Федеральное государственное бюджетное учреждение
«Национальный исследовательский центр «Курчатовский институт»

На правах рукописи

Баранов Антон Викторович

МЕТОДЫ И СРЕДСТВА УПРАВЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ В СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМАХ КОЛЛЕКТИВНОГО ПОЛЬЗОВАНИЯ

2.3.5 – Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
доктора технических наук

Москва – 2025

Оглавление

Список сокращений и условных обозначений

API	Application Programming Interface
HPC	High Performance Computing
MPI	Message Passing Interface
QoS	Quality of Service
БД	База данных
ВМ	Виртуальная машина
ВП	Вычислительный процессор
ВС	Вычислительная система
ВУ	Вычислительный узел
ВЧ	Вычислитель
КТ	Контрольная точка
ЛДП	Локальная дисковая память
МВС	Многопроцессорная вычислительная система
МСЦ РАН	Межведомственный суперкомпьютерный центр РАН
ОПП	Одна последовательная программа
ОС	Операционная система
ПК	Программный комплекс
ПО	Программное обеспечение
ПУВМ	Подсистема управления виртуальными машинами
РВС	Распределенная вычислительная система
СВЧ	Сервер вычислителя
СП	Связной процессор
ССРВ	Сетевая среда распределенных вычислений
СУЗ	Система управления заданиями
СУППЗ	Система управления прохождением параллельных заданий
СХД	Система хранения данных
ТРС	Территориально распределенная вычислительная среда

Введение

Суперкомпьютеры в 21 веке прочно вошли в практику научных исследований, как неотъемлемый инструмент сверхточного моделирования сложных природных явлений и общественных процессов. Примерами могут служить задачи гидродинамики, физики высоких энергий, астрофизические, биологические, медицинские и фармакологические исследования, наноэлектроника и синтез новых материалов, климатология, науки о Земле и океане, энергетика, а также задачи машинного обучения и искусственного интеллекта. С каждым годом сфера применения суперкомпьютерных технологий расширяется, растет сложность фундаментальных и прикладных вычислительных задач, увеличивается число пользователей суперкомпьютерных систем.

Уровень развития суперкомпьютерных технологий является для государства одним из факторов стратегического значения. Мировые лидеры в области науки, образования, промышленности и бизнеса – США, КНР, Европейский Союз, Япония – не первое десятилетие реализуют национальные и наднациональные проекты развития суперкомпьютерных технологий [1]. Целями подобных проектов являются получение конкурентных преимуществ [2, 3] за счет сокращения сроков научных исследований и разработок промышленных изделий, а также достижение технологического лидерства в таких областях, как создание новых материалов, энергетика, транспорт, медицина, обеспечение безопасности государства.

Создание инфраструктуры и условий для проведения научных исследований и разработок, внедрения наукоемких технологий, отвечающих современным принципам организации научной, научно-технической и инновационной деятельности, на основе лучших российских и мировых практик является одной из основных задач Стратегии научно-технологического развития Российской Федерации [4]. Современная научная инфраструктура немыслима без применения высокопроизводительных вычислительных систем, на основе которых реализуется такой приоритет научно-технологического развития, как переход к передовым технологиям проектирования и создания высокотехнологичной продукции. Важность

и актуальность развития суперкомпьютерных технологий в полной мере осознается российским научным сообществом [1, 5, 6]. Примерами развития и использования научной инфраструктуры мирового уровня являются суперкомпьютерные центры коллективного пользования МГУ им. М.В. Ломоносова [7], НИЦ «Курчатовский институт» [8], Российской академии наук [9, 10], Объединенного института ядерных исследований [11].

Основное предназначение высокопроизводительных вычислительных систем – решение таких научно-технических задач, нередко называемых «большими задачами», для которых недостаточно вычислительной мощности отдельного компьютера. Для ускорения решения большая задача разделяется на части, которые могут выполняться одновременно на разных вычислительных устройствах. Такое разделение называют распараллеливанием задачи, а одновременно выполняемые на разных устройствах вычисления соответственно называют параллельными вычислениями. Результатом распараллеливания является параллельная программа, т.е. программа, содержащая части, которые могут выполняться одновременно на разных устройствах. Распараллеливание больших задач, организация и производство параллельных вычислений образуют специальную технологию высокопроизводительных вычислений (англ. – High Performance Computing, HPC). В соответствии с ГОСТ Р 57700.27-2020 высокопроизводительные вычисления определяются как вычисления, выполнение которых требует большого объема расчетов и (или) обработки больших объемов данных за сравнительно небольшой промежуток времени, и, как правило, специальных вычислительных ресурсов. Такими специальными вычислительными ресурсами являются суперкомпьютерные системы, они же параллельные или HPC-системы.

В суперкомпьютерных центрах высокопроизводительные вычислительные системы, как дорогостоящее научное оборудование, используются главным образом в режиме коллективного пользования. Для производства высокопроизводительных расчетов пользователи формируют задания, каждое из которых включает расчетную параллельную программу, входные данные и требования к ресурсам.

Входной поток заданий последовательно проходит технологические этапы обработки, первоначально поступая в очередь одной из территориально распределенных суперкомпьютерных систем. После прохождения очереди каждому заданию выделяются вычислительные узлы суперкомпьютера, внутри которых запускаются процессы расчетной программы. В каждом узле эти процессы распределяются по вычислительным ядрам, а внутри ядра распараллеливаются выполняемые процессом отдельные вычислительные операции.

Таким образом, распараллеливание входного потока заданий осуществляется одновременно на нескольких иерархических уровнях, соответствующих этапам единой технологической цепочки обработки. Общая эффективность использования суперкомпьютерных систем очевидным образом зависит от эффективности управления вычислительными ресурсами на каждом технологическом этапе обработки (уровне распараллеливания) входного потока заданий. Постоянное усложнение архитектуры суперкомпьютеров, применение для их построения новейших технических решений, увеличение числа процессорных узлов и ядер, повышение степени их разнородности обуславливают актуальность развития существующих и создания новых методов и средств управления вычислительными ресурсами суперкомпьютерных систем коллективного пользования на разных уровнях распараллеливания входного потока заданий. При этом важно не только достигнуть эффективного распределения вычислительной работы по имеющимся ресурсам, но и обеспечить надежность и отказоустойчивость параллельных вычислений, поскольку рост степени распараллеливания ведет к соответствующему росту числа сбоев и отказов.

Поскольку доступ пользователей к вычислительным ресурсам суперкомпьютера обеспечивается посредством заданий, многие методы и алгоритмы управления этими ресурсами реализуются в виде специальных программных систем управления заданиями (СУЗ) [12]. Как отдельный вид системного программного обеспечения, СУЗ начали формироваться в середине 1990-х годов [13], научное осмысление проблематики построения СУЗ было завершено к 2000-м годам [14].

К этому времени в мировой практике применялось свыше 15 различных систем управления заданиями [15]. Разработка значительной части систем, рассмотренных в работе [15], к настоящему времени прекращена, но системы-лидеры, такие как SLURM [16] и IBM Platform LSF [17], активно развиваются, пополняя свой арсенал новыми функциональными возможностями. Несмотря на то, что часть ведущих систем, подобно SLURM, свободно распространяются в открытых исходных кодах, разработка и развитие этих систем осуществляются западными компаниями. Необходимость сохранения компетенций и научно-технологического паритета в области управления суперкомпьютерными ресурсами обуславливает актуальность исследований и разработок по созданию и развитию отечественных систем управления заданиями, обладающих набором функциональных возможностей, соответствующим сложившейся мировой практике. Для построения системы управления заданиями необходима разработка архитектуры, отражающей иерархический характер распараллеливания входного потока заданий. Другими словами, в основе архитектуры СУЗ должна лежать иерархическая модель управления вычислительными ресурсами, которая позволит интегрировать решения частных научно-технических задач на каждом технологическом этапе обработки заданий в единый комплекс архитектурных, технических и технологических решений.

Центральной функцией любой системы управления заданиями является их планирование, которое заключается в организации одной или нескольких очередей поступивших заданий и формировании расписаний их запусков. Качество планирования оценивается рядом показателей, таких как загрузка вычислительных ресурсов, среднее время ожидания задания в очереди, средний коэффициент замедления заданий и другими [18]. Показатели качества планирования являются взаимосвязанными и противоречивыми, оптимизация системы по одному показателю часто приводит к ухудшению значений другого, что делает актуальной задачу поиска баланса между разными показателями.

В большинстве суперкомпьютерных систем применяются методы планирования, базирующиеся на предоставляемых пользователями оценках времени вы-

полнения заданий и необходимого объема ресурсов, причем эти оценки не изменяются на протяжении жизненного цикла задания, т.е. являются фиксированными параметрами. Для методов планирования заданий с фиксированными параметрами характерна невытесняющая приоритетная дисциплина обслуживания с применением принципов справедливого распределения ресурсов [19, 20] и обратного заполнения [21-24]. Справедливое распределение означает обратную зависимость приоритета пользователя от объема потребленных его заданиями ресурсов. Принцип обратного заполнения разрешает запуск вне очереди некоторых заданий, если этот запуск не повлияет на время старта заданий, стоящих в очереди выше.

Разнообразие решаемых вычислительных задач порождает различные требования пользователей к объему и времени использования суперкомпьютерных ресурсов. В очереди одного суперкомпьютера могут одновременно находиться тестовые задания, требующие незначительного времени выполнения в целях отладки, и задания, сформированные для длительных расчетов. В исследованиях [25, 26] отмечается, что наличие в единой очереди множества разнородных заданий является одной из главных проблем планирования. При этом из-за разных размеров заданий в расписании запусков неизбежно образуются окна, ведущие к простою вычислительных узлов. Неточная оценка пользователями времени выполнения заданий [27] повышает стохастичность образования окон и снижает эффективность таких алгоритмов, как обратное заполнение. Заполнение динамических окон в расписании заданий с фиксированными параметрами возможно за счет адаптивных заданий [28-31], размеры которых подстраиваются по размеры появляющихся окон. В этом случае адаптивные задания образуют дополнительный поток, который необходимо эффективно совмещать с основным потоком.

Одним из широко распространенных методов повышения эффективности обработки данных является применение облачных вычислений, в основе которых лежат технологии гипервизорной и контейнерной виртуализации. Известные облачные платформы, такие как OpenStack, OpenNebula, предоставляют потребителям и поставщикам вычислительных ресурсов широкий спектр возможностей для

построения и применения облачных сервисов – предоставляемых по сети информационно-вычислительных услуг. Одним из важных направлений в этой области является создание облачных сервисов для высокопроизводительных вычислений [32-37]. Однако, при переносе суперкомпьютерных приложений в облачные платформы возникает ряд противоречий. Многолетняя практика применения СУЗ в суперкомпьютерных центрах привела к образованию полноценных цифровых экосистем, в которых исторически сложился порядок взаимодействия пользователей и персонала, выработаны политики предоставления ресурсов и обеспечения информационной безопасности, пользователями и системными администраторами накоплен значительный багаж инструментального программного обеспечения для взаимодействия с СУЗ. Прямой переход на облачные платформы нарушит целостность сложившейся цифровой экосистемы и повлечет значительные временные и трудовые затраты пользователей и персонала суперкомпьютерных центров. Выходом видится совмещение работы СУЗ и облачных платформ, однако, и те, и другие системы требуют монопольного управления вычислительными ресурсами, и это противоречие необходимо разрешать. Кроме этого, технологии виртуализации вносят существенные накладные расходы [32, 38, 39] и лишают пользователя непосредственного доступа к вычислительным ресурсам, что критически важно для большого числа суперкомпьютерных приложений. Перечисленные противоречия обуславливают актуальность исследований и разработок методов и средств совместного использования систем управления заданиями и облачных платформ.

Таким образом, разнообразие решаемых вычислительных задач, применение облачных вычислений и связанных с ними технологий виртуализации и контейнеризации приводит к появлению разнородных потоков заданий на входе суперкомпьютерной системы, что обуславливает актуальность исследований и разработок методов и средств повышения качества планирования заданий путем эффективного совмещения разнородных потоков заданий и поиска баланса между противоречивыми показателями качества.

Не менее важной функцией системы управления является распределение процессов прикладной параллельной программы по вычислительным узлам суперкомпьютера. Для каждого задания необходимо найти такое отображение информационного графа прикладной программы на граф вычислительных узлов, при котором минимизируется время, затрачиваемое процессами программы на информационные обмены. Эта задача в общем случае является NP-полной [40, 41], ее сложность многократно возрастает в связи с непрерывным ростом числа процессорных ядер и вычислительных узлов в суперкомпьютерных системах, и использование точных методов решения влечет высокие временные затраты даже для графов малых или средних порядков. Современные исследования [42-51] направлены на применение приближенных эвристических алгоритмов, при этом широко используются параллельные алгоритмы [49-53], а также различные комбинации базовых эвристик [48, 54]. В условиях режима коллективного пользования информационный граф программы и граф вычислительных узлов, как правило, неизвестны заранее, и задача поиска оптимального отображения должна решаться системой управления при каждом запуске задания и за ограниченное время. В такой постановке задача отображения в современных исследованиях либо не рассматривается, либо предлагаемые решения [55] фактически нарушают иерархию уровней управления. Анализ научных публикаций показывает актуальность исследований и разработок методов и алгоритмов решения задачи поиска оптимального отображения средствами системы управления заданиями за приемлемое время.

Повышение доступности и универсальности суперкомпьютеров расширяет круг пользователей. При этом возрастает доля исследователей, не являющихся профессиональными специалистами в области параллельных вычислений. С другой стороны, эффективная организация вычислительного процесса требует достаточно высокой квалификации пользователя, в распоряжении которого оказывается суперкомпьютерная система со сложной иерархической структурой. Наиболее ярко это противоречие проявляется для вычислительных задач с распараллеливанием по данным, где подготовка заданий и организация параллельных вычисле-

ний носят рутинный характер. Известные технологии распараллеливания по данным, такие как MapReduce [56, 57], BOINC [58, 59] или разработка НИВЦ МГУ им. М.В. Ломоносова программный комплекс X-COM [60, 61], освобождая пользователя от организации параллельных вычислений, тем не менее, подразумевают разработку специальных служебных программ, от эффективности которых существенно зависит итоговое быстродействие вычислений. Кроме этого, при распараллеливании по данным для учета обработанных порций данных в перечисленных решениях применяются специализированные базы данных, ведение которых вносит существенные накладные расходы. Исследование и разработка методов и средств, обеспечивающих распараллеливание по данным с меньшими по сравнению с известными решениями накладными расходами, также является актуальной научной задачей.

Таким образом, каждый этап обработки входного потока заданий требует поиска, разработки и внедрения новых научно обоснованных архитектурных, технических и технологических решений по повышению эффективности использования суперкомпьютерных систем.

Существенное влияние на развитие суперкомпьютерных технологий и их применение, включая создание, эксплуатацию и развитие суперкомпьютерных систем и центров коллективного пользования оказали работы советских и российских ученых Г.И. Савина, В.К. Левина, Б.М. Шабанова, Вл.В. Воеводина, В.Ф. Тюрина, В.В. Корнеева, А.О. Лациса, Н.Н. Миренкова, В.В. Коренькова. Исследования высокопроизводительных вычислительных систем методами теории массового обслуживания представлены в работах И.А. Соколова, В.Ф. Матвеева, В.А. Балыбердина, А.И. Костогрызова, А.С. Румянцева, Р.В. Разумчика. Создание и развитие математических методов, алгоритмов, системного, инструментального и прикладного программного обеспечения для эффективного управления суперкомпьютерными ресурсами и организации высокопроизводительных вычислений широко представлено в работах Б.Н. Четверушкина, А.И. Аветисяна, М.В. Якобовского, А.Н. Томилина, В.В. Топоркова.

Цель и задачи работы. Цель работы состоит в повышении эффективности использования суперкомпьютерных систем коллективного пользования за счет разработки комплекса архитектурных, технических и технологических решений для управления вычислительными ресурсами.

Для достижения поставленной цели необходимо решить следующие задачи:

− разработать иерархическую модель управления вычислительными ресурсами в суперкомпьютерной системе коллективного пользования, построить архитектуру и создать систему управления заданиями, обладающую качественными и количественными характеристиками на уровне мировых систем-лидеров;

− разработать методы и средства планирования заданий, обеспечивающие распределение по вычислительным ресурсам суперкомпьютера разнородных потоков пользовательских заданий различных классов;

− разработать решения по повышению быстродействия прикладных программ за счет оптимизации их отображения на структуру вычислительных узлов суперкомпьютера;

− разработать методы и технические решения организации параллельных вычислений с распараллеливанием по данным.

Методология и методы исследования. Результаты диссертации были получены с привлечением моделей и методов, используемых при поиске архитектурных и системных решений. Математическую основу исследования составляют методы теории массового обслуживания, теории алгоритмов, математической логики, теории графов.

Научная новизна работы.

Научная новизна выполненного исследования заключается в следующем:

− разработаны новые методы планирования, совмещающие обработку классов ординарных, отладочных, фоновых и адаптивных заданий, в том числе представленных в виде виртуальных машин и контейнеров;

− разработан новый двухэтапный метод отображения параллельной программы на вычислительные узлы суперкомпьютера: на первом этапе производит-

ся выделение вычислительных узлов для параллельной программы и тем самым сокращается размер задачи отображения, на втором этапе выделенные узлы используются для выполнения параллельного алгоритма поиска отображения;

– разработан новый эвристический алгоритм выделения вычислительных узлов для задания, основанный на разрезании графа свободных вычислительных узлов на два минимально связанных друг с другом подграфа при помощи имитации отжига;

– разработан новый параллельный алгоритм поиска отображения программного графа на граф вычислительных узлов с использованием циклической смены фаз имитации отжига и генетического отбора;

– разработан новый метод иерархического деления данных для организации параллельных вычислений с распараллеливанием по данным, за счет разделения входного пула данных на наборы упорядоченных элементарных вычислительных работ обеспечивающий компактное представление состояния вычислений и низкие накладные расходы на распараллеливание;

– разработана новая архитектура системы управления заданиями пользователей, основанная на иерархической модели управления вычислительными ресурсами суперкомпьютерной системы коллективного пользования.

Достоверность полученных результатов подтверждается реализацией разработанных решений в составе Системы управления прохождением параллельных заданий (СУППЗ) и положительным опытом ее практического применения для управления вычислительными ресурсами отечественных суперкомпьютерных систем МВС-1000, МВС-1000/16, МВС-1000/32, МВС-1000М, МВС-15000ВМ, МВС-6000IM, МВС-100К, МВС-10П, МВС-Экспресс, К-100, К-60. Эффективность предложенных автором методов и средств подтверждается данными статистики использования указанных высокопроизводительных систем, результатами сравнительных вычислительных экспериментов и имитационного моделирования.

Теоретическая и практическая значимость. Практическая значимость диссертации определяется тем, что разработанный и реализованный комплекс

решений по управлению вычислительными ресурсами суперкомпьютерных систем коллективного пользования в течение десятилетий обеспечивает проведение высокопроизводительных расчетов для пользователей-исследователей суперкомпьютерных центров коллективного пользования МСЦ РАН – НИЦ «Курчатовский институт», ИПМ им. М.В. Келдыша РАН, ФГУП НИИ «Квант» и других научных организаций.

Разработанные автором эвристические алгоритмы наименьшего разреза графа и поиска отображения программного графа на граф вычислительных узлов суперкомпьютера вносят вклад в решение фундаментальной квадратичной задачи о назначениях.

Теоретические положения и накопленный практический опыт, представленные в диссертации, могут служить основой дальнейшего развития методов и средств построения систем управления вычислительными ресурсами и организации параллельных вычислений в суперкомпьютерных центрах.

Работа является развитием достижений отечественной и мировой науки и практики по созданию системного программного обеспечения высокопроизводительных вычислительных систем, в том числе разработок научных школ С.А. Лебедева, В.А. Мельникова, В.К. Левина, В.С. Бурцева, А.В. Забродина, В.П. Иванникова, Н.Н. Говоруна, В.В. Воеводина, Э.В. Евреинова.

Положения, выносимые на защиту:

− метод планирования, совмещающий обработку классов ординарных, отладочных и фоновых заданий, позволяет снизить средний коэффициент замедления для отладочных и фоновых заданий при сохранении высокой загрузки вычислительных ресурсов;

− метод постпланирования, совмещающий поток заданий с фиксированными параметрами и поток адаптивных заданий, повышает загрузку суперкомпьютерной системы и минимизирует коэффициент замедления адаптивных заданий;

− двухэтапный метод и алгоритмы отображения параллельной программы на вычислительные узлы суперкомпьютера позволяют ускорить высокопроизво-

дительные расчеты, повышают точность и быстродействие поиска отображения по сравнению с известными методами и алгоритмами;

− метод иерархического деления данных позволяет организовать параллельные вычисления с распараллеливанием по данным с меньшими накладными расходами на распараллеливание по сравнению с известными решениями;

− система управления прохождением параллельных заданий составляет основу информационно-вычислительной среды (цифровой экосистемы) проведения высокопроизводительных научных расчетов в суперкомпьютерных центрах коллективного пользования, обеспечивает качество решения задач управления вычислительными ресурсами, соответствующее мировому уровню.

Апробация диссертации. Материалы диссертации докладывались на следующих международных и всероссийских конференциях:

1. Всероссийская научная конференция «Высокопроизводительные вычисления и их приложения», 30 октября - 2 ноября 2000 года, Черноголовка.

2. 1-я Международная научно-техническая конференция «Распределенные вычисления и Грид-технологии в науке и образовании (GRID’2004)», 29 июня – 2 июля 2004 г., Дубна.

3. 2-я Международная научно-техническая конференция «Распределенные вычисления и Грид-технологии в науке и образовании (GRID’2006)», 26-30 июня 2006 г., Дубна.

4. Всероссийская научная конференция «Научный сервис в сети Интернет: технологии параллельного программирования», 18-23 сентября 2006 г., Новороссийск.

5. Всероссийская научная конференция «Научный сервис в сети Интернет: многоядерный компьютерный мир. 15 лет РФФИ», 24-29 сентября 2007 г., Новороссийск.

6. Всероссийская научная конференция «Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность», 21-26 сентября 2009 г., Новороссийск.

7. 4-я Международная научно-техническая конференция «Распределенные вычисления и Грид-технологии в науке и образовании (GRID’2010)», 28 июня - 3 июля 2010 г., Дубна.

8. Всероссийская научная конференция «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи», 20-25 сентября 2010 г., Новороссийск.

9. 5-я Международная научно-техническая конференция «Распределенные вычисления и Грид-технологии в науке и образовании (GRID’2012)», 16-21 июля 2012 г., Дубна.

10. Всероссийская научная конференция «Научный сервис в сети Интернет: поиск новых решений», 17-22 сентября 2012 г., Новороссийск.

11. Всероссийская научная конференция «Научный сервис в сети Интернет: все грани параллелизма», 23-28 сентября 2013 г., Новороссийск.

12. Всероссийская научная конференция «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров», 22-27 сентября 2014 г., Новороссийск.

13. Всероссийская научно-техническая конференция «Суперкомпьютерные технологии СКТ-2014», 29 сентября - 4 октября 2014 г., Дивноморское, Геленджик.

14. Национальный Суперкомпьютерный Форум (НСКФ-2015), 24-27 ноября 2015 г., Переславль-Залесский.

15. Международная конференция «Суперкомпьютерные дни в России», 26-27 сентября 2016 г., Москва.

16. Национальный Суперкомпьютерный Форум (НСКФ-2016), 29 ноября - 02 декабря 2016 г., Переславль-Залесский.

17. 14th International Conference on Parallel Computing Technologies, September 4-8, 2017, Nizhni Novgorod, Russia.

18. Международная конференция «Суперкомпьютерные дни в России», 25-26 сентября 2017 г., Москва.

19. 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 29 января - 1 февраля 2018 г., Москва.

20. Национальный Суперкомпьютерный Форум (НСКФ-2018), 27-30 ноября 2018 г., Переславль-Залесский.

21. 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019, 01-04 сентября 2019 г., Leipzig.

22. 5-я Международная научно-практическая конференция «Информационные технологии и высокопроизводительные вычисления», 16-19 сентября 2019 г., Хабаровск.

23. Международная конференция «Суперкомпьютерные дни в России», 23-24 сентября 2019 г., Москва.

24. 13th International Scientific Conference «Parallel computational technologies (PCT) 2020», March 31 - April 2, 2020, Perm, Russia.

25. Научная конференция в рамках международного конгресса «Суперкомпьютерные дни в России», 21-22 сентября 2020 г., Москва.

26. 16th International Conference «Parallel Computing Technologies (PaCT 2021)», September 13–18, 2021, Kaliningrad, Russia.

27. Научная конференция в рамках международного конгресса «Суперкомпьютерные дни в России», 27-28 сентября 2021 г., Москва.

28. Национальный Суперкомпьютерный Форум (НСКФ-2021), 30 ноября - 3 декабря 2021 г., Переславль-Залесский.

29. Международная конференция «Суперкомпьютерные дни в России», 26-27 сентября 2022 г., Москва.

30. Международная конференция «Суперкомпьютерные дни в России», 23-24 сентября 2024 г., Москва.

Кроме конференций, результаты диссертации докладывались на семинарах Отделения суперкомпьютерных систем и параллельных вычислений НИЦ «Курчатовский институт», Федерального исследовательского центра «Информатика и управление» Российской академии наук, Научно-исследовательского вычислительного центра МГУ им. М.В. Ломоносова.

Публикации и личный вклад автора.

Все выносимые на защиту результаты получены соискателем лично.

По теме диссертации автором опубликовано 45 печатных работ, из них 27 работ опубликованы в изданиях, рекомендованных ВАК. По тематике исследований оформлено 6 свидетельств о государственной регистрации баз данных и программ для ЭВМ.

Структура и объем работы. Диссертация состоит из введения, пяти глав и заключения. Содержание работы изложено на 295 страницах машинописного текста. Список использованных источников составляет 242 наименования.

В первой главе раскрывается содержание основных задач управления вычислительными ресурсами суперкомпьютерной системы коллективного пользования, формулируются требования к системам управления заданиями, приводится краткий обзор этих систем. Центральным местом главы является иерархическая модель управления суперкомпьютерными ресурсами, на основе которой предложена архитектура СУППЗ. В главе показано соответствие архитектуры СУППЗ выдвинутым требованиям, а также отражены особенности архитектуры для отечественных массивно-параллельных систем серии МВС-1000.

Во второй главе представлен обзор комплекса предложенных в составе СУППЗ технических и технологических решений по управлению суперкомпьютерными ресурсами, представленных на каждом уровне иерархической модели, произведено сравнение качественных и количественных характеристик СУППЗ и ведущих мировых систем управления заданиями, представлены результаты эксплуатации СУППЗ на ведущих отечественных суперкомпьютерах.

Главы 3-5 посвящены решению частных научных задач диссертации: разработке методов и средств планирования заданий, отображения параллельных программ на вычислительные узлы суперкомпьютера, организации параллельных вычислений с распараллеливанием по данным. В каждой главе предлагаемые решения предваряются анализом состояния предметной области решаемой научной задачи и обзором соответствующих этой области научных работ.

В заключении сформулированы основные результаты диссертационной работы.

Автор выражает глубокую благодарность своим учителям Б.М. Шабанову, В.В. Корнееву, А.О. Лацису, А.В. Киселёву, В.А. Тихонову, коллегам и соавторам научных публикаций А.Н. Сотникову, О.С. Аладышеву, П.Н. Телегину, А.П. Овсянникову, М.Ю. Храмцову, С.В. Шарфу, С.А. Дбар, сотрудникам коллективов МСЦ РАН и ИПМ им. М.В. Келдыша РАН, а также своим ученикам Е.А. Киселёву, Д.С. Ляховцу, А.И. Тихомирову за неоценимую помощь в подготовке диссертации.

Глава 1. Иерархическая модель управления вычислительными ресурсами и архитектура системы управления заданиями пользователей суперкомпьютера

1.1 Типовая архитектура суперкомпьютерной системы коллективного пользования

В 1990 г. Р. Дунканом было дано следующее актуальное по сей день определение параллельной архитектуры компьютера [62], как способа организации вычислительной системы, при котором допускается, чтобы множество процессоров (простых или сложных) могло бы работать одновременно, взаимодействуя по мере надобности друг с другом. Это определение в явном виде выделяет в архитектуре параллельного компьютера вычислительную и коммуникационную подсистемы. На современном этапе развития в роли условного «сложного» процессора выступает вычислительный узел, представляющий собой многопроцессорный (многоядерный) сервер. Вычислительные узлы объединяются в единый параллельный компьютер (суперкомпьютер) высокопроизводительной локальной сетью.

Множество вычислительных узлов и объединяющая их коммуникационная среда многократно увеличивают стоимость построения параллельной вычислительной системы по сравнению с персональным компьютером или сервером. Кроме этого, следует учитывать, что параллельные компьютеры имеют значительно меньшую серийность, чем обычные вычислительные системы, и, как следствие, являются достаточно дорогостоящими вычислительными установками. В эксплуатации параллельный компьютер потребляет значительное количество электроэнергии и выделяет большие объемы тепла, которое необходимо отводить при помощи подсистемы охлаждения. Все это делает невозможным функционирование параллельного компьютера вне специализированного центра обработки данных, обычно называемого суперкомпьютерным центром, эксплуатация и развитие оборудования которого обеспечивается высококвалифицированным персоналом.

Основным режимом эксплуатации суперкомпьютерных систем как дорогостоящего научного оборудования является в режим коллективного пользования. Режим коллективного пользования подразумевает одновременную работу в суперкомпьютерной системе как параллельном компьютере множества пользователей, которым суперкомпьютерный центр оказывает услуги по высокопроизводительным вычислениям. При этом для пользователей суперкомпьютерная система представляется в виде некоторой типовой архитектуры, представленной на рисунке 1.

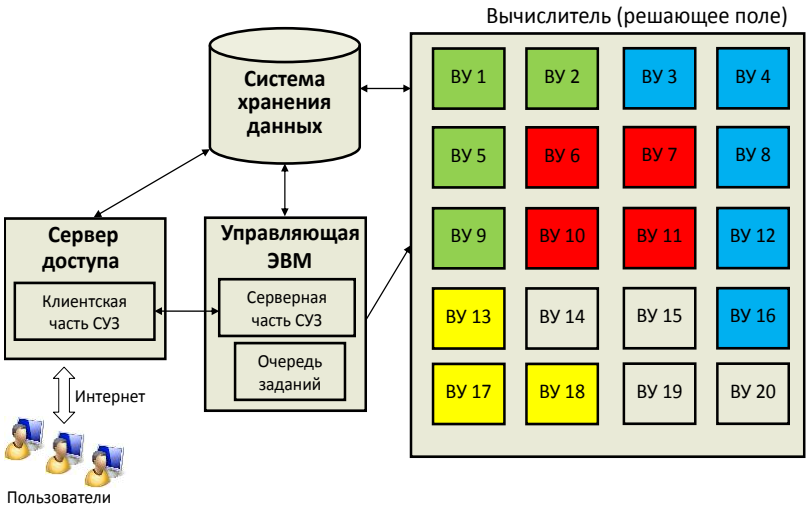

Рисунок 1. Типовая архитектура суперкомпьютерной системы коллективного пользования

Главным элементом суперкомпьютерной системы является **вычислительный узел** (ВУ). На разных этапах развития суперкомпьютерной техники вычислительный узел представлял собой разные технические решения. В массивно-параллельных системах 1990-х годов это был модуль (электронная плата) с микропроцессорами для вычислений и коммуникаций. Примером отечественных массивно-параллельных компьютеров могут служить многопроцессорные вычислительные системы МВС-100 и МВС-1000 [63, 64], разрабатывавшиеся НИИ «Квант» и рядом институтов РАН в период 1994-1999 гг. Вычислительный узел в этих системах представлял собой двухпроцессорный модуль, как показано на рисунке 2.

Один из процессоров модуля выполнял вычислительные функции и назывался вычислительным процессором. На другой процессор были возложены

функции информационного обмена, поэтому он назывался связным. Процессоры ВУ имели как общую разделяемую, так и свою собственную оперативную память. Идентификация ВУ в системе происходила по уникальному целочисленному номеру от 0 до N. Каждый СП имел несколько линков, посредством которых ВУ объединялись в сеть, составлявшую коммуникационную среду параллельного компьютера.

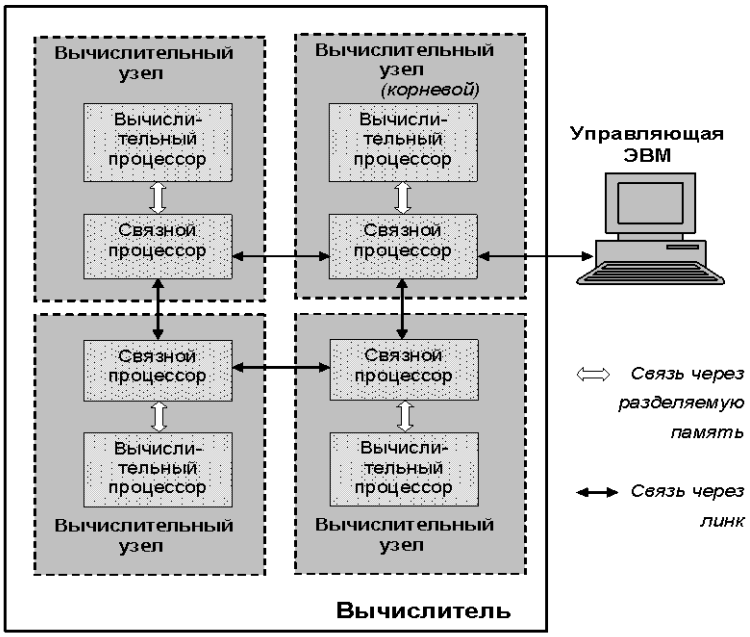

Рисунок 2. Архитектура многопроцессорных систем МВС-100 и МВС-1000

С наступлением эпохи кластерных вычислительных систем в качестве ВУ стал выступать высокопроизводительный сервер с одним или несколькими процессорами в составе. С распространением многоядерных микропроцессоров ВУ стали представлять собой многоядерные серверы с большим объемом оперативной памяти, т.е. фактически SMP-системы. При оснащении ВУ двумя или более микропроцессорами он превращался в систему NUMA. Наконец, в последнее десятилетие широко распространились гибридные решения, когда ВУ, помимо центральных универсальных процессоров, оснащается одним или более ускорителями вычислений, как правило, на базе высокопроизводительных графических процессоров.

Отличительными чертами современного ВУ являются:

– наличие собственной оперативной памяти и, как правило, собственного жесткого диска;

– ВУ управляется собственной операционной системой (в современных системах в качестве ОС в подавляющем большинстве случаев используется Linux), взаимодействуя с другими ВУ только по сети;

– ВУ имеет собственное уникальное сетевое имя, однозначно идентифицирующее его среди других ВУ.

ВУ объединяются друг с другом посредством высокоскоростных низколатентных коммуникационных сетей, часто называемых в литературе интерконнектом [65]. Обзор современных итерконнектов приведен в диссертации [66]. Среди интерконнектов выделяются серийно выпускаемые сети Ethernet 10G/25G/40G/100G, Infiniband EDR/FDR, OmniPath (OPA), Aries Interconnect, а также сети собственной разработки, например, Tofu Interconnect, Tianhe Express. В системе МВС-1000 интерконнектом служила сеть связных процессоров, объединенных линками.

Управление вычислителем, в т.ч. распределением вычислительной работы между ВУ осуществляет специально выделенный сервер – управляющая ЭВМ. В системе МВС-1000 управляющая ЭВМ, как показано на рисунке 2, через специальный адаптер соединялась с одним из линков нулевого (корневого) узла. В современных кластерных системах для целей управления решающим полем, как правило, добавляется отдельная управляющая сеть.

Доступ пользователей к параллельному компьютеру происходит через отдельный сервер доступа, к которому пользователи подключаются через Интернет при помощи собственных оконечных устройств: рабочих станций, мобильных телефонов, планшетов, ноутбуков и т.п. В системах с относительно малым числом ВУ функции управляющей ЭВМ и сервера доступа могут объединяться на одной машине.

Каждому пользователю выделяется определенное дисковое пространство в системе хранения (СХД) данных суперкомпьютерного центра. Это пространство монтируется на всех ВУ, сервере доступа и управляющей ЭВМ в каталоги с од-

ними и теми же именами, что позволяет пользователю иметь одинаковый доступ к своим файлам на всех компонентах параллельного компьютера.

1.2 Задания пользователей суперкомпьютерной системы

1.2.1 Понятие задания. Жизненный цикл задания

В системе коллективного пользования доступ к разделяемым суперкомпьютерным ресурсам должен быть упорядочен. С этой целью для получения доступа к решающему полю пользователь оформляет т.н. **задание** (англ. – job) – специальный информационный объект, в состав которого в общем случае входят:

– **параллельная программа**, реализующая алгоритм решения прикладной задачи и состоящая из нескольких взаимодействующих процессов (потоков), которые могут одновременно выполняться на нескольких ВУ (процессорных ядрах);

– требования к **параллельному ресурсу**, под которым мы будем понимать подмножество ВУ (процессорных ядер), выделяемое заданию на определенное время, при этом величина параллельного ресурса (число ВУ или ядер и время выполнения) называется **размером** задания (параллельного ресурса);

– **входные данные** параллельной программы.

Часто оперируют понятием **площади** задания (параллельного ресурса), под которой понимают произведение требуемого числа ВУ (процессорных ядер) и требуемого времени выполнения.

Кроме требований к размеру параллельного ресурса, задание может включать следующие дополнительные требования:

– тип ВУ в гетерогенных системах;

– подмножество ВУ с конкретными именами (идентификаторами);

– состав необходимого программного обеспечения – требуемый программный стек;

– необходимое окружение для процессов параллельной программы;

– сценарии запуска и завершения параллельной программы или нескольких параллельных программ;

– наличие связей с другими заданиями;

– иные требования.

Говорят, что совокупность требований определяется параметрами задания, которые оформляются пользователем в виде некоторого электронного документа, называемого паспортом задания. Разные задания различных пользователей образуют входной поток, обработку которого осуществляют специальные программные **системы управления заданиями** (СУЗ) [12].

Жизненный цикл задания включает следующие этапы.

1. Создание пользователем паспорта задания.

2. Направление пользователем задания на вход СУЗ (англ. – job submit).

3. Нахождение задания в очереди СУЗ.

4. Инициализация выделенных заданию ВУ после прохождения очереди.

5. Запуск задания на выделенных для него ВУ, заключающийся в порождении необходимых процессов (потоков) прикладной параллельной программы.

6. Выполнение прикладной параллельной программы.

7. Завершение задания, заключающееся в завершении запущенных на выделенных ВУ процессов (потоков) параллельной программы.

Каждый из этапов жизненного цикла требует ненулевого времени для своего выполнения. Наиболее длительными обычно являются этап 3 нахождения задания в очереди и этап 6 выполнения задания. Как будет показано в п.2.3 длительность именно этих этапов во многом определяет показатели качества СУЗ.

Следует отметить, что на этапе 3 пользователь имеет возможность удаления задания из очереди СУЗ, а на этапах 4-6 пользователь может досрочно завершить выполняющееся задание, что приводит к немедленному выполнению этапа 7.

1.2.2 Классификации пользовательских заданий

Задания пользователей суперкомпьютерной системы можно разбить на классы по различным основаниям. От класса обрабатываемых заданий часто зависит тот или иной тип СУЗ. Рассмотрим несколько распространенных классификаций заданий.

Если в основании классификации положить **изменяемость или неизменяемость** параметров задания, то задания можно разделить на два больших класса: задания с фиксированными и нефиксированными параметрами. Главными параметрами являются требуемое для выполнения задания число ВУ и требуемое время выполнения задания, т.е. параметры, задающие размер задания. **Задания с фиксированными параметрами** не изменяют их в течение всего жизненного цикла, в то время как задания с нефиксированными параметрами могут изменять требования к параллельному ресурсу на разных этапах жизненного цикла.

Как правило, изменение требований к параллельному ресурсу связано с необходимостью адаптации задания к текущему состоянию вычислений в суперкомпьютерной системе, например, к числу свободных ВУ. Такие задания назовем **адаптивными** [67] или **эластичными** (англ. – moldable jobs) [68]. Адаптивные задания могут быть выполнены на параллельном ресурсе произвольного размера в некотором заданном диапазоне. Как правило, для эластичных заданий неизменным остается площадь требуемого параллельного ресурса. Условно, такое задание может быть выполнено на 100 ВУ за 10 часов или на 10 ВУ за 100 часов. По этой причине адаптивные задания в литературе часто называют масштабируемыми.

Обработку **прерываемых (фоновых) заданий** [69] можно вести небольшими порциями – квантами. Как правило, фоновые задания требуют значительного времени для выполнения, которое не может быть одноразово предоставлено в режиме коллективного пользования. Фоновые задания сохраняют в процессе выполнения контрольные точки, позволяющие возобновить вычисления после прерывания. СУЗ, поддерживающие фоновые задания, периодически снимают их с выполнения, возвращая в очередь. Квант (единичная порция времени) выполне-

ния прерываемого задания зависит от алгоритма решения прикладной задачи и может составлять секунды, минуты и десятки минут, при этом общее время выполнения такого задания может измеряться часами и сутками.

По **соотношению этапов инициализации и выполнения задания** можно разделить на два класса: с коротким и длительным временем инициализации. Коротким временем инициализации обычно пренебрегают при расчетах показателей качества СУЗ. Под длительным временем инициализации будем понимать время, которым нельзя пренебречь по сравнению со временем выполнения. Можно выделить следующие типовые категории заданий с длительным временем инициализации [70].

1. Для краткосрочного задания требуется инициализация высокопроизводительного ускорителя (графического процессора или ПЛИС), при этом время инициализации ускорителя сравнимо со временем обработки задания [71]. Время инициализации может включать в себя время перепрограммирования ПЛИС или компиляции программы для графического процессора.

2. Для задания необходима специфическая программная платформа, развёртывание такой платформы перед стартом задания может занять существенное время [72, 73]. Время инициализации может включать в себя время запуска виртуальных машин, подготовки контейнеров, перезагрузки вычислительного узла.

3. Для задания необходимо до начала обработки загрузить на вычислительные узлы значительный объём входных данных.

Для обработки заданий с длительным временем инициализации необходимо применение специальных методов, таких как объединение однотипных заданий в группы (пакеты) [74].

По **требованиям к составу аппаратно-программной платформы** задания могут разделяться на **стандартные** и **нестандартные**. Стандартные задания используют для выполнения своих параллельных программ так называемый «стандартный» стек программного обеспечения для высокопроизводительных вычислений. Это стек, как правило, включает в себя [66, 75]:

– на нижнем уровне: операционные системы вычислительных узлов с драйверами адаптеров (контроллеров) высокоскоростной коммуникационной среды;

– над уровнем операционных систем вычислительных узлов через драйверы коммуникационной среды выстраивается слой коммуникационных библиотек для взаимодействия процессов параллельных программ, по факту – различные реализации стандарта MPI;

– следующим уровнем можно назвать инструментальные средства разработки параллельных программ, включающие библиотеки для высокопроизводительных расчетов, в том числе библиотеки прикладных программных пакетов;

– высшим уровнем является прикладная параллельная программа пользователя, включаемая им в состав стандартного задания.

Важно, что для запуска стандартного задания не требуются особые операционные системы, либо другие специфические компоненты стека программного обеспечения. Однако, как отмечается в исследовании [76], в суперкомпьютерных центрах коллективного пользования всё чаще появляются **нестандартные задания**, предъявляющие особые требования к ресурсам. Подобным заданиям могут потребоваться отличная от стандартной операционная система (например, MS Windows), специфическое окружение, особые программные пакеты и лицензии, то есть **собственная программная платформа (среда)**. Методы совмещения потоков стандартных и нестандартных заданий рассмотрены в п. 3.5

По **связности процессов параллельной программы** задания делятся на задания **с взаимодействующими процессами** и задания **с распараллеливанием по данным.** В заданиях с взаимодействующими процессами последние осуществляют друг с другом информационные обмены промежуточными результатами вычислений. Такие задания соответствуют классической схеме параллельных вычислений, подразумевающей смену фаз расчетов и коммуникаций. Для заданий с взаимодействующими процессами актуальна задача поиска оптимального отображения информационного графа параллельной программы на подмножество вы-

деленных для задания ВУ суперкомпьютера. Предлагаемый автором метод решения этой задачи представлен в главе 4.

В случае распараллеливания по данным процессы параллельной программы не взаимодействуют друг с другом и осуществляют одновременную обработку независимых порций входных данных. Предлагаемый автором метод иерархического разделения данных при организации параллельных вычислений с распараллеливанием по данным рассмотрен в главе 5.

По **способу учета приоритета** можно выделить задания с абсолютными и относительными приоритетами. При использовании **абсолютных приоритетов** [77] вновь поставленное в очередь задание с высшим приоритетом вытесняет с выполнения задания с низшим приоритетом. **Относительные приоритеты** заданий означают, что задание с высоким приоритетом только опережают в очереди задания с низким приоритетом, но не имеют права прерывать уже выполняющиеся низкоприоритетные задания. В подавляющем большинстве СУЗ применяется планирование заданий с относительными приоритетами.

1.3 Задачи управления вычислительными ресурсами суперкомпьютера при организации режима коллективного пользования

В монографии [78] рассмотрена иерархическая модель управления многопроцессорными вычислительными системами (МВС). В этой модели предполагается, что в МВС могут быть выделены отдельные подсистемы, и на каждую выделенную подсистему может быть назначен определённый вид вычислительных работ. Состояние МВС, связанное с определённым разбиением на подсистемы и назначением на подсистемы определённых видов работ, называется функциональным или **F-состоянием** системы. Периодически в такой системе необходимо производить перераспределение ресурсов, т.е. производить изменение F-состояния системы, в чем собственно и заключается сущность управления вычислительными ресурсами.

На вход системы поступает поток заданий на выполнение, которые могут ждать освобождения вычислительных ресурсов в одной или нескольких входных очередях. Вычислительные процессы, протекающие на выделенных подсистемах, могут находиться в разных состояниях (выполнения, ожидания, зацикливания) и выдавать разные запросы к СУЗ. Текущее F-состояние системы, состояние вычислений процессов на выделенных подсистемах, текущее содержимое входных очередей определяют текущую **мультипрограммную ситуацию**.

Рассмотрим основные задачи управления вычислительными ресурсами, которые решаются при организации режима коллективного пользования в суперкомпьютерном центре.

1. **Подготовка пользователями программ и данных для высокопроизводительных расчетов**. Размещая программы и данные в СХД суперкомпьютерного центра, на сервере доступа пользователи осуществляют редактирование входных данных, исходных текстов программ, трансляцию и сборку программ, подготовку их к выполнению, которая заключается в оформлении соответствующего задания в виде паспорта. Паспорт задания, как правило, представляет собой текстовый файл определенного формата. Паспорт задания направляется на вход СУЗ, которая, как показано на рисунке 1, разделяется на клиентскую и серверную части. Клиентская часть представляет собой набор команд или API, выполняемых пользователями на сервере доступа. Серверная часть функционирует на управляющей ЭВМ и реализует дальнейший порядок обработки задания.

2. Множество различных заданий разного размера от разных пользователей образует **входной поток заданий**, который СУЗ должна принять и обработать. Во время приема очередного задания СУЗ осуществляет:

– авторизацию пользователя, направившего задание в СУЗ;

– проверку правильности оформления паспорта задания;

– проверку действительности квот пользователя на доступ к запрашиваемым вычислительным узлам;

– присваивание заданию уникального имени или иного идентификатора;

– фиксацию в системных журналах времени поступления задания, его имени, пользователя и основных характеристик.

3. Велика вероятность возникновения ситуации, когда для очередного задания из входного потока в определенный момент времени требуемый параллельный ресурс будет занят. В этом случае СУЗ должна обеспечить **ведение очереди заданий**, которое осуществляется, как правило, непосредственно на управляющей ЭВМ. На рисунке 1 цветом выделены ВУ, занятые разными заданиями (один цвет соответствует одному заданию), а свободные ВУ показаны неокрашенными. Важнейшей задача СУЗ при ведении очереди **планирование заданий** (англ. – job scheduling), под которым понимают составление расписания запусков заданий на решающем поле. Обзор применяемых в известных СУЗ методов планирования, а также предлагаемые автором методы планирования заданий представлены в главе 3.

4. После прохождения очереди СУЗ производит **выделение заданию требуемых вычислительных ресурсов**, для чего необходимо выполнить следующие действия:

– выбрать для задания ВУ из числа свободных;

– проверить работоспособность выбранных ВУ;

– провести конфигурацию выбранных ВУ;

– предоставить доступа пользователю – владельцу задания на выделенные узлы;

– зафиксировать факт старта задания в системных журналах.

Выбор ВУ из состава свободных производится в соответствии с требованиями задания и далеко не всегда является тривиальным. Если ВУ однородны, выбор может сводиться к поиску наиболее связного подмножества узлов, чтобы обеспечить высокую скорость информационных обменов. Например, в работе [79] рассмотрена задача выбора узлов в суперкомпьютерной системе на базе отечественной коммуникационной среды «Ангара», которая решается таким образом, чтобы сетевой трафик разных заданий не пересекался.

В случае гетерогенного вычислителя могут применяться различные критерии выбора. Например, можно подобрать заданию ВУ, удовлетворяющие требованиям задания по минимуму, а можно, наоборот, выделять в первую очередь самые мощные узлы, чтобы задание выполнилось как можно быстрее.

На этапе проверки работоспособности ВУ производится диагностика узлов, в ходе которой тестируются доступность узла и СХД, а также готовность основных программных и аппаратных подсистем. Сценарий тестирования разрабатывается системным администратором с учетом нюансов, присущих конкретному суперкомпьютерному центру. Если какой-либо ВУ не проходит диагностику, он помечается как неисправный, выводится из состава решающего поля, а для задания подбирается другой, исправный узел.

Конфигурация выбранных ВУ для задания преследует две цели: предоставление заданию требуемой программной платформы и обеспечение **взаимной изоляции заданий.**

Предоставление заданию требуемой программной платформы соответствует этапу инициализации жизненного цикла задания и может заключаться в следующих видах подготовки ВУ:

– формировании программного окружения для используемых заданием прикладных программных пакетов;

– подготовке набора прикладных или коммуникационных библиотек, иных инструментальных средств, упакованных в контейнер;

– запуске отдельного экземпляра операционной системы ВУ в виде виртуальной машины;

– перезагрузке ВУ с переключением на другую операционную систему;

– инициализации ускорителя вычислений на базе ПЛИС или графического процессора для выполнения алгоритма решения прикладной задачи.

Взаимная изоляция заданий подразумевает конфигурирование выделенных заданию ВУ таким образом, чтобы доступ к этим ВУ мог получить только пользователь – владелец задания. Кроме того, должно быть исключено влияние выпол-

няющихся процессов одного задания на выполнение процессов другого. Под влиянием здесь понимается не только злонамеренное или случайное воздействие на выполняющийся процесс (например, посылка сигнала), но и конкуренцию процессов за ресурсы ВУ (процессорные ядра, оперативная память, графический ускоритель, устройства ввода-вывода и др.), приводящую к взаимному ожиданию процессов и итоговому снижению быстродействия вычислений.

5. После выделения заданию вычислительных ресурсов СУЗ производит **запуск задания, поддержку и контроль его выполнения**. Запускающиеся процессы – ветви параллельной программы – должны быть распределены по выделенным для задания ВУ так, как потребовал пользователь в паспорте задания.

Поддержка выполнения задания заключается в предоставлении пользователю актуальной и достоверной информации о статусе и состоянии задания, возможности интерактивного взаимодействия с процессами задания, в том числе – возможности доступа к стандартным потокам вывода и ошибок процессов задания.

Контроль выполнения задания производится в целях своевременного обнаружения анормального поведения процессов задания. Как минимум, СУЗ отслеживает время выполнения задания, которое не должно превышать заказанное пользователем время, указанное в паспорте при постановке в очередь. При превышении заказанного времени выполнения подавляющее большинство СУЗ принудительно завершает процессы задания. Другим анормальным поведением задания, которое может отслеживаться СУЗ, является «зависание» в результате дедлоков или ливлоков, либо аварийное завершение процессов параллельной программы. В этих случаях выполнение задания также принудительно завершается.

6. По завершении задания СУЗ **освобождает занятые заданием вычислительные узлы**. Освобождение ресурсов во многом зеркально процессу выделения ВУ и включает:

– прекращение доступа пользователя-владельца задания к занятым заданием ВУ;

– прекращение выполнения всех процессов задания на каждом выделенном для него ВУ;

– освобождение захваченных заданием разделяемых ресурсов ВУ, таких как очереди и семафоры IPC, области разделяемой памяти, временные файлы и другие;

– реконфигурацию вычислительных узлов, подразумевающую их возврат в то состояние, которое было до запуска задания;

– фиксацию факта и статуса завершения задания в системных журналах.

Следует отметить, что освобождение занятых заданием вычислительных ресурсов является отнюдь не тривиальным процессом. Поскольку на алгоритм и код параллельной программы не накладывается никаких ограничений, ее процессы «имеют право» совершать любые не запрещенные операционной системой анормальные действия, в том числе «зависать» самим и «подвешивать» операционную систему, не возвращать или терять занятую память, бесконтрольно размножаться и т.п. Если СУЗ не удается пресечь анормальные действия процессов задания и корректно завершить их выполнение, то ВУ либо перезагружается, либо помечается как аварийный и выводится из состава решающего поля.

7. Задание считается покинувшим СУЗ после освобождения ВУ. Однако, даже после того, как завершенное задание покидает СУЗ, необходимо **обеспечить доступ пользователя к результатам проведенных расчетов**. Обычно эти результаты автоматически сохраняются в СХД суперкомпьютерного центра, но бывают такие конфигурации параллельных компьютеров, которые требуют копирования результатов с локальных дисков ВУ в общую СХД. В любом случае пользователь должен быть проинформирован, в каком месте файловой системы размещены результаты выполненных расчетов.

8. Во время работы параллельного компьютера ведется **непрерывный мониторинг состояния его вычислительных ресурсов.** Помимо отслеживания работоспособности, готовности и загруженности ВУ решающего поля, систем энергоснабжения и охлаждения, подсистема мониторинга может в автоматическом

или автоматизированном режиме отслеживать эффективность использования вычислительных ресурсов каждым заданием. Некоторые системы мониторинга [80] способны выдавать пользователям рекомендации по оптимизации выполнения их заданий. Немаловажной функцией подсистемы мониторинга является наглядное интерактивное графическое отображение состояния подсистем параллельного компьютера [81], помогающее администраторам осуществлять оперативный контроль работоспособности и эффективности использования вычислительных ресурсов.

9. **Учет потребления пользовательскими заданиями ресурсов параллельного компьютера**. С этой целью СУЗ в том или ином виде ведет базу данных, способную собирать и обрабатывать статистическую информацию о работе параллельного компьютера [82]. В такой базе данных сохраняется информация об обработанных заданиях, включая время поступления и характеристики каждого задания, времена его старта и завершения, изменения в составе и статусе ВУ, изменения параметров планирования заданий и др. Пользователям и администрации суперкомпьютерного центра база данных предоставляет статистические отчеты, отражающие за заданный период объемы потребленных ресурсов каждым заданием, пользователем, группой пользователей, организацией. Кроме этого, в базу данных могут записываться данные подсистемы мониторинга, отражающие степень загруженности и эффективности использования вычислительных ресурсов.

1.4 Требования к системам управления заданиями

Формирование систем управления заданиями как отдельного вида системного программного обеспечения происходило в 1990-е годы. Необходимость появления подобных систем, способных решать указанные в п.1.3 задачи управления вычислительными ресурсами, назрела в связи с расширением круга пользователей суперкомпьютеров именно как параллельных вычислительных систем. Как справедливо отмечается в монографии [14], к середине 1990-х годов сформировалась типовая архитектура параллельного компьютера, представленная на рисунке

1, и наблюдалось бурное развитие технологий организации параллельных вычислений для компьютеров подобной архитектуры, связанное прежде всего с появлением множества реализаций стандарта MPI (Message Passing Interface) [65], который де-факто стал фундаментом для большинства прикладных параллельных программ и библиотек. Реализации MPI содержат средства запуска и развертывания процессов параллельной программы на подмножестве ВУ суперкомпьютера, но не решают при этом ни одной из указанных в п. 1.3 управленческих задач. Рост числа суперкомпьютерных систем и их пользователей потребовал организации режима коллективного пользования и создания СУЗ как неотъемлемого элемента стека системного программного обеспечения.

Одной из первых работ, в которой при участии автора были сформулированы принципы организации и требования к СУЗ, стал доклад [83]. В дальнейшем эти требования были уточнены в монографии А.О. Лациса [14] и работе [84]. Обобщим и систематизируем эти требования в виде следующего перечня.

1. Универсальность. На вид и характер пользовательских прикладных программ не накладывается никаких ограничений. Зарегистрированный в системе пользователь с помощью доступных инструментальных средств имеет право и возможности разработки и выполнения любой программы. Требование универсальности применимо также к обслуживаемым СУЗ вычислительным системам: СУЗ должна быть способна функционировать на любом суперкомпьютере, соответствующем рассмотренной в п.1.1 типовой архитектуре.

2. Автоматическое функционирование и круглосуточная доступность. Функционирование СУЗ в штатном режиме не должно предусматривать ручного вмешательства оператора или системного программиста (администратора). Последние должны воздействовать на СУЗ только в случае реконфигурации или аварийных ситуаций. Соответственно, автоматическое функционирование СУЗ подразумевает, что обслуживание пользовательских заданий должно производиться 24 часа в сутки и 7 дней в неделю, за исключением вынужденных простоев и плановой профилактики.

3. Гарантированное обслуживание без потерь. СУЗ должна гарантировать, что поступившее на ее вход задание не будет потеряно (уничтожено) и рано или поздно будет выполнено.

4. Императивность управления. Наиболее полно принцип императивности управления был сформулирован в монографии [14]. Принцип постулирует техническую невозможность таких действий пользователя и процессов его прикладной программы, которые могут потенциально привести к нарушению работы как других прикладных программ, так и всей суперкомпьютерной системы. Как справедливо отмечается в [14], на момент начала исследований автора в 1999 году в свободном распространении не было СУЗ, в полной мере удовлетворяющих принципу императивности управления, что во многом обусловило необходимость разработки собственной СУЗ.

5. Надёжность. СУЗ должна выполнять свои функции при любой, сколь угодно сложной мультипрограммной ситуации в системе. Например, пользовательские процессы могут зацикливаться, выдавать некорректные запросы к ОС узла и к СУЗ, может произойти разрушение операционной системы ВУ. Во всех этих случаях, если нет аппаратных неисправностей в системе, СУЗ должна быть способна решать задачи управления вычислительными ресурсами.

6. Возможность реконфигурации в процессе функционирования. Под возможностью реконфигурации понимается способность системы динамически изменять своё F-состояние. В частности, должна обеспечиваться возможность динамического разбиения системы на подсистемы с произвольным составом ВУ в каждой.

7. Автоматическая организация режимов профилактики в определенное время. СУЗ должна иметь возможность обеспечить отсутствие выполняющихся заданий на решающем поле к некоторому заданному времени. При этом должно продолжаться планирование очереди заданий, в которую пользователи могут добавлять новые задания. Соблюдение этого требования обеспечивает

удобство организации регулярных профилактических работ на суперкомпьютерной системе по заранее заданному расписанию.

8. Прозрачность алгоритмов планирования и справедливое распределение вычислительных ресурсов. Требование было сформулировано в работе [85], в которой под прозрачностью алгоритмов планирования понимается возможность однозначной оценки пользователем перспектив запуска его заданий, находящихся в очереди. Под справедливым распределением вычислительных ресурсов понимается прежде всего невозможность захвата одним пользователем некоторого параллельного ресурса на неопределенно длительное время. Принцип справедливого распределения выражается формулой «пользователь, считавший за определенный период мало, должен иметь преимущество перед пользователем, считавшим много за этот же период».

9. Автоматизация формирования типового программного окружения. Параллельные программы пользователей, как правило, выполняются в некотором типовом (стандартном) программном окружении, определяемом, например, конкретной реализацией MPI. Стандартное программное окружение должно формироваться в автоматизированном режиме максимально прозрачным для пользователя образом. Соблюдение этого требования позволяет создать для пользователя удобную и комфортную рабочую среду, что, как будет показано ниже, является основой цифровой экосистемы суперкомпьютерного центра.

10. Модульность архитектуры СУЗ. Соблюдение этого требования дает возможность применения в СУЗ взаимозаменяемых встраиваемых модулей, изменяющих тем или иным образом функционал СУЗ с сохранением ее общей архитектуры и соответствия сложившейся цифровой экосистеме суперкомпьютерного центра.

Проведенное под руководством автора исследование [84] показало, что практически все распространенные СУЗ соответствуют указанным требованиям с рядом оговорок, которые будут рассмотрены в п.2.2 Этот факт позволяет гово-

рить о том, что соответствие приведенным требованиям является базовым показателем, отражающим качество СУЗ.

1.5 Распространенные системы управления заданиями

Обзор наиболее распространенных СУЗ приведен в работе [12], в которой произведено их сравнение по перечню определенных авторами характеристик. В более ранней работе [15] произведена классификация СУЗ по областям применения: в кластерных HPC-системах, в грид-средах и облачных средах. Авторы [15] определили список характеристик СУЗ, во многом совпадающих с перечнем [12], и также произвели сравнение СУЗ по всем характеристикам. Рассмотрим ряд известных суперкомпьютерных СУЗ, в том числе из списков сравнения [12, 15].

Система управления заданиям PBS (Portable Batch System) [13] ведет свою историю с 1991 года от системы NQS (Network Queuing System), считающейся первым планировщиком заданий для суперкомпьютеров. NQS/PBS разрабатывались при финансировании Национального космического агентства США, но в 2003 году права на PBS выкупила компания Altair Engineering. В настоящее время существует несколько как открытых, так и коммерческих версий PBS, среди которых следует выделить OpenPBS/TORQUE [86] – свободно распространяемое решение, разрабатываемое компанией Adaptive Computing (ранее – компанией Cluster Resources), и PBS Professional (PBS Pro) – проприетарную версию PBS от Altair Engineering.

Распространенной коммерческой СУЗ является IBM Spectrum LSF (LSF – Load Sharing Facility) [17]. Изначально эта СУЗ разрабатывалась компанией Platform Computing, которая в 2007 году выпустила Platform Lava, упрощенную версию LSF под лицензией GNU General Public License. В январе 2012 г. компания IBM приобрела Platform Computing, и с этого времени продукт называется IBM Spectrum LSF и поставляется совместно с суперкомпьютерными системами фирмы IBM. Следует отметить производный от LSF проект OpenLAVA [87] – СУЗ с

открытым исходным кодом, которая по оценкам [12] обладает необходимым, но в то же время сбалансированным набором функциональных возможностей.

Другим известным коммерческим продуктом является Moab от компании Adaptive Computing. Moab ведет свою историю от свободно распространяемого проекта Maui [88], разработку которого с 1990-х годов вела компания Cluster Resources. В 2005 году проект был выкуплен Adaptive Computing, разработка Maui прекращена, а все преимущества проекта были воплощены в новом коммерческом продукте Moab HPC Suite [89]. Moab HPC Suite не только автоматизирует планирование и управление заданиями, осуществляет мониторинг ресурсов параллельного компьютера и учет их потребления, но также использует многомерные политики и расширенное предсказательное моделирование для оптимизации планирования заданий.

Полнофункциональной коммерческой СУЗ является продукт Grid Engine [90, 91], изначально разработанный и выпущенный под названием CODINE в 1993 году компанией Genias Software, позже приобретенный и доработанный Sun Microsystems и Oracle. В 2013 году права на продукт перешли к компании Univa, которая в 2020 году была поглощена Altair Engineering. В настоящее время разработчики версий Grid Engine от компаний Sun, Oracle и Univa продвигают новую версию СУЗ под названием Gridware Cluster Scheduler [92]. Существует также несколько версий Grid Engine с открытым исходным кодом, включая Son of Grid Engine и Open Grid Scheduler, но развитие этих проектов в настоящее время не осуществляется.

Среди коммерческих СУЗ следует отметить Windows HPC Server [93], разрабатывавшийся компанией Microsoft в 2005-2015 гг. Основу программного продукта составляла операционная система Windows, адаптированная для использования на ВУ суперкомпьютеров. Microsoft реализовала поддержку стандартов параллельного программирования MPI и OpenMP как на уровне среды разработки MS Visual Studio (включая отладку), так и на уровне операционной системы ВУ. Официальная поддержка Windows HPC Server была прекращена в 2023 году.

Несмотря на мощность и широкий спектр возможностей коммерческих СУЗ, наибольшее распространение получила система SLURM (Simple Linux Utility for Resource Management – простая утилита Linux для управления ресурсами) [16, 94]. Разработка SLURM началась в начале 2000-х годов в Национальной лаборатории Лоуренса в Ливерморе (США) для решения проблемы масштабируемости СУЗ. За 20-летнюю историю развития SLURM прошел путь от простой утилиты до развитого программного комплекса с множеством поддерживаемых функций и встроенных модулей. SLURM является свободно распространяемым в открытых исходных текстах программным продуктом, способным управлять параллельными компьютерами практически произвольной мощности. Отмечается [12], что главным преимуществом SLURM является модульная архитектура, позволяющая для добавления или изменения функционала подключать к системе различные модули третьих разработчиков. Такая архитектура позволяет настраивать SLURM для разных видов входных потоков заданий, различных программно-аппаратных стеков, политик планирования задания и т.п.

Среди отечественных разработок следует отметить систему Cleo [95], в начале 2000-х годов разработанную коллективом НИВЦ МГУ им. М.В. Ломоносова для эффективного управления ресурсами вычислительных кластеров. Система организует поток вычислительных заданий в одну или несколько очередей и позволяет управлять порядком их выполнения на кластерах. В качестве вычислительной единицы в Cleo используется процессор. Хотя система позволяет оперировать понятием вычислительного узла, но рассматривает узел как объединение нескольких равноправных процессоров. Система Cleo написана на интерпретируемом языке Perl, что позволяет портировать её практически на любую вычислительную систему. К сожалению, с появлением SLURM разработки и развитие Cleo были свернуты.

СУЗ Slurm-ВНИИТФ [25] разработана и функционирует в ФГУП РФЯЦ-ВНИИТФ им. акад. Е.И. Забабахина. В основе системы лежит менеджер ресурсов SLURM. Slurm-ВНИИТФ расширяет функции SLURM и предоставляет дополни-

тельные возможности управления заданиями. В п. 3.1.1 приведено более подробное описание системы Slurm-ВНИИТФ.

1.6 Иерархическая модель управления вычислительными ресурсами суперкомпьютерной системы

В фундаментальной работе [12] представлена модель СУЗ, основанная на разбиении системы на компоненты по функциональному признаку. Модель [12] выделяет следующие взаимодействующие подсистемы:

- управления жизненным циклом задания;
- планирования заданий;
- выполнения заданий;
- управления вычислительными ресурсами.

Подсистема управления жизненным циклом задания решает задачи приема входного потока заданий, управления очередями заданий, контроля выполнения заданий, учета и биллинга. Подсистема планирования составляет расписание запусков заданий, выделяет вычислительные ресурсы и назначает на них здания из очередей. Подсистема выполнения заданий отвечает за запуск заданий на выделенных им ресурсах и за завершение заданий, очевидно, включающее освобождение занятых ими ресурсов. Подсистема управления вычислительными ресурсами осуществляет мониторинг вычислительных ресурсов, собирая информацию о доступности и состоянии ВУ решающего поля, а также отслеживает состояния выполняющихся заданий.

Модель [12] определяет функционал практически любой СУЗ, но при этом не отражает иерархической структуры управления ресурсами в параллельных вычислительных системах, соответствующей уровням распараллеливания входного потока заданий. Построим иерархическую модель управления, взяв за основу фундаментальную модель, рассмотренную в монографии [78]. Разделим СУЗ на иерархические уровни, каждый из которых уменьшает неопределённость сложной

мультипрограммной ситуации, определяя и фиксируя ряд параметров для вышестоящего уровня. Выделим четыре базовых уровня принятия решений.

Первый уровень – уровень пользователя, который выбирает алгоритмы для своих задач, средства для записи алгоритмов, их анализа, отладки и т.п. На втором уровне принимаются локальные, автономные решения по распределению ресурсов в рамках одного или нескольких вычислительных узлов. На третьем уровне принимаются решения, которые связаны с управлением подсистемой вычислительных узлов. На четвёртом уровне принимаются системные решения, изменяющие F-состояние отдельной суперкомпьютерной системы. Эти решения определяются приоритетами режимов функционирования, очередью заданий, состоянием ресурсов системы, директивами оператора и т.п.

Дополним модель [78] пятым уровнем иерархии, на котором представлены средства управления несколькими параллельными компьютерами, объединенными в единую распределенную вычислительную систему (РВС). На этом уровне принимаются глобальные решения по распределению вычислительных работ по очередям суперкомпьютерных систем, входящих в РВС.

Представим СУЗ совокупностью блоков $\{ S_{ij}\ (i = 1,2,3,4,5;\ j = 1,2,\ldots,J(i) \}$, где i – соответствующий уровень иерархии, как показано на рисунке 3.

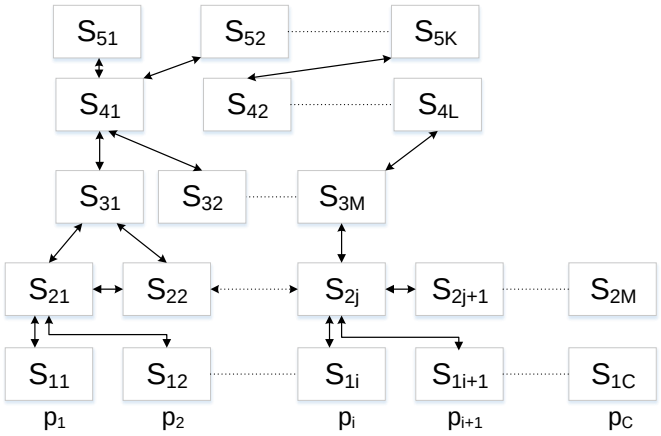

Рисунок 3. Иерархия блоков управления вычислительными ресурсами

Пусть $P = \{ p_i : i = 1 \ldots C \}$ – множество процессорных ядер, а $B = \{ n_i : i = 1 \ldots N \}$ – множество вычислительных узлов распределенной вычислительной си-

стемы, соответственно, C – общее число ядер, N – общее число узлов в РВС. РВС состоит из L МВС, каждая из которых ведет собственную (локальную) очередь заданий. Пусть в некоторый момент времени существует множество $W = \{ t_i: i = 1...T \}$ назначенных работ, т.е. множество выполняющихся в данный момент времени заданий, T – общее число работ (заданий). Каждая работа t_i является назначенной на некоторую подсистему $n^{t_i} = \{ n_k^{t_i} : k = 1... m^{t_i} \}$, состоящую из m^{t_i} ВУ.

Рассмотрим соответствие между блоками разных уровней иерархии и типовой архитектурой параллельного компьютера, показанное на рисунке 4.

Рисунок 4. Иерархия блоков управления вычислительными ресурсами

Заметим, что

$$\bigcup_{i=1}^{T} n^{t_i} \subseteq B$$

При этом существует некоторое множество n^{free}, такое что

$$n^{free} = B \setminus \bigcup_{i=1}^{T} n^{t_i}$$

Множество n^{free} – это множество не занятых ни одной работой, свободных ВУ. Общее число занятых под работы узлов равняется

$$m = \left| \bigcup_{i=1}^{T} n^{t_i} \right|$$

Разбиение множества B на подмножества n^{t_i} с назначением на каждое подмножество n^{t_i} определённой работы t_i фактически будет определять текущее F-состояние всей РВС.

Первый уровень иерархии будут представлять средства управления отдельным процессорным ядром, причем $J(1) = C$. Блоки S_{1j} представляют собой ветви параллельной программы: потоки либо процессы в зависимости от модели параллельного программирования. Каждый блок S_{1j} назначается на процессорное ядро p_j средствами вышестоящего второго уровня иерархии, и задача управления вычислительным процессом на этом уровне целиком возлагается на пользователя-программиста, который в зависимости от прикладного алгоритма определяет степень параллелизма своей программы. Например, программист может задействовать или не задействовать векторные команды, использовать механизмы распараллеливания графических процессоров и т.п.

Второй уровень иерархии образуют средства управления отдельным вычислительным узлом. Средства второго уровня осуществляют контроль над средствами первого уровня и имеют возможность вмешиваться в их работу, прерывать и даже уничтожать и порождать новые. Кроме этого, средства второго уровня контролируют взаимодействие как блоков первого уровня, так и блоков второго уровня. Число блоков второго уровня равно числу ВУ в системе, $J(2) = N$. Блоки S_{2j} представляют операционные системы вычислительных узлов, контролирующие взаимодействующие потоки и процессы параллельной параллельной программы. В состав ОС ВУ также включаются компоненты СУЗ, управляющие отдельным ВУ. Узлы $\{ n_i : i = 1 \ldots m \}$ являются занятыми определёнными работами (заданиями), соответствующие им управляющие средства $\{ S_{2i} : i = 1 \ldots m \}$ находятся в режиме функционирования. Узлы $\{ n_j : j = m+1 \ldots N \}$ свободны, соответствующие им управляющие средства $\{ S_{2j} : j = m+1 \ldots N \}$ находятся в режиме

ожидания (бездействия) или отсутствуют вовсе. Назначение работы t_i на некоторый узел n_j производится средствами вышестоящего, третьего уровня иерархии.

Третий уровень иерархии представлен средствами управления подсистемами (подмножествами узлов) n^{t_i}. Число блоков третьего уровня равно числу выделенных подсистем, т.е. числу назначенных заданий, $J(3) = T$. Средства третьего уровня полностью контролируют средства второго уровня в рамках подчинённых им подсистем. Блоки S_{3j} представляют собой компоненты СУЗ, выполняющиеся на управляющей ЭВМ и контролирующие запуск, выполнение и завершение отдельного задания, для которого выделено некоторое подмножество вычислительных узлов.

Введем два допущения. Первое допущение заключается в том, что каждый ВУ трактуется как минимальная и неделимая единица параллельного ресурса, которая может быть выделена заданию. Из этого следует, что выделяемые под различные задания подсистемы не должны пересекаться, т.е. один вычислительный узел не может разделяться между разными заданиями. Это неизбежно влечёт фрагментацию ресурсов в случае, когда число запрашиваемых заданием процессорных ядер не кратно числу ядер на узле. Однако, для первого допущения есть два серьезных основания:

– разделение одного узла между процессами разных заданий ослабляет изоляцию заданий и неизбежно влечет их конкуренцию за ресурсы узла и, как следствие, – снижение скорости вычислений в обоих заданиях;

– принятие первого допущения значительно упрощает схему управления ресурсами, что в свою очередь повышает надежность и готовность всей системы в целом.

Следует отметить, что для мощных многоядерных ВУ с несколькими ускорителями фрагментация может обойтись достаточно дорого, и в этом случае приходится разделять один ВУ между разными заданиями [96].

Второе допущение заключается в том, что обмениваться информацией друг с другом могут только процессы параллельной программы одного задания, т.е. информационный обмен невозможен между двумя различными заданиями.

Введённые допущения накладывают следующие ограничения на построенную иерархическую модель.

1. Для любых $i, j = 1 \ldots T$ выполняется

$$n^{t_i} \cap n^{t_j} = \varnothing$$

2. Для любых $i, j = 1 \ldots T$ взаимодействие между блоками S_{2i} и S_{2j} будет разрешено тогда и только тогда, когда существует $k = 1 \ldots T$ такое, что $n_i \in n^{t_k}$ и $n_j \in n^{t_k}$.

Четвертый уровень иерархии образуют средства, управляющие очередью отдельной МВС. Средства этого уровня полностью контролируют средства нижестоящего третьего уровня и имеют возможность формировать отображение множества заданий W на множество подсистем n^{t_i}. Число блоков $J(4) = L$ равно числу локальных очередей МВС в составе распределенной вычислительной системы. Блоки S_{4j} представляют собой локальные планировщики СУЗ, выполняющиеся на управляющей ЭВМ и составляющие расписание запусков заданий на контролируемых ими МВС. В состав блоков S_{4j} включаются также средства приема входного потока заданий от вышестоящего, пятого уровня иерархии.

Пятый уровень иерархии представлен средствами, обеспечивающими доступ пользователей и администраторов к ресурсам системы. На этом уровне формируется входной поток заданий для локальных очередей МВС. Это могут быть запросы пользователей локальной МВС, либо один или несколько планировщиков глобальной очереди заданий в распределенной вычислительной системе. Число блоков $J(5) = K$ можно рассматривать, как число различных управляющих функций пятого уровня иерархии, таких как команды отправки заданий в локальную и глобальную очереди, контроля выполнения заданий, средства ведения глобальной очереди заданий (метапланировщик) и др.

В отличие от функциональной модели [12], представленная иерархическая модель в явном виде отображает схему распараллеливания обработки информации в суперкомпьютерном центре коллективного пользования. На высшем, пятом уровне происходит распределение крупных вычислительных работ (заданий) между разными очередями, каждая из которых представляет собственную суперкомпьютерную систему или ее раздел. Уровнем ниже задания распределяются между подсистемами одного раздела, при этом обеспечивается мультизадачный режим работы, подразумевающий одновременное выполнение нескольких заданий на разных непересекающихся подсистемах. На третьем уровне в рамках одной подсистемы осуществляется распределение вычислительной работы между узлами суперкомпьютера. Дальнейшее распараллеливание происходит на втором уровне в рамках вычислительного узла, когда процессы или потоки прикладной программы распределяются по ядрам как центрального процессора, так и ускорителя вычислений. Наконец, на нижнем, первом уровне назначенный на ядро процесс (поток) пользовательской программы использует возможности распараллеливания процессоров, такие как векторизация кода [97], одновременная [98] и/или гетерогенная многопоточность [99].

1.7 Архитектура Системы управления прохождением параллельных заданий

Начиная с 1999 года, силами коллектива сотрудников Института прикладной математики им. М.В. Келдыша РАН (ИПМ им. М.В. Келдыша РАН), Межведомственного суперкомпьютерного центра РАН (МСЦ РАН) – отделения суперкомпьютерных систем и параллельных вычислений НИЦ «Курчатовский институт» и Института математики и механики им. Н.Н. Красовского УрО РАН (ИММ УрО РАН) разрабатывается и эксплуатируется отечественная СУЗ – Система управления прохождением параллельных заданий (СУППЗ) [100]. Архитектура СУППЗ построена в соответствии с рассмотренной в п.1.6 иерархической моделью управления вычислительными ресурсами суперкомпьютерной системы коллективного пользования и за годы своего развития претерпела ряд модификаций,

которые будут рассмотрены ниже. Современная архитектура СУППЗ [101] представлена на рисунке 5, а соответствие компонентов архитектуры уровням иерархической модели – на рисунке 6.

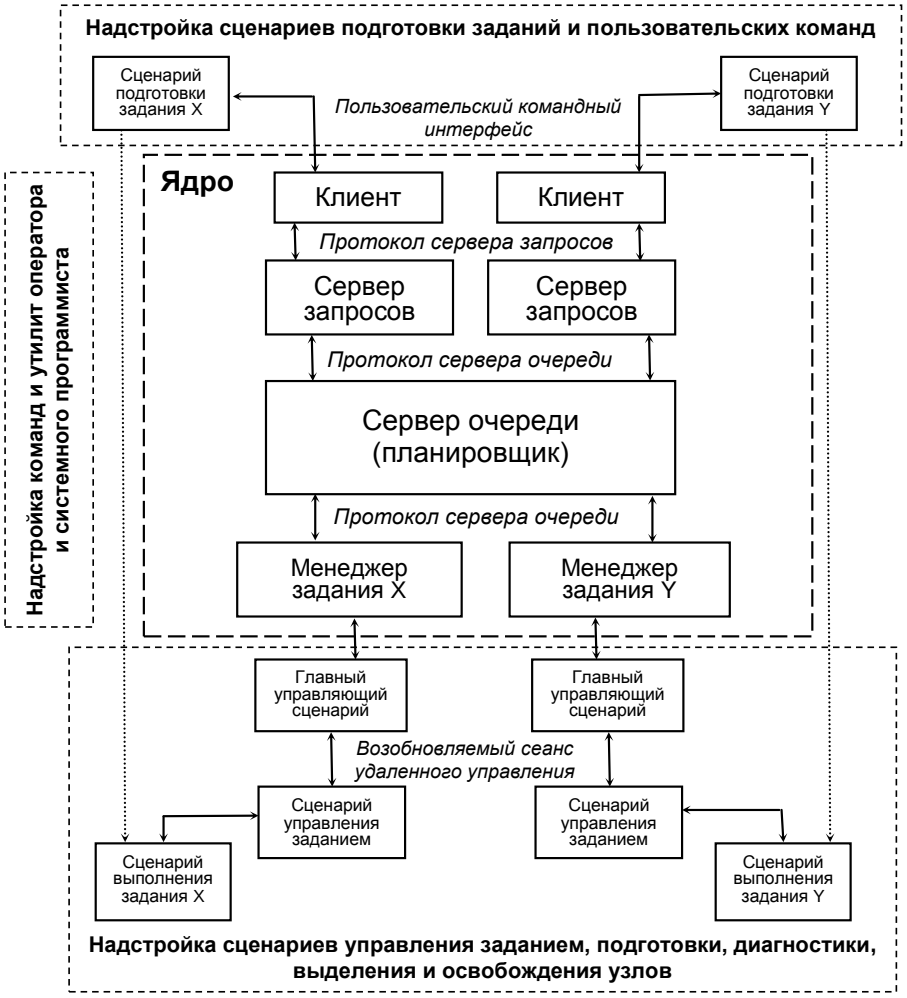

Рисунок 5. Архитектура СУППЗ

Архитектурно СУППЗ можно разделить на ядро и надстройки. Ядро включает планировщик (сервер очереди) и служебные процессы: клиентское приложение, серверы запросов и запуска и менеджеры заданий. Надстройками являются подсистемы:

– сценариев пользовательских команд;

– команд и утилит оператора и системного программиста;

– сценариев управления заданием;

– сценариев диагностики, выделения и освобождения узлов.

Сценарии надстройки подразделяются на две группы. Первую составляют сценарии, обеспечивающие базовый функционал соответствующей надстройки, одинаковый для всех суперкомпьютерных систем типовой архитектуры. Вторую группу составляют сценарии, разрабатываемые системным программистом в целях адаптации надстроек под особенности конкретной суперкомпьютерной системы.

Для стандартных сред параллельного программирования, например, различных реализаций MPI, СУППЗ предоставляет надстройку сценариев подготовки задания, предназначение которых – автоматизация подготовки паспорта задания для выполнения параллельной программы в заданной стандартной среде. При применении пользователем сценариев надстройки соответствующий сценарий выполнения задания будет сформирован системой автоматически. На рисунке 5 пунктирной стрелкой показано, что в результате работы сценариев подготовки заданий X и Y формируются соответствующие сценарии выполнения заданий.

Сценарии подготовки заданий, взаимодействуя с клиентской утилитой на сервере доступа, обеспечивают подготовку входного потока заданий и могут быть отнесены к пятому уровню управления. На этом же уровне располагаются не входящие в состав СУППЗ средства управления глобальной очередью.

Клиентская утилита обеспечивает авторизованный доступ пользователя к нижестоящему, четвертому уровню управления через связь с сервером запросов, выполняющимся на управляющей ЭВМ. При этом применяется сетевой протокол сервера запросов, обеспечивающий постановку задания в очередь, просмотр состояния очереди, снятие задания с выполнения или удаление его из очереди и другие действия управления. На четвертом уровне управления функционирует сервер очереди, осуществляющий планирование заданий в соответствии с рассмотренным в п. 3.3.1 методом. На этом же уровне выполняются команды оператора из соответствующей надстройки. Для взаимодействия со служебными процессами ядра и командами надстройки применяется специфицированный протокол сервера очереди.

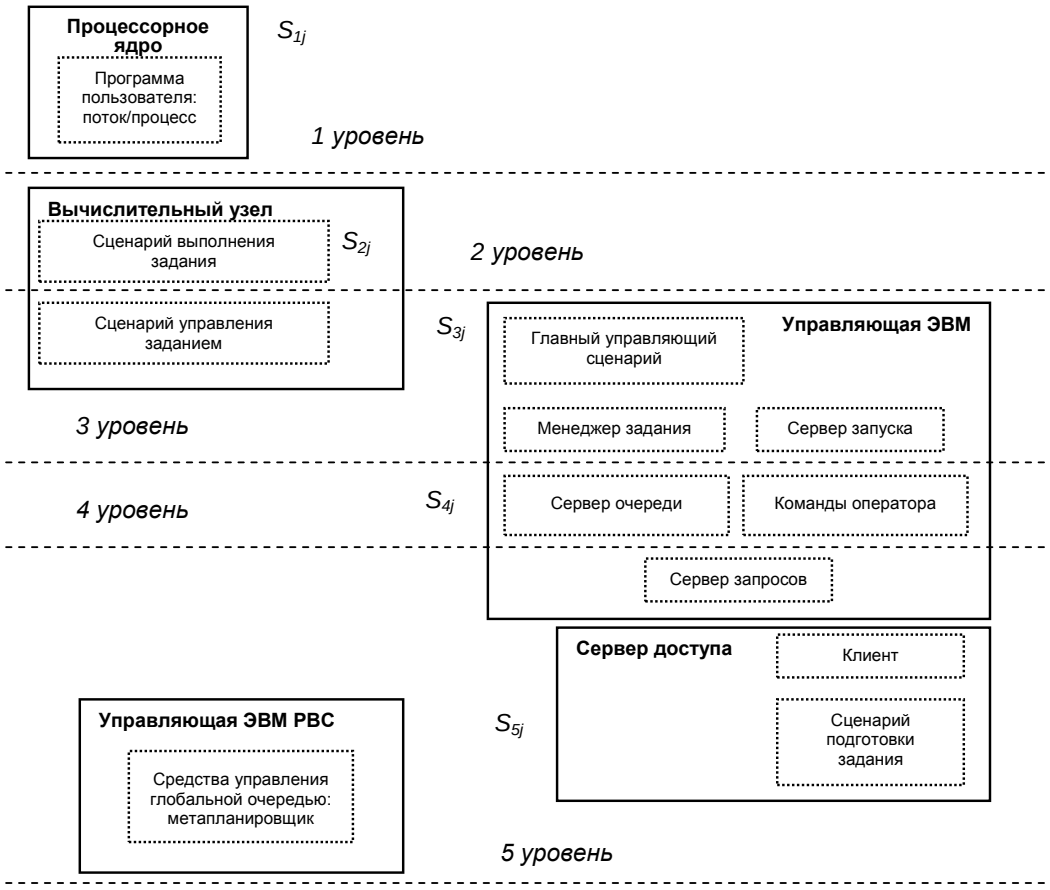

Рисунок 6. Соответствие компонентов архитектуры СУППЗ уровням иерархической модели

Третий уровень управления представлен служебными процессами ядра – сервером запуска и менеджером задания. Сервер запуска производит выделение ВУ для задания, их диагностику и конфигурацию, для чего применяет сценарии соответствующей надстройки. При обнаружении неисправных узлов последние автоматически выводятся из состава вычислителя, а задание возвращается в очередь на перепланирование. Менеджер задания открывает отдельный сеанс, в котором запускает на управляющей ЭВМ главный управляющий сценарий, определяющий действия по управлению выделенной подсистемой узлов. В частности, в этом сценарии возможно передача функций управления узлами сторонней СУЗ [102].

На первом из выделенных заданию узлов от имени администратора выполняется сценарий управления заданием, который поддерживает возобновляемый сеанс взаимодействия с главным управляющим сценарием, за счет чего, напри-

мер, возможна независимая перезагрузка управляющей ЭВМ без прерывания выполнения заданий.

Сценарий выполнения задания работает на вычислительных узлах от имени пользователя и представляет второй уровень иерархической модели, обеспечивая распределение процессов/потоков прикладной программы по процессорных ядрам. Процессы/потоки прикладной программы в соответствии с представленной моделью функционируют на нижнем, первом уровне иерархии управления.

1.8 Соответствие СУППЗ требованиям к системам управления заданиями

Покажем, что предложенная архитектура СУППЗ полностью удовлетворяет выдвинутым в п.1.4 требованиям.

Требование **универсальности** выполняется за счет выделения в архитектуре СУППЗ ядра системы, для которого задание представляет собой абстрактный информационный объект, запрашивающий некоторый виртуальный параллельный ресурс. Задача ядра – обеспечить выделение реального физического параллельного ресурса для нужд задания. Вид и характер пользовательских прикладных программ, которые будут выполнены на выделенном параллельном ресурсе, полностью определяются соответствующими надстройками подготовки заданий. При необходимости обеспечения поддержки конкретных программных сред в соответствии с правилами СУППЗ могут быть сформированы новые надстройки. Важно, что сценарии надстроек подготовки заданий выполняются на сервере доступа с правами пользователей, поэтому последние имеют возможность разрабатывать и применять собственные сценарии.

Автоматическое функционирование и круглосуточная доступность обеспечиваются за счет выделения надстройки сценариев управления заданием, подготовки, диагностики, выделения и освобождения ВУ. Сценарии диагностики автоматически выявляют возникающие неисправности ВУ и выводят отказавшие узлы из состава решающего поля, изменяя их статус на «неисправен». Факт отказа фиксируется в системных журналах, оператору отправляется электронное письмо.

Очередь заданий в случае сужения решающего поля из-за отказа ВУ автоматически перепланируется. СУППЗ продолжит обработку заданий до вмешательства оператора, который после детальной диагностики отказавших ВУ либо вернет их в состав решающего поля, либо отправит в ремонт.

В случае сбоев на управляющей ЭВМ задания продолжат выполнение на решающем поле под контролем соответствующих сценариев управления заданием. Система продолжит функционирование до вмешательства оператора, который после детальной диагностики управляющей ЭВМ осуществит её перезагрузку и восстановление менеджеров заданий. Последние возобновят удаленные сеансы со сценариями управления заданиями, после чего система продолжит функционирование в штатном режиме.

Гарантированное обслуживание без потерь обеспечивается за счет упоминавшегося механизма диагностики ВУ, исправность которых проверяется перед запуском каждого задания. В случае отказа одного или нескольких ВУ они выводятся из решающего поля, а для задания подбираются новые ВУ. В случае отсутствия свободных ВУ задание возвращается в очередь.

Императивность управления обеспечивается за счет выделенной в архитектуре надстройки сценариев подготовки и освобождения ВУ, ориентированных на соблюдение следующих принципов взаимной изоляции заданий:

– доступ пользователю и процессам его программы предоставляется только к выделенным для его задания ВУ и только на время выполнения задания; в остальное время или к другим ВУ доступ пользователю запрещается на уровне технической невозможности;

– ВУ, выделенный для выполнения одного задания, не может быть одновременно предоставлен еще какому-либо другому заданию, даже если на узле остаются свободные простаивающие процессорные ядра или другие ресурсы;

– принудительная очистка от процессов пользователя каждого ВУ после завершения параллельного задания; если на ВУ после очистки все еще остаются

пользовательские процессы, ВУ помечается как неисправный и выводится из состава решающего поля.

Надёжность в СУППЗ обеспечивается за счет **принципа делегирования функций управления** ресурсами высоконадежным компонентам и утилитам операционной системы с сохранением контроля за жизненным циклом заданий. Под высоконадежными понимаются такие компоненты, отказ которых означает неисправность все суперкомпьютерной системы в целом вне зависимости от исправности компонентов СУППЗ. Другими словами, высоконадежный компонент считается по умолчанию исправным, в противном случае суперкомпьютер нуждается в ремонте с приостановкой обслуживания пользователей.

В СУППЗ в качестве таких компонентов выступают сетевая файловая система (NFS) и ОС Linux на узлах. Сетевая файловая система используется в СУППЗ для следующих критических функций управления:

– авторизации клиента сервером запросов;

– ведение оперативной базы данных статусов вычислительных узлов («свободен», «занят», «неисправен»);

– контроля сценария управления заданием главным управляющим сценарием;

– перенаправления стандартных потоков ввода/вывода задания.

Подобное делегирование позволяет, в отличие от SLURM, LSF и PBS, отказаться от наличия активных компонентов СУППЗ на вычислительных узлах: сценарий управления заданием запускается в момент старта задания через систему удаленного выполнения команд. Отсутствие активных компонентов на узлах с одной стороны исключает их выход из строя и повышает надежность функционирования суперкомпьютера, а с другой стороны открывает возможность применять для управления ресурсами сторонние системы, такие как SLURM.

Возможность реконфигурации в процессе функционирования обеспечивается в СУППЗ за счет виртуализации параллельного ресурса на уровне надстройки сценариев подготовки заданий и конфигурации выделенных заданию узлов, соответствующих требуемому виртуальному параллельному ресурсу. По-

скольку распределение заданий по ВУ производится автоматически, пользователь не может влиять на процесс распределения и заранее знать, какие ВУ будут выделены его заданию. По умолчанию число процессов параллельной программы, запускаемых на каждом узле, равно числу процессорных ядер узла. СУППЗ дает пользователю возможность самостоятельно задавать, сколько процессов на каждом ВУ должно быть запущено, путем виртуализации параллельного ресурса. Пользователь должен предположить, что его заданию будет выделено нужное ему число ВУ с предопределенными виртуальными именами node1, node2,…, nodeN, где N – число необходимых ВУ. Распределение процессов по узлам пользователь осуществляет при помощи специального файла-шаблона стандартного формата. После того, как задание пройдет через очередь и поступит на выполнение, процесс-менеджер задания заменит виртуальные имена реальными именами ВУ, выделенных заданию. Процессы параллельной программы будут распределены по реальным ВУ точно так, как указал пользователь в файле-шаблоне.

Автоматическая организация режимов профилактики в определенное время, а также **прозрачность алгоритмов планирования и справедливое распределение вычислительных ресурсов** обеспечиваются применяемым в СУППЗ рассмотренным в п. 3.3.1 методом планирования заданий. В частности, режимы профилактики организуются за счет применения так называемых шкал доступа, определяющих возможности пользователей и системных администраторов по доступу к решающему полю в определенное время.

Автоматизация формирования стандартного программного окружения обеспечивается наличием в архитектуре СУППЗ надстройки сценариев подготовки заданий, которая позволяет автоматизировать подготовку паспорта задания для некоторого стандартного программного окружения. Одной из таких групп сценариев, изначально входившей в состав СУППЗ, являются сценарии поддержки MPI-программ. В рамках этих сценариев СУППЗ предлагает пользователю специальную команду mpirun, по синтаксису максимально приближенную к аналогичной команде, поставляемой различными реализациями MPI. Команда mpirun в

СУППЗ по набору заданных параметров автоматически формирует паспорт задания, содержащий в том числе сценарий (также формируемый автоматически) выполнения задания в окружении выбранной пользователем реализации MPI. Для этого окружения также автоматически происходит отображение виртуальных имен ВУ, заданных пользователем, в сетевые имена ВУ, выделенных заданию, таким образом, что MPI-программа получает на вход релевантный список ВУ.

При смене суперкомпьютерной системы или реализации MPI сценарии надстройки подготовки заданий MPI корректируются системным программистом СУППЗ для обеспечения формирования адекватного окружения MPI-приложения. Внешний интерфейс и набор ключей команды mpirun при этом остается неизменным для пользователя. Практика показала, что именно MPI-надстройка оказалась наиболее востребованной пользователями и стала характерной особенностью СУППЗ.

Модульность архитектуры в СУППЗ обеспечивается:

– соответствием архитектуры иерархической модели управления ресурсами;

– разделением на ядро, неизменяемые и изменяемые системным программистом надстройки;

– применением специфицированных протоколов взаимодействия компонентов.

Ядро СУППЗ, реализованное на языке высокого уровня, изменяется разработчиками сравнительно редко. Достаточно большое число функций управления и мониторинга, таких как диагностика вычислительных узлов, подключение разных сред программирования прикладных программных пакетов, реализовано в виде командных сценариев, что позволяет разработчикам при необходимости оперативно вносить изменения в систему, не затрагивая ядро. Наконец, разграничение функций компонентов по уровням иерархии управления и применение специфицированных протоколов, таких как протоколы сервера запросов и сервера очереди, дало возможность подключения к СУППЗ ключевых сторонних компонентов. Примерами могут служить интеграция в СУППЗ известного планировщи-

ка Maui, рассмотренная в п. 2.1.4.4 и применение для управления вычислительными узлами СУЗ SLURM, рассмотренное в п. 2.1.3

1.9 Особенности архитектуры СУППЗ для отечественных массивно-параллельных систем МВС-1000

Многопроцессорная вычислительная система МВС-1000 [64] была разработана в 1998 году усилиями коллективов НИИ «Квант», ИПМ им. М.В. Келдыша РАН и ИММ УрО РАН. Общая архитектура системы представлена на рисунке 2. В качестве вычислительного узла в МВС-1000 применялся двухпроцессорный модуль. Один из процессоров узла назывался вычислительным и соответственно выполнял вычислительные функции, другой служил для объединения вычислительных узлов в единое решающее поле и назывался связным. В МВС-1000 качестве вычислительного процессора использовался микропроцессор Alpha-21164 300-500 МГц фирмы DEC. В качестве связных процессоров использовались сигнальные процессоры – либо TMS320C44 фирмы Texas Instruments, либо ADSP21060 (Sharc) фирмы Analog Devices. Оба сигнальных процессора имели по несколько (C44 – четыре, Sharc – шесть) высокоскоростных линков. При помощи линков ВУ соединялись друг с другом, образуя коммуникационную топологию, представленную на рисунке 7.

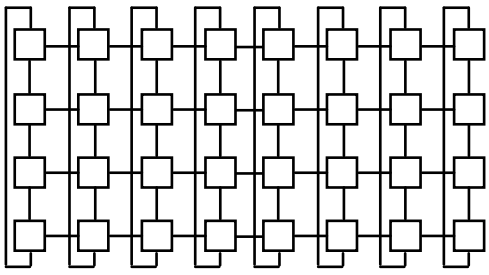

Рисунок 7. Топология коммуникационной среды МВС-1000 для варианта из 32 ВУ

Важным моментом в архитектуре ВУ МВС-1000 являлось то, что на аппаратном уровне связной процессор имел возможность обращения к оперативной памяти вычислительного процессора, в то время как оперативная память связного

процессора была недоступна для вычислительного. При этом связной процессор имел также возможность произвести аппаратный сброс вычислительного процессора и загрузку его операционной системы, в качестве которой применялась POSIX-совместимая ОС VxWorks [104].

На сети связных процессоров функционировала созданная в ИПМ им. М.В. Келдыша РАН под руководством А.О. Лациса операционная среда Router [105]. Операционная среда Router была разработана в 1994 году для системы предыдущего поколения МВС-100 [63] как основное средство информационного обмена и включала в себя одноименную библиотеку передачи сообщений. Библиотека предоставляла пользователю набор примитивов для организации обмена сообщениями между двумя любыми узлами по модели Хоара. В МВС-1000 операционная среда Router была дополнена функциями защиты, обеспечения файлового ввода-вывода и управления вычислительным процессором. Кроме этого, МВС-1000 впервые в истории отечественных параллельных ВС была обеспечена коммуникационной библиотекой, полностью реализовывавшей стандарт MPI.

На управляющей ЭВМ функционировала полнофункциональная сетевая операционная система Digital Unix производства компании DEC.

Возрастающее число пользователей МВС-1000 привело к необходимости организации полнофункционального режима коллективного пользования. С этой целью были сформулированы рассмотренные в п. 1.4 требования, произведен поиск удовлетворяющих этим требованиям свободно распространяемых решений. В 1998 году доступны для анализа были уже упоминавшаяся система NQS/PBS, а также СУЗ DQS (Distributed Queueing System) [15]. Проведенный анализ [83] показал, что доступные решения удовлетворяют далеко не всем выдвинутым требованиям. Это было обусловлено, прежде всего, направленностью рассмотренных систем на объединение в одно целое гетерогенных однопроцессорных вычислительных ресурсов и динамическое планирование в подобном кластере, как правило, непараллельных (однопроцессорных) задач. Ни одна из этих систем не была способна в полной мере управлять параллельными ресурсами с обеспечением им-

перативности управления. Кроме этого, системы не позволяли автоматизировать работу пользователя со стандартными программными средами, в частности, с MPI, что приводило, например, к ситуации, DQS выделяла для задания одни ВУ, а среда MPI запускала параллельную программу на других. В итоге было принято решение о разработке собственной системы управления задания, которой и стала СУППЗ.

С точки зрения организации управления главной сложностью было отсутствие полнофункциональной сетевой операционной системы для связных процессоров. Этот факт определял отсутствие следующих возможностей:

– сетевого имени ВУ, вместо которого применялся идентификатор в виде специального номера;

– сетевой доступности ВУ из управляющей ЭВМ по протоколам TCP/IP;

– средств идентификации, аутентификации пользователей и разграничения доступа;

– сетевой файловой системы.

В этих условиях было необходимо построить СУЗ, удовлетворяющую выдвинутым в п. 1.4 требованиям. С этой целью автором была предложена рассмотренная в п. 1.6 иерархическая модель управления вычислительными ресурсами. Единственным отличием было отсутствие пятого уровня иерархии, поскольку на тот момент времени задача организации территориально распределенной вычислительной среды еще не была поставлена. Распределение средств управления по уровням иерархической модели для системы МВС-1000 представлено на рисунке 8.

Блоки S_{1j} первого уровня иерархии – это пользовательские программы и операционная система, выполняющиеся на вычислительном процессоре узла. ОС вычислительного процессора (ОС ВП) полностью контролирует доступные пользователю ресурсы узла – вычислительный процессор и его оперативную память – и выделяет их по запросам от пользовательской программы.

Второй уровень иерархии представляется операционной средой, функционирующей на сети связных процессоров вычислителя (ОС СП). Протоколы данного

уровня отвечают за взаимодействие вычислительных узлов между собой. При этом управляющая программа каждого связного процессора контролирует ОС своего вычислительного процессора и имеет возможность осуществления его аппаратного сброса и перезагрузки.

Рисунок 8. Соответствие уровням иерархической модели компонентов архитектуры СУППЗ для суперкомпьютерной системы МВС-1000

Блоки S_{3j} третьего уровня иерархии образуются средствами управляющей ЭВМ, которые обеспечивают загрузку параллельного задания на определённые процессоры вычислителя, т.е. выделение некоторой подсистемы и назначение работы на эту подсистему. Средства третьего уровня обеспечивают доступ нижестоящим уровням к общим для системы внешним устройствам, подключённым к управляющей ЭВМ, – файловым системам, терминалам, сетевым службам.

Четвёртый уровень иерархии образуют средства управляющей ЭВМ, которые обеспечивают планирование очереди заданий, а также доступ пользователей и администраторов к ресурсам системы. На эти средства возлагается и большая часть функций защиты от НСД.

Вариант архитектуры ядра СУППЗ для суперкомпьютеров МВС-1000, соответствующей иерархической модели управления, показан на рисунке 9. Надстройки полностью идентичны общей архитектуре СУППЗ и для наглядности иллюстрации не показаны на рисунке.

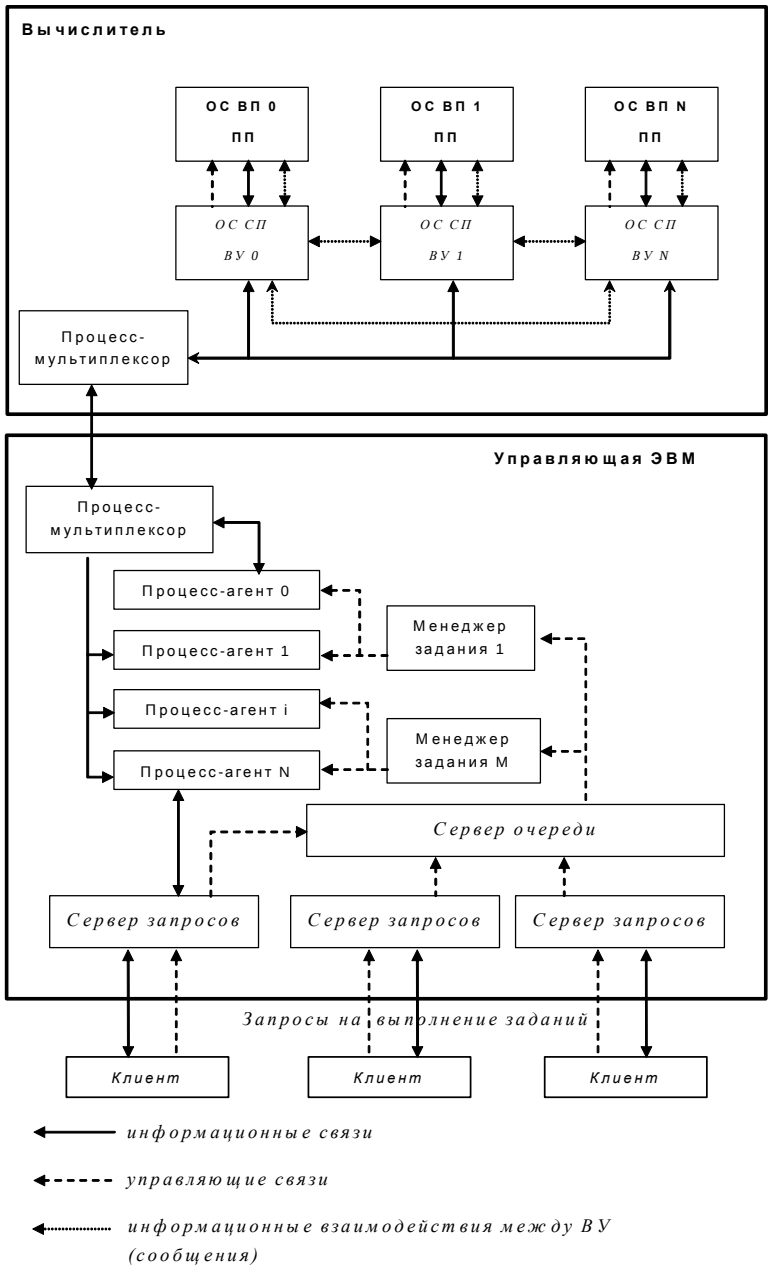

Рисунок 9. Архитектура ядра СУППЗ для суперкомпьютерной системы МВС-1000

Нетрудно видеть, что компоненты ядра СУППЗ МВС-1000 четвертого и третьего уровней иерархии полностью идентичны общей архитектуре. Различие начинается на втором уровне по причине упомянутого отсутствия полнофункциональной сетевой ОС на вычислительных узлах. Тем не менее, требования универ-

сальности, автоматической организации режимов профилактики, автоматизации формирования стандартного программного окружения, прозрачности алгоритмов планирования и справедливого распределения вычислительных ресурсов обеспечиваются теми же методами и средствами, что были рассмотрены в п.1.8

Рассмотрим выполнение требований автоматического функционирования и круглосуточной доступности, надежности и гарантированного обслуживания без потерь. Для удовлетворения этим требованиям средства управления процессорами были выделены в высоконадёжное программное ядро. Надёжность ядра была обеспечена за счёт полной независимости от пользовательских программ и операционной системы вычислительного процессора, а также на счёт качества разработанных и реализованных протоколов информационного обмена. Добиться этого получилось путём сосредоточения средств управления второго уровня исключительно в связных процессорах ВУ, в памяти, недоступной вычислительным процессорам. Общение с пользовательскими программами и ОС вычислительного процессора осуществлялось исключительно по схеме «запрос-ответ». При этом запросы специфицировались таким образом, чтобы исключить возможность реконфигурации или изменения управляющих средств со стороны пользовательских программ путём косвенного влияния через выдачу определённых запросов.

Был введен сеансовый режим работы управляющих средств, означавший, что ОС ВП и пользовательские программы существуют и функционируют лишь во временных рамках отведённого им сеанса работы, т.е. только во время выполнения определенного задания. В рамках отдельного сеанса работы функции управления вычислительным процессором полностью делегировались его операционной системе. Функции идентификации и аутентификации пользователей делегировались операционной системе управляющей ЭВМ.

Требование возможности реконфигурации во время функционирования выполнялось за счёт ввода логической нумерации ВУ во время сеанса работы. Управляющая программа связного процессора получала в начале каждого сеанса от средств вышестоящего уровня иерархической модели список ВУ, которые

должны были образовать связанную группу на время сеанса. Поскольку информационный обмен во время сеанса в соответствии со вторым допущением иерархической модели управления был разрешён только между ВУ одной группы, то внутри каждой группы было возможно введение логической нумерации ВУ. Управляющая программа производила во время сеанса отображение из логического номера в физический.

Рассмотрим требование императивности управления, выполнение которого также было обеспечено построением механизма изоляции заданий. С этой целью автором были разработаны специальные управляющие протоколы взаимодействия между ОС СП и специальными процессами-агентами на управляющей ЭВМ (см. рисунок 9). Для каждого ВУ в один момент времени на управляющей ЭВМ мог существовать только один процесс-агент. В свою очередь процесс-агент обслуживал только один, «собственный» ВУ. Соответствие между процессом-агентом и ВУ определялось физическим номером узла, который присваивался процессу-агенту при его инициализации от средств вышестоящего уровня иерархии.

Временные рамки каждого сеанса работы на определённом ВУ были жёстко связаны с порождением и завершением соответствующего этому ВУ процесса-агента на управляющей ЭВМ. Работа управляющих протоколов была построена таким образом, что *ОС ВП и пользовательские программы выполнялись на определённом узле тогда и только тогда, когда на управляющей ЭВМ был загружен и выполнялся соответствующий узлу процесс-агент*. Это важное свойство позволило организовать простое и эффективное программное управление вычислительными процессорами непосредственно с управляющей ЭВМ. Для загрузки одного или группы вычислительных процессоров было необходимо и достаточно запустить на управляющей ЭВМ один или группу процессов-агентов с соответствующими номерами. Для освобождения одного или группы вычислительных процессоров системы было необходимо и достаточно завершить на ЭВМ процессы-агенты с соответствующими номерами.

Указанные свойства средств управления вычислительными процессорами позволили на их основе построить средства следующего, третьего уровня иерархической модели. Это средства управления подсистемами, за счёт наличия которых была обеспечена возможность организации одновременного и независимого выполнения на вычислителе различных параллельных программ, т.е. организация мультизадачности на уровне ВС.

Каждому заданию выделялась определённая подсистема – группа вычислительных процессоров. Средства управления подсистемой открывали новый сеанс на ВУ выделенной группы путём запуска процессоров-агентов с номерами, соответствующими физическим номерам ВУ группы. При завершении задания средства управления подсистемой уничтожали процессы-агенты соответствующей группы и тем самым закрывали сеанс. По окончании сеанса работы, т.е. по завершении очередного задания, управляющая программа связного процессора производила аппаратный сброс вычислительного процессора. В начале нового сеанса происходила очистка оперативной памяти ВП, загрузка ОС ВП и пользовательской программы.

За счёт того, что средства управления процессорами разрешали информационный обмен только между ВП одной связанной группы, обеспечивалась изоляция друг от друга параллельных пользовательских программ разных заданий. При этом сеансовый режим с очисткой оперативной памяти ВП позволил устранять последствия сбоев или краха ОС ВП, а также делал невозможным несанкционированный доступ какого-либо пользователя к информации другого пользователя, оставшейся в оперативной памяти ВП после завершения задания.

Выделение ВУ, как подсистемы для выполнения задания, происходило на четвёртом уровне иерархической модели средствами планирования ресурсов системы. При запуске задания всем процессорам-агентам из выделенной группы передавался одинаковый список физических номеров ВУ группы. При этом сохранялась логическая нумерация ВУ внутри группы. ОС СП обеспечивала независимую загрузку ОС ВП одновременно на нескольких разных непересекающихся

группах ВУ. При этом обмен сообщениями между ОС ВП был возможен только внутри группы по логическим номерам.

Предложенный механизм позволил осуществить независимую загрузку вычислителя одновременно разными параллельными программами от разных пользователей и обеспечил мультизадачность на уровне системы в целом. Загрузка параллельной программы на вычислительные процессоры рассматривалась как выделение определённой подсистемы для соответствующего программе задания. Управление подсистемами осуществляли специальные процессы – серверы задания, по одному на каждую выделенную подсистему.

На рисунке 9 сплошными стрелками показаны информационные связи различных процессов вычислителя с процессами управляющей ЭВМ и между процессами управляющей ЭВМ. Пунктирными стрелками показаны информационные обмены между ВУ. Штриховыми стрелками обозначены управляющие обмены, связанные со стартом и завершением процессов.

Компоненты СУППЗ для МВС-1000 распределялись по уровням предложенной иерархической модели управления следующим образом. Первый уровень составляли ОС ВП и пользовательские программы. Второй уровень составляли ОС СП, процессы-мультиплексоры корневого ВУ и управляющей ЭВМ и процессы-агенты. Третий уровень составили процессы-серверы заданий. Сервер очереди и серверы запросов представляли четвертый уровень иерархической модели.

Выводы к главе 1

Большинство суперкомпьютерных систем соответствуют типовой архитектуре параллельного компьютера, функционирующего в режиме коллективного пользования. В основе такого функционирования лежит понятие параллельного ресурса, представляющего собой некоторое подмножество вычислительных узлов суперкомпьютера, доступных, занятых или неисправных в течение определенного времени. Число вычислительных узлов в параллельном ресурсе и время его простоя/занятости/неисправности определяют размер (площадь) ресурса. Для полу-

чения доступа к параллельному ресурсу пользователь должен сформировать задание, включающее прикладную параллельную программу, требования к параллельному ресурсу, как минимум, к его размеру и входные данные. Управление вычислительными ресурсами в этом случае сводится к распределению поступающих пользовательских заданий по вычислительным узлам суперкомпьютера. Задачи управления, такие как прием входного потока заданий, ведение очереди заданий, выделение и освобождение параллельных ресурсов для заданий, контроль выполнения заданий, мониторинг состояния и учет потребления параллельных ресурсов осуществляют специальные программные системы – системы управления заданиями (СУЗ).

Функционирование СУЗ определяется требованиями, такими как

− универсальность, как отсутствие ограничений на выполняемые пользователями прикладные программы, и связанные с этим требования надежности и императивности управления;

− круглосуточная доступность, автоматическое функционирование с организацией режимов профилактики и обслуживания;

− прозрачность алгоритмов планирования и справедливое распределение вычислительных ресурсов, под которым понимается невозможность длительного захвата одним пользователем большого объема вычислительных ресурсов;

− автоматизация формирования типового программного окружения, под которым понимается автоматическая настройка заданий пользователя на выбранный типовой стек системного, инструментального и прикладного программного обеспечения;

− модульность архитектуры СУЗ, позволяющая менять функционал системы за счет добавления или замены программных модулей;

− возможность реконфигурации в процессе функционирования, включающая динамическое изменение состава вычислительных узлов суперкомпьютера и связанную с этим виртуализацию представления параллельного ресурса для пользователя.

В целях упорядочивания решаемых системами управления заданиями задач предложена пятиуровневая иерархическая модель управления вычислительными ресурсами в суперкомпьютерной системе коллективного пользования. Уровни иерархии построенной модели отражают степени распараллеливания и технологические этапы обработки входного потока пользовательских заданий от их планирования на уровне распределенной вычислительной системы до векторизации программного кода на уровне процессорного ядра. На основе предложенной иерархической модели и в соответствии с выдвинутыми требованиями разработана архитектура системы управления заданиями, которая была реализована в виде программной Системы управления прохождением параллельных заданий (СУППЗ).

Глава 2. Построение системы управления заданиями пользователей суперкомпьютера на основе иерархической модели

2.1 Технические и технологические решения для управления вычислительными ресурсами суперкомпьютерных систем коллективного пользования на разных уровнях иерархической модели

За годы развития СУППЗ под руководством и при участии автора был проведен ряд исследований и разработок по созданию средств управления вычислительными ресурсами для каждого из уровней иерархии предложенной модели. Результатом стал комплекс технических и технологических решений, одни из которых были реализованы непосредственно в структуре СУППЗ, а другие функционируют в тесном взаимодействии с последней. Рассмотрим предложенные решения средства по уровням иерархической модели управления вычислительными ресурсами.

Для первого уровня иерархии, на котором решения принимаются пользователем в его прикладной программе, были проведены представленные в п. 2.1.1 исследования в области автоматизации создания контрольных точек приложения.

Для второго уровня иерархии, уровня управления отдельным ВУ, исследования и эксперименты проводились в области виртуализации и контейнеризации пользовательских заданий, результаты отражены в п. 2.1.2

На третьем уровне иерархии осуществляется управление подмножеством ВУ как подсистемой суперкомпьютера, выделенной определенному заданию. Для этих целей были созданы рассмотренные в п. 2.1.3 средства сопряжения СУППЗ и сторонних СУЗ, а также разработаны представленные в главе 4 метод и средства поиска отображения параллельной программы на вычислительные узлы суперкомпьютера. Кроме этого, были проведены исследования и разработки в области организации параллельных вычислений с распараллеливанием по данным, подробно рассмотренные в главе 5.

Четвертый уровень иерархии – уровень распределения подсистем суперкомпьютера между заданиями. На этом уровне были предложены методы планирования заданий, представленные в главе 3. Кроме этого рассмотренные в п. 2.1.4 технические решения для этого уровня иерархии включают методы и средства применения в составе СУППЗ сторонних планировщиков и многоресурсного планирования заданий, а также поддержки обработки заданий с заданным уровнем обслуживания.

Для пятого, высшего уровня иерархии проведены исследования по созданию территориально распределенной среды для высокопроизводительных вычислений. К этому же уровню следует отнести рассмотренные в п. 2.1.5 работы по созданию и развитию подсистемы сбора и обработки статистики СУППЗ.

2.1.1 Первый уровень иерархии: средства автоматизации создания контрольных точек для пользовательских параллельных программ

Практика суперкомпьютерных центров коллективного пользования показывает, что время нахождения задания в очереди может составлять от нескольких часов до нескольких дней. В этих условиях администраторы суперкомпьютерных центров вынуждены ограничивать время, которое может быть выделено для выполнения отдельного задания. Например, в МСЦ РАН это время ограничено 24 часами, за исключением отдельных суперкомпьютерных систем, где время ограничено 7 сутками. Подобные ограничения отрицательно сказываются на качестве обслуживания тех пользователей, которым необходимы длительные расчеты. В СУППЗ предусмотрен механизм т.н. фоновых заданий (см. главу 3), которые могут выполняться произвольное время, но при этом могут прерываться системой с возвратом в очередь. Для организации длительных расчетов пользователь может применить этот механизм, но задача сохранения состояния вычислений (контрольных точек) перед возвращением задания в очередь и восстановления состояния вычислений после возобновления выполнения задания целиком и полностью ложится на самого пользователя. Однако, самостоятельная организация контрольных точек

(КТ) возможна, как правило, только при использовании авторских кодов. Если же пользователь применяет в расчетах сторонние программные пакеты, далеко не всегда присутствует возможность сохранения состояния вычислений.

Для параллельных программ, способных масштабироваться до большого числа процессорных ядер, возникает еще одна проблема – суммарный размер контрольных точек всех параллельных процессов может быть настолько большим, что время сохранения и восстановления состояния вычислений может оказаться неприемлемо долгим. С другой стороны, наблюдается тенденция роста числа процессорных ядер как на отдельных кристаллах, так и на вычислительных узлах суперкомпьютеров. Например, вычислительный узел суперкомпьютера МВС-10П ОП [10] на базе процессора Intel Xeon Broadwell содержит 32 процессорных ядра, а узел на базе процессора Intel Xeon Icelake – 64 ядра. Для достаточно большого числа параллельных программ это является пределом масштабируемости, подобные задания заказывают для своего выполнения один вычислительный узел, при этом могут проводить в очереди длительное время. Несмотря на то, что потребление ресурсов заданиями, требующими для выполнения единственный узел, не превышает 5%, их число достаточно велико и может достигать до 40% от общего числа заданий. Для таких заданий актуально применение средств автоматического сохранения контрольных точек.

Для возможности эффективного применения в суперкомпьютерных системах коллективного пользования средств автоматического сохранения контрольных точек последние должны удовлетворять следующим требованиям.

1. Поддержка ядер операционных систем для суперкомпьютеров. Ядро ОС является фундаментом, на который опирается стек программного обеспечения суперкомпьютера [10]. Замена ядра критична для всего стека и поэтому, как правило, ядро ОС не меняется на протяжении всего жизненного цикла суперкомпьютерной системы. В то же время значительное число решений в области автоматического сохранения состояния вычислений часто рассчитаны на версии ядра Linux, не совместимые с применяемыми в суперкомпьютерах ядрами.

2. Средство автоматического сохранения контрольных точек не должно требовать модификации кода пользовательской программы. Напомним, что СУППЗ удовлетворяет требованию универсальности и не накладывает ограничений на применяемые пользователями при подготовке параллельных программ алгоритмы, языки программирования и инструментальные средства. На выполнение поступает широкий спектр различных типов параллельных программ, и фактически невозможно потребовать от разработчиков внесения изменений для поддержки автоматического сохранения контрольных точек.

3. Поддержка как многопоточных (разработанных при помощи OpenMP или pthread), так и многопроцессных (разработанных при помощи MPI) пользовательских программ. Средство автоматического сохранения КТ должно «понимать», что параллельная программа представляет собой довольно сложный динамически изменяющийся объект, состоящий из множества взаимодействующих друг с другом процессов и потоков.

4. Поддержка сохранения состояния объектов ядра ОС, таких как дескрипторы открытых файлов, области разделяемой памяти, очереди IPC, семафоры и т.п. Например, особая структура в области памяти ядра хранит информацию о местоположении открытого программой файла, режимах ввода-вывода, блокировках, текущем положении указателя чтения-записи, и эту структуру необходимо будет восстановить после возобновления выполнения программы с контрольной точки. Если параллельная программа состоит из нескольких взаимодействующих процессов, их идентификаторы также будет необходимо восстановить после рестарта программы.

5. Минимальное влияние на производительность вычислений. Средство автоматического сохранения КТ не должно привносить в процесс вычислений существенных накладных расходов, снижающих быстродействие.

6. Возможность применения без получения привилегий суперпользователя. Поскольку на характер пользовательских программ не накладывается ограничений, никакой административной проверке перед выполнением эти программы не

подвергаются. В такой ситуации применение средства, требующего привилегий суперпользователя, потребует, как минимум, дополнительного аудита и в любом случае снизит уровень информационной безопасности.

7. Лицензия должна позволять свободное применение в научном суперкомпьютерном центре коллективного пользования.

В рамках исследований [106] был проанализирован ряд средств автоматического создания контрольных точек. Были выделены свободно распространяемые продукты Checkpoint Restore In Userspace (CRIU) [107] и Distributed MultiThreaded Checkpointing (DMTCP) [108], для которых были проведены экспериментальное исследование их возможностей и характеристик и анализ влияния средств поддержки КТ на производительность вычислений.

Экспериментальное исследование [106] средств автоматического сохранения КТ показало, что наиболее полно сформулированным требованиям соответствует продукт DMTCP, оказывающий минимальное влияние на производительность вычислений, не требующий изменений в исходных текстах программ и способный функционировать с правами обычного пользователя. Оценка производительности DMTCP показала нецелесообразность использования сжатия при сохранении контрольных точек. Для интеграции продукта с СУППЗ была разработана соответствующая утилита, включение которой в сценарий выполнения задания позволит пользователю применять сохранение КТ в фоновых заданиях.

2.1.2 Второй уровень иерархии: виртуализация и контейнеризация пользовательских заданий

Цифровую экосистему суперкомпьютерного центра определяет прежде всего принятый порядок обслуживания пользователей и их заданий. Часто этот порядок оказывается несовместим с исторически сложившейся технологией организации расчетов, которую новые пользователи применяли до обращения в суперкомпьютерный центр. Освоение принятого порядка работы, т.е. вхождение в новую экосистему может повлечь необходимость полной реорганизации вычислительно-

го процесса, вплоть до реинжиниринга программных кодов. Подобная реорганизация может потребовать от пользователя в том числе отказа от работы с привычными программными средами и пакетами. Кроме этого, переход в новую экосистему повлечет затраты на переобучение сотрудников. Освоение суперкомпьютера новыми пользователями в такой ситуации будет проходить легче, если адаптировать привычные для пользователя технологию организации расчетов и соответствующие ей программную среду и пакеты.

Выходом видится применение технологий виртуализации для возможности переноса привычной пользователю программной среды в суперкомпьютерный центр. Как было показано в работе коллектива Института системного программирования им. В.П. Иванникова РАН [32], а также в проведенных в МСЦ РАН при участии автора работах [73, 109], гипервизорная виртуализация вносит неоправданно высокие накладные расходы в процесс высокопроизводительных вычислений. Более привлекательной с точки зрения накладных расходов выглядит контейнерная виртуализация [72, 110].

В работах [111, 112] в качестве одного из методов применения контейнерной виртуализации рассмотрена подготовка специального репозитория контейнеров с разными (в идеале – со всевозможными) стеками программного обеспечения. Метод предполагает, что необходимые для выполнения задания окружение и программная среда упаковываются в контейнер, который размещается системным администратором суперкомпьютерного центра в специально организованном репозитории. При запуске задания, прошедшего через очередь СУЗ, образ контейнера автоматически извлекается из репозитория, после чего на выделенных для задания ВУ суперкомпьютера разворачивается отдельная сеть контейнеров. Внутри этих контейнеров начинают выполняться процессы задания. При завершении задания контейнеры останавливаются и удаляются с модулей суперкомпьютера.

Пользователь при запуске задания в СКЦ выбирает из репозитория наиболее подходящий для него контейнер. В проведенном под руководством автора исследовании [112] показаны преимущества метода репозитория: простота и

надежность реализации и высокая безопасность. Метод был успешно применен при создании программного комплекса с распараллеливанием по данным XP-COM [113], который подробно рассмотрен в п. 5.6 а также для представленной в п. 2.1.3 интеграции СУППЗ с системой Sun Grid Engine.

Второй метод [75] подразумевает для пользователя возможность запуска любого подготовленного им контейнера с произвольным наполнением и выбранным пользователем стеком ПО. Реализация этого варианта значительно сложнее, для возможности ее осуществления был решен ряд следующих задач.

1. Были разработаны способ представления задания в виде контейнера, методика и средства применения этого способа пользователем. Способ позволяет пользователю в рамках заданных правил и ограничений упаковывать в контейнер произвольный стек инструментального и прикладного ПО. Предложенная методика формирования образа такого контейнера включает рекомендации по обеспечению работоспособности коммуникационной библиотеки MPI внутри контейнера.

2. Была обеспечена безопасность вычислений в пользовательских контейнерах с организацией технической невозможности превышения пользователем своих привилегий.

3. Был разработан способ конфигурирования подсети развёрнутых на ВУ контейнеров таким образом, чтобы выполняющиеся внутри контейнеров процессы параллельной программы «видели» друг друга и могли совершать информационные обмены.

4. Были разработаны сценарии для развёртывания на выделенных ВУ представленных в виде контейнеров пользовательских заданий, контроля их выполнения и сворачивания контейнеров после завершения заданий.

В итоге оба метода были реализованы для СУППЗ, что позволило представлять суперкомпьютерные задания в виде контейнеров в Центре коллективного пользования вычислительными ресурсами МСЦ РАН.

2.1.3 Третий уровень иерархии: совмещение управления вычислительными ресурсами со сторонними системами управления заданиями

Все функции СУЗ по уровням иерархической модели можно условно разделить на две группы. Первую группу составляют функции управления заданиями: прием входного потока различных заданий от разных пользователей, ведение их очереди, построение расписания запусков. Эти функции относятся к пятому и четвертому уровням иерархии управления. Вторую группу составляют функции управления ресурсами: выделение ресурсов для прошедших очередь заданий, конфигурация выделенных ресурсов, запуск заданий, контроль их выполнения, освобождение ресурсов после завершения заданий. Эти функции осуществляются на третьем и втором уровнях иерархии.

Нетрудно видеть, что функции управления заданиями одинаковы для любых типов суперкомпьютеров и не зависят от архитектуры ВУ. Функции управления ресурсами, наоборот, во многом определяются архитектурой и комплектацией ВУ. По этой причине функции управления ресурсами часто выносятся в отдельную подсистему командных сценариев, в которые системный администратор суперкомпьютера может вносить изменения, отражающие специфику вычислительной системы. В СУППЗ функции управления ресурсами реализованы в надстройке сценариев управления заданием, подготовки, диагностики, выделения и освобождения узлов. Замена сценариев в этой надстройке позволяет с минимальными трудозатратами подключить для управления вычислительными ресурсами возможности сторонних СУЗ. В практике применения СУППЗ было осуществлено два подобных интеграционных проекта: совместное управление ресурсами с известными системами Sun Grid Engine (SGE) и SLURM.

Для ряда пользователей МСЦ РАН исторически сложилась технология расчетов, основу которой составляет вычислительная инфраструктура под управлением Sun Grid Engine (SGE) [90, 91]. На каждом из находящихся под управлением SGE вычислительных узлов функционирует клиентский процесс SGE. Так же, как и в любой СУЗ, пользователи подключаются к серверу доступа (в терминологии

SGE – узлу входа), на котором формируют свои расчетные задания и направляют их в очередь SGE. SGE распределяет задания из очереди по клиентским ВУ.

В свою очередь, СУППЗ полностью управляет ВУ суперкомпьютера, поэтому простое совместное применение СУППЗ и SGE для управления одними и теми же ВУ суперкомпьютера невозможно. Поскольку производимые под управлением SGE расчеты носят не постоянный, а периодический характер, то выделение под управление SGE отдельного сегмента суперкомпьютера нецелесообразно. Для устранения этого противоречия было принято решение об использовании рассмотренного в п. 2.1.2 метода представления пользовательских заданий в виде контейнеров.

Для реализации среды контейнеризации была применена известная система Docker [114]. Весь необходимый набор программ, библиотек, пользовательского окружения и клиентский сервер SGE были упакованы в образ Docker, который был размещен в специально созданном репозитории. Образ из репозитория может быть извлечен запущенным заданием, прошедшим через очередь СУППЗ. Успешный запуск задания и, соответственно, контейнеров на ВУ суперкомпьютера фактически означает развертывание вычислительной инфраструктуры SGE в режиме коллективного доступа. Далее пользователи могут обычным порядком обращаться к мастер-серверу SGE и помещать свои расчетные задания в очередь SGE. Эти задания планировщиком SGE будут автоматически распределяться по контейнерам, запущенным под управлением СУППЗ на ВУ суперкомпьютера. Полная очистка ВУ от всех возможных остатков вычислений достигается простым удалением контейнера по завершении задания СУППЗ.

В результате была создана подсистема разворачивания среды выполнения SGE под управлением СУППЗ. Подробное изложение предложенных для ее создания технических решений приведено в публикации [112].

Подсистемы сценариев управления заданием, подготовки, диагностики, выделения и освобождения узлов имеют в своем составе практически все современные СУЗ, в том числе широко распространенная система SLURM [16, 94]. Осо-

бенностью SLURM является значительный объем программного обеспечения управления ресурсами (командных сценариев), разработанного производителями ВУ и системными интеграторами для поставляемого ими суперкомпьютерного оборудования. По этой причине в МСЦ РАН, начиная с суперкомпьютера МВС-10П (2013 г.) [115], практикуется совместное функционирование СУППЗ и SLURM. На СУППЗ возлагаются функции управления заданиями, а часть функций управления ресурсами осуществляется путем применения готовых командных сценариев из состава SLURM.

Однако, при появлении в 2017 году ВУ на базе микропроцессоров Intel Xeon Phi KNL [116] ситуация изменилась. Архитектура KNL предполагает несколько альтернативных режимов работы, поддержку которых компания Intel осуществила не в виде командных сценариев подсистемы управления ресурсами, а внедрила непосредственно в ядро SLURM [117]. В результате управление ВУ на базе процессоров KNL без применения SLURM стало невозможным. Суперкомпьютер МВС-10П МП2 KNL [10] был предоставлен пользователям под управлением SLURM, но оказался слабо востребованным по сравнению с системами на базе СУППЗ. По этой причине было принято решение об интеграции СУППЗ и SLURM, для осуществления которого под руководством и при участии автора были проведены соответствующие разработки [102, 103].

Основной принцип интеграции, представленный на рисунке 10, состоит в следующем. Все функции управления заданиями, включая командный интерфейс с пользователями и ведение очереди заданий, возлагаются на СУППЗ. Прошедшее через очередь СУППЗ задание помещается в SLURM, очередь которой всегда искусственно поддерживается пустой. При передаче задания СУППЗ сообщает SLURM список ВУ, на которых следует запустить задание.

Поскольку очередь SLURM пуста, переданное задание немедленно поступает на выполнение на указанных в списке узлах. При этом за тем, чтобы один узел не выделялся одновременно более чем одному заданию, отвечает СУППЗ. Она же предоставляет пользователям привычный командный интерфейс.

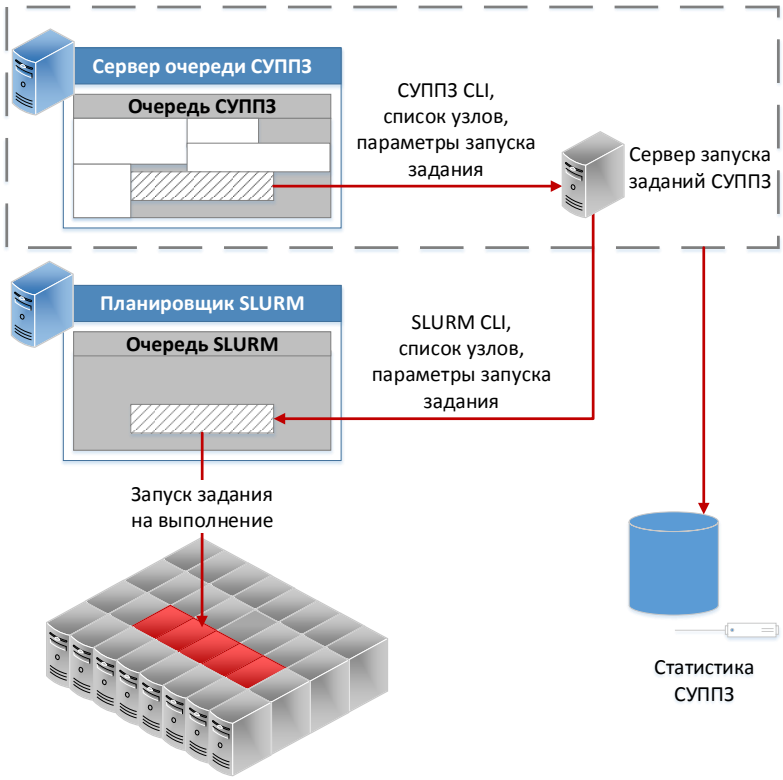

Рисунок 10. Принцип интеграции СУППЗ и SLURM

SLURM при такой схеме выполняет все функции управления ресурсами суперкомпьютера, независимо от того, реализованы они в системе командных сценариев или в ядре SLURM. При этом пользовательскому заданию становится доступным окружение SLURM, пользуясь которым можно осуществлять желаемую настройку суперкомпьютерного оборудования в полном соответствии с документацией и рекомендациями производителя. Другими словами, при такой схеме интеграции получается оптимальное сочетание функций управления: СУППЗ обеспечивает привычные интерфейс и порядок работы с очередью заданий, SLURM позволяет использовать все богатство возможностей высокопроизводительной программно-аппаратной платформы. Отдельно следует отметить, что предложенный подход позволил в полной мере продолжать использовать подсистему сбора и обработки статистики СУППЗ.

При практической реализации рассмотренного принципа интеграции был решен ряд научно-технических задач. Во-первых, был разработан механизм предотвращения несанкционированного доступа пользователей к SLURM в обход очередь СУППЗ. Во-вторых, в SLURM обеспечена поддержка фоновых заданий

СУППЗ. В-третьих, компенсировано различие в подходах к резервированию ресурсов. В-четвертых, осуществлена поддержка командных сценариев SLURM для обеспечения полноты функций управления ресурсами.

Интеграция SLURM и СУППЗ в полной мере продемонстрировала принципы модульности иерархической архитектуры СУППЗ и делегирования полномочий управления в высоконадежные компоненты. SLURM в этом случае выступил в качестве такого компонента. Делегирование функций управления второго и третьего уровней иерархии позволила совместить порядок обслуживания пользователей, принятый в СУППЗ и составляющий экосистему суперкомпьютерного центра МСЦ РАН, и возможности управления вычислительными ресурсами, включенные производителями оборудования в функции SLURM. Разработанная интегрированная система была внедрена в Центре коллективного пользования вычислительными ресурсами МСЦ РАН – отделения МСЦ Курчатовского высокопроизводительного вычислительного комплекса НИЦ «Курчатовский институт» и обслуживает пользовательские задания на всех суперкомпьютерах линейки МВС-10П.

2.1.4 Технические и технологические решения четвертого уровня иерархии

2.1.4.1 Логические подсистемы СУППЗ

СУППЗ поддерживает разбиение на две и более логических систем. ВУ, отнесенные к той или иной логической системе, могут пересекаться, но могут быть и разными. Кроме этого, механизм логических систем позволяет работать одной управляющей ЭВМ сразу с несколькими вычислителями. Логические системы различаются по именам. На рисунке 11 показаны две логические системы – sys1 и sys2. Каждая система имеет собственный сервер очереди – qserver1 и qserver2, соответственно. Сервер запросов обращается к серверам очередей разных систем по разным идентификаторам. Сервер очереди каждой из логических систем может использовать для запуска заданий свой собственный сервер запуска. Имя системы автоматически добавляется к имени задания в качестве префикса.

Рисунок 11. Пример логических подсистем СУППЗ

За счет механизма логических подсистем обеспечивается функционирование нескольких разделов суперкомпьютера МВС-10П ОП [10], установленного в МСЦ РАН – отделении МСЦ КВВК НИЦ «Курчатовский институт».

2.1.4.2 Многоресурсное планирование в СУППЗ

Достаточно часто суперкомпьютерные системы подвергаются модернизации в процессе эксплуатации. При этом нередко проводится модернизация не всей системы, а только некоторой ее части, что приводит к неоднородности решающего поля. Разные вычислительные узлы могут содержать процессоры разной мощности, разные объемы оперативной памяти и т.п. Если разнородность не учитывать при планировании, то выделение ресурсов будет производиться неэффективно. Более оснащенные ВУ могут простаивать или выделяться под задания, которым не требуется их полная мощность. Для поддержки планирования множества вычислительных ресурсов в ядро СУППЗ была добавлена соответствующая возможность [118]. При постановке задания в очередь пользователь может указать, какие дополнительные вычислительные ресурсы (помимо процессоров) требуются для его задания. Дополнительные вычислительные ресурсы специфицируются системным администратором. При этом каждый специфицированный дополнительный ресурс имеет имя и может быть числовым или строковым. Числовые вычислительные ресурсы выделятся по принципу «не меньше, чем задано

пользователем», а строковые – по принципу «точного соответствия значению, заданному пользователем».

Частным случаем многоресурсного планирования в СУППЗ является поддержка планирования ресурсов локальной дисковой памяти (ЛДП) вычислительных узлов, которая помогает в решении проблемы сохранения больших объемов промежуточных результатов долгосрочных вычислений. Если задание пользователя фоновое, т.е. может периодически прерываться системой, то, заказав ресурс ЛДП, пользователь получает возможность сохранять контрольные точки своего задания от одного рестарта до другого. СУППЗ гарантирует пользователю, что при каждом рестарте его задание будет выполняться на тех ВУ, где размещен заказанный ресурс ЛДП.

2.1.4.3 Поддержка уровней обслуживания пользователей

При наличии большого числа пользователей время пребывания задания в очереди может достигать нескольких часов и даже дней. Для ряда пользователей, чьи задания носят срочный характер, длительное ожидание в очереди неприемлемо. Простое повышение приоритета таких заданий часто оказывается недостаточным – даже находясь первым в очереди, высокоприоритетное задание будет ожидать завершения ранее запущенных заданий. Кроме этого, в системе одновременно может оказаться несколько высокоприоритетных заданий, которым неизбежно придётся ожидать друг друга. Для таких случаев в СУЗ предусматривается механизм урегулирования конфликтов между высокоприоритетными заданиями, обычно называемый «уровнем обслуживания» или «соглашением об уровне обслуживания» (англ. – Service Level Agreement, SLA).

В таких СУЗ, как коммерческие продукты LSF от компании IBM Platform и Moab от компании Adaptive Computing, присутствуют механизмы поддержки различных уровней обслуживания. Предполагается, что пользователь может заключить с суперкомпьютерным центром отдельное соглашение SLA, и СУЗ будет сконфигурирована так, чтобы требования соглашения соблюдались автоматически.

В Moab для соглашений об уровне обслуживания реализован механизм обеспечения заданного качества обслуживания (англ. – Quality of Service, QoS) [119]. С механизмом QoS можно производить немедленный запуск задания с вытеснением остальных заданий, назначение приоритета, подбор ВУ заданной конфигурации, запрет обратного заполнения при обработке приоритетных заданий и др. Среди этих возможностей наиболее важное значение имеют обработка заданий к заданному сроку (англ. – deadline) и предварительное резервирование ресурсов. Для отдельного задания или группы заданий сроки выполнения могут быть указаны специально. Для таких заданий сначала оценивается возможность соблюдения этих сроков. Если такая возможность есть, Moab добавляет задание в список срочных и выделяет ресурсы, гарантируя, что все принятые срочные задания будут обработаны не позднее запрошенных сроков. При этом планировщик оптимизирует время использования ресурсов, старясь выполнить все задания в кратчайшие сроки.

Возможность определения крайних сроков для обработки задания обеспечивается и в СУЗ LSF [120]. Так же, как и в Moab, эта возможность реализуется за счёт предварительного резервирования ресурсов в определённый период времени. В LSF это называется «временны́м окном». Кроме резервирования в определённый период, LSF поддерживает уровни обслуживания, обеспечивающие обработку определённого числа заданий в течение заданного времени или к заданному сроку.

Для поддержки уровней обслуживания в СУППЗ при участии автора было проведено исследование [121], по результатам которого пользователям было предоставлено две возможности. Первая – гарантированный запуск задания не позднее определённого срока, вторая – наличие определённого объёма свободных ресурсов к заданному пользователем времени (резервирование ресурсов на заданное время). Последнее подразумевает запуск привилегированных заданий в заранее выделенных временны́х окнах, т.е. на заранее зарезервированных на определённое время под запуск задания ВУ суперкомпьютера.

Конкретные значения по гарантированным срокам запуска заданий, числу этих заданий, объёму и времени резервируемых ресурсов пользователи готовы оговаривать в SLA. В соглашении также отражается, что в ситуации, когда привилегированный запрос на гарантированный запуск заданий или на резервирование ресурсов не может быть удовлетворён в силу высокой загрузки суперкомпьютера, пользователь извещается об этом при постановке задания в очередь. В этом случае соглашение SLA оставляет за пользователем выбор: либо перенести сроки запуска задания или резервирования ресурсов, либо поставить задание в очередь с высоким приоритетом, но без гарантии сроков запуска, либо отказаться от запроса.

2.1.4.4 Применение сторонних планировщиков в составе СУППЗ

Модульность СУППЗ позволяет заменять компоненты с сохранением общего построения и характеристик архитектуры. Проведенные под руководством автора исследования [112, 122, 123] продемонстрировали практическую возможность замену штатного сервера очереди СУППЗ на сторонний планировщик. В практике применения СУППЗ было осуществлено два таких проекта: подключение к СУППЗ планировщиков Maui и Sun Grid Engine. Последний был рассмотрен в п. 2.1.3 , здесь осветим вопросы подключения Maui.

Планировщик Maui [88] в начале 2000-х годов получил широкое распространение в мировой практике построения и применения СУЗ. Maui разрабатывался с середины 1990-х годов компанией Cluster Resources как программный продукт с открытым исходным текстом. Это обстоятельство привлекло к его развитию значительное число специалистов мирового сообщества HPC. Интеграцию с Maui осуществили многие известные СУЗ: OpenPBS/Torque, LoadLeveler, Sun Grid Engine, LSF. Для интеграции были разработаны соответствующие интерфейсы, главными недостатками являлись их ориентация на конкретную СУЗ и отсутствие документации. Первоначально исследование пошло по следующему пути. В качестве основы было решено рассмотреть связку Maui-OpenPBS/Torque, т.к. данный вариант на практике доказал свою стабильную работоспособность. Было вы-

двинуто предположение, что существует четкий интерфейсный разрез между Maui и Torque, через который взаимодействуют эти системы. В этом случае задача сводилась бы к выявлению интерфейсного разреза и его поддержке со стороны СУППЗ. Другими словами, система OpenPBS/Torque заменялась на СУППЗ с сохранением интерфейса с Maui. При этом Maui, работая в уже в составе СУППЗ, «считал» бы, что находится в составе OpenPBS/Torque.

К сожалению, указанный путь привел в тупик. Отсутствие документации на интерфейсный модуль Maui-OpenPBS заставило обратиться к исходным текстам, исследование которых показало, что четкого интерфейсного разреза между Maui и OpenPBS/Torque не существует. Более того, нечеткое соблюдение принципа модульности в архитектуре OpenPBS/Torque привело к тому, что, например, Maui получил возможность непосредственного управления вычислительными узлами. При этом во время трансляции исходных текстов Maui для включения поддержки интерфейса OpenPBS/Torque было необходимо наличие исходных кодов OpenPBS/Torque.

В итоге было принято решение воспользоваться другим преимуществом Maui – наличием отдельного административного интерфейса Wiki [124]. Интерфейс Wiki предполагает клиент-серверную архитектуру взаимодействия, в роли сервера (Wiki-сервер) выступает сторонняя система управлением заданиями, а роли клиента (Wiki-клиент) – планировщик Maui. Задача интеграции в этом случае сводится к разработке некоторого сервера-посредника, обеспечивающего трансляцию протокола Maui в протокол СУЗ и обратно. Отметим, что на момент начала работ не существовало готовых решений интеграции с Maui через интерфейс Wiki, публикации о реализации подобных проектов появились позднее [125].

На рисунке 12 представлена структура разработанного сервера-посредника. Для обеспечения взаимной трансляции протоколов сервера очереди СУППЗ и интерфейса Wiki была организована [123] подсистема внутреннего представления вычислительных ресурсов и заданий. Изначальная конфигурация ВУ определялась конфигурационными файлами СУППЗ, изменения в конфигурации осу-

ществлялись через запросы серверов запуска и серверов заданий СУППЗ. Информация о заданиях извлекалась из паспортов заданий, передаваемых анализатору конфигурационных файлов сервером запросов СУППЗ при постановке заданий в очередь. Через анализатор запросов Maui информация о текущем состоянии заданий и ВУ передавалась планировщику Maui. Последний составлял расписание запусков заданий и выдавал команды на запуск очередного задания, которые через сервер-посредник транслировались серверу запуска СУППЗ.

Под управление СУППЗ с интегрированным планировщиком Maui был помещен отдельный сегмент суперкомпьютера МВС-100К [66, 126], установленного в МСЦ РАН. Сегмент получил название Mixa, включал в себя 32 ВУ МВС-100К с общим числом процессорных ядер 256 и функционировал в 2009-2012 гг. в качестве учебного кластера. За это время на сегменте было обслужено свыше 70 пользователей, выполнивших более 10 тыс. заданий.

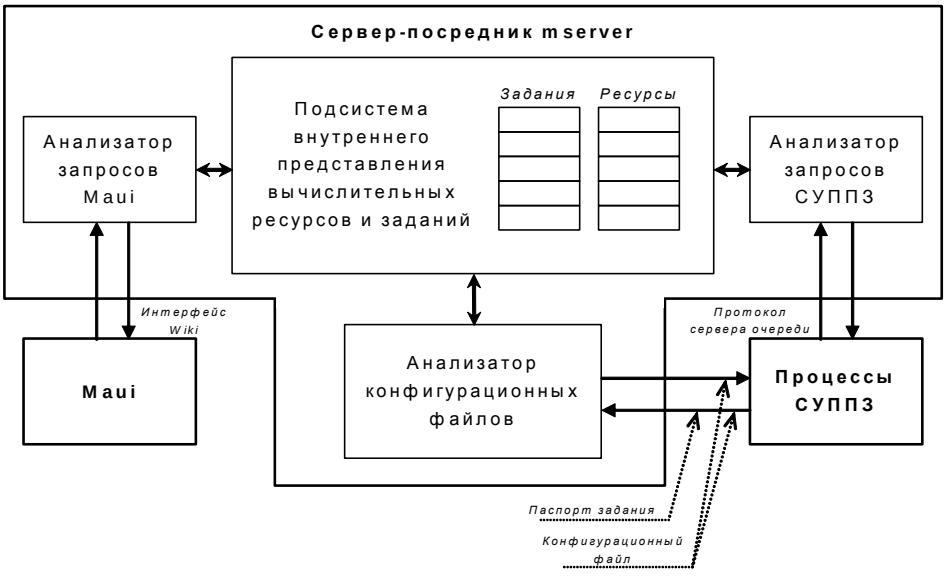

Рисунок 12. Структура сервера-посредника для интеграции планировщика Maui в состав СУППЗ

К сожалению, после приобретения Cluster Resources компанией Adaptive Computing планировщик Maui был включен в состав коммерческой СУЗ Moab HPC Suite, поддержка изначального проекта Maui была прекращена, что постепенно в течение нескольких лет остановило его развитие. В связи с этим плани-

ровщик Maui не смог вытеснить со своего места штатный планировщик СУППЗ. Тем не менее, успешная интеграция Maui продемонстрировала преимущества модульной архитектуры СУППЗ, позволившей осуществить такой проект.

2.1.5 Технические и технологические решения пятого уровня иерархии

2.1.5.1 Сбор и обработка статистики работы суперкомпьютерной системы

Одной из высокоуровневых задач управления вычислительными ресурсами сбор и обработка различной статистической информации, позволяющей пользователям оценивать результаты своей работы на суперкомпьютере, системным администраторам – осуществлять мониторинг работоспособности и функциональности вычислительных систем, руководству центра коллективного пользования – анализировать эффективность работы суперкомпьютерного оборудования центра и планировать его дальнейшую деятельность. Для этих целей в СКЦ организуются специальные системы, включающие как базы данных (БД), накапливающие соответствующую статистическую информацию, так и средства доступа к этой информации и ее анализа.

В состав СУППЗ входит подсистема сбора, хранения и обработки статистической информации (подсистема «Статистика»), включающая базу данных [127] и специальное программное средство «МСЦ-КроСтат» [128].

Подсистемы сбора и обработки статистической информации о своей работе содержат все распространенные СУЗ. Сбор информации заключается в фиксации статистически значимых событий, фиксируются время наступления события и его параметры. К основным статистически значимым событиям относятся:

– поступление нового задания (постановка задания в очередь);

– запуск задания на выполнение после прохождения очереди и, соответственно, занятие заданием определенного параллельного ресурса;

– завершение выполняющегося задания и, соответственно, освобождение занятых заданием вычислительных ресурсов;

– снятие задания из очереди до его запуска;

– старт и остановка СУППЗ;

– смена режима работы СУППЗ;

– изменение состава ВУ решающего поля по причине отказа (неисправности) определенных ВУ или, наоборот, их восстановления после отказа (возврата из ремонта).

Обработка собранной информации заключается в формировании статистических отчетов, отражающих за определенный период времени как потребление вычислительных ресурсов пользователями (биллинг), так и показатели качества работы суперкомпьютерной системы, такие как загрузка ресурсов, коэффициент замедления заданий, число обработанных заданий и др. Статистическая информация может собираться отдельно по пользователям, по организациям, по научным проектам.

Информация о происшедших статистически значимых событиях фиксируется СУППЗ на управляющей ЭВМ в специальном журнале событий. С помощью «МСЦ-КроСтат» информация из журналов событий заносится в БД «Статистика» под управлением свободно распространяемой СУБД PostgreSQL. После разбора журналов событий и заполнения БД становится возможным формирование ряда статистических отчетов, содержащих как биллинговую информацию, так и информацию о загрузке ресурсов.

2.1.5.2 Пользовательские интерфейсы СУППЗ

СУППЗ обладает собственным пользовательским командным интерфейсом, архитектурно реализованным в виде соответствующей надстройки. Этот интерфейс сложился исторически, реализация многих команд производилась в ответ на запросы пользователей для удовлетворения их актуальных потребностей. Для обеспечения соответствия стандарту POSIX 1003.2d Batch Environment Standard на системы пакетной обработки для СУППЗ был разработан соответствующий стандарту интерфейс [118], совместимый с СУЗ PBS.

Следование рекомендациям POSIX необходимо для возможности интеграции в состав СУППЗ соответствующих стандарту компонентов различных изготовителей, а также для обеспечения эффективности переноса прикладного программного обеспечения. Пользователи, работавшие с POSIX-совместимыми системами семейства PBS, накопили большое количество командных сценариев для запуска и сопровождения задания. При переходе таких пользователей на СУППЗ они смогут использовать ранее созданные программные инструменты. При создании POSIX-совместимого командного интерфейса СУППЗ было решено ориентироваться на СУЗ OpenPBS/Torque [86], которая соответствует спецификации POSIX 1003.2d Batch Environment Standard. В результате был реализован [118] POSIX-совместимый набор команд, обеспечивающий проведение полного цикла работы пользователя с системой управления заданиями.

2.1.5.3 Организация распределенных вычислительных сред

Одним из ключевых методов повышения доступности и эффективности использования ресурсов суперкомпьютерных центров является их объединение в единую распределенную сеть. Появление в начале 2000-х годов фундаментальных работ [129, 130] дало старт организации высокопроизводительных распределенных вычислительных сред в виде т.н. грид-систем. Подобные исследования и разработки более 20 лет осуществляются при участии автора в МСЦ РАН – Отделении суперкомпьютерных систем и параллельных вычислений НИЦ «Курчатовский институт».

В работе [131] отражены результаты работ по созданию сетевой среды распределенных вычислений (ССРВ) – грид-системы, представляющей собой совокупность связанных коммуникационными каналами вычислительных систем (ВС), доступную для пользователей как единый вычислительный ресурс. ВС ССРВ могут различаться по архитектуре, составу программных и аппаратных средств, политике администрирования т.п. и могут использоваться как в составе ССРВ, так и вне ее. Входящие в состав ССРВ ВС могут иметь собственную ло-

кальную систему планирования заданий, которая получает задания либо непосредственно от пользователей локальной ВС, либо из глобальной очереди ССРВ. В рамках реализованного проекта [131, 132], в качестве локальной системы планирования заданий была использована СУППЗ, для сопряжения с которой был разработан специальный интерфейс ССРВ. Разработанный интерфейс, в том числе, обеспечивал возможность синхронного запуска ветвей параллельной программы из одного задания на разных ВС ССРВ.

Развитие ССРВ получила в виде проекта Российской инфраструктуры для суперкомпьютерных приложений [133], в рамках которого были объединены вычислительные ресурсы филиалов МСЦ РАН в Москве, Санкт-Петербурге и Казани. Суперкомпьютерные сегменты в каждом из филиалов работали под управлением СУППЗ.

Актуальность вопросов создания территориально распределенной высокопроизводительной вычислительной среды (ТРС) отражена в работах [134, 135]. ТРС предполагает объединение разных вычислительных систем, принадлежащих разным организациям, в конфедерацию суперкомпьютерных центров. По решению руководства суперкомпьютерного центра, только часть из имеющихся в центре ВС может войти в состав ТРС, оставшаяся часть оборудования может использоваться исключительно под нужды и задачи центра. Для управления отдельной ВС используется локальная система управления ресурсами (ЛСУР), действия всех ВС в сети координирует глобальная система управления ресурсами (ГСУР). В качестве ЛСУР может выступать любая система управления заданиями (СУППЗ, SLURM, PBS и т.п.).

После включения ВС в состав сети ее вычислительные ресурсы не отчуждаются от их владельца и продолжают использоваться для выполнения локальных заданий, которые образуют локальный поток заданий. Задания этого потока поступают на вход определенной ВС и могут выполняться на ресурсах только этой ВС.

После включения ВС в состав сети совместно с заданиями локального потока на вычислительные ресурсы ВС начинают поступать задания с других ВС – из

глобального потока заданий. В отличие от локального, задания глобального потока допускают обработку на вычислительных ресурсах любой ВС сети (либо ВС из заданного списка), ГСУР распределяет задание глобального потока в ту ВС сети, время обработки задания в которой будет минимальным. Вопросы создания методов и алгоритмов глобального планирования заданий в ТРС подробно рассмотрены в работах [75, 136].

Важно отметить, что ГСУР не планирует задания на локальные ресурсы ВС, а только выбирает т.н. целевую ВС для размещения задания. Размещенное задание глобального потока поступает под управление ЛСУР целевой ВС и планируется наряду с заданиями локального потока.

2.2 Сравнение качественных характеристик СУППЗ и ведущих систем управления заданиями

Под качественными характеристиками (свойствами) системы управления заданиями мы будем понимать наличие или отсутствие определенных функций или возможностей. С точки зрения определения перечня качественных характеристик систем управления заданиями наиболее полной является фундаментальная работа научного коллектива Массачусетского технологического института [12]. В этой работе качественные характеристики (англ. – features) классифицированы по группам, и проведено сравнение наиболее распространенных систем по всем характеристикам. Рассмотрим выделенные авторами [12] группы качественных характеристик и определим их наличие или отсутствие у СУППЗ по выделенным группам. Данные о наличии или отсутствии определенных свойств у других систем возьмем из [12].

1. К общим характеристикам систем управления заданиями относятся следующие.

Развитие СУППЗ продолжается, новые возможности добавляются.

Распространение. Исходные коды СУППЗ открыты.

Поддерживаемая ОС: Linux.

Языки программирования. Поскольку СУППЗ удовлетворяет требованию универсальности, ограничений на использование пользователями языков программирования нет.

Защита от НСД. СУППЗ обеспечивает авторизацию, разграничение и контроль доступа пользователей к вычислительным ресурсам.

Отказоустойчивость. В случае отказа управляющей ЭВМ существуют возможность перезапуска СУППЗ с сохранением стоящих в очереди и выполняющихся заданий.

Сводная информация по сравниваемым системам приведена в таблице 1.

Таблица 1. Общие характеристики систем управления заданиями

Характеристика	LSF	Open Lava	Grid Engine	SLURM	PBS	СУППЗ
1	2	3	4	5	6	7
Развитие	+	+	+	+	+	+
Распространение	платно	открыт.	платно и открыт.	открыт.	платно и открыт.	открыт.
Поддерживаемая ОС	Linux	Linux Cygwin	Unix	Unix	Unix	Linux
Языки программирования	все	все	все	все	все	все
Защита от НСД	+	+	+	+	+	+
Отказоустойчивость	+	+	+	+	+	+

2. Виды поддерживаемых заданий.

Разнотипные задания: поддерживает ли система разнотипные задания, другими словами различает ли СУЗ задания разных классов. Рассмотренный в п. 3.3 метод планирования заданий в СУППЗ основан на разделении заданий на классы: отладочные, ординарные, фоновые.

Параллельные задания. СУППЗ поддерживает как задания с взаимодействующими процессами, так и задания с распараллеливанием по данным.

Массивы заданий: поддерживает ли система синхронные зависимые параллельные и/или асинхронно независимые параллельные задания. СУППЗ поддерживает только асинхронно независимые задания.

Ведение нескольких очередей: поддерживает ли система одну или несколько очередей заданий. СУППЗ поддерживает несколько очередей заданий через рассмотренный в п. 2.1.4.1 механизм логических подсистем.

Метапланирование: поддерживает ли система многоуровневое планирование в распределенной вычислительной среде. СУППЗ может быть элементом распределенной вычислительной среды, участвуя в рассмотренной в п. 2.1.5.3 двухуровневой системе глобального планирования в территориально распределенной вычислительной среде.

Сводная информация о видах заданий по сравниваемым системам приведена в таблице 2.

Таблица 2. Виды поддерживаемых заданий

Характеристика	LSF	Open Lava	Grid Engine	SLURM	PBS	СУППЗ
1	2	3	4	5	6	7
Разнотипные задания	+	+	+	+	+	+
Параллельные задания	+	+	+	+	+	+
Массивы заданий	+	+	+	+	+	
Несколько очередей	+	+	+	+	+	+
Метапланирование						+

3. Методы планирования заданий.

Разделяемость ВУ: поддерживает ли система одновременное выполнение разных заданий на одном ВУ. СУППЗ поддерживает разделение ВУ, однако поскольку это противоречит требованиям императивности управления и надежности, такая возможность, как правило, на практике не задействуется.

Обратное заполнение: использует ли планировщик стратегию обратного заполнения. Планировщик СУППЗ поддерживает эту стратегию.

Пакетирование заданий: поддерживает ли объединение однотипных заданий в группы (пакеты). Для СУППЗ разработана отдельная подсистема пакетирования заданий [137].

Упаковка заданий: это метод распределения ресурсов, при котором планировщик выбирает группы заданий для одновременного запуска на ВУ или подмножестве ВУ, чтобы повысить загрузку вычислительных ресурсов. Поскольку подобная упаковка неизбежно приведет к возрастанию конкуренции заданий за ресурсы ВУ и, как следствие, – к ослаблению изоляции заданий, СУППЗ не поддерживает этот метод.

Групповое планирование позволяет пользователю помещать в очередь несколько процессов в рамках одного слота задания. Только один из этих процессов может выполняться в один момент времени, т.е. обычно это последовательно выполняющиеся процессы. В СУППЗ такая возможность доступна в рамках сценария выполнения задания, в котором пользователь может указать последовательность запусков процессов в рамках одного задания.

Зависимости заданий: есть ли у пользователя возможность определять зависимости выполнения между заданиями при помощи направленных ациклических графов. У СУППЗ не поддерживает зависимости между заданиями.

Сводная информация о методах планирования заданий по сравниваемым системам приведена в таблице 3.

Таблица 3. Поддерживаемые методы планирования заданий

Характеристика	LSF	Open Lava	Grid Engine	SLURM	PBS	СУППЗ
1	2	3	4	5	6	7
Разделяемость ВУ	+	+	+	+	+	+
Обратное заполнение	+	+	+	+	+	+
Пакетирование заданий			+			+
Упаковка заданий				+		
Групповое планирование				+		+
Зависимости заданий	+	+	+	+		

4. Управление ресурсами.

Гетерогенность: поддерживает ли система управление гетерогенными ВУ. В п. 2.1.4.2 было рассмотрено многоресурсное планирование в СУППЗ, направленное на поддержку гетерогенных вычислительных ресурсов.

Политика распределения ресурсов: есть ли возможность разработки и применения правил использования ресурсов. СУППЗ обладает развитым механизмом планирования заданий, правила планирования задаются специальным расписанием, составляемым системным администратором. Подробно применямые методы планирования изложены в главе 3.

Динамические ресурсы: есть ли возможность помимо управления статическими ресурсами, такими как процессорные ядра, планировать потребление динамически изменяющихся ресурсов, таких как оперативная память. СУППЗ не поддерживает планирование динамических ресурсов.

Отображение заданий: учитывается при распределении заданий по ВУ топология коммуникационной сети. Предлагаемые автором для СУППЗ метод и алгоритмы отображения заданий рассмотрены в главе 4.

Сводная информация о методах управления ресурсами по сравниваемым системам приведена в таблице 4.

Таблица 4. Поддерживаемые методы управления ресурсами

Характеристика	LSF	Open Lava	Grid Engine	SLURM	PBS	СУППЗ
1	2	3	4	5	6	7
Гетерогенность	+	+	+	+	+	+
Политика распределения ресурсов	+	+	+	+	+	+
Поддержка динамических ресурсов	+	+	+	+		
Отображение заданий	+		+	+	+	+

5. Управление очередью заданий.

Интеллектуальное планирование: поддерживает ли система методы планирования заданий, отличные от стратегии FCFS (First Come First Serve). Метод и алгоритмы планирования СУППЗ основаны на достаточно сложном разделении заданий на классы и приоритеты с поддержкой стратегии обратного заполнения.

Схема приоритетов определяет возможности управления приоритетами заданий разных пользователей. В СУППЗ наличествует развитая система динамических приоритетов заданий, подробно рассмотренная в главе 3.

Резервирование ресурсов определяет возможности пользователя по заказу одного или нескольких слотов заданий в расписании запусков. Такая возможность доступна в СУППЗ как в виде отдельной команды предоставления параллельного ресурса, так и при помощи рассмотренного в п. 2.1.4.3 механизма поддержки уровней обслуживания пользователей.

Замена и переупорядочивание заданий. СУППЗ не предоставляет пользователю возможности заменять и/или переупорядочивать задания в очереди.

Учет энергопотребления: возможность управлять числом включенных ВУ для минимизации потребления энергии суперкомпьютерной системой. В СУППЗ подобные механизмы представлены в виде результатов исследований [138, 139]. Кроме этого, за счет рассмотренного в п. 2.1.3 механизма сопряжения СУППЗ со сторонними системами возможности последних могут быть использованы для управления энергопотреблением ВУ.

Управление размещением заданий на ВУ: есть ли у пользователя возможность указывать, может ли его задание (или нет) быть размещено на ВУ, на которых выполняются другие задания, в т.ч. задания другого пользователя. Поскольку СУППЗ соблюдает принцип неделимости ВУ, такая возможность не предоставляется.

Назначение заданий на ВУ с требуемыми данными возможно в СУППЗ за счет механизма ЛДП.

Сводная информация о возможностях управления очередью заданий по сравниваемым системам приведена в таблице 5.

Таблица 5. Возможности управления очередью заданий

Характеристика	LSF	Open Lava	Grid Engine	SLURM	PBS	СУППЗ
1	2	3	4	5	6	7
Интеллектуальное планирование	+	+	+	+	+	+
Схема приоритетов	+	+	+	+	+	+
Резервирование ресурсов	+		+	+	+	+
Замена и изменение порядка заданий	+	+	+	+	+	
Учет энергопотребления	+		+	+	+	+
Управление размещением заданий на ВУ	+		+	+	+	
Назначение заданий на ВУ с требуемыми данными						+

6. Управление выполнением заданий.

Поддержка пролога/эпилога: это функция, которая позволяет выполнять командные сценарии до и/или после выполнения задания. В СУППЗ эта функция обеспечивается за счет надстройки сценариев диагностирования и подготовки ВУ.

Перемещение данных: возможность копирования вспомогательных файлов в локальное хранилище ВУ, в [12] отмечаются, что эту возможность можно считать устаревшей по мере развития сетевых хранилищ данных. Возможность включается системным администратором СУППЗ в соответствующий сценарий подготовки ВУ.

Контрольные точки: позволяют сохранять состояние вычислений запущенных заданий, чтобы в случае сбоя можно было перезапустить задания и восстановить их из контрольной точки. СУППЗ предоставляет механизм фоновых заданий (см. главу 3), контрольная точка может быть сохранена при помощи рассмотренных в п. 2.1.1 средств.

Миграция заданий: возможность переместить выполняющееся задание на другой параллельный ресурс. Такая возможность в СУППЗ не поддерживается.

Следует особо отметить, что отмеченное в [12] наличие подобной возможности в других СУЗ касается только тех заданий, которые состоят из одного процесса. Такие задания имеют слабое отношение к высокопроизводительным вычислениям, однако часто встречаются в анализируемых в [12] системах обработки больших данных. Миграция многопроцессного задания, использующего параллельный ресурс в режиме взаимодействия процессов, невозможна ни в одной из СУЗ.

Перезапуск заданий. Отмеченная в [12] возможность перезапустить задание в случае аварии в СУППЗ автоматически обеспечивается для фоновых заданий.

Вытеснение заданий: возможность вытеснить с выполнения низкоприоритетное задание, отправив его в режим гибернации на ВУ в СУППЗ не поддерживается. Опять-таки в отношении остальных СУЗ гибернация заданий возможно только в случае занятия ими одного узла одним процессом, что важно для систем обработки больших данных, но не для суперкомпьютеров.

Сводная информация о возможностях управления выполнением заданий по сравниваемым системам приведена в таблице 6.

Таблица 6. Возможности управления выполнением заданий

Характеристика	LSF	Open Lava	Grid Engine	SLURM	PBS	СУППЗ
1	2	3	4	5	6	7
Пролог/эпилог	+		+	+	+	+
Перемещение данных	+		+	+	+	+
Контрольные точки	+	+	+	+	+	+
Миграция заданий	+	+	+	+		
Перезапуск заданий	+	+	+	+	+	+
Вытеснение заданий	+	+	+	+	+	

Сравнительный анализ возможностей СУЗ показывает, что СУППЗ обладает подавляющим большинством качественных характеристик ведущих СУЗ, что позволяет говорить о соответствии СУППЗ мировому уровню развития СУЗ. Рассмотрим наиболее часто применяемые количественные показатели качества СУЗ.

2.3 Количественные характеристики систем управления заданиями

К количественным характеристикам СУЗ следует отнести измеряемые тем или иным способом показатели качества. Рассмотрим наиболее употребительные показатели качества СУЗ.

Пусть некоторое задание было поставлено в очередь в момент времени t_q, запустилось в момент времени t_e и завершилось в момент времени t_f. Тогда время $Q = t_e - t_q$ будет **временем ожидания задания в очереди**, а время $E = t_f - t_e$ будет **временем выполнения задания**. В работе [18] вводится понятие **времени пребывания задания в системе** (англ. – walltime) T_W, определяемое как

$$T_W = t_f - t_q = Q + E \quad (1)$$

Для снижения времени T_W необходимо минимизировать времена Q и E. Очевидно, что время T_W пребывания задания в системе есть некоторый общий показатель, характеризующий как качество планирования заданий, напрямую влияющее на время Q ожидания задания в очереди, так и производительность суперкомпьютерной системы, во многом определяющую время выполнения задания E. Однако, как будет показано в главе 4, СУЗ также может влиять на время E путем построения оптимального отображения графа параллельной программы на граф связности ВУ суперкомпьютера, выделенных для выполнения задания.

Возьмем некоторый, достаточно длительный интервал T работы суперкомпьютера и разобьём этот интервал на последовательные периоды $\Delta t_i,\ i = 1 \dots t,\ T = \sum_{i=1}^{t} \Delta t_i$. В течение каждого отдельного периода Δt_i не изменяются значения N_i общего числа доступных ВУ и M_i числа занятых выполнением заданий ВУ, т.е. за каждый отдельный период времени Δt_i не происходит добавления или изъятия ВУ из решающего поля, а также не запускаются или завершаются задания. Собственно, смена периодов Δt_i и Δt_{i+1} означает изменение либо значения N_i, либо значения M_i. Определим как N_{all} площадь параллельного ресурса, доступного для выполнения заданий за интервал времени T:

$$N_{all}(T) = \sum_{i=1}^{t} N_i \Delta t_i$$

Определим как M_{all} суммарную площадь выполнявшихся на суперкомпьютерной системе заданий за тот же интервал T:

$$M_{all}(T) = \sum_{i=1}^{t} M_i \Delta t_i$$

Определим как R_i заказанное время выполнения i-го задания, т.е. то время выполнения, которое пользователь указал при постановке задания в очередь как требование к необходимому параллельному ресурсу. Определим как Q_i время ожидания i-го задания в очереди, а как E_i – реальное время выполнения i-го задания.

Определим **загрузку** U вычислительных ресурсов суперкомпьютера за определённый интервал времени T как

$$U(T) = \frac{M_{all}(T)}{N_{all}(T)} \quad (2)$$

В некоторых случаях используют показатель простоя вычислительных ресурсов $I(T)$, как величину, противоположную загрузке:

$$I(T) = 1 - U(T) \quad (3)$$

Пусть за интервал времени T было выполнено n заданий. Тогда **среднее время ожидания задания в очереди** $Q(T)$ за интервал T может быть рассчитано как

$$Q(T) = \frac{\sum_{i=0}^{n} Q_i}{n} \quad (4)$$

Отметим, что в реальных системах разброс значений Q_i может быть значительным. По этой причине часто вместо среднего времени ожидания задания в очереди применяется медианное значение этого показателя.

Само по себе среднее время ожидания задания в очереди не является информативным показателем и может служить лишь для сравнения качества разных СУЗ на одном и том же входном потоке заданий. Поясним этот факт на следующем примере. Пусть два разных задания ожидали в очереди одно и то же время, например, 1 час. При этом первое задание выполнялось 10 часа, а второе –

10 минут. Очевидно, что время ожидания является достаточно малым для первого задания, поскольку составляет только 0,1 от времени его выполнения, и неприемлемым для второго задания, поскольку превышает время его выполнения в 6 раз. Для нормализации времени ожидания задания в очереди это время приводят либо ко времени выполнения задания *E*, либо к заказанному времени выполнения *R*. Последнее применяется для заданий с фиксированными параметрами, поскольку именно время *R* используется планировщиком СУЗ для составления расписания запусков заданий. Приведенное время ожидания задания характеризует его относительную задержку в очереди и поэтому часто в литературе называется **коэффициентом замедления** (англ. – slowdown).

В работе [18] коэффициент замедления предлагается ограничивать сверху, поскольку значительная часть заданий в реальной системе завершается аварийно по разным причинам (как правило, из-за ошибок в программе пользователя). Средний коэффициент замедления n заданий за интервал времени T в соответствии с [18] предлагается рассчитывать как

$$S(T) = \frac{1}{n}\sum_{i=1}^{n} \max(S_i, 1)\,, S_i = \frac{R_i + Q_i}{\max(R_i, \tau)} \qquad (5)$$

Параметр τ применяется для исключения из расчетов аварийно завершившихся заданий с аномально коротким временем выполнения, не превышающим τ. Очевидно, что минимальное значение коэффициента замедления равно 1, что соответствует отсутствию ожидания заданий в очереди.

Коэффициент замедления рассчитывается по фактическому времени выполнения задания, в то время как планировщик СУЗ составляет расписание, исходя из времени выполнения, заказанного пользователем при постановке задания в очередь. Пусть B_i – заказанное пользователем время выполнения i-го задания. Определим величину *V(T)* – **среднее значение времени нахождения задания в очереди относительно заказанного времени выполнения**:

$$V(T) = \frac{1}{n}\sum_{i=1}^{n} \frac{Q_i}{B_i} \qquad (6)$$

Рассмотрим систему управления заданиями с точки зрения системной инженерии. Тогда в соответствии с ГОСТ Р 59341 2021 величину T_W следует рассматривать как время реакции системы на запрос пользователя в виде задания. ГОСТ Р 59341 2021 определяет **своевременность предоставления требуемой информации в системе** как «свойство системы обеспечивать предоставление запрашиваемой или выдаваемой принудительно (автоматически) выходной информации в задаваемые сроки, гарантирующие выполнение соответствующей функции согласно целевому назначению системы».

Стандарт предполагает наличие в системе I различных типов запросов, для каждого типа задаются следующие величины:

– λ_i – интенсивность поступления в систему запросов i-го типа;

– $T_{зад\,i}$ – пороговое значение для критерия своевременности по среднему времени реакции: среднее время обработки запросов i-го типа T_i должно быть не более задаваемого $T_{зад\,i}$, в стандарте условие этого критерия упоминается как условие своевременности α_1;

– $P_{св\ зад\,i}$ – пороговое значение для вероятностного критерия: вероятность своевременной обработки запросов i-го типа в системе $P_{св\,i}(T_{зад\,i}) = P_{св\,i}(\tau_i \le T_{зад\,i})$ за заданное время $T_{зад\,i}$ должна быть не ниже задаваемой $P_{св\ зад\,i}$ (в стандарте условие этого критерия упоминается как условие своевременности α_2), где τ_i – случайная величина, означающая время реакции системы при обработке запросов i-го типа.

На основе заданных величин λ_i, $T_{зад\,i}$, $P_{св\ зад\,i}$ ГОСТ Р 59341 2021 определяет такой показатель, как **относительная доля своевременно обработанных в системе запросов** $C_{своевр}$. Показатель $C_{своевр}$ охватывает лишь те типы запросов, для которых выполнены требования заказчика, и в соответствии с ГОСТ Р 59341 2021 этот показатель вычисляют по формуле:

$$C_{\text{своевр}} = \sum_{i=1}^{I} \lambda_i P_{\text{св}\,i}\left(T_{\text{зад}\,i}\right)[Ind(\alpha_1) + Ind(\alpha_2)] \Big/ \sum_{i=1}^{I} \lambda_i \qquad (7)$$

Критерии своевременности обработки каждого типа запросов устанавливают с использованием индикаторной функции *Ind(α)*, которая принимает либо значение 1 при выполнении условия критерия α, либо значение 0 в противном случае.

Запросами в СУЗ являются задания пользователей. При постановке задания в очередь пользователь, как правило, не задает величин $T_{зад\,i}$, $P_{св\,зад\,i}$ и указывает только заказанное время выполнения задания, которое СУЗ использует для составления расписания и определении прогнозируемого времени запуска заданий. О своевременности обработки запроса пользователя можно сделать вывод по следующему признаку. Если пользователя не устраивает прогнозируемое время запуска его задания, то он удаляет это задание из очереди до начала его выполнения.

Пусть *i*-й пользователь за период времени T направил в систему n_i заданий. Тогда интенсивность поступления его заданий можно оценить как $\lambda_i = \frac{n_i}{T}$. Пусть d_i – число заданий, который *i*-й пользователь за время T удалил из очереди до начала выполнения, т.е. то число заданий, для которых не было выполнено условие своевременности α_1. Тогда вероятность $P_{св\,i}(T_{зад\,i})$ можно оценить как $P_{\text{св }i}\left(T_{\text{зад }i}\right) = 1 - \frac{d_i}{n_i}$.

Пусть I пользователей за время T направили в систему $n = \sum_{i=1}^{I} n_i$ заданий. Будем считать, что у каждого пользователя свои требования по своевременности обработки его заданий. В этом случае выполнение условия своевременности α_1 для каждого пользователя будет определяться неявно подразумевающимся этим пользователем временем $T_{зад\,i}$. Определим величину d удаленных из очереди заданий за время T как $d = \sum_{i=1}^{I} d_i$. Тогда с учетом (7) показатель $C_{своевр}$ приобретает следующий вид:

$$C_{\text{своевр}} = \sum_{i=1}^{I} \lambda_i P_{\text{св }i}\left(T_{\text{зад }i}\right) \Big/ \sum_{i=1}^{I} \lambda_i = \sum_{i=1}^{I} \frac{n_i}{T}\left(1 - \frac{d_i}{n_i}\right) \Big/ \sum_{i=1}^{I} \frac{n_i}{T} = 1 - \frac{d}{n} \qquad (8)$$

Показатель $C_{своевр}$, рассчитанный в соответствии с (8), является частным случаем показателя доли соевременно обработанных запросов, определенного в ГОСТ Р 59341 2021.

В [12] в качестве одного из основных показателей качества исследуется **задержка планирования** (англ. – scheduler latency) t_l, под которой авторы понимают накладные расходы СУЗ на запуск одного задания. При этом выделяется два случая:

– если среднее время выполнения задания $t \gg t_l$, то влияние накладных расходов СУЗ невелико и им можно пренебречь;

– если среднее время выполнения задания $t \lessapprox t_l$, то влиянием накладных расходов СУЗ пренебречь нельзя.

Второй случай соответствует заданиям с длительным временем инициализации, для планирования которых требуются специальные методы [70, 74].

Будем считать загрузку вычислительных ресурсов и средний коэффициент замедления заданий основными количественными показателями качества СУЗ. Отметим, что руководство и администрация суперкомпьютерного центра обычно стремится к максимизации загрузки вычислительных ресурсов, а пользователи заинтересованы в минимизации среднего коэффициента замедления. В дальнейшем в диссертации мы будем пользоваться главным образом основными показателями качества.

К дополнительным показателям качества СУЗ можно отнести следующие:

– средняя длина очереди, определяющая среднее число находившихся в очереди заданий за рассматриваемый период;

– полезная загрузка вычислителя, учитывающая время инициализации заданий как простой вычислительных ресурсов [70];

– пропускная способность системы как количество обслуженных заданий за временной период;

– сбалансированность загрузки как равномерная загруженность всех вычислительных ресурсов;

– «честность» по отношению СУЗ к заданиям как дисперсия времени ожидания заданий в очереди;

– полнота охвата как отношение числа обработанных к директивному сроку заданий к общему числу заданий.

Как уже было отмечено, все указанные показатели качества отражают прежде всего эффективность планирования заданий в СУЗ. Кроме этого, могут применяться показатели качества, отражающие возможности масштабирования СУЗ [12, 140]:

– максимальное число обслуживаемых вычислительных узлов;

– максимальное число одновременно планируемых заданий.

Обычно в современных СУЗ значения этих показателей заведомо превышают актуальные потребности суперкомпьютерного центра.

2.4 Сравнение количественных показателей качества СУППЗ и SLURM

В 2013 году под руководством автора было проведено исследование [140] по сравнению качества систем управления заданиями СУППЗ и SLURM. Для сравнения были выделены следующие количественные показатели.

1. Загрузка ресурсов, определяемая в соответствии с (2).

2. Масштабируемость СУЗ по числу ВУ, другими словами, максимальное число ВУ, которым СУЗ способна управлять.

3. Масштабируемость по числу заданий, т.е. максимальное число планируемых заданий.

4. Среднее значение времени нахождения задания в очереди относительно заказанного времени его выполнения в соответствии с (6).

Определение показателей качества планирования СУППЗ и SLURM производилось на статистических данных суперкомпьютера МВС-100К, установленного в МСЦ РАН. Подробно состав системы МВС-100К рассмотрен в п. 2.5 , а также в работах [66, 126]. Статистические данные по работе МВС-100К были взя-

ты за период с 6 по 12 декабря 2012 года. В этот период под планирование заданий в системе было отведено 5 824 процессора (728 8-процессорных ВУ).

Для определения показателей SLURM был использован симулятор [141], на основе которого был собран экспериментальный стенд с виртуальным вычислителем с параметрами, аналогичными кластеру МВС 100К (728 8-процессорных ВУ).

Для корректного сравнения показателей качества планирования разных СПО было необходимо обеспечить для них одинаковые входные потоки заданий. С этой целью из базы данных подсистемы «Статистика» СУППЗ [114] была извлечена следующая статистическая информация за рассматриваемый период с 6 по 12 декабря 2012 года, включающая:

– имя пользователя, поставившего задание в очередь;

– время поступления задания в систему;

– число ядер, требуемое для выполнения задания;

– время выполнения, заказанное пользователем;

– фактическое время выполнения задания.

На основе указанной информации был сформирован модельный поток заданий, поданный на вход симулятора SLURM на экспериментальном стенде. В процессе вычислительного эксперимента симулятор SLURM сохранял результаты симуляции (время поступления, запуска и останова заданий и др.) в собственную базу данных. После окончания симуляции показатели качества планирования были рассчитаны с помощью SQL-запросов к этой базе данных.

При расчёте показателей качества планирования было необходимо учесть, что на момент начала симуляции виртуальный вычислитель SLURM, в отличие от реального вычислителя МВС-100К, не был загружен никакими заданиями, и первые выполненные задания будут запускаться на счёт без ожидания в очереди. Поскольку это напрямую влияет на показатели качества, анализ статистики был проведен не с момента запуска, а с момента заполнения виртуального вычислителя. По этой причине статистика учитывалась, начиная с третьего модельного дня работы стенда.

На вход симулятора SLURM был подан модельный поток из 17 392 фиктивных заданий. Первое поступившее задание по времени соответствовало заданию, поставленному в очередь СУППЗ МВС-100К 4 декабря 2012 года в 0 часов 0 минут и 1 секунду. Показатели качества планирования для SLURM рассчитывались за тот же период времени, что и для СУППЗ, т.е. с 6 по 12 декабря 2012 года. За этот период SLURM успела запустить 13 500 заданий против 13 748, выполненных СУППЗ.

Результаты сравнения показателей представлены в таблице 7.

Таблица 7. Показатели качества СУЗ SLURM и СУППЗ

Показатель	SLURM	СУППЗ
1	2	3
Загрузка ресурсов за исследуемый период в соответствии с (2)	0,94	0,97
Средний коэффициент замедления заданий за исследуемый период в соответствии с (5)	9,67	9,05
Среднее значение времени нахождения задания в очереди относительно заказанного времени счёта за исследуемый период в соответствии с (6)	0,34	0,23
Доля своевременно обработанных заданий за исследуемый период в соответствии с (8)	–	0,978
Число запущенных заданий за исследуемый период	13 500	13 748

Из работы [12] известно, что SLURM может управлять системами размером до 65 536 ВУ и одновременно обрабатывать свыше 100 тыс. заданий. Максимальное число одновременно планируемых заданий в СУППЗ ограничено разработчиками системы и равняется 512. Задания, поступившие в систему сверх этого числа, принимаются СУППЗ, но на планирование не поступают, ожидая освобождения места в очереди. Опыт эксплуатации СУППЗ свидетельствует о том, что система способна обслуживать решающее поле, состоящее из 1 500 ВУ. Следует отметить, что указанные параметры СУППЗ заметно перекрывают потребности суперкомпьютерных центров, в которых эксплуатируется система. Полученные

результаты позволяют говорить о примерном паритете СУППЗ и SLURM и одинаково высоком качестве планирования заданий в обеих системах.

2.5 Результаты эксплуатации Системы управления прохождением параллельных заданий

Эксплуатация СУППЗ осуществляется с 1999 года по настоящее время на множестве суперкомпьютерных систем МСЦ РАН (ныне – отделения МСЦ Курчатовского высокопроизводительного вычислительного комплекса НИЦ «Курчатовский институт»), ИПМ им. М.В. Келдыша РАН. Кроме этого, СУППЗ применялась на ряде суперкомпьютерных систем, поставлявшихся НИИ «Квант» институтам РАН. Рассмотрим эти системы.

1. Серия вычислительных установок МВС-1000 [64] штатно оснащалась СУППЗ в варианте архитектуры, рассмотренном в п. 1.9 Системы серии МВС-1000 были установлены в МСЦ РАН и ИПМ им. М.В. Келдыша РАН в 1999 году и функционировали до 2007 года. Состав суперкомпьютеров МВС-1000 рассмотрен в п. 1.9 установки в разных комплектациях содержали 32 и 64 вычислительных узла.

2. Первый российский суперкомпьютер МВС-1000М [66, 142] сферы науки и образования, перешагнувший терафлопсный рубеж производительности и занявший 64-место в рейтинге Топ-500 самых производительных суперкомпьютеров мира. Место установки – МСЦ РАН. МВС-1000М состоял из 384 двухпроцессорных ВУ на базе микропроцессоров DEC Alpha 21264 667 МГц. ВУ объединялись в решающее поле при помощи низколатентной высокоскоростной сети Myrinet. Пиковая производительность МВС-1000М – 1024 ГФлопс.

Система эксплуатировалась с 2001 по 2006 год. С 2005 года проводилось демасштабирование системы [66]: отдельные сегменты были установлены в ИВМиМГ СО РАН и Казанском научном центре РАН.

3. Серия суперкомпьютеров МВС-1000/16 и МВС-1000/32 производства НИИ «Квант». Системы состояли из 16 и 32 ВУ соответственно. ВУ содержали по

одному процессору Intel Pentium III и объединялись друг с другом при помощи сети Fast Ethernet. Системы были поставлены в ИПМ им. М.В. Келдыша РАН, ИММ УрО РАН [143], МФТИ [144], Ивановский государственный энергетический университет им. В.И. Ленина [145], ИВМиМГ СО РАН [146], ИВМ СО РАН [147], ИАПУ ДВО РАН [148]. Системы активно эксплуатировались с 2001 по 2007 год.

4. Суперкомпьютер МВС-1000/RSC4, созданный и установленный в ИПМ им. М.В. Келдыша РАН. Система состояла из 64 двухпроцессорных ВУ на базе процессоров AMD Opteron, ВУ объединялись сетью Myrinet. Годы эксплуатации – 2004-2010.

5. Суперкомпьютер МВС-15000 [66] с пиковой производительностью 10,1 Тфлопс в 2005 году занял 56-е место в списке Топ-500 и стал самым мощным российским суперкомпьютером. В его состав входили 574 двухпроцессорных узла на базе процессоров IBM PowerPC 970FX. Коммуникационная сеть использовала технологию Myrinet и обеспечивала скорость передачи данных между двумя вычислительными узлами с использованием библиотек MPI находилась на уровне 170-180 Мбайт/с. Система эксплуатировалась в МСЦ РАН с 2004 по 2008 год. В 2008 году было проведено демасштабирование системы, отдельные сегменты были установлены в Санкт-Петербургском и Казанском филиалах МСЦ РАН.

6. Суперкомпьютер МВС-6000 [66] содержал 128 двухпроцессорных ВУ на на базе VLIW-процессоров Intel Itanium-2. ВУ объединялись коммуникационной сетью Myrinet. Пиковая производительность МВС-6000 составляла 1,54 Тфлопс (412 место в TOP500). Система эксплуатировалась в МСЦ РАН с 2006 по 2011 год.

7. Суперкомпьютер МВС-100К [66, 126] содержал 1275 вычислительных узлов, 10572 ядра, 21144 виртуальных процессора. 19 узлов были оснащены ускорителями Nvidia Tesla M2090, по 8 ускорителей в каждом узле. Суперкомпьютер занял 38-е место в рейтинге Топ-500 с пиковой производительностью 227,94 Тфлопс. Коммуникационная сеть была построена на базе InfiniBand DDR 20 Гбит/с. Система эксплуатировалась в МСЦ РАН с 2007 по 2021 год.

8. Суперкомпьютер К100 [9, 149] был создан и установлен в ИПМ им. М.В. Келдыша РАН в 2010 году. Является первым отечественным суперкомпьютером гибридной архитектуры, содержащей графические ускорители в составе ВУ. Состоит из 64 ВУ и имеет пиковую производительность 107 ТФлопс. В состав ВУ этой системы, наряду с двумя 6-ядерными процессорами общего назначения, входят три 448-ядерных графических ускорителя модели Fermi производства nVidia. Основу коммуникационной среды составляют сеть Infiniband производства QLogic и сеть собственной (ИПМ им. М.В. Келдыша, НИИ «Квант») разработки. Активная эксплуатация системы продолжалась с 2011 по 2023 год.

9. Суперкомпьютер МВС-Экспресс [150] создан и установлен в ИПМ им. М.В. Келдыша РАН в 2010 году. В состав системы входят шесть ВУ, каждый из которых содержит два четырехядерных процессора AMD Opteron и один спаренный ускоритель NVidia GeForce GTX 295. Узлы соединены между собой каналами PCI Exppress через коммутатор. Активная эксплуатация системы продолжалась с 2011 по 2023 год.

10. Суперкомпьютер МВС-10П Торнадо [66, 115] был установлен в МСЦ РАН в 2013 году. Пиковая производительность 524 ТФлопс позволила занять 59-е место в списке Топ-500. Система включает 208 ВУ, каждый из которых содержит два 8-ядерных процессора Intel Xeon E5-2690 и два 61-ядерных ускорителя Intel Xeon Phi 7110X. Для объединения ВУ применяется сеть InfiniBand FDR, 56 Гбит/с. Годы активной эксплуатации – 2013-2023.

11. Линейка суперкомпьютеров МВС-10П ОП [10, 66], установленных в МСЦ РАН. ВУ этих систем объединены сетью Intel OmniPath. МВС-10П состоит из следующих разделов:

– МВС-10П ОП Broadwell (2017 г.), пиковая производительность 181 ТФлопс, раздел включает 136 ВУ на базе 16-ядерных процессоров Intel Xeon с микроархитектурой Broadwell, каждый узел включает в себя 2 процессора Intel Xeon E5-2697Av4 и 128 ГБ оперативной памяти;

– МВС-10П ОП KNL (2018 г.), пиковая производительность 76 ТФлопс, раздел включает 22 ВУ на базе 72-ядерных процессоров Intel Xeon Phi 7290 с микроархитектурой Knights Landing, каждый узел имеет 96 ГБ оперативной памяти;

– МВС-10П ОП Skylake (2019 г.), пиковая производительность 200,4 ТФлопс, раздел включает 58 ВУ на базе 18-ядерных процессоров Intel Xeon с микроархитектурой Skylake, каждый узел включает в себя 2 процессора Intel Xeon Gold 6154 и 192 ГБ оперативной памяти;

– МВС-10П ОП Cascade Lake (2020 г.), пиковая производительность 872,3 ТФлопс, раздел включает 191 ВУ на базе 24-ядерных процессоров Intel Xeon с микроархитектурой Cascade Lake, каждый узел имеет 192 ГБ оперативной памяти;

– МВС-10П ОП Icelake (2024 г.), пиковая производительность 42,6 ТФлопс, раздел включает 8 ВУ на базе 32-ядерных процессоров Intel Xeon с микроархитектурой Icelake, каждый узел имеет 256 ГБ оперативной памяти.

С 2021 года из состава раздела МВС-10П ОП Cascade Lake был выделен раздел Optane, состоящий из 8 ВУ Cascade Lake, каждый из которых был снабжен дополнительными 768 ГБ оперативной памяти на базе технологии Intel Optane.

12. Суперкомпьютер К60 [9] был установлен в ИПМ им. М.В. Келдыша РАН в 2022 году. В состав системы входят 86 ВУ, каждый из которых содержит два 14-ядерных процессора Intel Xeon с микроархитектурой Broadwell и 10 ВУ с двумя 16-ядерными процессорами Intel Xeon Broadwell и 4-мя ускорителями NVidia Volta V100.

Подробное описание суперкомпьютерных систем МСЦ РАН приведено в диссертации [66]. Описание суперкомпьютерных систем ИПМ им. М.В. Келдыша РАН приведено в работе [9]. Обобщенная статистика о числе пользователей и выполненных ими заданиях на некоторых суперкомпьютерах под управлением СУППЗ приведена в таблице 8. По остальным системам статистика либо не собиралась, либо была утрачена после снятия системы с эксплуатации.

На каждой суперкомпьютерной системе велась отдельная очередь заданий, при этом за указанные в таблице 8 годы эксплуатации (столбец 3) в очереди пла-

нировалось различное число процессорных ядер. Максимальное число ядер указано в столбце 4. Столбец 7 представляет число научных проектов, реализованных с применением высокопроизводительных вычислений исследовательскими группами пользователей, а столбец 8 – число научных и образовательных организаций, сотрудниками которых являлись пользователи.

Таблица 8. Число пользователей и обработанных заданий на некоторых суперкомпьютерных системах под управлением СУППЗ в 2001-2024 гг.

Система	Размещение	Годы	Число ядер, макс.	Число польз.	Число заданий	Число научных проектов	Число орган.
1	2	3	4	5	6	7	8
МВС-1000М	МСЦ РАН – отделение МСЦ КВВК НИЦ «Курчатовский институт»	2001-2006	768	372	339 984	н/д	86
МВС-15000		2004-2008	1146	357	221 903	н/д	81
МВС-6000		2006-2011	254	206	78 089	н/д	49
МВС-100К		2007-2021	9728	964	1 825 797	353	117
МВС-10П		2013-2023	2800	448	389 293	220	75
МВС-10П ОП Broadwell		2017-2024	4352	444	169 695	204	74
МВС-10П ОП KNL		2018-2024	2160	248	43 547	143	57
МВС-10П ОП Skylake		2019-2024	2088	329	46 890	160	61
МВС-10П ОП Cascade Lake		2020-2024	8880	357	118 830	168	67
МВС-10П ОП Optane		2021-2024	288	178	6 604	96	45
МВС-10П ОП Icelake		2024	512	72	3 883	51	32
К100	ИПМ им. М.В. Келдыша РАН	2010-2023	768	258	467 001	32	12
К60		2022-2024	2408	90	63 348	н/д	6

Общее число выполненных под управлением СУППЗ заданий составляет более 3,7 млн. Общее число исследователей, воспользовавшихся услугами СУППЗ, составляет в МСЦ РАН свыше 1240 человек из 135 организаций, а в ИПМ им. М.В. Келдыша РАН – свыше 280 человек. Общее число научных проек-

тов, реализованных при помощи суперкомпьютеров МСЦ РАН, превышает 530, а реализованных при помощи суперкомпьютеров ИПМ им. М.В. Келдыша РАН – превышает 32.

На рисунке 13 представлена диаграмма загрузки вычислительных ресурсов суперкомпьютеров под управлением СУППЗ, достигнутой в длительные (свыше 6 дней) периоды между профилактиками оборудования.

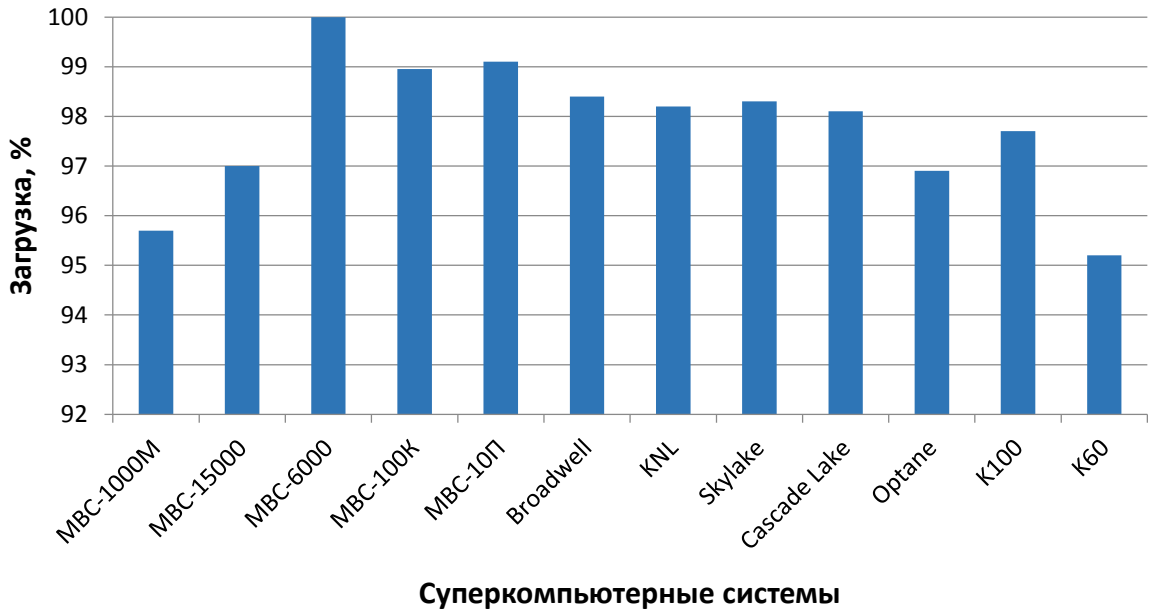

Рисунок 13. Загрузка суперкомпьютеров под управлением СУППЗ

Десятилетия успешной эксплуатации СУППЗ позволяют говорить о сформировавшейся на её основе цифровой экосистемы. В работе [151] такая экосистема определяется как цифровое пространство, построенное на базе одной или нескольких цифровых платформ и включающее в себя совокупность сервисов, которые позволяют пользователям удовлетворять разнообразные потребности в рамках реализации единого бесшовного процесса. В нашем случае именно СУППЗ предоставляет пользователям сервисы высокопроизводительных вычислений в рамках реализации процесса научных исследований. В диссертации [66] под экосистемой суперкомпьютерного центра понимается динамичная совокупность научного оборудования центра, его персонала, пользователей и отношений между ними. Во многом эти отношения определяются системой управления заданиями, на базе которой пользователи выстраивают инструменты и среды разра-

ботки для проведения своих исследований и через которую получают доступ к высокопроизводительным вычислениям.

Выводы к главе 2

Для каждого уровня построенной иерархической модели предложены технические и технологические решения по управлению вычислительными ресурсами этого уровня иерархии. На основе предложенной архитектуры, разработанных технических и технологических решений построена Система управления прохождением параллельных заданий (СУППЗ), обеспечивающая комплекс возможностей, качественных характеристик и количественных показателей, соответствующий мировому уровню.

Для оценки систем управления заданиями был определен ряд основных количественных показателей, к которым отнесены загрузка вычислительных ресурсов и средний коэффициент замедления заданий, отражающие соответственно интересы поставщика ресурсов в виде руководства и персонала суперкомпьютерного центра и потребителя ресурсов в виде пользователей. Сравнительный анализ СУППЗ и ведущей систему управления заданиями SLURM показал примерный паритет сравниваемых систем по рассмотренным количественным показателям.

Практическая значимость СУППЗ подтверждается ее успешным применением в ведущих российских суперкомпьютерах, установленных в 1999-2024 гг. в МСЦ РАН, ИПМ им. М.В. Келдыша РАН и ряде других научных и образовательных организаций. Услугами СУППЗ воспользовались более 1240 пользователей-исследователей из 135 организаций, выполнивших свыше 3,7 млн. заданий при реализации более чем 530 научных проектов.

Объем и характеристики предоставляемых под управлением СУППЗ услуг по высокопроизводительным вычислениям позволяют говорить о формировании на основе СУППЗ цифровой экосистемы в виде информационно-вычислительной среды суперкомпьютерного центра коллективного пользования, доступной практически для всех российских ученых.

Глава 3. Планирование пользовательских заданий

3.1 Существующие методы и средства планирования суперкомпьютерных заданий

Под планированием заданий (англ. – job scheduling) понимается функция составления расписания запусков заданий, находящихся в очереди. Эта функция возлагается на специальный компонент СУЗ, называемый планировщиком (англ. – scheduler). Планировщик является ядром СУЗ, вокруг которого выстраиваются остальные компоненты системы. Роль функции планирования центральная, и поэтому в англоязычной литературе термином «scheduler» часто называют СУЗ целиком [12]. Планировщик управляет всей суперкомпьютерной системой, выделяя для нужд заданий подсистемы ВУ, т.е. средства планирования относятся к четвертому уровню иерархии управления вычислительными ресурсами.

Классификацию и обзор существующих методов планирования заданий можно найти в работе [152], посвященной в большей степени организации вычислений в распределенных средах. В п.1.2.2 была приведена классификация пользовательских заданий. Тип или класс обрабатываемых в СУЗ заданий во многом определяют те методы, алгоритмы и средства, которые будут применяться для планирования заданий этого класса или категории. В настоящей работе предлагаются методы планирования заданий с фиксированными параметрами, адаптивных (эластичных) заданий и нестандартных заданий, поступающих на вход СУЗ в виде виртуальных машин или контейнеров, в том числе от облачных платформ.

3.1.1 Методы и средства планирования заданий с фиксированными параметрами

В большинстве случаев применяемые в практике научных суперкомпьютерных центров методы планирования заданий с фиксированными параметрами опираются на следующие основные принципы.

1. Расписание запусков заданий строится на основе сообщаемых пользователем-владельцем задания характеристик, главными их которых являются число необ-

ходимых для задания ВУ суперкомпьютера и требуемое время вычислений. Эти характеристики задаются пользователем при постановке задания в очередь и в процессе обработки задания не могут быть изменены.

2. Дисциплина обслуживания заданий – **приоритетная невытесняющая**. Другими словами, при планировании применяются относительные приоритеты заданий, которые определяются главным образом приоритетом пользователя-владельца и его группы. Вновь поступающие высокоприоритетные задания не вытесняют выполняющиеся задания с низким приоритетом, а лишь занимают более высокое место в очереди.

Основным методом приоритетного обслуживания практически во всех современных СУЗ является механизм **динамического справедливого распределения (**англ. – fair share**)** [19]. Основная идея здесь состоит в «справедливом» распределении ресурсов – чем больше ресурсов пользователь потребил, тем ниже приоритет его заданий. Рассмотрим этот принцип на примере ведущей мировой СУЗ SLURM. В SLURM по умолчанию задействован алгоритм т.н. «дерева справедливости» (англ. – fair tree) [20].

Для определения приоритетов в этом алгоритме используется упорядоченное дерево, образуемое на основе иерархии учётных записей пользователей. Например, пользователь может быть членом одной проектной группы, несколько таких групп могут составлять один проект, несколько проектов могут принадлежать одной организации и т.д.

Предполагается, что администратор СУЗ распределяет квоты на использование ресурсов на верхнем уровне иерархии, например, среди организаций. Полученные организациями квоты распределяются между проектами администраторами соответствующих организаций. Далее администраторы проектов делят доставшиеся им квоты между группами и т.д. Приоритет конечного пользователя складывается из его квоты и квот его иерархических «предков» в дереве справедливости. Важным моментом при этом является то, что квоты определяют только базовую часть прио-

ритета. Окончательное значение приоритета формируется с учётом того, сколько ресурсов (например, в ядро-минутах) пользовательские задания уже использовали.

Потребление ресурсов в прошлом разбивается на ряд учётных периодов, и чем более удалён по времени учётный период, тем меньшее влияние он оказывает на итоговое значение приоритета нового задания. В SLURM для этого применяется формула «полураспада», в которой вес более раннего учётного периода равен половине веса последующего за ним периода, т.е. вес учётных периодов уменьшается в геометрической прогрессии по мере их отодвигания в прошлое. Период «полураспада» в SLURM задаётся администратором СУЗ.

Альтернативой иерархической схеме директивного распределения приоритетов являются экономические методы планирования, которые позволяют [153] добиться справедливого распределения вычислительных ресурсов на конкурентной основе. Противоречие разрешается через систему бюджетов, которые пользователи могут тратить на оплату ресурсов. Бюджет может быть основан как на виртуальных, так и реальных финансовых средствах. Приоритет пользователя при применении экономических методов планирования определяется той ценой, которую пользователь готов заплатить за место в очереди для своего задания. Для определения этой цены могут быть применены различные схемы аукционов [136].

3. В рамках одного приоритета задания обрабатываются строго в порядке поступления в очередь (принцип FCFS). Однако, в большинстве СУЗ разрешен запуск вне очереди некоторых заданий, которые не только поступили позже других, но могут иметь более низкий приоритет. Необходимым условием для внеочередного запуска является то, что он не повлияет на время старта заданий, стоящих в очереди выше (по причине высокого приоритета или более раннего поступления в очередь). Такое возможно, например, в случае, если для задания А, стоящего в очереди выше, недостаточно ресурсов, и при этом задание Б, стоящее в очереди ниже, успеет завершиться до момента, когда освободится достаточное количество ресурсов для запуска задания А. Принцип такого перераспределения заданий впервые был представлен в [154] и получил название «**обратного заполнения**» (англ. – backfilling). В

настоящее время стратегия обратного заполнения является основной для всех современных СУЗ, поскольку она позволяет радикально повысить загрузку ресурсов.

Исследования в области развития методов и средств планирования суперкомпьютерных заданий можно разделить на следующие направления.

В таких исследованиях, как [27], принцип планирования заданий с фиксированными параметрами справедливо критикуется, поскольку качество планирования в этом случае существенно зависит от точности оценки пользователями времени выполнения их заданий. Как показано в [155], из-за боязни принудительного снятия задания пользователи многократно завышают оценку времени выполнения своих заданий. Задания с завышенными оценками времени выполнения с точки зрения планировщика завершаются преждевременно, что приводит к внеочередному перестроению расписания запусков заданий и негативно влияет на качество их планирования.

В [27] для преодоления этого влияния предлагается введение т.н. «мягких» ограничений, основанных на предсказании времени выполнения каждого задания. Авторы разработали модуль прогноза, встроенный в планировщик СУЗ PBS Pro. Прогноз времени выполнения задания осуществляется на основе статистики работы суперкомпьютера за некоторый исторический период. Предсказанное время используется в PBS Pro в качестве т.н. «мягкого ограничения», на которое ориентируется планировщик при составлении расписания запусков заданий. При превышении мягкого ограничения задание не снимается с выполнения и может продолжать расчеты до истечения заказанного пользователем времени. За счет применения при планировании предсказанных времен авторам удалось улучшить такие характеристики, как время ожидания задания в очереди и коэффициент замедления заданий.

Большое число работ, например [21-24], посвящено развитию и оптимизации метода обратного заполнения.

В [23] классифицированы основные проблемы, возникающие при исследовании методов и алгоритмов планирования заданий методом имитационного моделирования (симуляции). Авторы отметили невозможность точной симуляции работы планировщика, а также отсутствие метрик, позволяющих оценить точность симуля-

ции, обозначили задачу формирования статических и динамических входных потоков заданий (рабочей нагрузки). Особо выделены проблема неточных пользовательских оценок времени выполнения заданий и ее влияние на качество планирования. В статье рассмотрены два основных вида алгоритма обратного заполнения: **консервативный** и **агрессивный**. Консервативный алгоритм при размещении заданий в обход основной очереди не допускает сдвига запланированного времени старта ни одного из заданий основной очереди. Агрессивный (EASY – Earliest Available Start-time Yielding) алгоритм не разрешает сдвиг времени старта только первого стоящего в очереди задания.

В статье [21] рассмотрены различные стратегии выбора задания для запуска в обход очереди при агрессивном алгоритме обратного заполнения. В работе [22] авторы использовали статистику пяти суперкомпьютерных систем при симуляции процесса планирования. В работе произведено сравнение четырех политик планирования в сочетании с агрессивным алгоритмом обратного заполнения.

Одна из последних работ [24] в этой области посвящена направлению исследований, связанному с предсказанием времени выполнения заданий. Как мы уже отмечали, одной из основных проблем планирования является неточная оценка пользователями времени выполнения своих заданий. Основываясь на пользовательских оценках, авторы при помощи метода k средних дают прогнозную оценку реального времени выполнения каждого задания. Это предсказанное время используется при выборе задания для обратного заполнения очереди. В статье показано преимущество предлагаемого подхода перед стандартными стратегиями обратного заполнения. По мнению авторов, полученные результаты не только продемонстрировали потенциал по повышению эффективности планирования, но и способствуют повышению мотивации пользователей для более точной оценки времени выполнения своих заданий.

Проблеме предсказания времени выполнения заданий посвящены работы [155-157]. В [156] определены пределы повышения качества планирования при 100% точности прогноза времени выполнения заданий. В статье [155] на базе стати-

стики суперкомпьютеров МСЦ РАН показано, что для улучшения показателей качества планирования точность предсказания времени выполнения заданий должна быть не ниже 80%. В [157] авторы исследовали различные алгоритмы машинного обучения для предсказания времени выполнения заданий. На основе статистики работы суперкомпьютеров Северо-западного политехнического университета (КНР) авторы продемонстрировали улучшение таких показателей качества планирования, как среднее время ожидания, среднее время ответа и средний коэффициент замедления заданий.

В работе [158] исследуются подходы к совместному распределению ресурсов для гарантированного выполнения заданий в системах с гетерогенными узлами. Отмечается, что сложные вычислительные системы часто работают в условиях неопределенности доступности ресурсов, вызванной непредсказуемостью текущей мультипрограммной ситуации. В то же время у пользователей существует высокая потребность в гарантированных сроках выполнения их заданий. Однако, гарантированное резервирование определенного объема вычислительных ресурсов неизбежно приводит к их простою в режиме ожидания и снижению загрузки вычислителя. Предлагаемое авторами [158] решение позволяет оптимизировать процедуру распределения и резервирования ресурсов для нужд заданий с учетом статических и динамических особенностей использования ресурсов, при этом доступность ресурсов выдвигается в качестве целевого критерия.

Ряд исследований посвящен оптимизации планирования в условиях специализации суперкомпьютерной системы. В [159] справедливо отмечается, что целевые показатели качества планирования заданий могут существенно отличаться в зависимости от специализации разных суперкомпьютерных центров, определяемой автором как цель использования вычислительных ресурсов. В работе предложен новый подход к оценке эффективности функционирования суперкомпьютерной системы, позволяющий сравнивать разные суперкомпьютеры исходя из их специализации (цели использования).

В работе [25] рассмотрена система планирования заданий, функционирующая в ФГУП РФЯЦ-ВНИИТФ им. акад. Е.И. Забабахина. В основе системы лежит менеджер ресурсов SLURM. Система, названная Slurm-ВНИИТФ, расширяет функции SLURM и предоставляет дополнительные возможности планирования. Главной из этих возможностей является разделение заданий на классы в зависимости от срочности их выполнения. Авторы отмечают, что одной из проблем планирования является наличие в единой очереди множества разнородных (относящихся к разным классам) заданий. Выделяются класс интерактивных заданий, необходимых для отладки и визуализации расчетов, и класс фоновых заданий, которые при поступлении более приоритетных заданий могут вытесняться с выполнения с сохранением промежуточных результатов расчетов. За счет возможности перезапуска фоновых заданий, система Slurm-ВНИИТФ позволила увеличить ежегодное количество выполненных заданий примерно на 20%. В то же время авторы отмечают в высокой степени специализированный характер предлагаемых ими подходов.

3.1.2 Методы и средства планирования адаптивных заданий

СУЗ, в которых осуществляется планирование исключительно адаптивных заданий, имеют достаточно узкую специализацию и слабо представлены в научных публикациях. В большинстве случаев адаптивные задания планируются наряду с заданиями с фиксированными параметрами, образуя входной поток разнородных заданий. Сложность планирования в одной очереди разнородных заданий отмечается в исследовании [26], в котором так же, как и в [25], выделяется класс заданий с высоким уровнем отклика (High-Responsiveness-Requesting jobs – HRR). Такие задания применяются пользователями для отладки своих программ, во время которой важно быстрое прохождение очереди. Авторы отмечают, что простое приоритетное обслуживание HRR-заданий ухудшает положение в очереди остальных заданий и предлагают для потока HRR использовать переподписку ресурсов – выделение для разных HRR-заданий одних и тех же узлов суперкомпьютера.

Рассмотрим приведенные в научной литературе методы и способы обработки адаптивных заданий.

1. Обработка адаптивных заданий в общей очереди. Пользователи самостоятельно занимаются поиском окон в расписании запусков и добавляют адаптивные задания в очередь в соответствии с размерами найденных окон. Этот процесс может быть автоматизирован как самими пользователями, так и разработчиками СУЗ.

В статье [160] исследуется вопрос динамического выделения ресурсов для MPI-приложения. Работа [161] посвящена интеллектуальному совместному планированию, когда несколько заданий могут совместно использовать ресурсы на уровне узлов для повышения эффективности использования узлов и снижения времени обработки заданий.

2. Создание отдельного сегмента («песочницы») для обработки адаптивных заданий небольшого размера.

В работе [28] осуществлена попытка моделирования ситуации обработки дополнительных небольших по вычислительным ресурсам, но длительных по времени заданий на суперкомпьютере IBM Summit. Отмечено негативное влияние на измеряемые показатели эффективности и в качестве рабочего решения предложено создание отдельного вычислительного контура для обработки небольших заданий.

3. Применение грид-технологий или облачных технологий.

В работах [162, 163] рассматривается система Production and Distributed Analysis (PanDA), которая успешно использовалась в эксперименте ATLAS в качестве СУЗ. Авторами предложен новый компонент под названием Harvester для посредничества в управлении и потоке информации между PanDA и вычислительными ресурсами, что позволяет интеллектуально управлять рабочей нагрузкой и динамически предоставлять ресурсы на основе детального знания возможностей ресурсов и их состояния в реальном времени.

В работе [164] Google Cloud использовался для оценки эластичных пакетных вычислений до 100 тыс. ядер для обработки заданий, требующих быстрого выполнения. Успех этапов проверки концепции привел к расширению проекта Google Cloud,

в котором ATLAS будет изучать вопросы организации облачного сайта, полностью интегрированного с распределенными вычислительными ресурсами грид.

4. Автоматическая адаптация параметров заданий планировщиком.

В работе [165] предлагается эластичный планировщик, который на основе времени выполнения и потребляемой электроэнергии способен подбирать наилучшую конфигурацию в целях оптимизации времени выполнения или потреблённой энергии.

В работе [166] рассматривается инструмент для умного планировщика, который сможет подбирать компромисс между требуемыми вычислительными ресурсами и временем выполнения. Отмечается, что такой инструмент неполон без учёта времени ожидания задания в очереди.

В статье [167] представлен метод Artemis подбора параметров задания с помощью машинного обучения для оптимизации времени выполнения задания. Artemis отслеживает задания во время выполнения и создает адаптивные модели для настройки параметров выполнения, минимально вмешиваясь в разработку приложений и обеспечивая невысокие накладные расходы во время выполнения заданий.

В статье [168] предлагается целостная динамическая политика планирования заданий Slowdown Driven (SD-Policy), которая использует адаптивность заданий для повышения темпов обработки и снижения времени отклика заданий. SD-Policy основана на стратегии обратного заполнения и совместном использовании узлов несколькими заданиями.

В нескольких работах предлагается использовать нейронные сети для назначения заданий в окна простоя. В проекте FreeTrain [29] задача определения соответствия заданий динамически изменяющимся окнам решается при помощи детерминированного алгоритма распределения ресурсов с применением смешанного целочисленного линейного программирования (MILP). По утверждению авторов, задача MILP может быть эффективно решена во время выполнения заданий. Однако Free Train полагается на то, что пользователи предоставят точную информацию о време-

ни выполнения заданий, что увеличивает нагрузку на пользователей и делает предлагаемый подход труднопримениым на практике.

5. Применение постпланирования.

В [30] авторы предложили использовать свободные вычислительные ресурсы суперкомпьютеров путем организации дополнительной очереди низкоприоритетных непараллельных заданий. Такие задания предлагается выполнять в контейнерах, разбивая выполнение на отдельные интервалы с помощью инструментов миграции контейнеров. Авторы разработали систему управления контейнерами, которая поддерживает дополнительную очередь во взаимодействии с планировщиком СУЗ.

В диссертации [31] предложен и реализован алгоритм, позволяющий увеличить эффективность использования высокопроизводительных вычислительных систем, ориентированных на выполнение больших параллельных задач (использующих до нескольких тысяч вычислительных узлов), посредством динамической дозагрузки простаивающих ресурсов суперкомпьютера малыми задачами под контролем внешней системы управления нагрузки. Автору удалось повысить загрузку суперкомпьютера Titan на 2% при практическом отсутствии какого-либо воздействия на время ожидания для крупных и средних заданий в очереди.

В исследовании [169] была предпринята попытка создания постпланировщика в виде программного средства под названием «Квазипланировщик», способного заполнять простаивающие ресурсы небольшими заданиями. Эффект от применения «Квазипланировщика» был показан на реальном входном потоке длительностью 7 дней из более чем 4000 заданий, взятых из статистики работы суперкомпьютера МВС-100К. За счёт заполнения свободных окон в расписании была незначительно повышена загрузка вычислительных ресурсов с 93,3% до 94,2% при отсутствии статистически значимого влияния на время нахождения заданий в очереди.

3.1.3 Методы и средства планирования нестандартных заданий

В большинстве суперкомпьютерных центров принят примерно одинаковый порядок работы. Зарегистрированный пользователь получает учётную запись и

пароль, которые позволяют осуществлять удалённый доступ к суперкомпьютерным ресурсам и сервисам. После подготовки задания и постановки в очередь СУЗ пользователю необходимо дождаться его старта и завершения, после чего забрать результаты. При работе на нескольких суперкомпьютерных установках пользователь вынужден повторять однотипные действия для каждой из них, что далеко не всегда является удобным и эффективным. При этом очень часто пользователи сами не разрабатывают новые программы, а используют для расчётов готовые прикладные пакеты. Установка и настройка пакета для нескольких суперкомпьютерных установок – трудоёмкий процесс. Вместе с тем необходимый для работы пользователя стек программного обеспечения может быть оформлен в виде виртуальной машины (ВМ) или контейнера, в том числе с использованием рассмотренных в п. 2.1.2 методов и средств. Задание, оформленное в виде ВМ или контейнера, в соответствии с введенной нами в п. 1.2.2 классификацией следует отнести к нестандартным заданиям.

Для СУЗ возникает новая задача – обеспечение возможности автоматического обслуживания потока нестандартных заданий, представленных набором виртуальных машин (контейнеров), совместно с потоком обычных стандартных заданий. Решить эту задачу возможно через введение дополнительного уровня абстракции – облачного сервиса, который обеспечит единый интерфейс управления. При этом необходимо учитывать и традиционный порядок работы пользователей через СУЗ.

В существующих решениях в области организации облачных вычислений, как и в суперкомпьютерных расчётах, подразумевается определённый типовой порядок действий пользователя. Пользователь при помощи клиентских инструментов инициирует запрос на создание/использование необходимых ему виртуальных машин. С помощью специального компонента – контроллера облака – облачная платформа подбирает наиболее подходящие вычислительные ресурсы, на которых начинается выполнение виртуальных машин. При подборе вычислительных ресурсов и запуске на них виртуальных машин контроллер облака непосред-

ственно взаимодействует с гипервизорами физических вычислительных узлов (серверов), анализируя текущую загруженность решающего поля и балансируя вычислительную нагрузку. Другими словами, решающее поле, которое в суперкомпьютерах управляется СУЗ, должно быть передано под полное управление облачной платформе. Это противоречие обуславливает актуальность разработок методов планирования, совмещающих потоки стандартных и нестандартных заданий.

Концепция облачных вычислений подразумевает, что конечному пользователю по требованию и на определённое время будет предоставлен удобный сетевой доступ к некоторому набору настраиваемых вычислительных ресурсов: подмножеству узлов решающего поля суперкомпьютера, сетям передачи данных, устройствам хранения данных, программным приложениям и пакетам. В терминологии облачных вычислений предоставляемый пользователю доступ к ресурсам называется услугой или сервисом, а сам процесс предоставления сервиса – обслуживанием. Среди основных моделей обслуживания выделяют программное обеспечение как услугу (SaaS), платформу как услугу (PaaS) и инфраструктуру как услугу (IaaS).

К обязательным характеристикам облачных вычислений обычно относят следующие:

− самообслуживание по требованию – услуга предоставляется по запросу пользователя, который самостоятельно определяет её объём и продолжительность;

− универсальный доступ по сети – услуга доступна пользователям по сети вне зависимости от используемого оконечного устройства (терминала, клиента);

− динамическое перераспределение вычислительных мощностей в условиях постоянного изменения нагрузок за счёт объединения ресурсов в единый пул;

− эластичность – объём предоставляемых услуг по требованию пользователя может быть в любой момент времени изменён как в большую, так и в меньшую сторону;

− оплата пользователем только фактически потреблённых ресурсов.

С точки зрения пользователя, облака упрощают доступ к необходимым сервисам, делая прозрачным путь от запроса к вычислительным ресурсам. При этом пользователь может существенно сэкономить как на организации вычислений, так и на том, что оплатит лишь фактически потреблённые ресурсы. Выгода владельца ресурсов состоит в кратном увеличении прибыли за счёт более эффективной загрузки ресурсов и консолидации затрат на управление ими. Для эффективной организации облачных вычислений на базе имеющихся ресурсов создаётся специальная инфраструктура с помощью облачных платформ и систем виртуализации.

Нетрудно заметить, что любая СУЗ в рамках контролируемой вычислительной системы в некоторой степени абстракции обладает всеми перечисленными характеристиками облачных вычислений:

− самообслуживание по требованию и эластичность – зарегистрированный пользователь для непосредственного производства расчётов формирует и направляет в СУЗ вычислительное задание, самостоятельно задавая необходимые число узлов решающего поля и время для выполнения задания;

− универсальный доступ по сети – зарегистрированный пользователь имеет круглосуточный удалённый доступ к суперкомпьютеру и может использовать для доступа произвольное оконечное устройство (терминал);

− система управления заданиями динамически выделяет вычислительные узлы для каждого задания, обеспечивая полноту и равномерную загрузку решающего поля;

− биллинговая подсистема СУЗ ведёт точный учёт фактического потребления пользователями суперкомпьютерных ресурсов.

Отметим также принципиальные отличия высокопроизводительных вычислений от облачных, не позволяющие осуществить их простую интеграцию:

− уникальность вычислительных ресурсов суперкомпьютеров не всегда поддерживается универсальными гипервизорами;

− обеспечивая минимальное время выполнения своего задания и максимальную эффективность использования вычислительных устройств, пользователь суперкомпьютера предпочитает иметь непосредственный, а не опосредованный системой виртуализации, доступ к суперкомпьютерным ресурсам;

− для широкого класса суперкомпьютерных задач использование виртуализации не оправдано, т.к. последняя привносит существенные накладные расходы на вычисления [73, 32].

Тем не менее, попытки интеграции облачных и высокопроизводительных вычислений осуществляются постоянно на протяжении многих лет как за рубежом, так и в России. Среди исследований и разработок в этой области можно выделить три направления.

Первое направление связано с применением в суперкомпьютерах технологий виртуализации как основы организации облачных вычислений. Успехи по этому направлению связаны с технологиями контейнерной виртуализации, привносящей наименьшие накладные расходы в вычислительный процесс [72, 172]. В работе [111] содержится не только достаточно полный обзор исследований в области контейнерной виртуализации, но и предлагаются решения по представлению вычислительных заданий в виде контейнеров в системе управления заданиями SLURM.

По второму направлению ведутся разработки облачных сервисов для высокопроизводительных вычислений на основе построения собственных облачных платформ [33, 34], в т.ч. и с использованием программных комплексов организации добровольных вычислений [35].

Третье направление связано с исследованиями в области построения и применения облачных платформ для высокопроизводительных вычислений. Например, в работе [36] описывается концепция динамического разделения решающего поля на стандартно управляемую часть и виртуализированный на базе OpenStack раздел и интеграцию данного механизма с СУЗ LSF. В работе [37]

предлагается в облаке, управляемом платформой OpenStack, выделять по запросу пользователя виртуальный кластер не из виртуальных машин, а из части физических узлов, т.е. некий гибридный подход к построению облака для высокопроизводительных вычислений. В работах [32, 38] рассмотрены основные проблемы, возникающие при переносе высокопроизводительных вычислений в облако, и представлен подход к организации высокопроизводительного облачного сервиса с использованием виртуализации. Отмечается [32] основное препятствие – высокие накладные расходы, которые даже при достаточно глубоких исследованиях авторов [38] составили не менее 10%. Схожий опыт демонстрирует работа [39], где попытка внедрения облачной платформы OpenStack на вычислительных узлах суперкомпьютера Cray привела к накладным расходам в 20%. Кроме этого, для эффективного управления высокопроизводительными ресурсами авторам работы [38] потребовалось создание собственного планировщика [173], учитывающего топологию коммуникационной среды. Как показывает опыт автора настоящей диссертации, собственные разработки [174] при всей их эффективности на узкоспециализированном участке исследований часто оказываются несовместимыми с общепринятыми стандартными подходами.

3.2 Планировщик заданий как система массового обслуживания

Отдельное научное направление представляют собой исследования планировщиков заданий как систем массового обслуживания, преследующие цель построения и анализа математических моделей суперкомпьютерных систем коллективного пользования. Рассматривая это направление, следует отметить успехи отечественной школы теории массового обслуживания, представители которой в своих работах постоянно совершенствуют математические модели, методы и средства исследования усложняющихся вычислительных машин и систем, в том числе суперкомпьютеров.

В монографии [175] впервые на русском языке были представлены результаты аналитического исследования приоритетных систем обслуживания, в том числе рассмотрены многоэтапные приоритетные системы как наиболее общие модели высокопроизводительных расчетов на ЭВМ. В работе [176] рассмотрены проблемы эффективного использования вычислительных ресурсов управляющих ЭВМ, работающих в режиме реального времени в составе автоматизированных систем управления. В частности, приводятся основные характеристики методов организации вычислительного процесса: приоритетных и бесприоритетных методов диспетчеризации с ограниченной и неограниченной буферной памятью, в том числе методов диспетчеризации с квантованием времени обслуживания. Методы анализа мультипрограммных систем представлены в [177], а в [178] аналитические методы были применены для исследования сетей ЭВМ. Дальнейшее развитие методов теории массового обслуживания в целях исследования приоритетных вычислительных систем представлено в [179, 180], где рассматриваются вопросы пакетной обработки заявок и комбинации различных дисциплин приоритетного обслуживания заявок в вычислительных системах.

Новый импульс аналитическое моделирование вычислительных систем получило при появлении многоядерных микропроцессоров, которые превратили практически любое компьютерное устройство в достаточно сложную параллельную вычислительную систему. В теории массового обслуживания параллельные системы получили название многосерверных систем [181]. Обзор моделей многопроцессорных систем приведен в работе [182], в ней же предпринята попытка моделирования работы суперкомпьютерной системы коллективного пользования на примере вычислительного кластера Карельского научного центра РАН (КарНЦ РАН). Суперкомпьютер представлен как m-процессорная система массового обслуживания $GI/G/m$ с независимыми одинаково распределенными интервалами между заявками $\{T_n\}$ и независимыми одинаково распределенными временами обслуживания $\{S_n\}$, в которой i-й приходящей заявке требуется одновременно случайное число процессоров $N_i \in [1, m]$. Если число свободных процессоров

меньше N_i, то заявка (вычислительное задание) ожидает в буфере освобождения недостающего числа процессоров. Модель показала хорошее согласие со статистическими данными кластера КарНЦ РАН, что позволило авторам сделать вывод об определенном потенциале ее практического применения для анализа существующих и проектирования новых многопроцессорных систем.

В работе [183] предложена модель суперкомпьютерного кластера, основанная на типовой системе массового обслуживания $M/M/\infty$. Как и в [182], каждая заявка представляет собой задание, требующее для своего выполнения выделения случайного числа каналов обслуживания (процессоров). Авторы ввели в модель важное дополнение, отражающее особенность функционирования суперкомпьютера, – ограничение времени обработки каждого задания некоторой случайной величиной, распределенной по показательному закону. Это позволило получить в явном виде закон распределения числа занятых процессоров, а также вероятности успешного завершения заданий в виде некоторой функции от среднего ограничения по времени.

В то же время в работах [182, 184, 185] отмечаются особенности, затрудняющие анализ суперкомпьютерных систем коллективного пользования при помощи методов теории массового обслуживания. К основным трудностям относят т.н. «тяжелые хвосты распределений», ограничивающие использование экспоненциальных распределений для моделирования процессов в современных компьютерных системах [182] и неконсервативность процесса нагрузки (узлы или процессоры суперкомпьютера могут простаивать при непустой очереди) [184]. Отмечается [185], что нахождение явных решений для характеристик производительности таких систем затруднено даже для систем малого размера. Значительное число задач, связанных с анализом суперкомпьютерных систем, являются открытыми проблемами [186], и часто исследователи сосредотачивают свои усилия на небольших системах, например, двухпроцессорных [185].

Отдельно отметим, что заявки в виде заданий в суперкомпьютерную систему, как правило, направляют люди – пользователи системы. Пользователи отсле-

живают текущие параметры планирования заданий и адаптируют требования своих заданий к изменениям этих параметров. Таким образом, мы получаем систему массового обслуживания с обратной связью, обусловленной в том числе человеческим фактором. Это обстоятельство еще сильнее усложняет анализ суперкомпьютерных систем коллективного пользования. В монографии [186] отмечается, что аналитическое или численное моделирование столь сложных систем либо затруднено, либо невозможно традиционными методами теории массового обслуживания, что заставляет исследователей прибегать к методам имитационного моделирования и машинного обучения.

Имитационное моделирование является одним из часто используемых методов исследования СУЗ как систем массового обслуживания. В работе [187] на базе анализа статистики десятков тысяч заданий нескольких суперкомпьютеров была предложена модель, способная формировать входной поток заданий, статистически неотличимый от входного потока реальной вычислительной системы. Для этого каждая из характеристик задания моделируется случайной величиной с определённым распределением и заданным параметрами. Предложенный в работе [187] подход был применен при имитационном моделировании СУЗ для разработки метода совмещения разнородных потоков заданий.

3.3 Планирование заданий с фиксированными параметрами

3.3.1 Метод планирования заданий с фиксированными параметрами

Рассматриваемый метод планирования заданий применяется в СУППЗ, начиная с 1999 года. Программная реализация метода в виде сервера очереди СУППЗ осуществлена сотрудником Института математики и механик им. Н.Н. Красовского УрО РАН С.В. Шарфом.

Метод предполагает, что пользователи для каждого задания указывают число необходимых процессорных ядер и время выполнения задания. В зависимости от значения этих параметров задания разбиваются на три категории – отладочные, ординарные (пакетные) и фоновые. Размер отладочных заданий ограничивается

сравнительно небольшим числом ядер $P_{отл}$ и коротким временем выполнения $T_{отл}$. Параметры $P_{отл}$ и $T_{отл}$ определяются администратором. Задания, чьи требования не превышают этих значений, категорируются как отладочные. Ординарными считаются однократно запускаемые задания, время исполнения которых сравнительно велико и ограничивается сверху параметром $T_{орд}$. Число ядер для ординарного задания ограничивается только общим числом ядер в суперкомпьютерной системе.

Фоновые задания могут выполняться произвольное время, но при этом периодически прерываться системой управления. Для фонового задания пользователь явно указывает квант – минимальное время выполнения фонового задания. СУППЗ гарантирует, что если фоновое задание было запущено, то ему для выполнения будет предоставлено время, не меньшее указанного кванта. Если по истечении кванта будут обнаружены задания, претендующие на занятые фоновым заданием ресурсы, фоновое задание будет снято с выполнения и заново поставлено в очередь. При этом общее время счета фонового задания будет уменьшено на число минут, прошедших с его последнего запуска.

Расписание планировщика представляет собой последовательность сменяющих друг друга режимов планирования. Планирование очереди в каждый момент времени производится в соответствии с параметрами текущего режима. В системе может быть несколько расписаний, переключение между которыми осуществляется по директиве администратора, а переключение режимов внутри каждого расписания – автоматически.

Режим планирования определяется следующими параметрами.

1. Дата и время включения режима, определяющие время, начиная с которого параметры режима вступают в силу. Параметры режима действуют вплоть до включения следующего режима.

2. Шкала доступа к режиму определяет группы пользователей, которым разрешено выполнение заданий в этом режиме. С помощью шкалы доступа организуются режимы профилактики оборудования, во время которой запуск заданий

разрешается только администраторам системы. Планировщик гарантирует, к началу режима профилактики узлы суперкомпьютера будут свободны от заданий пользователей.

3. Общее число планируемых процессорных ядер.

4. Максимальное время $T_{отл}$ и число ядер $P_{отл}$, отведенные для отладочных заданий. Задания, требующие для исполнения времени, не превышающего $T_{отл}$, и числа ядер, не превышающего $P_{отл}$, автоматически будут отнесены к категории отладочных. Все ординарные задания в сумме не могут на время, большее $T_{отл}$, занимать ядер больше, чем разность между общим числом ядер и значением $P_{отл}$. Фактически ядра числом $P_{отл}$ будут использоваться только для выполнения отладочных заданий, другими словами, будут «зарезервированы» для отладочных заданий. Следует пояснить, что «резервирование» не означает выделение конкретных ядер или узлов под отладочные задания. Планировщик СУППЗ гарантирует, что $P_{отл}$ процессоров не будет использовано для ординарных заданий на время, не превышающее $T_{отл}$, а какие конкретно узлы будут «зарезервированы» для этой цели, зависит от текущей ситуации. Подход с «плавающим» резервом отладочных ядер позволяет сократить простои ресурсов при отсутствии в системе отладочных заданий.

5. Максимальное время, отведенное для ординарных заданий $T_{орд}$. Задания, чьи требования превышают время $T_{орд}$, будут ожидать в очереди смены режима или расписания без включения в расписание запусков.

6. Шкала приоритетов пользователей. Приоритеты напрямую зависят от суммарного времени выполнения всех заданий пользователя за некоторый учетный период. Размер учетного периода задается администратором при составлении расписания очереди. При планировании определяются k приоритетов, $1 \le k \le 6$, для заданий путем задания шкалы вида (t_1, t_2, t_3, ... t_k, 0). Обозначим как T_{user} суммарное время выполнения всех заданий пользователя за учетный период. Отметим, что в сумму T_{user} включается также сумма заказанных времен выполнения заданий пользователя, находящихся в очереди.

Наивысшим приоритетом (очередь 0) будут обладать задания пользователей с суммарным временем $T_{user} < t_1$, чуть меньшим (очередь 1) приоритетом – тех пользователей, у которых $T_{user} < t_2$, еще меньшим (очередь 2) – тех пользователей, у которых $T_{user} < t_3$ и т.д. Низшим для текущего режима (очередь k) приоритетом будут обладать задания пользователей, у которых $t_k \leq T_{user}$.

При суммировании времени заданий пользователя за учетный период, время каждого задания умножается на специальный коэффициент – цену задания. Цена задания определяется администратором для каждого пользователя при составлении расписания. При вычислении приоритета задания учитывается время выполнения, указанное пользователем при постановке задания в очередь. Это время прибавляется к суммарному времени заданий пользователя за учетный период.

Планировщик пытается выделить ресурсы из числа свободных сначала для заданий из очереди 0, потом – из очереди 1 и т.д. Внутри одной очереди ресурсы выделяются в порядке поступления заданий (принцип FCFS). Если свободных ресурсов для задания нет, определяется момент времени, когда нужные ресурсы освободятся, и устанавливается время запуска задания. Никакое менее приоритетное задание не может занять ресурсы так, чтобы это отодвинуло запуск более приоритетного задания. При отсутствии конфликта по ресурсам менее приоритетное задание может стартовать раньше более приоритетного, реализуя консервативный алгоритм обратного заполнения.

Отметим основные характеристики метода планирования СУППЗ:

– метод реализует принцип планирования с фиксированными параметрами и невытесняющей приоритеной дисциплиной обслуживания;

– за счет шкалы приоритетов реализуется «справедливый» принцип планирования faire share – чем больше времени выполнялись задания пользователя за учетный период, тем ниже его приоритет, и наоборот;

– метод применяет консервативный алгоритм обратного заполнения.

Перечисленные характеристики метода присущи большинству используемых в СУЗ стратегий планирования заданий. Во многом по этой причине иссле-

дование [140] показало паритет между планировщиками СУППЗ и SLURM по основным показателям качества планирования для одних и тех же входных потоков заданий.

Отличительными особенностями метода планирования в СУППЗ является выделение в отдельные классы отладочных и фоновых заданий. Аналогичная классификация применяется в [25], однако в СУППЗ подобный подход был применен на два десятилетия раньше. Рассмотрим статистику работы суперкомпьютеров под управлением СУППЗ и исследуем, как выделение отладочных и ординарных заданий влияет на показатели качества планирования.

3.3.2 Методика обработки данных статистики для определения коэффициента замедления

В расчетах использовались данные статистики работы рассмотренных в п. 2.5 вычислительных систем МСЦ РАН за период с 2001 по 2024 годы. Информация о выполненных на суперкомпьютерных системах МСЦ РАН и ИПМ им. М.В. Келдыша РАН заданиях приведена в таблице 9.

Каждая суперкомпьютерная система управлялась через отдельную очередь заданий, при этом за указанные в таблице 9 годы эксплуатации в очереди планировалось различное число процессорных ядер. Максимальное число ядер указано в столбце 3 таблицы 9. Таблица содержит также общее число выполненных заданий, число выполненных отладочных и фоновых заданий. Процент использования ресурсов отладочными и фоновыми заданиями рассчитывался от общего объема ресурсов, потребленного всеми заданиями. Для большинства суперкомпьютерных систем максимальное время выполнения ординарного задания равно 1 суткам (1440 минут). Исключение составляет раздел Optane суперкомпьютера МВС-10П ОП, на котором возможен запуск ординарных заданий длительностью до 1 недели (10080 минут).

Из таблицы 9 видно, что при сравнительно небольшом числе фоновых заданий они потребляют существенную долю вычислительных ресурсов. С отладоч-

ными заданиями мы имеем обратную ситуацию – при достаточно большом их числе доля потребленных ресурсов мала. Для ряда суперкомпьютеров режим отладочных заданий не вводился, и соответствующие ячейки таблицы 1 пусты.

Таблица 9. Суперкомпьютерные системы МСЦ РАН в 2001-2024 гг.

Система	Годы	Число ядер, макс.	Выполнено заданий			Использовано ресурсов, %	
			Всего	Отл.	Фон.	Отл.	Фон.
1	2	3	4	5	6	7	8
МВС-1000М	2001-2006	768	339 984	78 031	9 233	1,2%	28,5%
МВС-15000	2004-2008	1146	221 903	60 672	6 073	1,0%	18,1%
МВС-6000	2006-2011	254	78 089	15 970	928	3,2%	17,9%
МВС-100К	2007-2021	9728	1 825 797	231 067	40 190	0,3%	8,4%
МВС-10П	2013-2023	2800	389 293	21 140	7 603	0,1%	8,7%
МВС-10П ОП Broadwell	2017-2024	4352	150 843	20 427	1 933	0,1%	3,1%
МВС-10П ОП KNL	2018-2024	2160	41 900	31	659	0,01%	9,3%
МВС-10П ОП Skylake	2019-2024	2088	42 356		831		7,0%
МВС-10П ОП Cascade Lake	2020-2024	8880	99 238		1381		2,2%
МВС-10П ОП Optane	2021-2024	288	5 938	8	110	0,01%	13,9%
К100	2010-2023	768	467 001	150 611	19 829	0,01%	22,6%
К60	2022-2024	2408	63 348	19 284	3584	0,01%	28,9%

Всего на функционировавших под управлением СУППЗ суперкомпьютерах механизмом фоновых заданий воспользовались более 260 пользователей при реализации свыше 110 научных проектов.

Традиционно коэффициент замедления рассчитывается в соответствии с формулой (5). В СУППЗ минимально возможное время выполнения задания равно 1 минуте, и именно это значение использовано в качестве параметра τ для отладочных заданий. Для длительных по времени ординарных и фоновых заданий время выполнения в 1 минуту означает фактически аварийный запуск, поэтому значение параметра τ для этих классов задавалось равным 2 минутам.

Кроме этого, для фоновых заданий, которые могут многократно запускаться и возвращаться в очередь, коэффициент замедления следует представить как

$$S_i = \frac{\sum_{j=1}^{k}(R_j + Q_j)}{\sum_{j=1}^{k}\max(R_j, \tau)} \quad (9)$$

где k – число запусков i-го фонового задания, Q_j и R_j – соответственно времена ожидания в очереди и выполнения для j-го запуска задания.

Коэффициент замедления является интегральным показателем качества и должен определяться на сравнительно больших временных промежутках. Учитывая, что фоновые задания могут находиться в системе несколько месяцев, целесообразно рассчитывать коэффициент замедления по годам. При выполнении статистических расчетов из базы данных статистики МСЦ РАН [114] извлекалась следующая информация:

– время поступления задания в очередь;

– число ядер и требуемое время выполнения для задания, квант для фонового задания;

– параметры режима планирования на момент поступления задания в очередь, в соответствии с которыми определялся класс задания;

– число доступных процессорных ядер на все время пребывания задания в системе;

– времена старта и завершения задания.

Сумма времен Q_j и R_j в формулах (5) и (9) есть время пребывания задания в системе. Из этого времени вычитались периоды профилактики оборудования, в течение которых задания ожидали в очереди без возможности выполнения. Кроме этого, из рассмотрения исключались периоды, во время которых были зафиксированы аварийные ситуации, связанные с массированным отказом оборудования.

3.3.3 Средние коэффициенты замедления для различных классов заданий суперкомпьютеров МСЦ РАН разных поколений

Средние коэффициенты замедления заданий разных заданий для суперкомпьютеров из таблицы 9 приведены на диаграммах рисунков 14-25. На каждой диаграмме по годам представлены четыре столбца. Первый соответствует среднему коэффициенту замедления для всех заданий за год. Остальные три соответствуют

коэффициентам замедления отладочных, ординарных и фоновых заданий. На некоторых системах в определенные годы режим отладочных заданий не включался, и соответствующие столбцы на диаграммах отсутствуют.

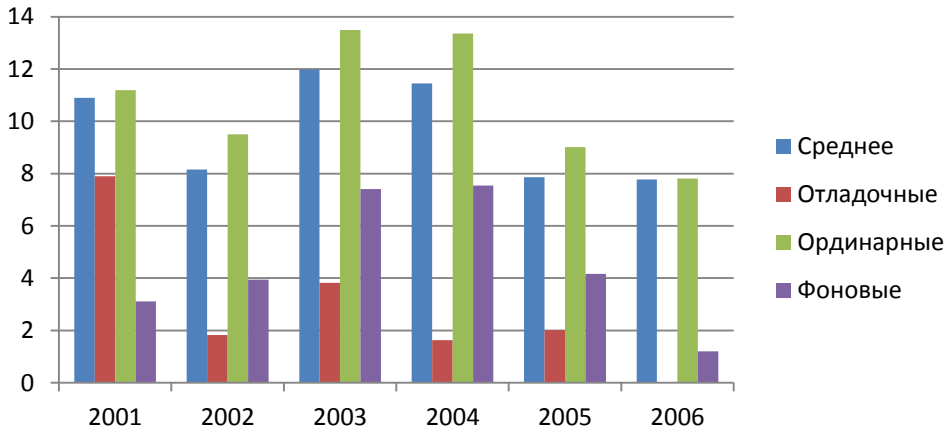

Рисунок 14. Коэффициент замедления разных классов заданий суперкомпьютера МВС-1000М

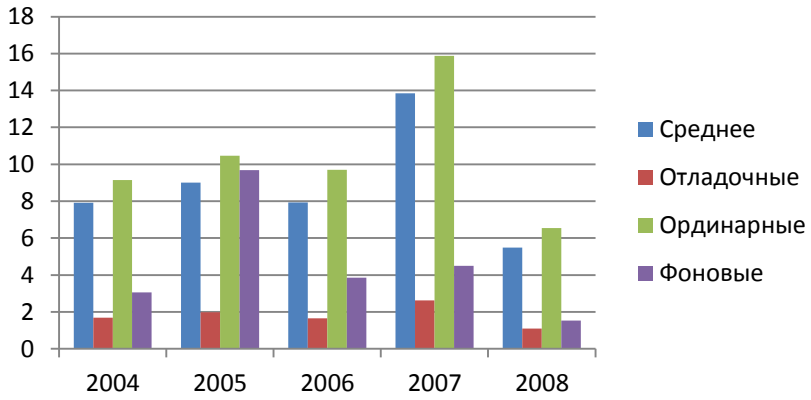

Рисунок 15. Коэффициент замедления разных классов заданий суперкомпьютера МВС-15000

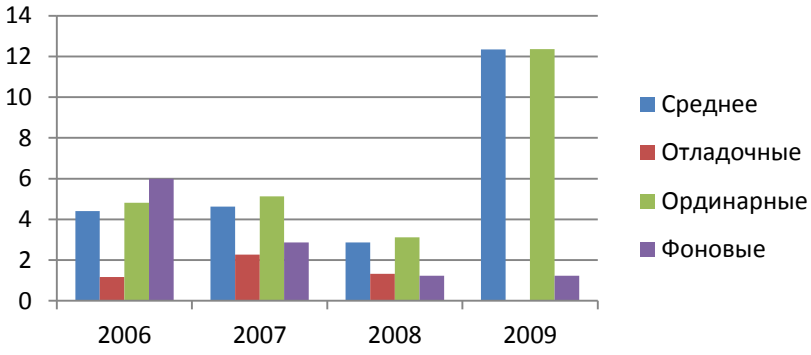

Рисунок 16. Коэффициент замедления разных классов заданий суперкомпьютера МВС-6000

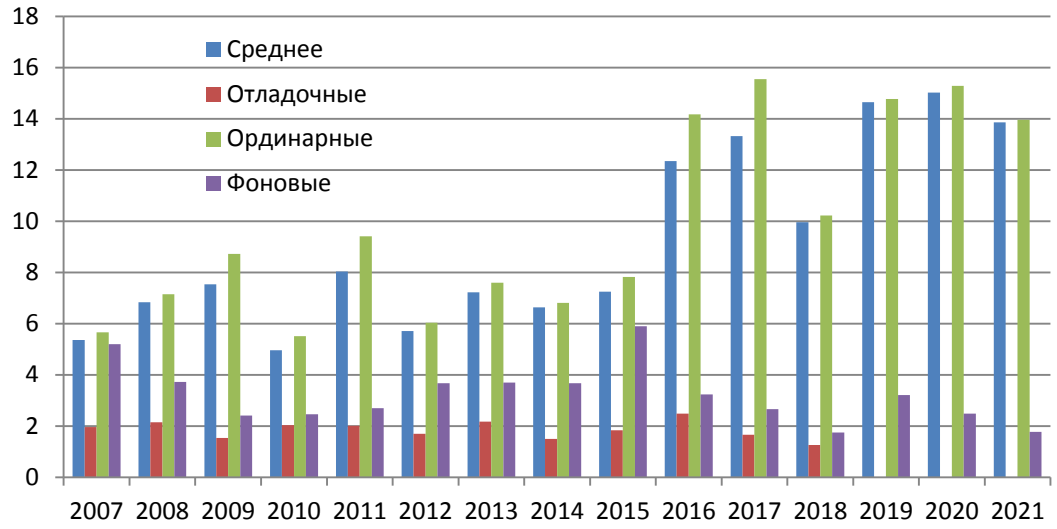

Рисунок 17. Коэффициент замедления разных классов заданий суперкомпьютера МВС-100К

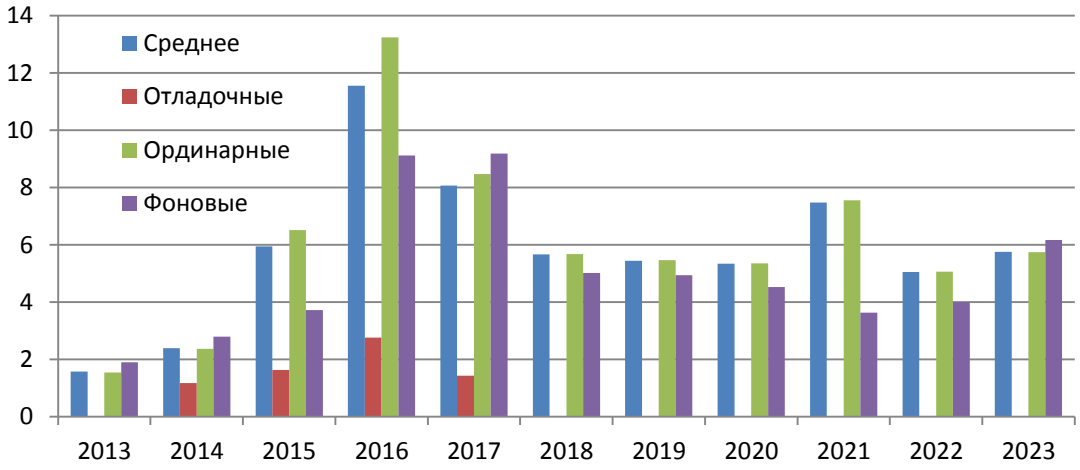

Рисунок 18. Коэффициент замедления разных классов заданий суперкомпьютера МВС-10П

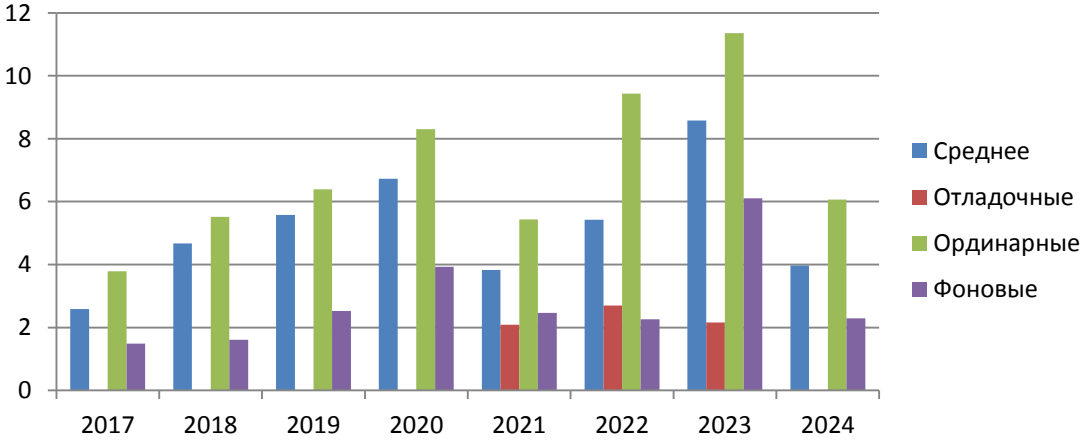

Рисунок 19. Коэффициент замедления разных классов заданий раздела Broadwell суперкомпьютера МВС-10П ОП

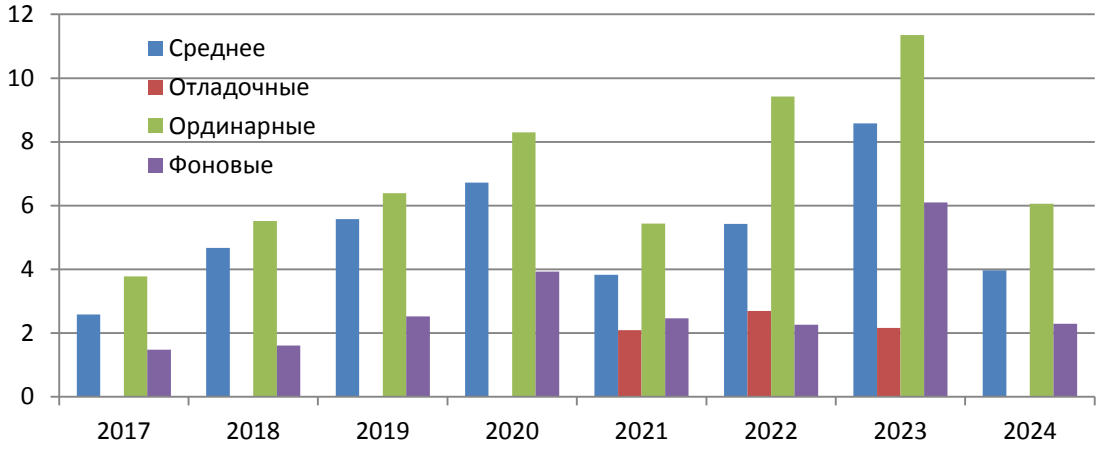

Рисунок 20. Коэффициент замедления разных классов заданий раздела KNL суперкомпьютера МВС-10П ОП

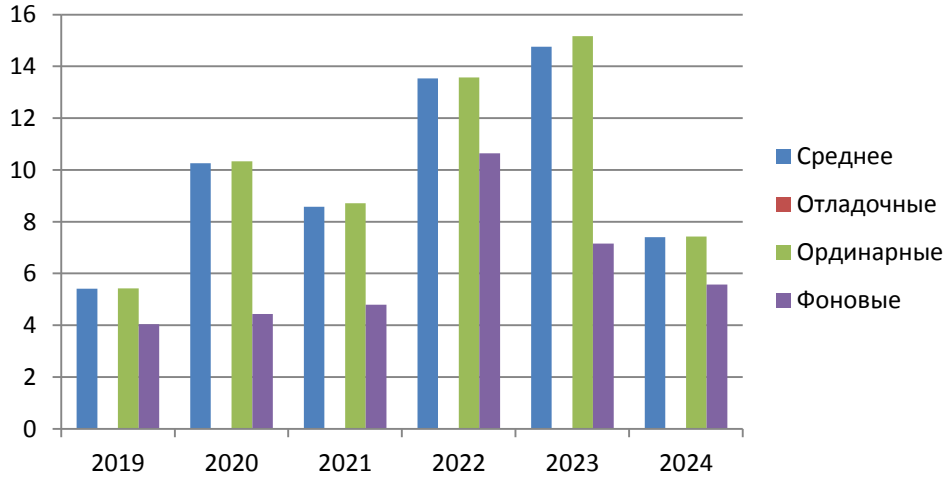

Рисунок 21. Коэффициент замедления разных классов заданий раздела Skylake суперкомпьютера МВС-10П ОП

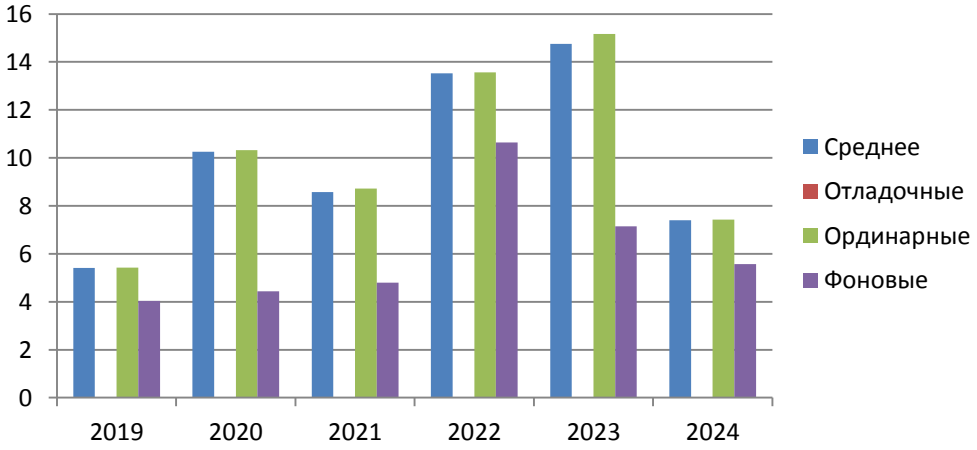

Рисунок 22. Коэффициент замедления разных классов заданий раздела Cascade Lake суперкомпьютера МВС-10П ОП

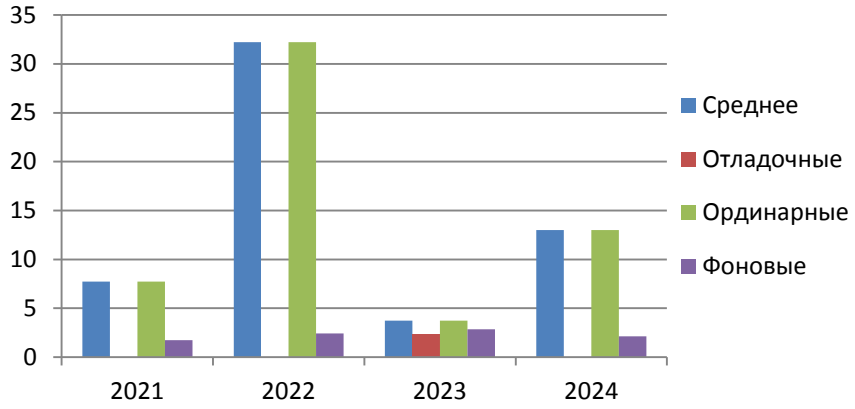

Рисунок 23. Коэффициент замедления разных классов заданий раздела Optane суперкомпьютера МВС-10П ОП

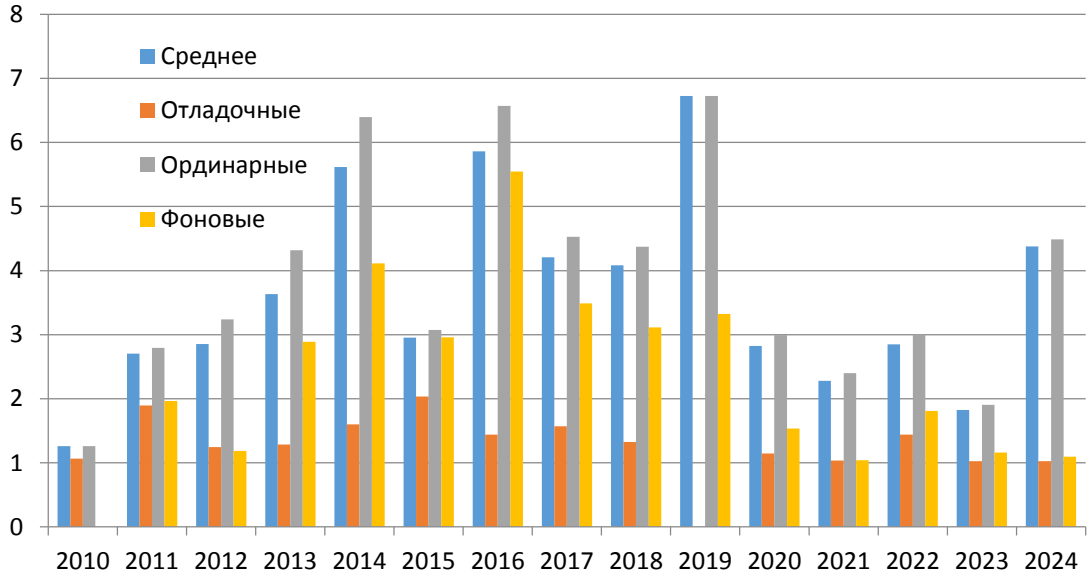

Рисунок 24. Коэффициент замедления разных классов заданий суперкомпьютера К100

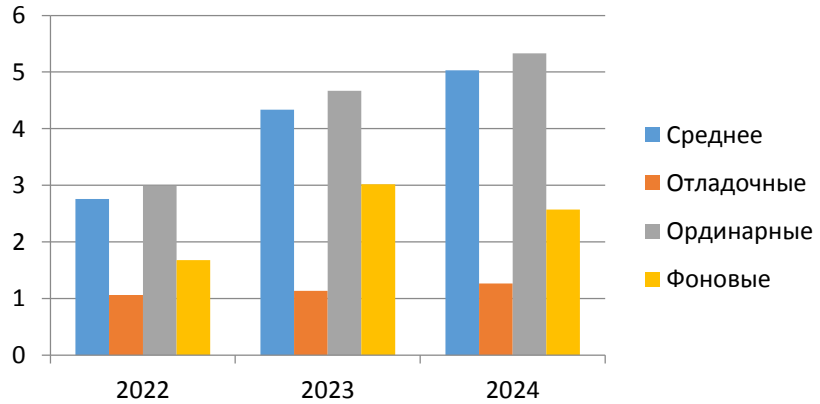

Рисунок 25. Коэффициент замедления разных классов заданий суперкомпьютера К60

Анализ представленных результатов позволяет сделать следующие выводы. Коэффициент замедления для отладочных заданий всегда меньше среднего коэффициента от 2 до 8 раз. Для фоновых заданий коэффициент замедления в подавляющем большинстве случаев ниже среднего значения и значения для ординарных заданий, исключения носят единичный характер. Темп обработки фоновых заданий выше темпа обработки ординарных заданий от 20% до 900%.

3.3.4 Особенности планирования отладочных заданий

Интересен пример раздела Broadwell суперкомпьютера МВС-10П ОП, приведенный в таблице 10. Режим отладочных заданий в этой системе действовал с 5 августа 2021 года по 30 мая 2023 года с неизменными параметрами. Отладочными считались задания, запрашивавшие не более 128 ядер не более чем на 30 минут. Отметим, что правами отладочных заданий обладают также фоновые задания с квантом, не превышающим время отладочного задания (в нашем примере – 30 минут) и запрашивающие число ядер не более значения, установленного для отладочных заданий (в нашем примере – 128 ядер). Такие фоновые задания мы также учитывали при расчете коэффициента замедления.

Сравним коэффициент замедления заданий с указанными параметрами для периода с отладочным режимом и для непосредственно предшествующих периодов до введения этого режима. Выключение отладочного режима в 2023 г. связано с передачей значительной части процессорных ядер из очереди СУППЗ в монопольное использование для решения специализированных задач. По этой причине коэффициент замедления для всех заданий очереди кратно вырос, и учет его при сравнении с периодом действия отладочного режима будет некорректным. Статистические показатели представлены в таблице 10.

Из таблицы 10 видно, что с момента введения отладочного режима (строка № 3) коэффициент замедления для указанной категории заданий существенно снизился и оставался на низком уровне вплоть до выключения отладочного режима. Более того, рисунок 21 показывает, что отладочный режим вызвал заметное

снижение как общего коэффициента замедления, так и коэффициента замедления ординарных и фоновых заданий.

Таблица 10. Статистические показатели раздела Broadwell суперкомпьютера МВС-10П ОП до и после введения режима обработки отладочных заданий

№	Период	Число заданий	Коэффициент замедления
1.	2021, 1 квартал	882	4,86
2.	01.04.2021 – 05.08.2021	2093	4,54
3.	05.08.2021 – 01.10.2021	674	1,9
4.	2021, 4 квартал	1518	2,13
5.	2022, 1 квартал	1643	3,05
6.	2022, 2 квартал	5233	3,27
7.	2022, 3 квартал	1924	3,84
8.	2022, 4 квартал	4367	1,93
9.	2023 1 квартал	1924	3,84
10.	01.04.2023 – 30.05.2023	2620	1,23

Таблица 11 демонстрирует значения показателя $C_{своевр}$ доли своевременно обработанных заданий, рассчитанной в соответствии с (8).

Таблица 11. Доля своевременно обработанных заданий разных классов

Система	Общая	Отладочные	Ординарн.	Фоновые
1	2	3	4	5
МВС-1000М	0,967	0,977	0,967	0,917
МВС-15000	0,963	0,973	0,960	0,950
МВС-6000	0,965	0,979	0,962	0,951
МВС-100К	0,978	0,983	0,977	0,978
МВС-10П	0,977	0,983	0,965	0,964
Broadwell	0,944	0,984	0,940	0,938
KNL	0,954	0,940	0,954	0,961
Optane	0,948	1	0,948	0,948
К100	0,985	0,992	0,983	0,973
К60	0,989	0,997	0,986	0,978

Из таблицы 11 видно, что доля своевременно обработанных заданий для подавляющего большинства систем заметно выше как общей доли своевременно обработанных заданий, так и значений этого показателя для классов ординарных и фоновых заданий.

За счет применения механизма «плавающего резерва» и занятия зарезервированных для отладочных заданий узлов заданиями других классов удалось повысить загрузку для суперкомпьютера МВС-100К на 9-14 тыс. узло-часов в год, а для суперкомпьютера МВС-10П ОП Broadwell на 4,5 тыс. узло-часов в год.

3.4 Планирование адаптивных заданий

3.4.1 Метод совмещения потоков адаптивных заданий и заданий с фиксированными параметрами (метод постпланирования)

При планировании заданий с фиксированными параметрами из-за разных размеров заданий в плане запуска неизбежно возникают так называемые **окна**, во время которых часть узлов простаивает. Для заполнения окон большинство планировщиком применяют одну или несколько рассмотренных выше стратегий обратного заполнения. Обратное заполнение позволяет помещать менее приоритетные задания в образующиеся окна, если это не задержит запуск более приоритетного задания. Обратное заполнение позволяет существенно поднять загрузку суперкомпьютера и минимизировать простои узлов.

Выделим адаптивные (эластичные) задания в отдельный класс и отдельный входной поток. Для адаптивных заданий допускается варьировать требуемый объём вычислительных ресурсов и заказанное время счёта. Рассмотрим задачу совмещения потоков адаптивных заданий и заданий с фиксированными параметрами в одной очереди.

Предлагаемый метод совмещения потоков адаптивных заданий и заданий с фиксированными параметрами основан на поиске окон в расписании запусков заданий с фиксированными параметрами и помещении в найденные окна адаптивных заданий. Размер адаптивного задания должен соответствовать размеру окна.

По этой причине процесс обработки адаптивных заданий совместно с потоком заданий с фиксированными параметрами назовем **постпланированием**. Выделим часть планировщика СУЗ, ответственную за обработку потока адаптивных заданий, которую назовем **постпланировщиком**. Метод совмещения двух разнородных потоков соответственно назовем **методом постпланирования**.

При 100% загрузке вычислителя заданиями с фиксированными параметрами обработка потока адаптивных заданий не имеет смысла, так как нет окон для заполнения. Очевидно, при отсутствии в очереди заданий с фиксированными параметрами (0% загрузке) поток адаптивных заданий сможет эффективно поднять загрузку суперкомпьютера. Важным является вопрос определения границы загрузки от 0 до 100%, при которой обработка потока адаптивных заданий позволит статистически значимо улучшить качество планирования. Актуальными вопросами настоящего исследования являются следующие. Насколько такая обработка сможет поднять загрузку вычислителя? Как сильно при этом замедлится обработка заданий с фиксированными параметрами?

Предлагаемый метод постпланирования представлен на рисунке 26 и заключается в следующем.

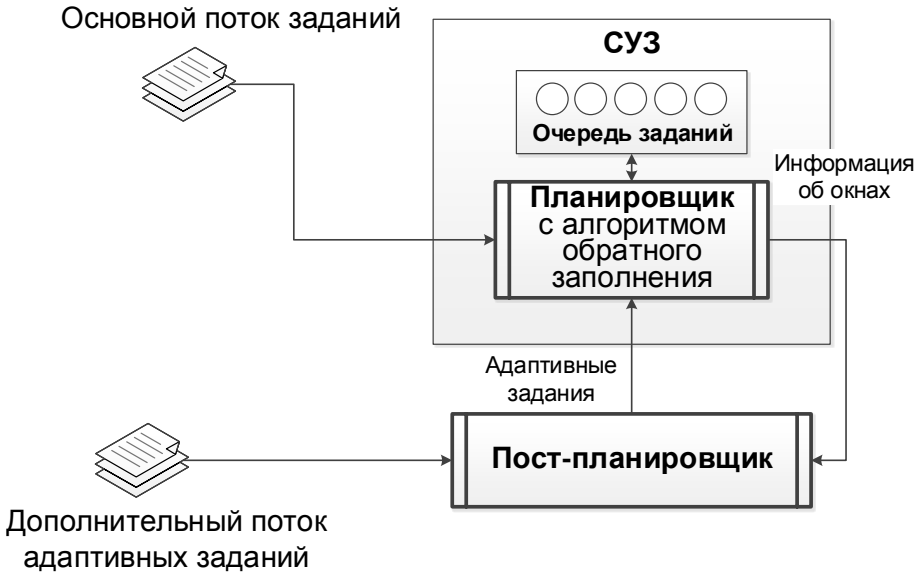

Рисунок 26. Метод постпланирования

1. Будем считать очередь заданий с фиксированными параметрами *основной*. Эту очередь обрабатывает соответственно *основной планировщик* СУЗ. Предполагается, что основной планировщик реализует одну из стратегий обратного заполнения, т.е. позволяет заполнять окна в расписании запусков заданиями соответствующих окнам размеров.

Рассмотрим понятия окна и обратного заполнения подробнее.

Расписание (план) запусков заданий основной очереди формируется на основе заданных пользователем фиксированных параметров: числа узлов суперкомпьютера (процессорных ядер) и времени выполнения задания. Из-за разных параметров заданий в плане запусков возникают так называемые окна, что демонстрирует рисунок 27. Высотой окна назовём число простаивающих узлов, а шириной окна назовём его длительность.

Рисунок 27. Пример окна в расписании запусков заданий

С помощью алгоритма обратного заполнения окна заполняются менее приоритетными заданиями, если это не помешает запуску более приоритетных заданий. По статистике, пользователи указывают заказанное время счёта с большим запасом [170]. На примере рисунка 27, если раньше завершится задание 1, то запущенное в окне менее приоритетное задание продолжит выполнение. В результате задание 2 не сможет запуститься раньше, как если бы окно не было заполнено. Более того, как показано на рисунке 28, образуется новое окно.

Образование таких новых окон по причине неточной пользовательской оценки времени выполнения является одним из главных факторов снижения эффективности планирования. Этот фактор вносит существенную стохастическую

составляющую в план запуска, что не позволяет делать точные прогнозы времен старта и завершения заданий.

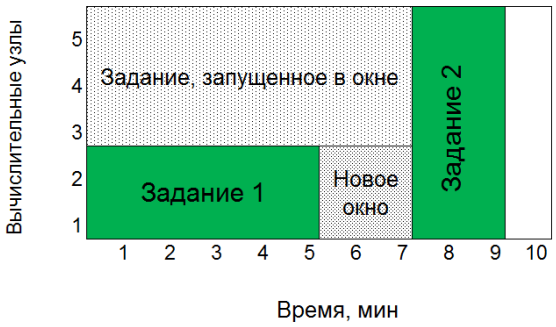

Рисунок 28. Пример отрицательного влияния обратного заполнения при досрочном завершении задания 1

2. Поток адаптивных заданий поступает на вход постпланировщика и образует тем самым дополнительную очередь. В отсутствие адаптивных заданий основная очередь обрабатывается применяемым в основном планировщике методом.

3. Постпланировщик анализирует текущее состояние основной очереди и определяет окна в плане запуска заданий, в которые можно поместить задания из дополнительной очереди. Анализ производится после каждого перестроения расписания основным планировщиком. Перестроение производится после поступления нового задания в очередь, удаления задания из очереди, запуска или завершения задания, иных событий.

4. Параметры очередного адаптивного задания задаются в соответствии с характеристиками найденного окна. Заказанное число узлов подбирается так, чтобы задействовать все узлы свободного окна. Время выполнения адаптивного задания не должно превышать длительности заполняемого окна и определяется в соответствии с коэффициентом заполнения окна (КЗО). Этот коэффициент принимает значение от 0 до 1 и показывает, какую часть окна по длительности можно использовать для адаптивного задания. На рисунке 29 представлено окно на 3 узла продолжительностью 10 минут. При коэффициенте заполнения равном 0,7 в рассматриваемое окно может быть помещено адаптивное задание на 3 узла с продолжительностью 7 минут.

Длительность появившегося окна может быть большой и даже неограниченной, если в основной очереди нет заданий, запланированных на некоторых узлах, как показано на рисунке 30. Для корректной обработки случая неограниченного окна введём параметр – максимальную длительность адаптивного задания.

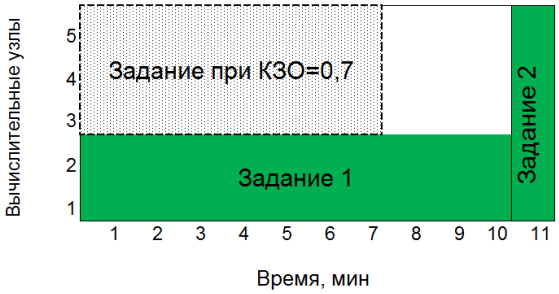

Рисунок 29. Пример длительности временно̀го окна при коэффициенте заполнения, равном 0,7

Рисунок 30. Пример неограниченного окна

5. Адаптивное задание с заданными постпланировщиком параметрами направляется в основную очередь. Согласно принципу обратного заполнения основной планировщик немедленно запускает это задание в соответствующем окне.

3.4.2 Исследование эффективности метода постпланирования

Рассмотрим факторы, влияющие на эффективность метода постпланирования:

1. Исходная загрузка вычислителя (без постпланированияя).

2. Характеристики основного входного потока заданий с фиксированными параметрами.

3. Характеристики дополнительного входного потока адаптивных заданий.

4. Настройки постпланировщика (коэффициент заполнения окна, максимальная длительность адаптивного задания).

Рассмотрим влияние исходной загрузки вычислителя на эффективность метода постпланирования при фиксированных значениях остальных факторов. Для решения этой задачи требуется провести экспериментальное исследование с множеством входных потоков с разной загрузкой, провести с каждым потоком эксперимент без постпланировщика и с ним, и рассчитать показатели эффективности СУЗ. Для проведения экспериментов использована модель СУЗ на основе симулятора СУППЗ с виртуальным вычислителем, рассмотренная в [171]. Для получения основной очереди заданий был применен генератор Lublin&Feitelson [187].

Для формирования дополнительного потока заданий мы рассмотрим идеальную ситуацию, когда для постпланировщика доступен бесконечный поток идеально масштабируемых заданий. Для проведения экспериментов было сформировано множество входных потоков по 5000 заданий каждый. В качестве решающего поля использовалась виртуальная вычислительная установка из 500 вычислительных модулей. Интервалы времени между заданиями были уменьшены так, чтобы последнее задание поступило через 120 часов (завершение 5-го дня) после начала эксперимента. Такое масштабирование поступления заданий в очередь позволяет обеспечить одинаковую длительность каждого эксперимента.

Для 5000 заданий и параметров по умолчанию было сгенерировано 50 наборов входных потоков различной интенсивности, обеспечивающих разную загрузку вычислителя. Со сгенерированными потоками были проведены эксперименты с целью отбора потоков, обеспечивающих без постпланирования загрузку от 60% до 100%. Величина этой загрузки включена в название соответствующего потока после префикса flow. Например, поток с названием flow89.1 обеспечивает загрузку без постпланирования в 0,984 или 98,4%.

Для оценки эффективности СУЗ при применении постпланирования будем использовать следующие показатели:

– загрузка, определяемая в соответствии с (2);

– время ожидания задания в очереди, определяемое в соответствии с (4);

– длина очереди;

– число одновременно выполняющихся заданий;

– коэффициент замедления, определяемый в соответствии с (5).

Для всех показателей рассчитывается среднее значение за период. В качестве периода для расчёта показателей использовался промежуток времени от 24 до 120 часов (2-5 дни). Последнее задание во всех экспериментах поступало до конца 6 дня после начала эксперимента. После завершения поступления заданий в очередь эксперимент ещё длится некоторое время до окончания обработки всех заданий. Для расчёта показателей эффективности используется диапазон с функционированием СУЗ в устоявшемся режиме.

Для времени ожидания задания в очереди и коэффициента замедления в дополнение к среднему рассчитывалось медианное значение.

Пример графика загрузки для потока низкой интенсивности flow59.6 представлен на рисунке 31. В первый день постпланировщик не работает, поэтому графики без постпланировщика и с постпланировщиком совпадают. После 1-го дня включается постпланировщик, обеспечивая почти всегда 100% загрузку вычислительных ресурсов.

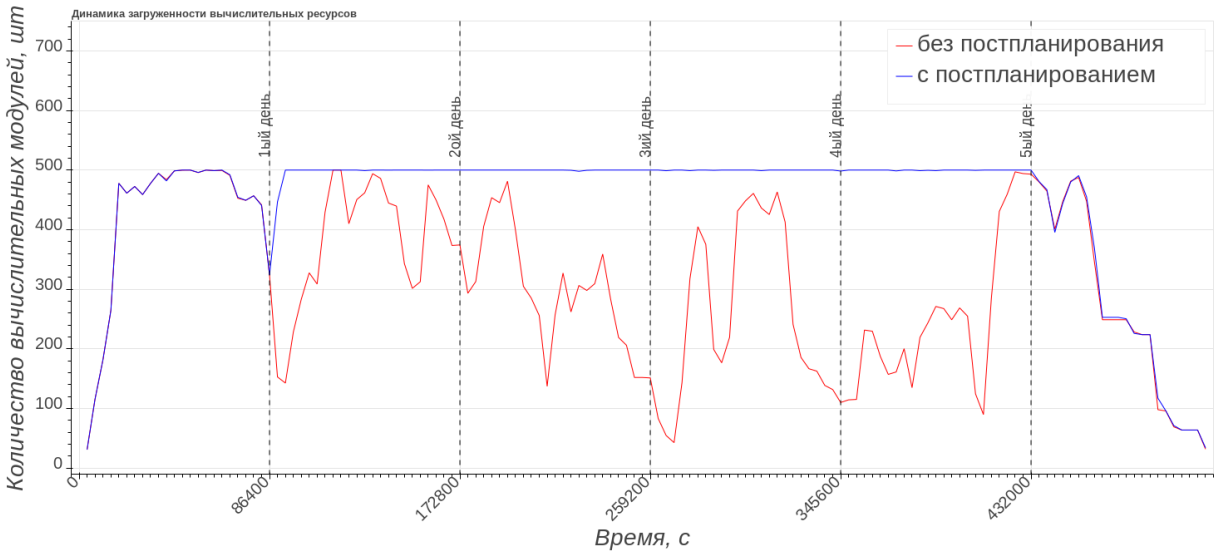

Рисунок 31. Загрузка вычислителя для потока flow59.6. Красная линия без постпланирования, синяя – с постпланированием

На рисунке 32 видно, что количество заданий в очереди относительно мало. Применение постпланирования увеличивает очередь заданий, то есть адаптивные задания в некоторой степени мешают заданиям основного потока.

Число одновременно запущенных заданий также увеличивается при применении постпланирования, как показано на рисунке 33.

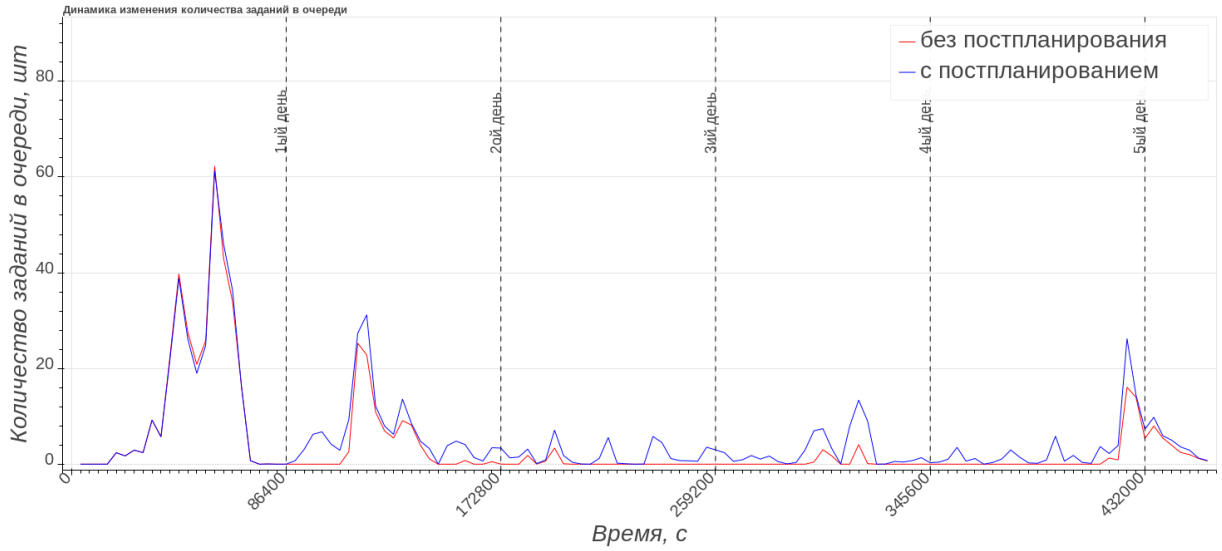

Рисунок 32. Число заданий в очереди для потока flow59.6. Красная линия без постпланирования, синяя – с постпланированием

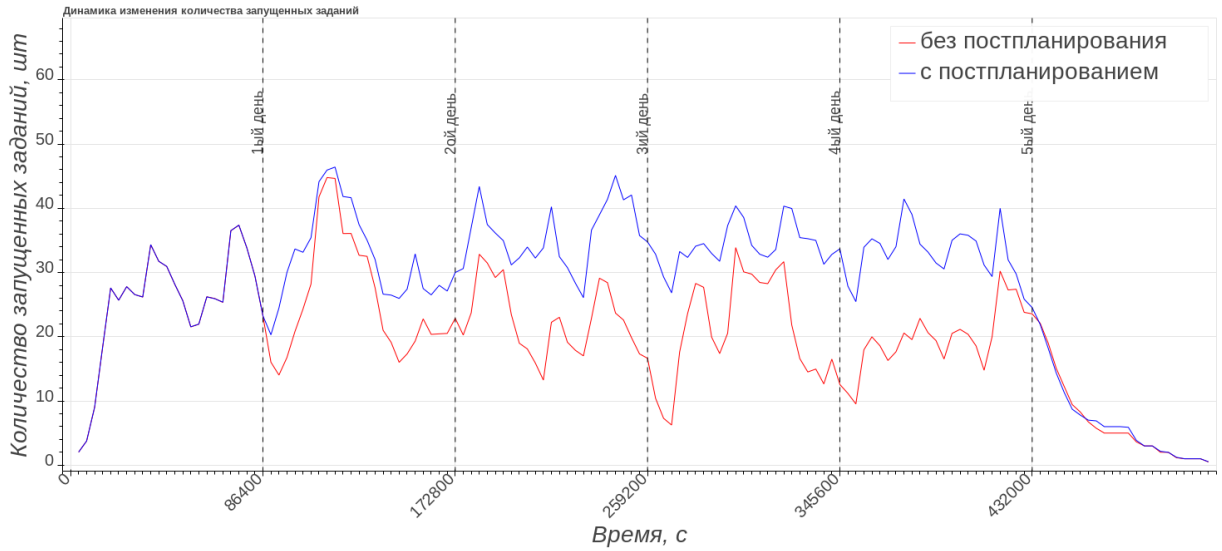

Рисунок 33. Число выполняющихся заданий для потока flow59.6. Красная линия без постпланирования, синяя – с постпланированием

Рассмотрим серию аналогичных графиков для потока высокой интенсивности. На рисунке 34 представлен график загрузки для потока flow98.4 высокой интенсивности. Загрузка без постпланирования изначально близка к 100%.

Рассмотрим график числа заданий в очереди для того же входного потока, представленный на рисунке 35. В очереди постоянно находится множество заданий, и с постпланировщиком эта очередь увеличивается.

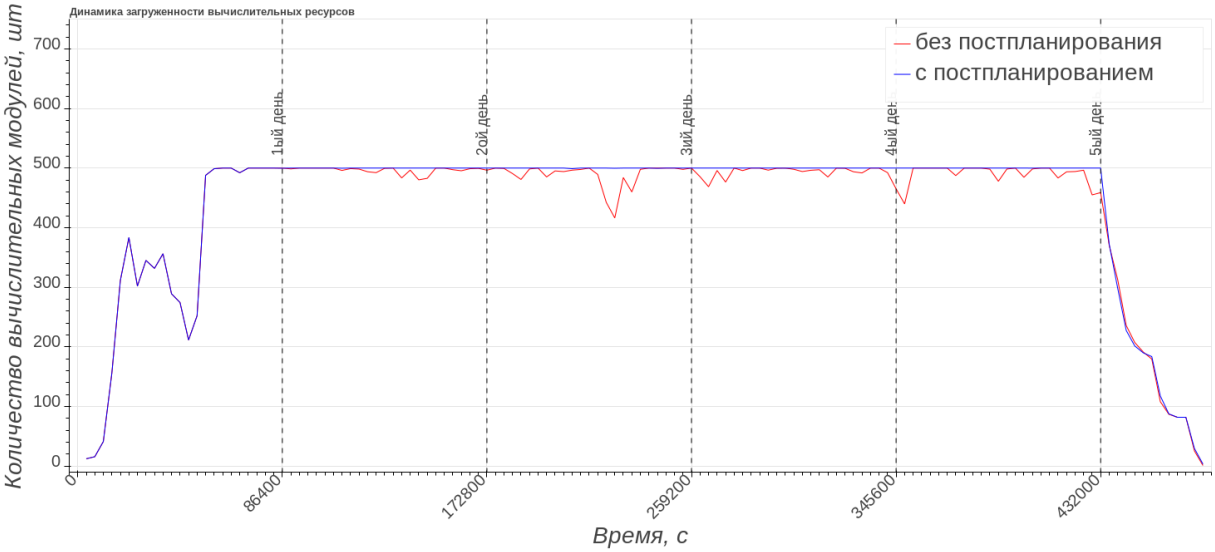

Рисунок 34. Загрузка вычислителя для потока flow98.4. Красная линия без постпланирования, синяя – с постпланированием

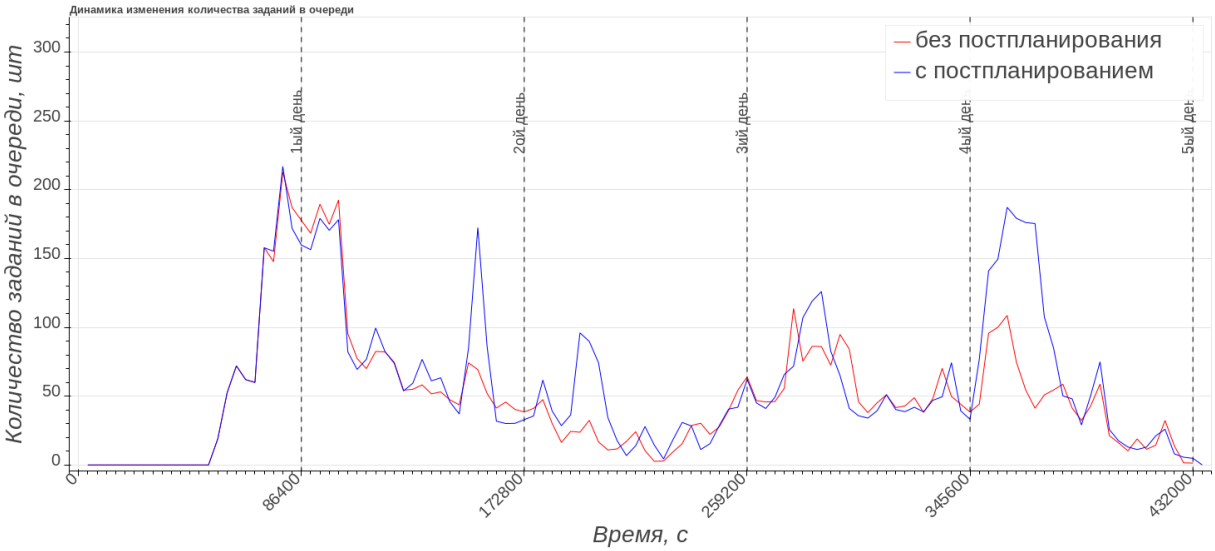

Рисунок 35. Число заданий в очереди для потока flow98.4. Красная линия без постпланирования, синяя – с постпланированием

Рисунок 36 демонстрирует, что число одновременно запущенных заданий увеличивается при постпланировании, но разница существенно меньше, чем для потока с невысокой интенсивностью.

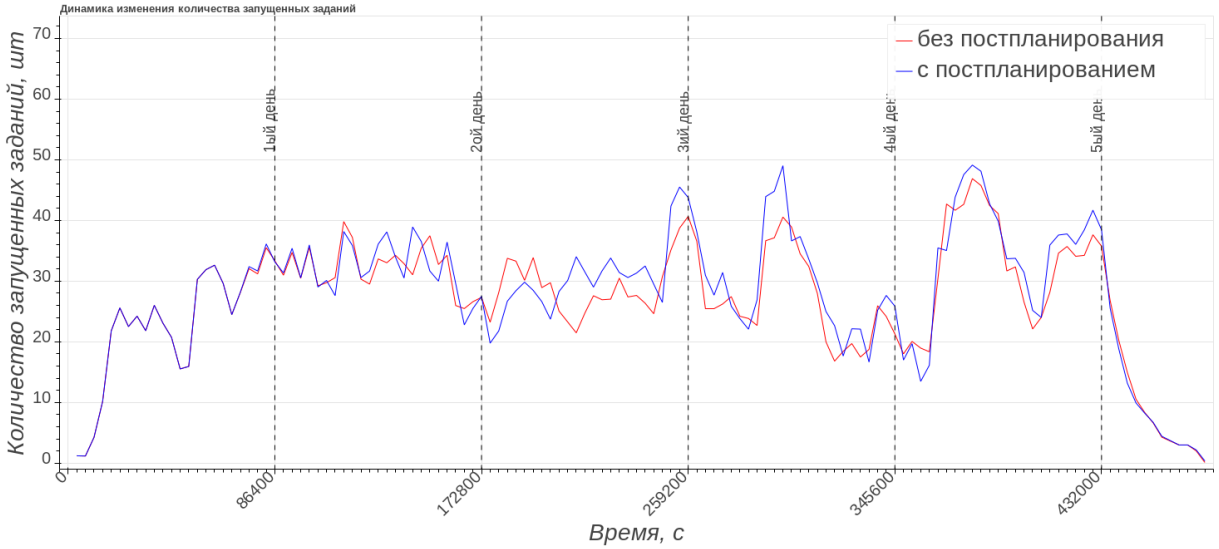

Рисунок 36. Число выполняющихся заданий для потока flow98.4. Красная линия без постпланирования, синяя – с постпланированием

Для оценки влияния постпланирования на загрузку вычислителя для каждого из 17 входных модельных потоков рассчитаем среднее значение загрузки за устоявшийся период (за часы с 24-го по 120-й, то есть за дни 2-5, убрав из расчётов 1-й день и окончание эксперимента после 5-го дня). Рассмотрим сводный график исходной загрузки и загрузки с постпланированием, представленный на рисунке 37. Каждая линия на графике соответствует набору из 17 экспериментов. По оси абсцисс указана загрузка вычислителя потоком без постпланирования, по оси ординат – загрузка в результате проведения эксперимента. Видно, что постпланирование позволяет обеспечить практически 100% загрузку.

Рассмотрим влияние постпланирования на средний коэффициент замедления, показанное на рисунке 38. Сплошная линия соответствует экспериментам без постпланирования, штриховая – экспериментам с постпланированием. Для отслеживания влияния постпланирования на задания основного потока дополнительно отобразим пунктирную линию, соответствующую эксперименту с постпланированием, но коэффициент замедления для этой кривой рассчитан только для зада-

ний основного потока без учёта заданий, добавленных постпланировщиком. Эта линия позволяет оценить изменение характеристик заданий исходного основного потока. Видно, что с уменьшением загрузки средний коэффициент замедления также уменьшается. Постпланировщик незначительно улучшает значение коэффициента замедления для всех заданий, но при этом увеличивает этот коэффициент для заданий основного потока.

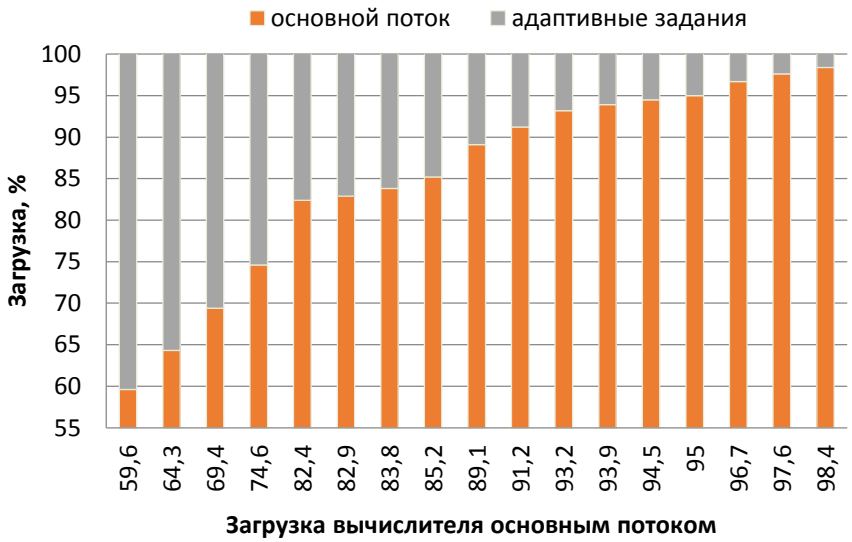

Рисунок 37. Загрузка для входных потоков различной интенсивности

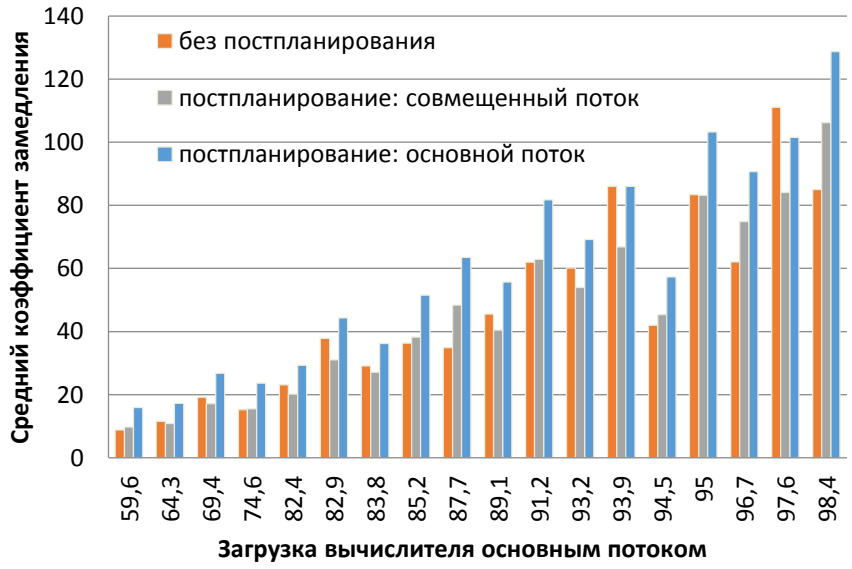

Рисунок 38. Средний коэффициент замедления для входных потоков различной интенсивности

Постпланирование также несколько ухудшает медианный коэффициент замедления, как показано на рисунке 39. Однако, начиная с 85% загрузки, медианный коэффициент замедления стабилизируется для совмещенного потока адаптивных заданий и заданий с фиксированными параметрами.

Постпланирование незначительно увеличивает среднее число заданий в очереди, как показано на рисунке 40.

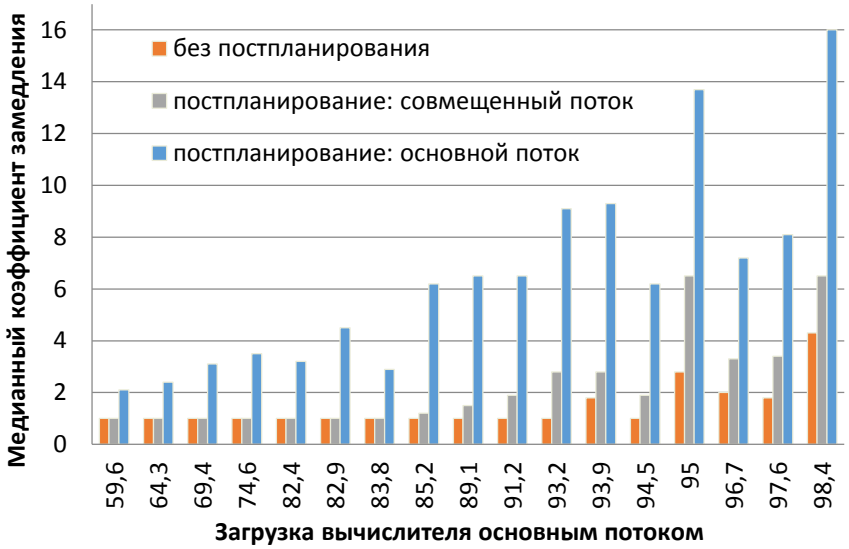

Рисунок 39. Медианный коэффициент замедления для входных потоков различной интенсивности

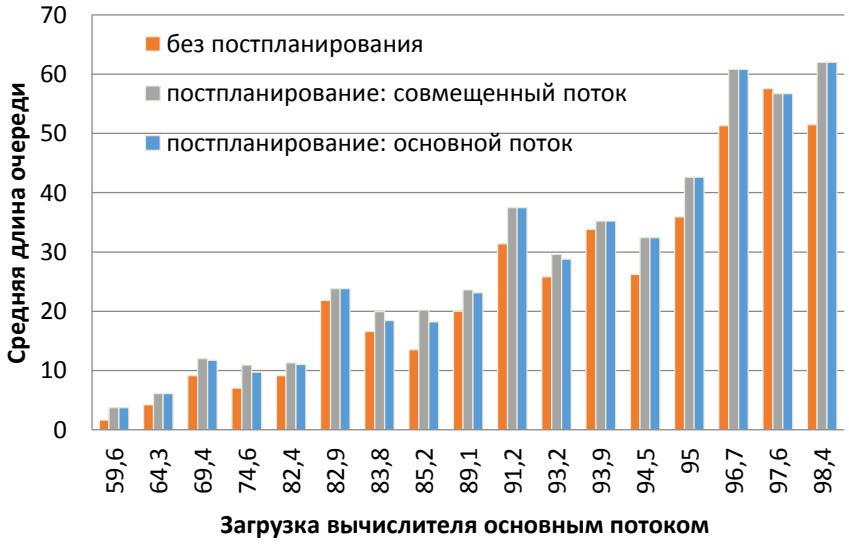

Рисунок 40. Среднее число заданий в очереди для входных потоков различной интенсивности

Рисунок 41 демонстрирует, что среднее время ожидания заданий совмещенного потока в очереди уменьшается, но для заданий основного потока увеличивается. При этом со снижением интенсивности входного потока отрицательное влияние постпланирования уменьшается.

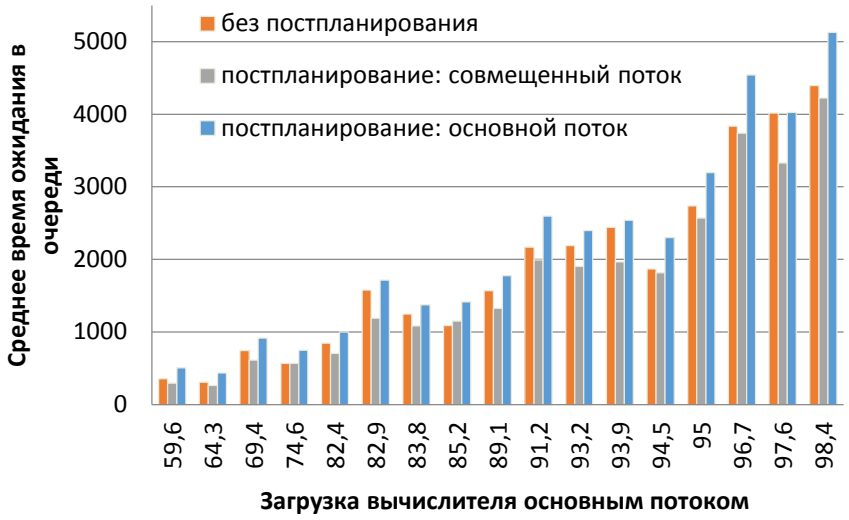

Рисунок 41. Среднее время ожидания в очереди для входных потоков различной интенсивности

На рисунке 42 представлено наглядное уменьшение негативного влияния постпланирования на медианное время ожидания заданий в очереди, которое существенно меньше для входных потоков малой интенсивности.

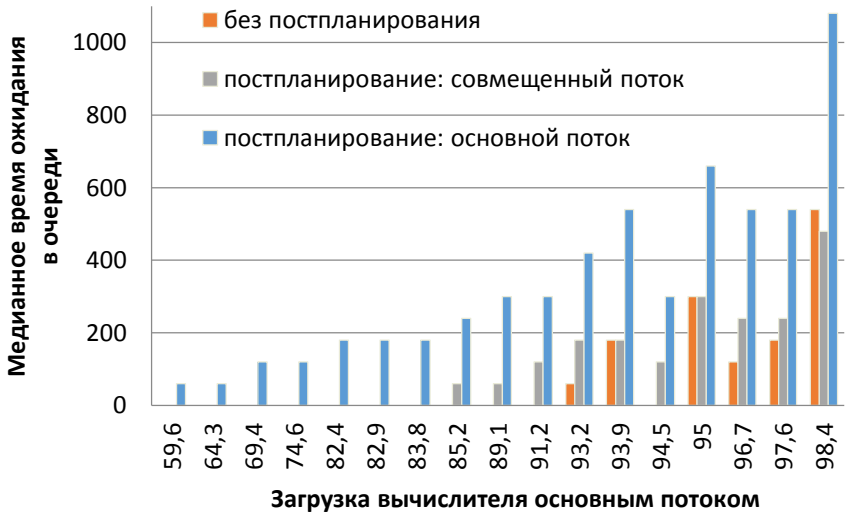

Рисунок 42. Медианное время ожидания в очереди для входных потоков различной интенсивности

Число одновременно выполняющихся заданий на вычислителе растёт при уменьшении интенсивности исходного потока, как показано на рисунке 43.

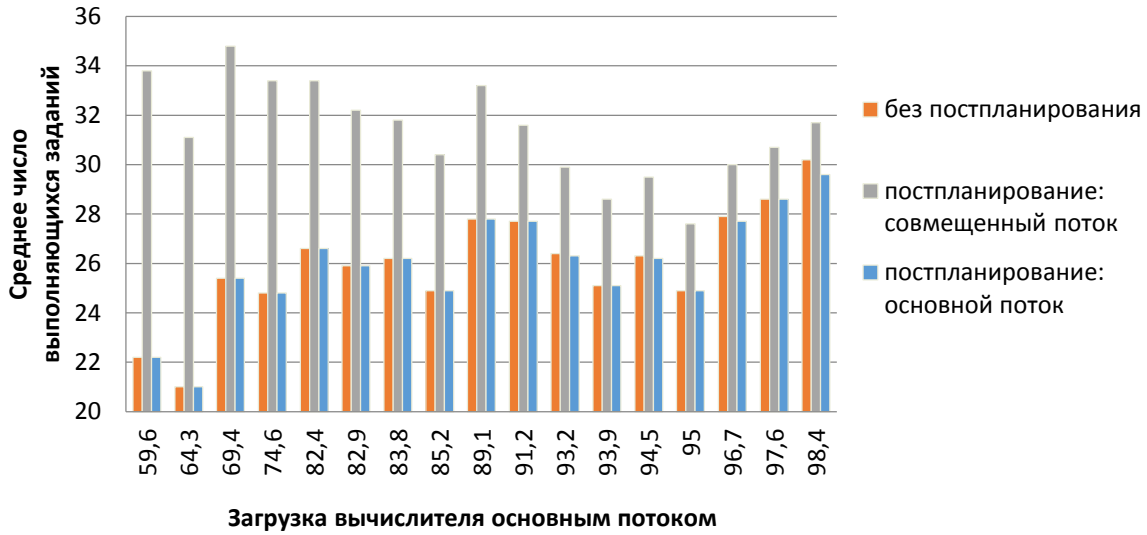

Рисунок 43. Среднее число выполняющихся заданий для входных потоков различной интенсивности

Постпланирование вносит отрицательный вклад в задержку заданий в очереди, увеличивая коэффицент замедления, однако с уменьшением интенсивности входного потока это отрицательное влияние уменьшается. На рисунке 44 по оси ординат отложен прирост медианного коэффициента замедления, обусловленный применением постпланирования. Рисунок показывает, что чем меньше интенсивность входного потока, тем меньше отрицательное влияние постпланирования на задержку заданий в очереди.

Рисунок 44. Изменение медианного значения коэффициента замедления заданий для входных потоков различной интенсивности

3.5 Методы совмещения потоков стандартных и нестандартных заданий

3.5.1 Система управления заданиями в составе облачной платформы

Как уже упоминалось, прямое использование СУЗ для обработки потока облачных заданий, как и прямое использование облачных платформ для выполнения традиционных суперкомпьютерных вычислений невозможно. При участии автора в МСЦ РАН были проведены исследования [72, 73, 75, 76, 112, 174, 188, 189] по представлению нестандартных заданий в виде виртуальных машин или контейнеров и разработке методов совмещения потока таких заданий с потоком стандартных заданий. Целью этих исследований был поиск перспективных решений, максимально совместимых с общепринятыми стандартами в облачных вычислениях, и позволяющих обрабатывать нестандартные задания наряду со стандартными, не изменяя при этом типовой порядок работы СУЗ.

Возможны два варианта применения СУЗ в составе облачной платформы. В первом из них [188] облачный сервис по высокопроизводительным вычислениям предоставляется пользователю через интерфейс облачной платформы, а СУЗ непосредственно управляет решающим полем, принимая задания из облачной платформы. Этот вариант можно условно назвать «облако управляет СУЗ». Для такого варианта необходимо разработать метод совмещения локального потока стандартных заданий и потока нестандартных заданий, поступающих из облачной платформы (потока облачных заданий). Второй вариант [76], который можно условно назвать «СУЗ управляет облаком», подразумевает использование системой управления заданиями облачной платформы для развертывания виртуальных машин и контейнеров при обработке потока нестандартных заданий.

3.5.2 Представление СУЗ в качестве гипервизора

Рассмотрим вариант [188] организации облачного сервиса для высокопроизводительных вычислений, предоставляемого пользователю через

интерфейс и посредством некоторой облачной платформы. Решение этой задачи должно удовлетворять следующим требованиям:

− сохранение исторически сложившегося порядка обслуживания пользователей через СУЗ;

− ограничение на применение гипервизорной виртуализации с целью снижения накладных расходов и обеспечения непосредственного доступа пользователей к уникальным возможностям суперкомпьютерного оборудования;

− максимально использование стандартных механизмов облачных вычислений.

Методом, позволяющим удовлетворить перечисленным требованиям, является представление СУЗ в виде гипервизора, что позволяет встроить СУЗ в стандартный стек облачных решений.

В исследовании [188], проведенным под руководством автора, был рассмотрен ряд облачных платформ, свободно распространяемых в исходных текстах, среди которых была выделена платформа OpenStack. Со многими облачными платформами OpenStack объединяет специализированная библиотека libvirt [190]. Библиотека управления виртуализацией libvirt представляет собой дополнительный уровень абстракции, позволяющий работать с виртуальными машинами под управлением различных гипервизоров. Библиотека поддерживает большое количество гипервизоров, таких как KVM, Xen, VMware ESX, VMware Workstation, и входит в состав множества программных продуктов, а также является одним из основных средств по управлению гипервизорами в облачной платформе OpenStack.

В состав библиотеки libvirt входит отдельный драйвер для каждого гипервизора. При этом сохраняется возможность реализации и встраивания в библиотеку собственного драйвера для своего гипервизора. Основная идея заключается в представлении СУППЗ в виде некоторого специализированного гипервизора путём разработки соответствующего драйвера для библиотеки libvirt, что должно обеспечить прозрачную интеграцию СУППЗ суперкомпьютера в стек

программного обеспечения облачной платформы. Для сопряжения СУППЗ с облачной платформой OpenStack был создан собственный драйвер для библиотеки libvirt. Драйвер преобразует команды управлению виртуальными машинами облачной платформы в команды управления заданиями суперкомпьютера. Для этого были введены следующие абстракции.

1. СУППЗ является гипервизором в абстракциях облачного сервиса.

2. Задание является виртуальной машиной в абстракциях облачного сервиса. Как и у ВМ, у задания может быть несколько состояний: «в очереди», «выполняется», «не выполняется». Были введены аналогичные абстракции: «ВМ работает» – «задание выполняется», «ВМ не работает» – «задание не выполняется», «ВМ на паузе» – «задание в очереди».

3. Суперкомпьютерные ресурсы, требуемые для выполнения задания, являются ресурсами ВМ. Были введены соответствующие абстракции: «число процессоров ВМ» – «число процессоров для задания», «размер оперативной памяти ВМ» – «максимальное время выполнения задания».

В процессе работы выяснилось, что отсутствует обратная совместимость версий библиотеки libvirt, в связи с чем разработанный для более ранней версии драйвер не может быть непосредственно подключен к библиотеке более поздней версии. Для преодоления несовместимости авторами была разработана специальная методика создания драйвера библиотеки libvirt, подробно рассмотренная в публикации [188].

3.5.3 Облачный сервис для высокопроизводительных вычислений на базе платформы OpenStack и СУППЗ

Структура разработанного облачного сервиса представлена на рисунке 45. Облачный сервис состоит из двух частей: управляющей и вычислительной. В управляющей части размещаются следующие основные компоненты облачной платформы OpenStack.

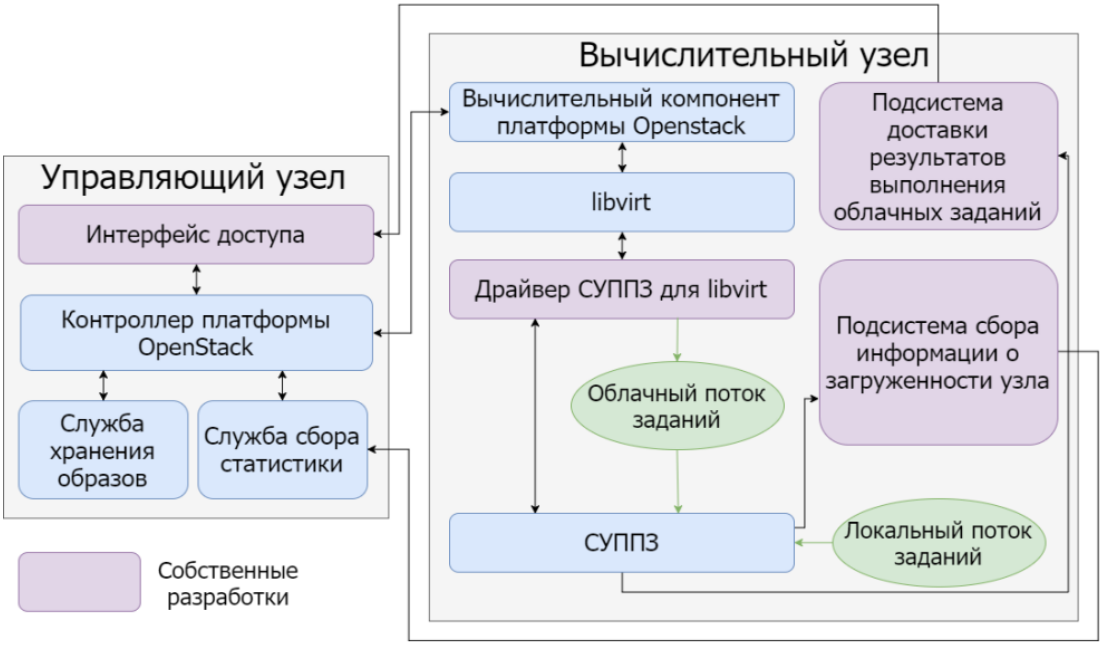

Рисунок 45. Структура облачного сервиса для высокопроизводительных вычислений на базе платформы OpenStack, библиотеки libvirt и СУППЗ

1. Контроллер облачной платформы OpenStack. Осуществляет обработку поступающих платформе команд и подготовку данных для запуска суперкомпьютерных заданий, а также отвечает за балансировку нагрузки между вычислительными узлами.

2. Служба хранения образов платформы OpenStack предназначена для хранения файлов, необходимых для запуска суперкомпьютерного задания.

3. Служба сбора статистики облачной платформы OpenStack предназначена для сбора информации о состоянии вычислительных узлов, входящих в состав платформы.

4. Интерфейс доступа пользователей к платформе, транслирует команды пользователей в программные вызовы облачной платформы OpenStack.

Вычислительная часть может состоять из нескольких вычислительных узлов, каждый из которых представляет отдельный суперкомпьютер и имеет следующий состав.

1. Вычислительный компонент облачной платформы OpenStack транслирует команды облачной платформы OpenStack в вызовы библиотеки виртуализации libvirt.

2. Драйвер СУППЗ транслирует вызовы библиотеки libvirt в вызовы команды СУППЗ управления заданиями.

3. СУППЗ получает от драйвера команды управления заданиями.

4. Подсистема доставки результатов облачных заданий на управляющий узел.

5. Подсистема сбора информации о загруженности вычислительного узла.

Макет представленного на рисунке 45 облачного сервиса был реализован на суперкомпьютерных ресурсах МСЦ РАН. В состав макета вошли один управляющий узел OpenStack и два вычислительных узла, каждый из которых представлял отдельный сегмент суперкомпьютера МВС-100К под управлением СУППЗ. В платформе OpenStack сегменты суперкомпьютера были видны как вычислительные узлы с большим числом процессорных ядер, представлявших ядра суперкомпьютера, и значительным объёмом оперативной памяти, представлявшим ёмкость очереди заданий СУППЗ. Запросы к OpenStack через драйвер СУППЗ библиотеки libvirt трансформировались в команды постановки в очередь суперкомпьютерных заданий. Платформа OpenStack адекватно отображала загрузку вычислительных узлов с учётом поступающих напрямую в СУППЗ локальных заданий, автоматически производя балансировку нагрузки между двумя суперкомпьютерными сегментами.

3.5.4 Облачная среда для высокопроизводительных вычислений на базе платформы Proxmox и СУППЗ

Рассмотрим вариант облачного сервиса для высокопроизводительных вычислений [76], использующий облачную платформу в качестве инструмента управления виртуальными машинами и контейнерами из состава нестандартных пользовательских заданий. Схема такой облачной среды для высокопроизводительных приложений, построенной в МСЦ РАН, представлена на рисунке 46.

Рисунок 46. Вариант построения облачной среды для высокопроизводительных приложений

Облачная среда включает в себя высокопроизводительную вычислительную систему (суперкомпьютер), работающую под управлением СУППЗ. Суперкомпьютер был дополнен специально разработанной подсистемой управления виртуальными машинами (ПУВМ). Каждый вычислительный узел суперкомпьютера был оснащён гипервизором KVM. В СХД кластера была выделена область для хранения образов виртуальных машин (ВМ) и контейнеров. Для обеспечения возможности развёртывания и управления виртуальными машинами была применена свободно распространяемая платформа виртуализации Proxmox Virtual Environment (Proxmox VE) [191].

Облачная среда предоставляет возможность пользователю сформировать нестандартное задание в виде виртуальной машины на базе любой программной платформы (Linux/Windows и т.п.) и направить это задание в СУППЗ. Когда нестандартное задание попадает в СУППЗ, оно проходит через очередь наряду с обычными (стандартными) заданиями, рассчитанными на выполнение в рассмотренном в п. 1.2.2 стандартном стеке программного обеспечения. При запуске нестандартного задания задействуется ПУВМ, которая, в свою очередь, используя платформу Proxmox VE, извлекает из СХД образ виртуальной машины, содержащий необходимую заданию программную платформу, и разворачивает набор виртуальных машин из этого образа на выделенных для задания ВУ суперкомпьютера. При этом автоматически конфигурируется виртуальная

локальная сеть, доступная пользователю для запуска его параллельного приложения на развёрнутой виртуальной программной платформе.

Процесс подготовки и запуска нестандартного задания состоит из следующих этапов. На первом этапе пользователь или системный администратор подготавливают образ виртуальной машины, который сохраняется в специальном хранилище образов ВМ и контейнеров. На следующем этапе пользователь формирует паспорт нестандартного задания с включением в него информации о том, какой образ ВМ из хранилища будет использован, какие ресурсы (число ядер, объём оперативной памяти) требуются для функционирования виртуальной машины, развёрнутой из этого образа. Далее нестандартное задание направляется в очередь СУППЗ.

После того, как нестандартное задание пройдёт через очередь, ПУВМ развернёт на выделенных вычислительных узлах суперкомпьютера виртуальные машины из указанного в паспорте задания образа. По завершении процесса развёртывания и конфигурирования в стандартный вывод задания будут помещены IP-адреса для подключения к запущенным виртуальным машинам. Используя выданные IP-адреса, пользователь (или запущенная им прикладная программная система) подключаются к виртуальным машинам и производят прикладные вычисления.

Для автоматической настройки сетевого оборудования для динамической организации виртуальной среды А.П. Овсянниковым были предложены и реализованы соответствующие методы [76]. После реализации облачной среды авторами [76] была произведена оценка влияния средств виртуализации KVM и Proxmox VE, установленных на вычислительных модулях кластера, на производительность стандартных заданий. Оценка влияния применённых средств виртуализации на производительность MPI-программ осуществлялась с использованием стандартных тестов NAS Parallel Benchmarks 3.3 (NPB).

Тестирование выполнялось в два этапа. На первом этапе тесты NPB запускались на вычислительных узлах под управлением ОС Linux Debian Jessie.

На втором этапе выполнялся запуск тестов NPB на тех же узлах, но с установленными программными компонентами KVM и Proxmox VE. Подробно методика проведения и результаты экспериментов рассмотрены в работе [76]. Результаты позволяют сделать вывод, что компоненты KVM и Proxmox VE не оказывают существенного влияния на производительность стандартных заданий.

Для оценки влияния виртуальной среды на производительность MPI-программ была проведена серия экспериментов, в ходе которых тесты NPB запускались в виртуальных машинах KVM. Подробно методика проведения и результаты экспериментов также рассмотрены в работе [76]. По итогам экспериментов, потери производительности составили от 3% до 9%.

Итогом исследований стала иерархическая архитектура облачной среды для проведения научных исследований [189], представленная на рисунке 47. Облачная среда состоит из трех уровней. Каждый уровень предоставляет свой тип облачного сервиса: IaaS, PaaS и SaaS. Каждый пользователь может одновременно работать как с одним, так и с несколькими типами сервисов.

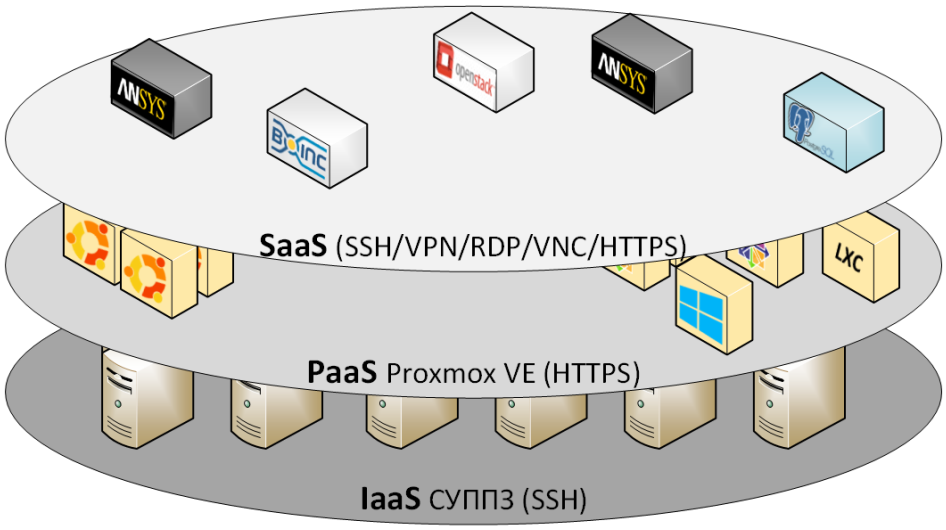

Рисунок 47. Иерархическая архитектура облачной среды для проведения научных исследований

Доступ пользователей к IaaS-сервисам осуществляется через командный интерфейс СУППЗ по протоколу SSH. Пользователю выделяется необходимое количество вычислительных узлов и предоставляется беспарольный доступ на каждый из них. Время доступа определяется через командный интерфейс СУППЗ

и ограничивается заданными системным администратором настройками. Для каждого пользователя автоматически создается проектный каталог, расположенный на сетевом дисковом пространстве, и предоставляется набор средств разработки MPI-, OpenMP-, Python-программ, отладки и контроля версий. Стандартные потоки ввода, вывода и ошибок перенаправляются в рабочий каталог задания пользователя.

PaaS-сервисы функционируют под управлением платформы виртуализации Proxmox VE. По запросу пользователя СУППЗ выделяет требуемое количество вычислительных узлов и сохраняет их IP-адреса в стандартный поток вывода задания. Выделенные узлы образуют виртуальный вычислительный кластер, на котором пользователь может разворачивать как виртуальные машины KVM, так и контейнеры. Доступ к виртуальному кластеру и управление виртуальными машинами осуществляется пользователем по протоколу HTTPS через браузер. В качестве URL-адреса для подключения к виртуальному кластеру может быть использован любой IP-адрес из выделенных при старте задания. В автоматическом режиме выделения вычислительных узлов СУППЗ ограничивает время аренды виртуального кластера, но по запросу пользователей системный администратор может увеличить время аренды путем блокировки узлов суперкомпьютера после старта задания.

SaaS-сервисы представляют собой виртуальные машины или контейнеры с запущенными сервисами: удаленной визуализации результатов моделирования и разработки CAD-моделей для проведения расчетных экспериментов, системы коллективной разработки программного обеспечения, организации вычислений с распараллеливанием по данным, почтовые службы, базы данных, VPN, виртуальные лаборатории, удаленные рабочие столы и др. Виртуальные машины и контейнеры функционируют в круглосуточном режиме на выделенных администраторами МСЦ РАН вычислительных модулях, объединенных в отказоустойчивый виртуальный кластер средствами Proxmox. В зависимости от вида SaaS-сервиса используется свой протокол доступа: SSH, RDP, VNC, HTTPS

или VPN. Для организации контроля доступа пользователей к каждому SaaS-сервису используется LDAP или Active Directory.

Отметим ключевые особенности предложенной архитектуры:

– использование отечественного программного обеспечения, либо программного обеспечения, распространяемого в открытых исходных текстах;

– возможность одновременного функционирования трех типов сервисов (IaaS, PaaS и SaaS) на одной вычислительной системе;

– возможность объединения нескольких суперкомпьютерных систем в единое облако.

Выводы к главе 3

Для планирования заданий предложен метод, основанный на приоритетной невытесняющей дисциплине обслуживания с реализацией принципа «faire share» и применением консервативной стратегии обратного заполнения. Особенностью метода, определяющей его научную новизну, является выделение во входном потоке классов отладочных, ординарных и фоновых заданий. Отладочные задания предполагают малое время выполнения на небольшом числе резервируемых процессорных ядер. Ординарные задания однократно запускаются на произвольном числе ядер на продолжительное, но ограниченное время. Фоновые задания могут выполняться произвольное время, но периодически прерываются системой и возвращаются в очередь. Резерв процессорных ядер под отладочные задания является «плавающим», что позволяет на непродолжительное время занимать зарезервированные ядра заданиям других классов и повышать тем самым загрузку суперкомпьютера.

Предложенный метод был реализован и применяется с 1999 года в составе планировщика СУППЗ. Анализ накопленной статистики суперкомпьютеров, установленных в МСЦ РАН и ИПМ им. М.В. Келдыша РАН, позволяет утверждать, что средний коэффициент замедления отладочных и фоновых заданий в

подавляющем большинстве случаев кратно ниже, чем общий коэффициент замедления, а темп обработки заданий этих классов соответственно выше.

При планировании суперкомпьютерных заданий в расписании запусков неизбежно образуются простаивающие вычислительные ресурсы (окна). Наличие окон снижает загрузку суперкомпьютера и вызывает простой его узлов. Для борьбы с окнами планировщики применяют различные алгоритмы обратного заполнения, позволяющие заполнять окна менее приоритетными заданиями подходящего размера. Однако, по причине неточной оценки пользователями времени выполнения заданий расписание постоянно перестраивается, и образуются новые окна.

Предложен метод постпланирования, который позволяет динамически отслеживать появление окон и заполнять их за счёт совмещения разнородных потоков заданий двух классов. К первому классу (основной поток заданий) относятся задания с фиксированными параметрами – требуемыми числом суперкомпьютерных узлов и временем выполнения. Ко второму классу (дополнительный поток заданий) относятся адаптивные (эластичные) задания, которые могут выполняться разное время на разном числе узлов. Дополнительный поток адаптивных заданий обрабатывается специальным компонентом, названным постпланировщиком.

Суть предлагаемого метода заключается в том, что постпланировщик при обнаружении окна помещает очередное адаптивное задание в основную очередь. Время выполнения и число узлов адаптивного задания подбирается в соответствии с размером обнаруженного окна, что за счет принципа обратного заполнения влечёт практически немедленный запуск адаптивного задания основным планировщиком.

Впервые были получены результаты имитационного моделирования, которые на входных потоках разной интенсивности продемонстрировали максимизацию загрузки суперкомпьютера, минимизацию коэффициента замедления адаптивных заданий при незначительном увеличении коэффициента замедления основного потока заданий, а также позволили определить границы применимости метода постпланирования.

Исследованы решения для обработки входного потока пользовательских заданий, в котором совмещаются стандартные и нестандартные задания. Последние требуют для своего выполнения отдельной программной платформы, с отдельными операционной системой, прикладным и инструментальным программным обеспечением. Автором предложено два метода совмещения потоков стандартных и нестандартных заданий.

Первый метод заключается в представлении системы управления заданиями в виде гипервизора виртуального многоядерного вычислительного узла с большим объемом оперативной памяти. Основным средством реализации метода явилось создание собственного драйвера для библиотеки управления виртуальными машинами libvirt, что позволило осуществить прозрачное включение СУППЗ в состав облачной платформы OpenStack. Предложенный метод был реализован в виде облачного сервиса для высокопроизводительных вычислений, развёрнутого на ресурсах суперкомпьютера МВС-100К.

Второй метод основан на использовании облачной платформы, как инструмента автоматического развертывания при старте нестандартного задания виртуальных машин и контейнеров на вычислительных узлах суперкомпьютера. При этом вычислительные узлы для запуска набора виртуальных машин динамически выделяются средствами системы управления заданиями. Проведённые эксперименты показали, что применённые при реализации метода средства виртуализации KVM и Proxmox VE не оказывают значительного влияния на производительность стандартных заданий, что позволяет успешно совмещать обработку стандартных и нестандартных заданий в одном входном потоке. Результаты исследований позволили предложить трехуровневую архитектуру облачной среды для научных исследований, предоставляющей пользователям Iaas-, PaaS- и SaaS-сервисы для высокопроизводительных вычислений.

Глава 4. Отображение параллельной программы на вычислительные узлы суперкомпьютера

4.1 Задача отображения параллельной программы на вычислительные узлы суперкомпьютера

Среди рассмотренных в п. 1.3 задач управления вычислительными ресурсами суперкомпьютерной системы коллективного пользования была отмечена задача выделения заданию требуемых вычислительных ресурсов, т.е. некоторого подмножества ВУ решающего поля. Эта задача включает в себя выбор для задания ВУ из числа свободных и их конфигурацию. Задача выбора ВУ относится к третьему уровню иерархии управления и решается в СУППЗ средствами сервера запуска, а конфигурация ВУ обеспечивается менеджером задания и надстройкой подготовки и диагностики ВУ.

Одной из целей выбора ВУ и их конфигурации является предоставление заданию такого подмножества ВУ и в такой конфигурации, чтобы обеспечить как можно более высокую скорость выполнения прикладных расчетов. Другими словами, необходимо так подобрать подмножество ВУ из числа свободных и таким образом распределить процессы параллельной программы по выбранным ВУ, чтобы минимизировать время выполнения параллельной программы из задания пользователя. Часто такую задачу называют **задачей отображения параллельной программы на вычислительные узлы суперкомпьютера** или кратко **задачей отображения.**

Вернемся к введенной в п. 1.6 иерархической модели управления вычислительными ресурсами и уточним ее. В общем случае модель описывает распределенную вычислительную систему, состоящую из некоторого числа вычислительных узлов, подмножество которых в текущий момент времени занято определенными работами (заданиями). Представим распределенную вычислительную систему в виде некоторого графа $G_S = (V_S, E_S)$, где V_S – множество вершин, каждая из которых представляет вычислительный узел, при этом в общем случае разные ВУ могут принадлежать разным суперкомпьютерным системам из состава распределенной

вычислительной среды, E_S – множество дуг, представляющих линии связи между узлами. Вершины графа взвешиваются весами p_i, ($i = 1 \ldots N_S$), обозначающими производительность i-го вычислительного узла. Пропускная способность линии связи между i-м и j-м узлами характеризуется весом m_{ij} дуги $(i, j) \in E_s$.

В каждый момент времени существует некоторый подграф $G_N = (V_N, E_N)$, где V_N – множество вершин, представляющих свободные ВУ, E_N – множество дуг, представляющих линии связи между свободными ВУ. Обозначим число свободных ВУ как $N = | V_N |$. Отметим, что состав и структура подграфа G_N постоянно изменяются в зависимости от текущей мультипрограммной ситуации. Пусть в некоторый момент из очереди поступает задание, содержащее параллельную программу из M взаимодействующих процессов. Программа может быть представлена графом $G_M = (A_M, E_M)$, где A_M – множество вершин, соответствующее процессам программы, E_M – множество дуг, представляющих информационные связи между этими процессами. Дугам графа G_M приписываются веса c_{ij}, отражающие интенсивность информационного обмена между i-м и j-м процессами. Граф G_M называется **программным** или **информационным графом**.

Задача отображения сводится в такой постановке к поиску отображения информационного графа G_M параллельной программы на структуру свободных ВУ, заданную графом G_N. Это отображение обозначается $\varphi : A_M \to V_N$ и представляется матрицей $X = \{ X_{ij} : i \in A_M, j \in V_N \}$, где $X_{ij} = 1$, если $\varphi(i) = j$, и $X_{ij} = 0$, если $\varphi(i) \neq j$. Критерием оптимальности отображения служит целевая функция $F(X)$ [40]:

$$F(X) = k_1 \sum_{l=1}^{N} \sum_{i=1}^{M} (X_{ij} a_i - M_x p_l)^2 + k_2 \sum_{i=1}^{M} \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{l=1}^{N} m_{ij} c_{lp} X_{ip} X_{jl}, \qquad (10)$$

где первый член оценивает равномерность загрузки процессоров, причем

$$M_x = \frac{\sum_{i=1}^{N} p_i}{\sum_{j=1}^{M} a_j}$$

коэффициент, характеризующий идеальную загрузку ВУ. Второй член оценивает загрузку межпроцессорных связей, при этом m_{ij} – пропускная способность линии связи между i-м и j-м ВУ; c_{ij} – объем информации, передаваемой между i-м и j-м процессами, k_1, k_2 – коэффициенты пропорциональности, отражающие баланс между вычислительной и коммуникационной составляющими суперкомпьютерной системы.

В большинстве практических случаев задача отображения решается в условиях, когда ВУ, на которые отображается параллельная программа, принадлежат одной однородной суперкомпьютерной системе, а процессы параллельной программы выполняют равные объемы вычислений. В этом случае коэффицент k_1 принимается равным 0, а целевая функция (10) принимает следующий вид:

$$F(X) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{p=1}^{M} \sum_{k=1}^{M} m_{ij} c_{kp} X_{ki} X_{pj} \longrightarrow \min \qquad (11)$$

Если в качестве весов m_{ij} используются величины, обратные пропускной способности линии связи между i-м и j-м ВУ, то целевая функция (11) будет определять время информационных обменов между процессами параллельной программы.

Решение задачи требуется обеспечить в системе управления заданиями суперкомпьютера в условиях обработки потока пользовательских заданий. При запуске очередного задания узлы для него выделяются из числа свободных на момент запуска. Поскольку состав графа G_N определяется текущей мультипрограммной ситуацией, какие конкретно узлы будут свободны при поступлении из очереди задания, заранее неизвестно. Для каждого задания требуется найти оптимальное в соответствии с (11) отображение программного графа на заранее неизвестный граф подмножества свободных узлов. Отображение должно строиться за приемлемое время, то есть быть существенно меньше времени выполнения задания (в среднем, несколько часов) и не превышать при этом времени системных таймаутов (до 5-15 минут).

4.2 Актуальные методы и средства поиска отображения информационного графа программы на граф связей вычислительных узлов

4.2.1 Обзор исследований в области поиска оптимального отображения программного графа

Задача поиска оптимального отображения является частным случаем фундаментальной математической проблемы, известной как задача о квадратичных назначениях (англ. QAP – Quadratic Assignment Problem) [192]. Многочисленные публикации представляют разнообразные подходы и алгоритмы решения задачи QAP. Подробные обзоры существующих методов и алгоритмов можно найти в [42, 43].

Для нахождения оптимального отображения могут быть применены точные алгоритмы, которые всегда находят минимальное значение целевой функции (11). К точным алгоритмам относятся полный перебор всех возможных отображений и метод ветвей и границ. Однако еще в 1976 году [41] было показано, что задача о квадратичном назначении относится к классу NP-полных задач. Использование точных алгоритмов влечет неоправданно высокие временные затраты даже для графов малых или средних порядков.

Сокращение времени до приемлемых величин возможно за счет применения приближенных параллельных алгоритмов. Эти алгоритмы в общем случае находят субоптимальное отображение, обеспечивая близкое к минимальному значение целевой функции (11). Среди приближенных алгоритмов выделяют эвристические алгоритмы, которые можно разделить на итеративные и популяционные. К итеративным относятся алгоритм поиска с запретами (поиск Табу), алгоритм жадного случайного адаптивного поиска, алгоритм имитации отжига. К популяционным алгоритмам относятся генетические алгоритмы, алгоритм оптимизации роя частиц и муравьиный алгоритм.

В работе [44] сравниваются 11 эвристик, применяемых для поиска отображения. Наиболее точное решение даёт применение генетического алгоритма. Ал-

горитм имитации отжига находит решение за самое короткое время, но при этом страдает точность отображения. Причина плохой точности в том, что на начальных этапах работы алгоритм имитации отжига может попасть в локальный минимум целевой функции (11), из которого тяжело выйти на поздних этапах.

В работе [193] авторы разработали алгоритм, основанный на муравьином алгоритме, применение которого в среднем даёт решение на 16% лучше, чем другие эвристики. Алгоритм имитации отжига почти всегда находит решение хуже, но показывает минимальное среднее время работы.

В статье [52] авторы решают проблему отображения с помощью параллельной версии алгоритма имитации отжига. Для всех узлов генерируется начальное общее решение с нулевой меткой времени. На каждом узле каждый поток генерирует решение относительно глобального решения. На каждом узле существует главный поток, который обновляет решение в рамках узла. Затем он пытается обновить общее глобальное решение. После обновления глобального решения все узлы генерируют решения относительно глобального.

В статье [55] отмечают главный недостаток алгоритма имитации отжига – рассмотрение слишком большего числа потенциальных решений. Авторы предлагают применение жадной стратегии построения начального отображения, при которой задания сортируются в соответствии со своими коммуникационными потребностями, выбирается определённый процент заданий из верхней части сортированного списка и назначается на смежные узлы, все остальные задания назначаются на узлы случайным образом. Во время генерации нового решения задания с большим числом связей стараются оставлять рядом друг с другом. Отметим, что предложенное в этой работе решение фактически нарушает иерархию уровней управления, что является недопустимым в рамках режима коллективного пользования суперкомпьютером.

В статье [194] авторы представили метод имитации отжига, который может «правильно» определять начальную и конечную температуры и количество необходимых итераций для каждого уровня температуры с целью устранения лишних

итераций оптимизации. В работе [45] представлен сравнительный анализ использования алгоритма имитации отжига в задаче отображения. Авторы приводят оптимальные функции перемещения во время генерации нового отображения, сравнивают функции принятия решения, описывают способы задания начальной и конечной температур и способ выбора количества итераций при заданных начальных параметрах алгоритма.

Авторы работы [41] отмечают, что при поиске отображения наилучший результат достигается путём применения генетического алгоритма, но время поиска отображения больше, чем у других алгоритмов. В статье [195] авторы сравнивают результаты поиска отображения при помощи генетического алгоритма и методов линейного программирования. Генетический алгоритм, в отличие от методов линейного программирования, способен находить субоптимальное решение за приемлемое время. В статье [196] производительность генетического алгоритма рассмотрена в сравнении с другими стандартными эвристиками. Отмечается, что стандартных операторов генетического алгоритма для получения прироста производительности недостаточно. С целью повышения производительности авторы используют алгоритм локального поиска – алгоритм Кернигана-Лина [197], который позволяет находить лучшие ближайшие решения. Модифицированная версия генетического алгоритма ведёт себя лучше, чем алгоритмы имитации отжига и многократного локального поиска.

В статье [198] рассмотрен параллельный генетический алгоритм, усовершенствованный с помощью алгоритма восхождения на вершины и позволяющий ограничить пространство решений. Авторы [199] разработали параллельный генетический алгоритм, который разбивает популяции между процессами. Периодически процессы обмениваются лучшими членами популяции и удаляют худшие. Для минимизации времени сетевого взаимодействия лучшими решениями обмениваются только соседние процессы, добиваясь этим сверхлинейного ускорения алгоритма.

В работе [46] авторы применяют генетический алгоритм для решения задачи отображения в системах на кристалле. Предлагают подход, в котором маршру-

тизация рассматривается как функция пригодности, в отличие от классической аналитической оценки, в которой основными показателями оптимизации являются максимальная задержка прохождения пакетов и средняя общая задержка пакетов. Авторы экспериментально сравнили предложенное решение с другими алгоритмами, продемонстрировав его эффективность почти для всех приложений. Приложения продемонстрировали прирост производительности от 2% до 30% по различным метрикам.

В статье [200] вводится модель задержки вычислений, как сумма задержки обработки данных и задержки на коммуникации. На предложенную модель устанавливаются следующие ограничения: данные передаются только по кратчайшему пути, с любым узлом в произвольный момент времени взаимодействует только один другой узел. В ходе выполнения алгоритма сначала минимизируется общая задержка, затем происходит назначение заданий на вычислительные узлы. Для решения задачи используются операторы скрещивания и мутации.

В работах [53, 201, 202] авторы описывают наиболее часто используемую при реализации параллельного генетического алгоритма схему «Мастер-Рабочие». «Мастер» отвечает за генерацию новых членов популяции. Все остальные вычислители «Рабочие» рассчитывают значение целевой функции для новых членов популяции. В этих работах авторы рассматривают параллельный генетический алгоритм, основанный на схеме кольцевого обмена. На каждой итерации кольцевого обмена различные популяции обмениваются лучшими членами, что позволяет передавать лучшие признаки между популяциями. Количество передаваемых членов популяции не должно быть большим, так как это может привести к максимальной схожести популяций на начальных итерациях.

В работе [47] предлагается эвристический алгоритм для распределения процессов MPI-программ по ядрам процессора с целью минимизации общего времени информационных обменов. Алгоритм показал снижение времени выполнения MPI-программ на вычислительных кластерах, использующих коммуникационную сеть «Ангара».

В [48] для решения задачи о квадратичном назначении авторами предложен гибридный алгоритм, основанный на комбинировании генетических и эволюционных алгоритмов. Разработанный гибридный алгоритм продемонстрировал более высокую точность по сравнению с другими эвристическими подходами для графов небольшого размера (порядок графов 12-25), при этом обеспечивается отклонение точности от известных оптимальных решений не более чем на 4,2%. Отметим, что авторы не привели в статье данных о времени выполнения алгоритма.

В исследовании [49] авторы использовали параллельный генетический алгоритм с некоторыми модификациями для решения задачи о назначениях на гибридной вычислительной системе с применением графических ускорителей. Авторы обнаружили, что время работы алгоритма на ГПУ не сильно зависит от увеличения размера популяции или размера задачи по сравнению со временем на универсальном процессоре. Авторы протестировали свой алгоритм на максимальном размере задачи 70, время решения которой составило около 120 секунд.

Использование параллелизма позволяет одновременно повысить скорость и точность отображения. В статье [50] авторы предлагают алгоритм, основанный на параллельном алгоритме распространения меток (LPA). Алгоритм решает задачи разбиения и отображения одновременно. Авторы рассмотрели большие графы с миллионами вершин, значительно уменьшили их и отобразили графы на блоки из нескольких сотен измерений. Кроме того, этот алгоритм учитывает специфику поиска отображения для иерархических архитектур, в первую очередь для неявной древовидной топологии.

В работе [51] авторы использовали комбинацию генетического алгоритма и оптимизации локальных решений. Авторы провели эксперимент, включавший проверку метода на различных наборах графов. Стоит отметить, что даже на средних наборах из сотен вершин время поиска решения составляет сотни секунд.

Исследователи [203] отмечают, что качество решений, полученных с помощью методов машинного обучения, пока неконкурентоспособно по сравнению с современными метаэвристиками. По мнению авторов, пройдет достаточно про-

должительное время, прежде чем методы поиска отображения, основанные на машинном обучении, превзойдут традиционные методы оптимизации.

В работе [54] авторы используют улучшенный гибридный алгоритм, который включает генетический алгоритм и алгоритм имитации отжига. Авторы продемонстрировали результаты комбинированного алгоритма, которые превосходят чистые алгоритмы отжига или генетического отбора.

Анализ научных публикаций по теме поиска оптимального отображения программного графа на граф вычислительной системы позволяет сделать следующие выводы.

1. Большинство авторов отмечают экспоненциальную вычислительную сложность точных алгоритмов, которая делает последние неприменимыми для оперативного поиска отображения. В то же время, эвристические подходы позволяют добиться высокой скорости поиска при достаточной точности находимого субоптимального решения.

2. Среди эвристических алгоритмов наиболее широкое распространение получили алгоритмы имитации отжига и генетического отбора. В большинстве публикаций отмечается, что генетический алгоритм превосходит остальные эвристики по точности отображения, в то время как алгоритм имитации отжига является одним из самых быстрых алгоритмов.

3. Результаты многих исследований подтверждают целесообразность применения параллельных версий алгоритмов поиска отображения.

4. В последние годы многие исследователи сосредотачивают свои усилия на комбинировании различных базовых эвристик с целью улучшить характеристики (точность и скорость поиска отображения) комбинируемых алгоритмов.

В этой связи рассмотрим общие схемы наиболее часто применяемых эвристик – имитации отжига и генетического отбора.

4.2.2 Алгоритмы имитации отжига и генетического отбора

Широкое распространение при решении оптимизационных задач получил эвристический метод имитации отжига [204], который относится к классу стохастических методов оптимизации [205]. Метод некоторым образом имитирует процесс отжига металла, заключающийся в нагревании и контролируемом охлаждении металла. Техника отжига в металлургии позволяет увеличить прочность кристаллической решетки металла и повысить его качественные характеристики. При охлаждении жидкого металла переход термодинамической системы из состояния с энергией E_1 в состояние с энергией E_2 происходит с вероятностью, рассчитываемой исходя из распределения Больцмана-Гиббса:

$$p = e^{\frac{-(E_2 - E_1)}{kT}}$$

где k – постоянная Больцмана, а T – температура.

В задачах оптимизации аналогом энергии является целевая функция, а «температура отжига» вступает в роли безразмерного параметра в формуле вероятности перехода. Для выхода из областей притяжения локальных минимумов с этой вероятностью разрешается переход в точки с худшим значением целевой функции. Это позволяет «проскочить» локальные минимумы оптимизируемой функции и попасть в область притяжения глобального минимума. Попадание в эту область происходит с определенной вероятностью, поэтому метод имитации отжига не гарантирует достижения глобального минимума целевой функции. Тем не менее, отмечается [205], что при правильной стратегии выбора траектории изменения «температуры отжига» T происходит не только значимое улучшение начального решения, но существенное приближение к глобальному экстремуму. К нахождению оптимальной стратегии управления траекторией «температуры отжига» сводятся в конечном итоге все решаемые при помощи метода имитации отжига оптимизационные задачи. Важное значение при этом имеют удачное начальное приближение, а также выбор начальной, достаточно высокой «температуры отжига», которая будет понижаться с каждой итерацией алгоритма.

Основные шаги алгоритма имитации отжига для нахождения минимума целевой функции (11) следующие.

1. Генерируется стартовое исходное решение, которое становится текущим.

2. Генерируется новое решение путём перестановки двух произвольных элементов матрицы *X* местами.

3. Если в результате перестановки приращение значения целевой функции (11) $\Delta F(X) < 0$, то новое решение становится текущим. Если $\Delta F(X) > 0$, то новое решение становится текущим с вероятностью, определяемой функцией принятия решения. Функция принятия решения зависит от текущей температуры отжига, понижающейся в процессе работы алгоритма.

4. Температура системы понижается в соответствии с видом функции понижения температуры.

5. Алгоритм останавливается по достижении одного из значений: определенного числа итераций, финальной температуры системы, числа подряд идущих итераций без улучшения значения целевой функции. Иначе – переход к п.2.

Эффективность (скорость и точность) метода зависят от начальной температуры, предельного числа итераций (конечной температуры), размера области сходимости (числа последовательных итераций, в течение которых значение целевой остается неизменным). При увеличении начальной температуры и области сходимости возрастает точность, но падает быстродействие алгоритма.

При применении алгоритма генетического отбора [205] каждый вариант входной матрицы *X*, подаваемой на вход целевой функции (11), представляется в виде вектора (массива) *P*, *i*-й элемент этого массива содержит номер ВУ, на который будет назначена *i*-я ветвь (*i*-й процесс) параллельной программы. Массив *P* в терминах генетического алгоритма называется особью, а каждый элемент массива *P* представляется как ген конкретной особи. Особи составляют популяцию, размер которой равен или больше числа вершин программного графа.

Над популяцией и особями в ней производят генетические операции. Операция мутации случайным образом изменяет один или несколько генов у одной

или нескольких особей популяции. Операция скрещивания заключается в обмене генами между двумя особями популяции. Операция селекции выбирает из популяции особи с наименьшим значением целевой функции (11).

Суть генетического алгоритма заключается в итеративном повторении операций мутации, скрещивания и селекции с постепенным улучшением значения целевой функции (11). После очередной операции селекции худшие особи популяции отбрасываются, и их места занимают новые особи, которые формируются путем выполнения операций мутации и скрещивания над выбранными в результате селекции лучшими особями. Алгоритм завершает работу либо по истечении определенного числа итераций, либо при отсутствии улучшения значения целевой функции в течение определенного числа итераций.

Для оценки эффективности эвристических алгоритмов поиска отображения можно использовать два показателя: точность A как процент совпадений с оптимальным решением и среднее отклонение D от оптимального значения целевой функции (11). Допустим, что нам заранее известно некоторое обеспечивающее минимальное значение функции (11) отображение $\varphi_{optimum}$ информационного графа G_p параллельной программы на граф G_N свободных ВУ. Пусть отображение $\varphi_{optimum}$ представляется матрицей $X_{optimum} = \{ X_{ij} : i \in A_M, j \in G_N \}$, где $X_{ij} = 1$, если $\varphi_{optimum}(i) = j$, и $X_{ij} = 0$, если $\varphi_{optimum}(i) \neq j$. Из-за стохастического характера эвристических алгоритмов при каждом запуске на выходе будут получаться разные решения даже на одних и тех же входных данных. Пусть некоторый алгоритм был выполнен K раз, и каждый i-й раз на выходе алгоритма получалось отображение φ_i, представляемое матрицей X_i. Обозначим как b_i величину, принимающую значение 1, если $X_{optimum} = X_i$, и 0, если $X_{optimum} \neq X_i$. Тогда точность A алгоритма отображения можно оценить как

$$A = \frac{\sum_{i=1}^{K} b_i}{K} \times 100\% \qquad (12)$$

Пусть для каждого отображения φ_i значение целевой функции (11) равно F_i, а минимальное значение целевой функции (11) равно F_{min}.Тогда среднее отклонение D целевой функции (11) можно представить как

$$D = \frac{\sum_{i=1}^{K} (F_i - F_{\min})}{K \cdot F_{\min}} \times 100\% \qquad (13)$$

Следует отметить, что показатели A и D далеко не всегда коррелируют друг с другом. Например, алгоритм может каждый раз находить очень близкое к оптимальному решение и обеспечивать тем самым малое значение D, но при этом очень редко находить оптимальное отображение и иметь низкий показатель точности A. Другой алгоритм может иметь высокий процент A совпадений с оптимальным решением, но находимые алгоритмом субоптимальные решения будут иметь существенное превышение значения целевой функции (11) над её минимумом.

4.3 Метод отображения параллельной программы на вычислительные узлы суперкомпьютера

4.3.1 Общая схема и этапы метода отображения

Предположим, что информационные графы параллельных программ из состава поступающих заданий произвольны, топология вычислителя известна, и производительность всех ВУ решающего поля одинакова. Для каждой отдельной программы задача распределения ее процессов по свободным ВУ сводится к поиску такого отображения, при котором минимизируется значение целевой функции (11). Очевидно, в этом случае одновременно будут решены две подзадачи управления вычислительными ресурсами:

– подзадача 1: выбор и выделение требуемых для выполнения параллельной программы вычислительных узлов из числа свободных;

– подзадача 2: назначение процессов (ветвей) параллельной программы на вычислительные узлы.

При одновременном решении этих подзадач фактически решается сразу подзадача 2, т.е. выделенными для параллельной программы считаются все свободные ВУ системы. В результате отображения происходит назначение ветвей программы на определённые ВУ. Эти ВУ становятся занятыми, а свободными остаются те узлы, на которые не была назначена ни одна ветвь параллельной программы. Такой подход обладает рядом недостатков.

Во-первых, размерность задачи поиска отображения будет определяться числом свободных ВУ, которых почти всегда больше, чем число требуемых ВУ для задания. Учитывая экспоненциальную сложность поиска отображения, увеличение размера задачи крайне нежелательно. В некоторой мере компенсировать возросший размер задачи возможно за счет применения параллельных эвристических алгоритмов поиска отображения, для выполнения которых можно использовать свободные ВУ. Однако, следует отметить, что во время работы параллельного алгоритма поиска отображения будет невозможен запуск других заданий, поскольку в это время все свободные ВУ будут заняты выполнением алгоритма поиска отображения.

Во-вторых, нахождение оптимального решения (11) для одной отдельно взятой программы совершенно не означает, что для программ из состава следующих заданий удастся произвести хорошее отображение на оставшиеся ВУ. Возьмем для примера топологию вычислителя МВС-1000, представленную на рисунке 7. Пусть на выходе очереди у нас имеются два задания, требующих 24 и 8 ВУ соответственно. Допустим, для первого задания было найдено отображение с минимальным значением целевой функции (11). На рисунке 48 ВУ, на которые распределено первое задание, отмечены цветом. Очевидно, что второе задание получит для выполнения «плохое», слабо связанное подмножество ВУ.

Контрпример, представленный на рисунке 48, иллюстрирует факт, что при поиске отображения необходимо обеспечить сокращение как времени выполнения отдельно взятого задания, так и суммарного времени выполнения всего потока задания. Другими словами, хотелось бы добиться по возможности минималь-

ного суммарного времени исполнения заданий, но без «ущемления прав» какой-либо отдельно взятого задания.

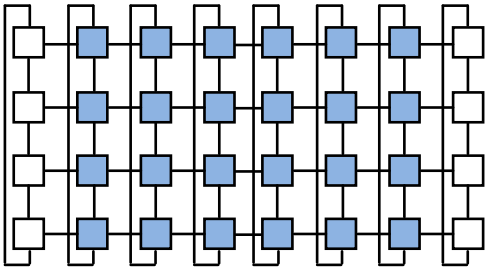

Рисунок 48. Неоптимальное распределение заданий по ВУ

Альтернативой является последовательное решение подзадач 1-2, при котором сначала производится выбор требуемого параллельной программе количества ВУ из числа свободных, а затем решается задача назначения ветвей программы на ВУ из выделенного подмножества.

При решении подзадачи 1 нам необходимо выделить из подграфа G_N некоторый подграф $G_{job} = (V_{job}, E_{job})$, где множество вершин $V_{job} \subseteq V_{free}, |V_{job}| = M$, представляет собой подмножество ВУ, выделенных для задания, а множество дуг E_{job} представляет линии связи между выделенными ВУ. Напомним, что в режиме коллективного пользования СУЗ обрабатывает поток поступающих заданий, и необходимо, чтобы выделенный для очередного задания подграф G_{job} обеспечивал минимизацию как времени исполнения очередного поступившего задания, так и суммарного времени исполнения всего потока задания.

В условиях выполнения требования универсальности СУЗ не вправе требовать от пользователя обязательного задания информационного графа параллельной программы. В этих условиях при выборе вычислительных узлов для очередного задания СУЗ должна считать информационный граф параллельной программы неизвестным, т.е. произвольным. В этом случае весьма разумным выглядит следующее предположение. Из всех возможных выборок подграфа G_{job} из графа G_{free} лучшей будет выборка с наибольшим количеством внутренних межузловых связей E_{job}. Учитывая, что поступившее задание – не последнее, и для последующих заданий в свою очередь тоже понадобится возможно большее число межузловых связей, выбор ВУ для очередного задания можно свести к разбиению

множества свободных ВУ на два подмножества из M и N-M узлов с минимальным количеством связей между этими подмножествами. Получаем известную задачу о разбиении графа на подграфы с наименьшим числом связей между этими подграфами. Предлагаемое автором решение этой задачи рассмотрено в п. 4.3.2

Выбор ВУ для очередного задания методом разрезания графа свободных ВУ на наименее связанные между собой подграфы обладает следующими преимуществами. Во-первых, каждому заданию предоставляется подмножество максимально связных ВУ, что само по себе обеспечивает улучшение значения целевой функции (11). Во-вторых, обеспечивается сокращение размера подзадачи 2 до числа ветвей параллельной программы, что поможет сократить время решения этой подзадачи.

Для решения подзадачи 2 предлагается применить параллельный эвристический алгоритм, для выполнения которого использовать подмножество выделенных для задания ВУ. В этом случае не только сокращается время поиска отображения, но одновременно СУЗ может продолжить распределять последующие задания на оставшиеся свободные в системе вычислительные узлы. Предлагаемые автором параллельные эвристические алгоритмы решения подзадачи 2 рассмотрены ниже по тексту настоящей главы.

В итоге суть предлагаемого автором метода отображения параллельной программы на вычислительные узлы суперкомпьютерной системы коллективного пользования состоит в следующем. Отображение очередного задания производится в два этапа. На первом этапе методом разрезания графа свободных ВУ на два минимально связанных между собой подграфа производятся выбор и выделение ВУ для задания. На втором этапе подмножество выделенных ВУ используется для выполнения параллельного алгоритма поиска оптимального отображения программного графа на граф топологии подмножества выделенных ВУ.

4.3.2 Выбор вычислительных узлов для очередного задания

Выбор вычислительных узлов для очередного задания будем производить путем решения задачи разбиения графа свободных ВУ на подграфы в следующей постановке.

Имеется неориентированный помеченный граф со взвешенными ребрами $G = (X, U, C, A)$, где $X = \{ x_i \}$ – множество вершин, $U = \{ u_{ij} \}$ - множество ребер, $C = \{ c_i \}$ – множество весов вершин, $A = \{ a_{ij} \}$ – множество весов ребер. Поскольку ВУ в общем случае неоднородны, под весом вершины можно понимать, например, число процессорных ядер на ВУ. Необходимо разбить G на k подграфов $G_i = (X_i, U_i, C_i, A_i)$, таких, что

$$\bigcup_{i=1}^{k} x_i = X \, , x_i \bigcap_{i \neq j} x_j = \varnothing \, ,$$

таким образом, чтобы сумма весов ребер, связывающих эти подграфы, была минимальной, а сумма весов вершин, входящих в подграф G_i, не превышала D_i, другими словами, обеспечивала бы необходимое для задания число процессорных ядер во множестве выделенных для задания ВУ Сумма весов ребер, связывающих вершины из различных подграфов называется **связностью разрезания**. Множество ребер, соединяющих вершины из различных подграфов, называют **разрезом графа**. Постановка задачи может варьироваться в зависимости от свойств графа, которые задаются моделируемым объектом.

Обзор различных решений этой задачи представлен в работе [206], где приводится классификация имеющихся алгоритмов, и дан анализ их эффективности в зависимости от размера и вида разрезаемого графа. По размерам графы классифицируются на малые (до 6 вершин), средние (от 6 до 30 вершин) и большие (свыше 30 вершин). Алгоритмы разрезания делятся в свою очередь на точные (дающие точное решение задачи) и эвристические (дающее приближенное решение задачи, но за относительно малое время).

Среди точных алгоритмов выделяются метод ветвей и границ, впервые рассмотренный в [207], и его модифицированный и более эффективный вари-

ант [208]. Эти алгоритмы основаны на упорядоченном переборе всех возможных вариантов и не гарантируют того, что во время решения задачи не будет произведен полный перебор. В обзоре [206] отмечается целесообразность применения этих алгоритмов на графах не более средней размерности.

В общем случае задача разрезания графа на два минимально связанных между собой подграфа является NP-полной. Например, если предположить, что СУЗ управляет суперкомпьютерной системой, состоящей 1024 ВУ, то в этом случае максимальное число вариантов выбора свободных ВУ составляет около 10^{320}, что делает точные алгоритмы неприменимыми.

Среди эвристических алгоритмов наиболее быстрым и простым для реализации является алгоритм, рассмотренный в [209]. Его рандоминизированный вариант представлен в [210]. Эвристические алгоритмы дают приближенное решение задачи, зато требуют значительно меньше времени для своего выполнения, что дает возможность их применения на графах большой размерности.

Рассмотрим частный случай сформулированной задачи, когда веса всех вершин графа одинаковы и равны 1, т.е. мы имеем дело с однородными ВУ. Задача выбора M ВУ из N свободных, как уже говорилось, сводится к разбиению множества N свободных ВУ на два подмножества из M и из N-M ВУ с минимальным количеством связей между этими подмножествами. Очевидно, это эквивалентно разрезанию графа $G_N = (X_N, U_N, A_N)$ свободных ВУ на два подграфа, $G_1 = (X_1, U_1, A_1)$ и $G_2 = (X_2, U_2, A_2)$, $|X_1| = M$, $|X_2| = N - M$, причем

$$X_1 \bigcup X_2 = X, X_1 \bigcap X_2 = \varnothing,$$

таким образом, чтобы сумма весов ребер, попавших в разрез графа G_N, была минимальной.

Учитывая большую размерность разрезаемого графа, для разбиения графа G_N можно применить следующий эвристический алгоритм, идея которого описана в [209].

В начале множества X_1 и X_2 выбираются произвольным образом, лишь бы выполнялись условия задачи. Вычисляется связность начального разрезания q_0.

Пусть некоторая вершина $x_i \in X$ попала в подграф G_j. Внутренней связностью вершины x_i будем называть величину, равную сумме весов всех ребер, выходящих из вершины x_i и соединяющих ее с вершинами подграфа G_j. Соответственно, внешней связностью вершины x_i будем называть величину, равную сумме весов всех ребер, выходящих из вершины x_i и соединяющих ее с вершинами противоположного подграфа $G_N \backslash G_j$.

На каждом шаге алгоритма для всех вершин, составляющих подграфы G_1 и G_2, вычисляются значения внутренней и внешней связности. В каждом подграфе выделяется вершина с наибольшей разностью между внутренней и внешней связностью. Далее осуществляется обмен выделенными вершинами между подграфами, и вычисляется связность q_k полученного нового разрезания, где k – номер шага. Переход к следующему шагу осуществляется, если значение q_k не больше значения q_{k-1}, полученного на предыдущем шаге алгоритма, или в подграфах G_1 и G_2 еще остаются нерассмотренные вершины. В противном случае алгоритм завершает работу.

Для того чтобы в ходе работы алгоритма не «ухудшить» характеристики текущего решения, была введена следующая эвристика. На каждом шаге вычисляется связность выделенного подграфа как сумма весов дуг графа G_1, целиком лежащих внутри. Максимальное значение запоминается и в конце работы алгоритма сравнивается с найденным на последнем шаге разрезанием. Лучшим признается разрезание с максимальным значением связности.

Рассмотренный алгоритм достаточно быстр и прост в реализации, каждая итерация имеет сложность, не более чем квадратичную. Однако, будучи реализованным в составе СУППЗ для МВС-1000, алгоритм показал невозможность нахождения точного решения задачи разрезания графа даже при небольших размерах выделяемого подграфа G_{job}, продемонстрировав при этом сильную зависимость от начального разрезания.

Для улучшения характеристик алгоритма [209] автором диссертации для нахождения оптимального разрезания графа было предложено [211] применить

метод имитации отжига. Соответствующий алгоритм получил название РГО (Разрезание Графа Отжигом). Алгоритм РГО, так же, как и алгоритм [209], заключается в последовательности взаимных перестановок вершин из X_1 и X_2, только переставляемые вершины выбираются случайно. Если связность разрезания уменьшается, то перестановка фиксируется, а если нет – то фиксируется с вероятностью $e^{-\Delta q/T}$, где Δq – приращение связности, а T – текущая температура отжига, понижающаяся в ходе работы алгоритма по закону $T = \frac{T_0}{t}$, где T_0 – начальная температура отжига, а t – текущая итерация алгоритма.

Экспериментально было определено оптимальное значение начальной температуры, равное 2000º. В ходе трассировки алгоритма было установлено, что на заключительных фазах отжига, когда текущая температура становилась менее 1°, происходит циклический перебор близко расположенных вершин графов G_1 и G_2. При этом каждое новое разрезание оказывается очень близко к оптимальному, не на «своих» местах стоят 2-4 вершины исходного графа G_N, однако алгоритм не может выйти на оптимальное решение. Для преодоления выявленного недостатка была предложена следующая эвристика. До температуры отжига выше 1° выполняется имитация отжига, при достижении температуры 1° осуществляется переход к алгоритму [209], который за несколько итераций достигает оптимального решения.

4.3.3 Поиск отображения программного графа на граф связей выделенных для задания ВУ

Рассмотрим вторую подзадачу – распределение ветвей (процессов) параллельной программы по выделенным ВУ. Автором для решения которой автором был предложен параллельный алгоритм имитации отжига с генетическими операциями (ПОГ – Параллельный Отжиг и Генетика) [211].

Алгоритм ПОГ выполняется несколькими процессами, среди которых выделяются мастер и процессы-рабочие. Начальная популяция заполняется случайно выбранными особями, которые распределяются между процессами-рабочими.

Каждый из процессов-рабочих осуществляет над своей частью популяции следующий алгоритм имитации отжига.

1. Полагаем начальную температуру равной T_0 и значение счетчика итераций t равным 1.

2. В качестве начального отображения X выбираем переданное процессом-мастером решение и вычисляем целевую функцию (11).

3. На очередной итерации случайным образом выбираем одну из вершин (i) программного графа. Поочередно перебираем оставшиеся вершины программного графа. Меняем местами выбранную вершину i и перебираемые вершины. Вычисляем приращение целевой функции $\Delta F(X)$ при обмене местами выбранной вершины i и очередной рассматриваемой нами вершины j. Если $\Delta F(X) < 0$, то замена закрепляется, и осуществляется переход к новой итерации. Если $\Delta F(X) > 0$, то замена закрепляется с вероятностью $e^{-\frac{\Delta F}{T}}$, где T – текущая температура отжига, и осуществляется переход к новой итерации. Если закрепления не происходит, осуществляется выбор новой, $(j + 1)$-й, вершины программного графа и ее обмен местами с выбранной в начале итерации вершиной i.

4. Понижаем температуру по закону $T = \frac{T_0}{t}$.

5. По истечении определенного числа итераций, либо если зафиксирован выход целевой функции $F(X)$ на стационарное значение, то завершаем алгоритм, в противном случае переходим к п. 3.

Результаты каждого процесса-рабочего собираются в процессе-мастере, который осуществляет над популяцией операции селекции, отбраковывает худшие особи и путем скрещивания формирует новые особи, которые вместе с отобранными в результате селекции особями образуют новую популяцию. Новая популяция распределяется по процессам-рабочим, и начинается очередной шаг алгоритма. Алгоритм останавливается, если по истечении заданного числа шагов не произошло улучшения целевой функции (11).

Первоначальная версия алгоритма не содержала генетических операций, каждый процесс-рабочий производил определенное число итераций, после чего

отправлял результат процессу-мастеру. Последний выбирал лучшее решение и рассылал его рабочим в качестве начального решения следующего шага. Начальное значение температуры отжига составляло 2000°, конечной температуры – 0,1°, предельное число итераций, производимых за один шаг процессом-рабочим, – 20000, область сходимости – 30 шагов. К сожалению, параллельный алгоритм имитации отжига с указанными параметрами для графов порядка 32 обеспечивал точность *A* в соответствии с (12) не более 8% при среднем отклонении *D* целевой функции, определяемом в соответствии с (13), равном 40%.

Таблица 12 отражает ход исследования и модификации параллельного алгоритма. Алгоритм выполнялся на 32-процессорном суперкомпьютере МВС-1000, который был установлен в ИПМ им. М.В. Келдыша РАН и производил отображение программного графа с топологией «решетка» 4×8 на граф ВУ с такой же топологией. При запуске алгоритма каждый процесс-рабочий выполнялся на отдельном процессоре.

Таблица 12. Процесс добавления генетических операций в параллельный алгоритм имитации отжига

Тест	Число процессов-рабочих	Среднее время, с	Среднее отклонение *D*	Точность *A*
1	15	4,3	40%	8%
2	31	5,7	34%	12%
3	31	15,8	26%	28%
4	31	29,9	22%	32%
5	23	6,9	20%	35%
6	23	7,1	10%	50%
7	23	6,7	5%	3%

Тесты №1 и №2 заключались в выполнении параллельного алгоритма имитации отжига разным числом процессов-рабочих. Тест №2 за примерно то же время перебрал примерно в два раза больше потенциальных решений, что положительно сказалось на среднем отклонении *D* точности *A*. По этой причине было

решено сконструировать и выполнить тесты №3 и №4, в каждом из которых параллельный алгоритм имитации отжига повторялся некоторое количество шагов, при этом полученное на *i*-м шаге решение подавалось в качестве начального на вход *(i+1)*-го шага. Тест №3 состоял из 16 шагов, а тест №4 – из 32 шагов. Из таблицы 12 видно, что такой экстенсивный подход привел только росту времени работы алгоритма, существенно не повлияв на точность и среднее отклонение.

Тесты №5 и №6 демонстрируют постепенный переход к алгоритму ПОГ. В тесте №5 была добавлена операция селекции, а в тесте №6 – дополнительно к селекции операция скрещивания. Таблица 12 показывает, что каждая генетическая операция внесла ощутимый вклад в улучшение показателей точности и среднего отклонения.

Тест №7 демонстрирует отображение алгоритмом ПОГ программного графа из 32 вершин в топологии «кольцо» на граф ВУ топологии «решетка» 4×8. Алгоритм ПОГ продемонстрировал малое среднее отклонение от минимума целевой функции, но при этом практически никогда не находил оптимальное отображение.

4.3.4 Результаты применения метода в суперкомпьютерах серии МВС-1000

Алгоритмы РГО и ПОГ были реализованы в составе сервера запуска СУППЗ для суперкомпьютеров серии МВС-1000. Апробация алгоритмов была произведена при помощи известного стандартного набора тестов NPB (NAS Parallel Benchmark) [212] версии 2.3, которые выполнялись 32-процессорной системе МВС-1000, установленной в 1999 году в ИПМ им. М.В. Келдыша РАН. Каждый тест NPB помимо времени своей работы выдает оценку производительности *P* суперкомпьютера в миллионах операций в секунду (Мопс). Каждый тест NPB выполнялся без применения алгоритмов РГО и ПОГ и с применением этих алгоритмов. Если обозначить производительность, оцененную тестами без применения алгоритмов отображения, как P_1, а производительность, оцененную с применением алгоритмов, как P_2, то прирост производительности *W* можно определить как

$$W = \frac{P_2 - P_1}{P_1} \times 100\%$$

Таблица 13 демонстрирует результаты применения алгоритма РГО. Запуск без применения алгоритма РГО означал выделение ВУ подряд в порядке их нумерации.

Таблица 13. Результаты выполнения тестов NPB на суперкомпьютере МВС-1000 без применения и с применением алгоритма РГО

Тест NPB	Число процессоров	Выделение ВУ подряд		Выделение по алгоритму РГО		Прирост производительности W, %
		Время, с	Производ. P_1, Мопс	Время, с	Производ. P_2, Мопс	
SP	4	954	89,1	880,2	101,1	13,5
SP	9	880,7	96,5	650,3	136,9	41,9
SP	16	680,2	130,9	540,8	164,6	25,7
LU	4	747,4	159,6	692,6	172,2	7,9
LU	8	651,6	183,1	497,5	239,6	23,6
LU	16	856,8	139,2	531	224,7	61,4
BT	4	1073,8	156,7	1028,7	163,6	4,4
BT	9	698,8	240,8	581,6	289,4	20,2
BT	16	472	356,5	415	405,5	13,7
EP	4	68,2	7,9	68,1	7,9	0
EP	8	34,2	15,7	34,1	15,7	0
EP	16	17,2	31,2	17,1	31,3	0
CG	4	38,6	38,8	38,1	39,3	1,3
CG	8	46	32,5	41,3	36,2	11,4
CG	16	59,5	25,1	48,8	30,7	22,3
MG	8	34,5	112,8	29,5	132	17
MG	16	22,1	176,2	18,2	213,9	21,4
FT	8	128,6	55,5	101,8	70,1	26,3
FT	16	73,6	97	63,4	112,6	16,1
IS	4	35,7	2,3	27,3	3,1	34,8
IS	8	40,1	2,1	28,5	2,9	38,1
IS	16	27,3	3,1	21,5	3,9	25,8

Усредненный по всем тестам прирост производительности от применения алгоритма РГО в суперкомпьютере МВС-1000 представлен в таблице 14. Максимальный прирост достиг 61,4% на тесте LU на 16 процессорах.

Алгоритм ПОГ при решении задачи отображения размером 32 показал 50% совпадений с оптимальным отображением при среднем отклонении от оптимального значения целевой функции около 10%, что существенно превосходило из-

вестные на момент 1999 года результаты. Таблица 15 демонстрирует результаты применения алгоритма ПОГ. Тест EP не выполнялся, поскольку он не содержит информационных обменов, т.е. у теста фактически отсутствует программный граф.

Таблица 14. Средний по всем тестам NPB прирост производительности от применения алгоритма РГО в суперкомпьютере МВС-1000

Число процессоров (ВУ)	4	8 (9)	16
Прирост производительности W	10,1%	22,3%	23,3%

Таблица 15. Результаты выполнения тестов NPB на суперкомпьютере МВС-1000 без применения и с применением алгоритма ПОГ

Тест NPB	Число процессоров	Без применения алгоритма ПОГ		С применением алгоритма ПОГ		Прирост производительности W, %
		Время, с	Производ. P_1, Мопс	Время, с	Производ. P_2, Мопс	
SP	4	444	191,4	444,1	191,3	0
SP	9	316,4	268,7	289,9	293,2	9,1
SP	16	341	249,3	239,8	354,6	42,2
SP	25	305,8	278	206,2	412,3	48,3
LU	4	367,5	324,5	367,4	324,6	0
LU	8	311,6	382,9	266,5	447	16,7
LU	16	340,9	350	233,7	510,4	45,8
LU	32	277,7	430	234,4	509,5	18,5
BT	4	455,7	369,3	455,4	369,6	0
BT	9	251,2	700	242	670	4,5
BT	16	225,6	845,8	189,9	886	18,8
BT	25	195,4	861,3	139,2	1208,7	40,3
CG	4	18,9	79	18,8	79,1	0
CG	8	19,9	75,4	13,2	113,5	50,5
CG	16	22,5	66,4	17,4	85,8	29,2
CG	32	23,5	63,7	14,9	100,3	57,5
MG	8	13,2	294,5	9,7	401	36,1
MG	16	10	391,1	7,9	492,3	25,9
MG	32	7,0	556,1	5,3	734,5	32

Таблица 16 отражает усредненный по всем тестам NPB прирост производительности от применения алгоритма ПОГ в зависимости от числа процессоров. Максимальный прирост достиг 57,5% на тесте CG на 32 процессорах.

Таблица 16. Средний по всем тестам NPB прирост производительности от применения алгоритма ПОГ в суперкомпьютере МВС-1000

Число процессоров (ВУ)	4	8 (9)	16	25	32
Прирост производительности W	0%	23,4%	32,4%	44,3%	36%

Апробация алгоритма ПОГ производилась также и на реальных задачах. Таблица 17 отражает результаты применения алгоритма ПОГ для решения задачи моделирования процессов структурообразования рибонуклеиновых кислот программным комплексом ГЕН-2 [213]. Прирост производительности составил от 37,1% до 47,5% в зависимости от числа процессоров.

Таблица 17. Результаты выполнения расчетов программным комплексом ГЕН-2 на суперкомпьютере МВС-1000 без применения и с применением алгоритма ПОГ

Число процессоров	4	8	16	24	29
Время выполнения расчетов без применения алгоритма, секунды	145	566	1240	1831	1217
Время выполнения расчетов с применением алгоритма ПОГ, секунды	146	397	884	1336	825
Ускорение, %	0	29,9	28,7	27	32,2
Прирост производительности, %	0	42,6	30,3	37,1	47,5

Во многом хорошие показатели роста скорости расчетов при применении предложенного метода отображения были обусловлены некоторым дисбалансом вычислительной и коммуникационной составляющих суперкомпьютеров МВС-1000. Коммуникационная среда МВС-1000, показанная на рисунке 7, представляет собой распределенный составной коммутатор с отсутствием прямых связей между несмежными ВУ. За счет этого при информационных обменах вычислительных процессоров несмежных ВУ приходилось осуществлять промежуточные передачи данных между связными процессорами. Это крайне отрицательно сказывалось на показателях латентности и пропускной способности коммуникационной

среды и приводило к указанному дисбалансу с вычислительными процессорами. В этих условиях применение предложенного автором метода отображения параллельных программ на структуру вычислительных узлов суперкомпьютера позволило существенно поднять скорость производимых расчетов. Отметим, что заметный прирост производительности наблюдался даже при применении алгоритма РГО отдельно от алгоритма ПОГ. Последняя ситуация возникала в том случае, когда пользователь по разным причинам не мог задать информационный граф своей параллельной программы, что наблюдалось для большинства заданий.

4.4 Развитие метода отображения параллельной программы на вычислительные узлы суперкомпьютера

4.4.1 Программные средства и инструменты для поиска отображения

С момента реализации алгоритмов РГО и ПОГ прошло значительное время, и появились новые программные инструменты решения задачи отображения. Например, для рекурсивного разбиения графа может быть использована библиотека Scotch [214], которая активно применяется исследователями. Другой известной библиотекой является LibTopoMap [215], в ней реализован ряд алгоритмов отображения, в том числе жадный алгоритм поиска самого загруженного процесса в коммуникационном плане и вычислительного элемента, имеющего наибольшую коммуникационную связь, а также алгоритм Катхилла-Макки, позволяющий уменьшить пропускную способность разреженной матрицы расстояний. Для улучшения результатов отображения используются алгоритмы имитации отжига и алгоритм восхождения на вершину. Еще одним известным инструментом является MPIPP [216], который позволяет осуществлять автоматический поиск оптимизированного отображения с целью минимизации затрат на связь типа «точка-точка» для приложений с произвольным характером обмена сообщениями.

При развитии алгоритма ПОГ в качестве основы была использована свободно распространяемая библиотека UGR-Metaheuristics [217]. Библиотека UGR-Metaheuristics написана на языке программирования C++ и реализует большин-

ство известных эвристических алгоритмов поиска отображения. Библиотека содержит в своем составе ряд базовых алгоритмов отображения, в том числе параллельных алгоритмов генетического отбора и имитации отжига. Базовые алгоритмы образуют иерархию классов, допускающую наследование и добавление новых классов. Это позволяет разрабатывать новые алгоритмы отображения в том числе путем модификации базовых алгоритмов библиотеки UGR-Metaheuristics.

Для целей развития и исследования характеристик алгоритма ПОГ под руководством автора было разработано программное средство GraphHunter [218], основанное на библиотеке UGR-Metaheuristics. Входными данными для программного средства GraphHunter являются программный граф и граф подмножества ВУ суперкомпьютера. При помощи GraphHunter возможны запуск и исследование как базовых алгоритмов библиотеки UGR-Metaheuristics, так и вновь разработанных алгоритмов поиска отображения.

4.4.2 Комбинированный параллельный алгоритм PGSA

Для развития алгоритма ПОГ при помощи библиотека UGR-Metaheuristics был реализован комбинированный параллельный алгоритм, характеристики которого были сравнены с предоставляемые этой библиотекой базовыми эвристическими параллельными алгоритмами имитации отжига и генетического отбора. Основные результаты проведенного исследования указанных трех алгоритмов были изложены в публикации [219].

В параллельной версии алгоритма имитации отжига несколько процессов (потоков) участвуют в поиске решения. Лучшее найденное решение сообщается всем процессам (потокам), после чего каждый из них выбирает полученное решение в качестве текущего. Параллельный имитационный отжиг позволяет за одно и то же время просмотреть большее количество вариантов отображения из пространства решений по сравнению с рассмотренной в п. 4.2.2 последовательной версией алгоритма. Благодаря этому, полученные решения покрывают большее количество локальных минимумов значений целевой функции, и повышается ве-

роятность нахождения решения, максимально близкого к глобальному минимуму целевой функции (11).

Параллельный генетический алгоритм основан на кольцевой схеме информационного обмена. Каждый процесс в ходе алгоритма выполняет следующие шаги.

1. Генерация начальной популяции на основе псевдослучайной последовательности и установка популяции в качестве текущей.

2. Получение новых потомков применением операции скрещивания к выбранным особям текущей популяции.

3. Применение операции мутации к потомкам, полученным на предыдущем шаге.

4. Замена худших решений в текущей популяции новыми потомками.

5. Выбор на текущей итерации лучшего члена популяции.

6. Коммуникационный обмен лучшими решениями между соседними процессами.

7. Замена худшего решения в текущей популяции новым решением, если оно лучше.

8. Если не пройдено заданное число итераций – переход к п. 2.

9. Выбор особи с минимальным значением целевой функции (11) среди всех процессов. Выбранная особь принимается в качестве решения задачи отображения.

В отличие от алгоритма имитации отжига, сгенерированные решения в генетических алгоритмах достаточно сильно отличаются от начального. Благодаря этому, удаётся охватить больше локальных минимумов целевой функции и максимально приблизиться к значению глобального минимума. Преимуществом алгоритма имитации отжига является меньшее время работы поиска приемлемого решения, в то время как преимуществом генетического алгоритма является получение более точного решения. Продуктивной видится идея комбинировать два этих алгоритма, чтобы обеспечить высокую точность отображения за приемлемое

время. Идея была реализована в виде комбинированного параллельного алгоритма, получившего название PGSA (Parallel Genetic & Simulated Annealing).

На первом этапе комбинированного алгоритма работает параллельный вариант имитации отжига. В отличие от рассмотренного выше имитационного отжига, предложенный алгоритм не подразумевает обмена между процессами в ходе поиска решений. Каждый процесс генерирует заданное количество решений, которые для параллельного генетического алгоритма становятся текущей популяцией. Выбор типов операторов генетического алгоритма осуществляется в соответствии с заданной конфигурацией.

Параллельный комбинированный алгоритм поиска отображения состоит из следующих шагов.

1. Параллельный поиск решений каждым процессом с применением имитационного отжига.

2. Создание популяции решений для работы параллельного генетического алгоритма из сгенерированных решений на шаге 1.

3. Запуск параллельного генетического алгоритма на заданное число итераций.

4. Выбор лучшего решения в каждом процессе.

5. Выбор лучшего глобального решения.

Отсутствие обмена на этапе работы параллельного алгоритма имитации отжига гарантирует, что каждый процесс сгенерирует уникальную популяцию решений. За счёт миграции решений удаётся передавать между популяциями лучшие признаки, которые будут наследованы потомками.

4.4.3 Методика исследования характеристик параллельных алгоритмов отображения

Параллельные алгоритмы имитационного отжига, генетический алгоритм, комбинированный алгоритм PGSA были реализованы в составе программного средства GrpahHunter [218]. Экспериментальные запуски исследуемых алгорит-

мов производились на разделе Broadwell суперкомпьютера МВС-10П ОП, установленного в МСЦ РАН. Раздел состоит из 136 узлов со следующими характеристиками:

– 2 процессора Intel Xeon E5-2697Av4;
– 32 физических, 64 виртуальных ядра в узле;
– 128 Гб оперативной памяти;
– коммуникационная сеть Intel Omni-Path.

В качестве входных данных использовался набор taiXeyy [220, 221] графов различных порядков, где X – порядок графов, yy – вариант графов заданного порядка. В наборе для каждого порядка указаны граф вычислительной системы и информационный граф программы, представленные в виде матриц расстояний. Для каждой пары графов из набора заранее известны матрицы расстояний и минимальное значение целевой функции (11). Наборы taiXeyy используются учёными для тестирования алгоритмов отображения при решении задач о квадратичном назначении [221].

В экспериментах использовались наборы графов tai27e01, tai45e01, tai75e01, tai125e01, tai175e01, tai343e01, tai729e01. Эксперимент заключался в нахождении отображения для заданной пары графов с фиксированными параметрами алгоритма и числом процессов. Для усреднения результатов было выполнено не менее 10 запусков с одинаковыми параметрами для каждого эксперимента. На первом этапе экспериментов производился подбор оптимальных параметров алгоритма. На втором этапе было выполнено экспериментальное сравнение параметров алгоритмов с найденными оптимальными параметрами.

4.4.4 Результаты сравнения параллельных алгоритмов имитации отжига, генетического отбора и PGSA

На характеристики параллельного алгоритма имитации отжига влияет множество параметров, оптимальные значения которых необходимо определить в ходе исследования. Поиск оптимальных значений параметров осуществлялся в фор-

мате проведения вычислительного эксперимента. Эксперимент заключался в многократном поиске отображения для графов заданной размерности с разным числом ветвей параллельной программы и изменяющимся значением исследуемого параметра при фиксации значений остальных. Подробно поиск оптимальных значений параметров параллельного алгоритма имитации отжига рассмотрен в публикации [219]. В результате экспериментов были установлены следующие значения основных параметров алгоритма:

– количество просматриваемых решений при фиксированном значении температуры – 50;

– в качестве функции понижения температуры выбрана функция Коши [219];

– число последовательных итераций работы алгоритма – 100;

– для графов порядка не более 256 достаточно 50 000 итераций параллельного алгоритма;

– для графов порядка не более 1024 достаточно 100 000 итераций параллельного алгоритма;

– для графов порядка не более 100, число «решателей» совпадает с размерностью графа, для графов порядка до 1024 число решателей должно составлять 125.

Для параметров, отвечающих за количество найденных решений при одном значении температуры и формирование начальной температуры, были сохранены значения, предложенные автором библиотеки UGR-Metaheuristics.

При выполнении параллельного генетического алгоритма после каждой итерации процессы обмениваются лучшими особями. Единственным отличием от последовательного алгоритма является наличие информационных обменов. Для параллельного генетического алгоритма были применены параметры, рекомендуемые автором библиотеки UGR-Metaheuristics, за исключением размера популяции:

– вероятность применения операции скрещивания 1;

– вероятность применения операции мутации – 0,001;

– число мигрирующих решений должно быть небольшим, эксперименты показали, что более 1 мигрирующего решения ухудшает качество финального решения;

– число членов популяции совпадает с порядком исходных графов;

– для графов разных порядков задано фиксированное число итераций алгоритма. Фиксированное число итераций работы алгоритма для графов больших порядков позволяет за приемлемое время получать приемлемое решение.

Комбинированный алгоритм PGSA фактически состоит из последовательного алгоритма имитации отжига и параллельного генетического алгоритма. Решения, полученные путём применения алгоритма имитации отжига, становятся начальной популяцией для параллельного генетического алгоритма. Для последовательного алгоритма имитации отжига задаются те же параметры, что и для параллельного алгоритма имитации отжига. Параметры параллельного генетического алгоритма не были изменены.

На втором этапе алгоритмы сравнивались по точности, определяемой значением целевой функции, и времени выполнения. Отметим, что параллельная схема алгоритмов такова, что увеличение числа процессов приводит к расширению пространства рассматриваемых решений. Соответственно, изменение числа процессов влияет на точность отображения, практически не сказываясь на времени работы алгоритмов.

Эксперименты показали, что параллельный генетический алгоритм при поиске отображения графов больших порядков в среднем получает решение лучше, чем параллельный алгоритм имитации отжига. Благодаря миграции решений между различными популяциями, удаётся передавать лучшие признаки. Параллельный генетический алгоритм при отображении графов больших порядков имеет большее время поиска отображения по сравнению с алгоритмом имитации отжига. Это объясняется тем, что для любого нового потомка необходимо заново рассчитывать значение целевой функции, в отличие от имитации отжига, где зна-

чение целевой функции рассчитывается относительно внесённых изменений в отображение.

Сравнение качества полученных решений от числа процессов для набора данных tai343e01 представлено на рисунке 49.

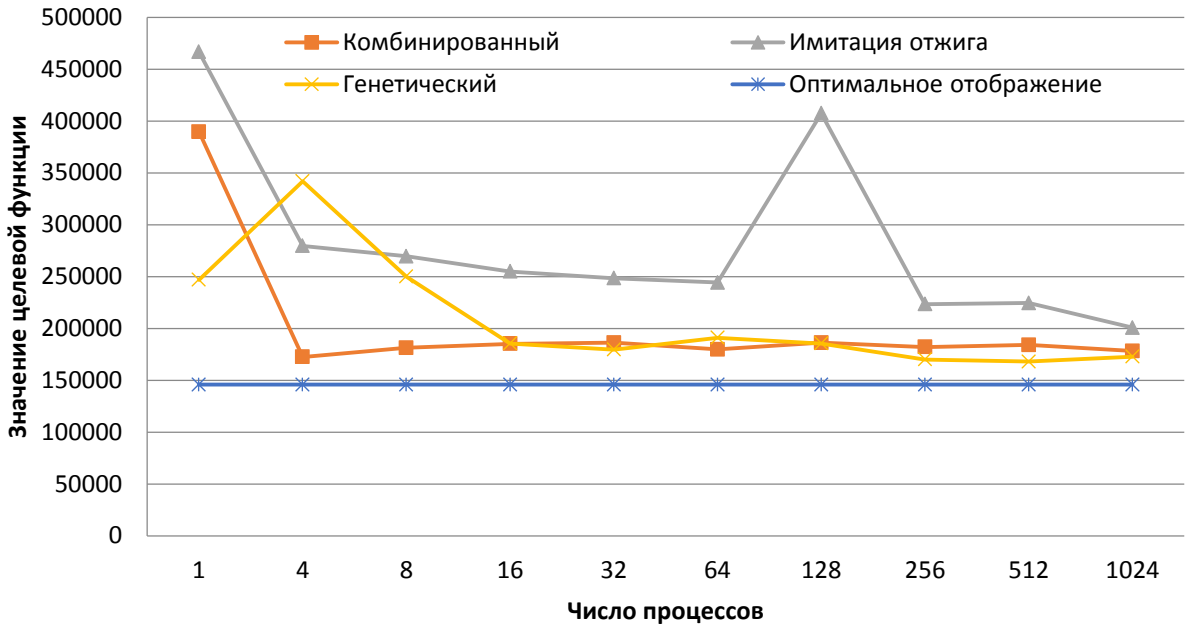

Рисунок 49. Зависимость значений целевой функции от числа процессов для набора данных tai343e01

Сравнение качества полученных решений от числа запущенных процессов для набора данных tai729e01 представлено на рисунке 50.

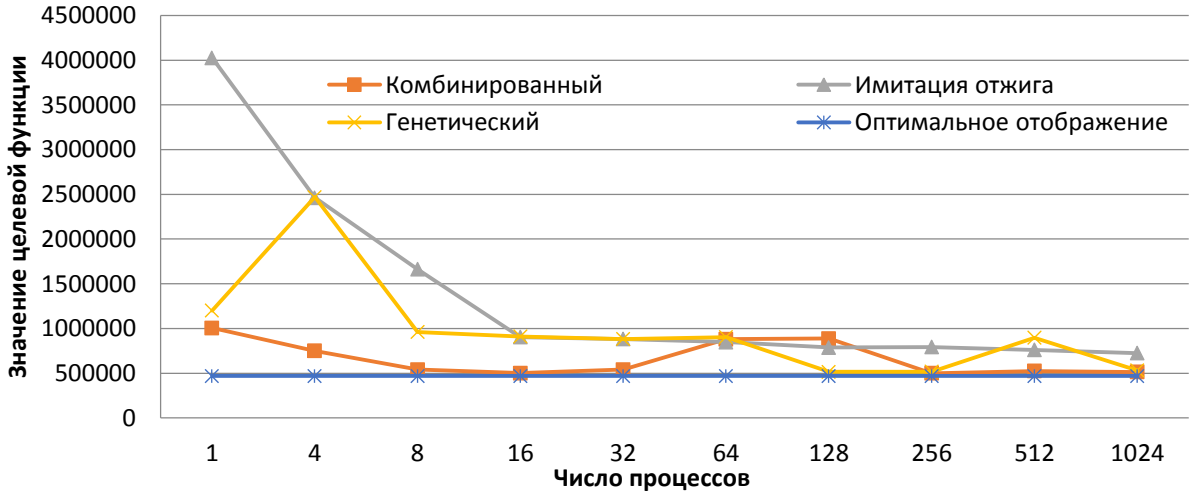

Рисунок 50. Зависимость значений целевой функции от числа процессов для набора данных tai729e01

Сравнение среднего времени выполнения алгоритмов для различных наборов данных представлено на рисунке 51.

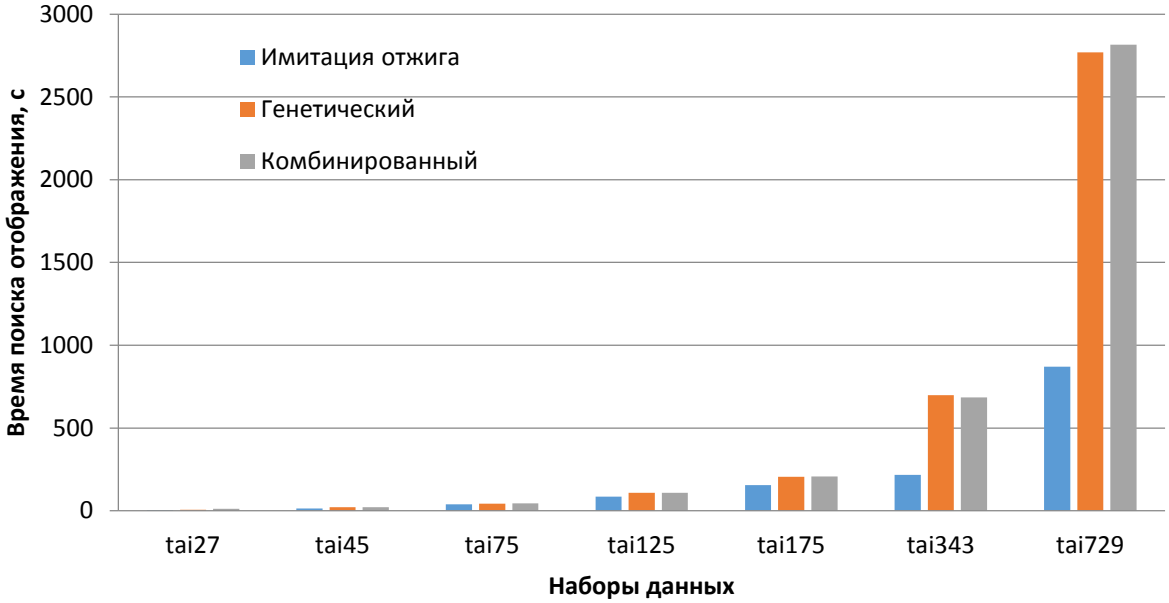

Рисунок 51. Среднее время выполнения алгоритмов для различных наборов данных

Таблица 18 содержит оценки точности и времени выполнения алгоритмов по итогам проведенных экспериментов. Оптимальное решение для наборов было взято из набора данных [222]. В столбце F указано лучшее полученное в ходе экспериментов значение целевой функции. В столбце T указано время в минутах, затраченное алгоритмом на получение лучшего решения. В столбце F_0 указано оптимальное значение целевой функции для каждого из наборов. В столбце T_0 указано время получения оптимального отображения, полученное в работе [223]. В столбце D приведена оценка точности полученного решения, показывающая насколько отличается значение целевой функции от оптимального в процентах: $D = 100 \cdot (F - F_0) / F_0$.

Для 256 процессов была определена масштабируемость программы GraphHunter при ее запуске на разном числе узлов раздела Broadwell суперкомпьютера МВС-10П ОП. Представленные на рисунке 52 результаты позволяют говорить о достаточно хорошей масштабируемости разработанного приложения.

Таблица 18. Оценка точности решений, полученных при помощи параллельных алгоритмов имитации отжига, генетического отбора и PGSA

Алгоритмы / Набор данных	Параллельный имитационный отжиг			Параллельный генетический алгоритм			Параллельный комбинированный алгоритм PGSA			Оптимальное решение	
	F	T	D	F	T	D	F	T	D	F_0	T_0
Tai27	2558	0,05	1	3176	0,1	24	2600	0,27	2	2558	0,02
Tai45	6724	0,3	5	8564	0,45	34	7332	0,5	14	6412	0,03
Tai75	19380	0,6	34	18268	0,7	26	18810	0,75	29	14488	8
Tai125	50780	1,6	43	47816	2	35	50792	1,75	43	35426	166
Tai175	72688	2,8	26	74602	5	29	74880	3,1	29	57540	181
Tai343	200856	3,5	37	168120	12,8	15	172466	10,1	18	145862	1026
Tai729	724820	18,2	54	514846	50	9	498454	53,2	6	469650	1187

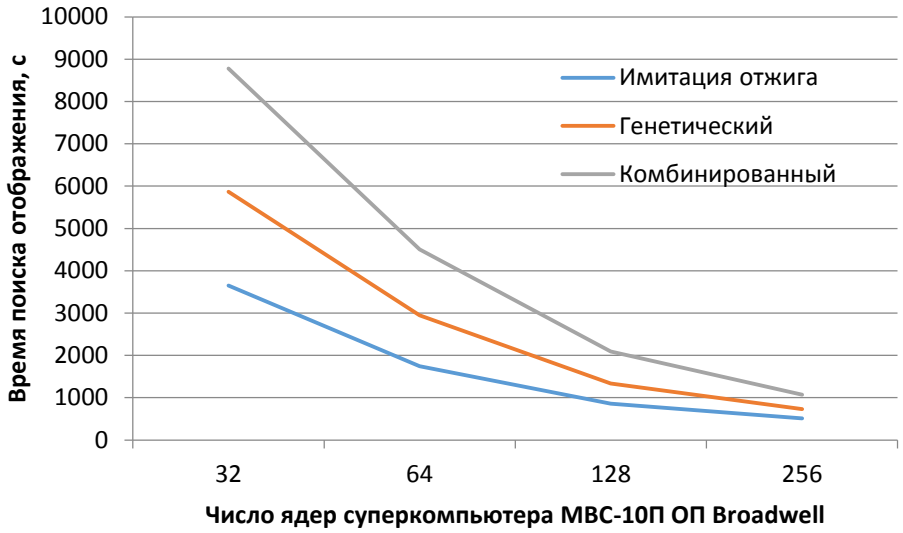

Рисунок 52. Масштабируемость программного средства GraphHunter при поиске отображения 256-ю процессами

По результатам проведенных на суперкомпьютере МВС-10П ОП экспериментов можно сделать следующие выводы.

1. Программное средство GraphHunter позволяет подобрать оптимальные параметры реализованных алгоритмов, обеспечивающие высокую точность отображения при приемлемом для систем коллективного пользования времени выпол-

нения. GraphHunter позволяет реализовать и сравнить по точности и времени выполнения произвольный параллельный алгоритм поиска отображения.

2. GraphHunter показал близкую к линейной масштабируемость при заданной точности отображения.

3. Приемлемое время выполнения для графов больших размеров демонстрирует только имитационный отжиг. Только этот алгоритм затрачивает на поиск отображения время, сопоставимое с системными таймаутами (не более 15 минут). Генетический и комбинированный алгоритм, хотя и находят лучшие решения, выполняются неприемлемо долго.

4. Двухэтапный метод РГО-ПОГ предполагает запуск алгоритмов отображения на выделенных для очередного задания узлах суперкомпьютера. Если применять этот метод, сравнение точности параллельных алгоритмов необходимо производить на числе процессов, равным числу выделенных для задания ядер (узлов). Как показывают результаты экспериментов, на числе процессов, равным порядку графов, точность имитационного отжига сопоставима с точностью генетического и комбинированного алгоритмов. Исключение составляет случай графа из 729 вершин, для которого генетический и комбинированный алгоритмы демонстрируют очень высокую точность с ошибкой в 9% и 6% соответственно.

4.4.5 Циклический комбинированный параллельный алгоритм CPGSA

Экспериментальное сравнение характеристик трех алгоритмов: имитации отжига, генетического отбора и комбинированного PGSA показало, что точность отображения на графах малых порядков у алгоритма PGSA сопоставима с точностью алгоритма отжига. Однако на графах больших порядков, где алгоритм отжига показывает себя не очень хорошо, точность и скорость алгоритма PGSA сопоставимы с алгоритмом генетического отбора. При этом алгоритм PGSA существенно уступал имитации отжига по времени поиска. С целью улучшения характеристик алгоритма PGSA в рамках настоящего исследования было реализовано несколько его модификаций.

Учитывая результаты [211] для графов малых порядков, было принято решение зациклить алгоритм PGSA с многократным повтором его фаз. После нахождения решения в первой итерации комбинированного алгоритма найденное отображение повторно подается на вход этого алгоритма. В связи со случайной природой алгоритма имитации отжига генерируется новый набор решений, которые затем передаются в генетический алгоритм. Процесс чередования фаз имитации отжига и генетического отбора повторяется несколько раз. В результате на каждой итерации нового составного алгоритма искомое решение демонстрирует возрастающую точность отображения от одной итерации к другой. На каждом шаге находится отображение, которое становится все ближе к оптимальному.

По аналогии с [219] модифицированный алгоритм получил название Cycled PGSA или CPGSA [224]. Алгоритм CPGA состоит из следующих фаз.

1. Параллельный поиск решений каждым процессом с использованием имитации отжига.

2. Генерация начальной популяции решений для параллельного генетического алгоритма из шага 1.

3. Запуск параллельного генетического алгоритма для указанного количества итераций.

4. Выбор оптимального решения для каждого процесса.

5. Выбор наилучшего глобального решения.

6. Если это не последняя итерация, генерация начальных значений для шага 1.

7. Повторение шагов 1-5 указанное количество раз.

Алгоритм CPGSA был реализован в составе ПО GraphHunter. Экспериментальное исследование характеристик модифицированного алгоритма проводились с использованием раздела Broadwell суперкомпьютера МВС-10П ОП в соответствии с рассмотренной в п.4.4.3 методикой.

На рисунке 53 показаны сравнительные результаты экспериментов. Наилучшие значения решения для кривой «Оптимальный» были получены из

[222]. Значения для кривых «Отжига», «Генетический» и «PGSA» взяты из представленных в п. 4.4.4 результатов.

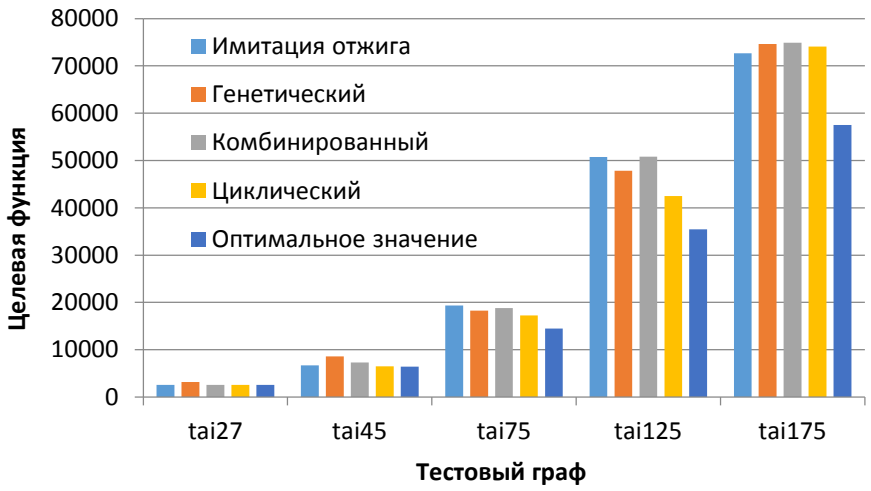

Рисунок 53. Зависимость значения функции потерь от различных программных графов и алгоритмов

Рисунок 53 показывает, что на графах малых и средних порядков комбинированный алгоритм PGSA находит решение, сопоставимое с наилучшим решением, полученным алгоритмом имитации отжига или генетическим алгоритмом. Однако, модифицированный алгоритм CPGSA предлагает более точное решение для отображения графа программы на граф многопроцессорной системы. Точность отображения, выраженная через отклонение значения целевой функции в соответствии с (13), увеличивается с 3% на графах малых порядков (набор tai45) до 12% на графах средних порядков (tai125). На наборе экземпляров tai175 значения целевой функции (11) сопоставимы со значением этой функции алгоритма имитации отжига. Вместе с тем время работы CPGSA существенно меньше, что показывает рисунок 54.

Графики на рисунке 54 демонстрируют, что алгоритм CPGA позволяет значительно сократить время, необходимое для поиска решения. Ускорение составляет от 1,41 раз на наборе tai175 до 2,68 раз на наборе tai75. Стоит отметить, что подобное уменьшение времени работы алгоритма существенно зависит от используемых параметров. В проведенных экспериментах применялся подход, при котором происходит множество коротких итераций циклического алгоритма.

Кроме того, улучшение времени работы дала проведенная соавтором [224] К.А. Брагиным оптимизация исходного кода.

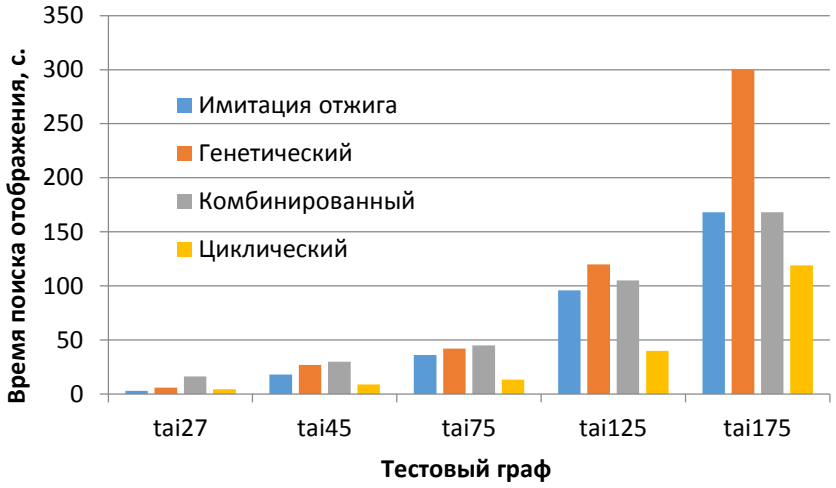

Рисунок 54. Время выполнения алгоритмов в зависимости от набора графов

Рассмотрим поведение модифицированного алгоритма CPGSA на примере экземпляра tai175. Точность решения увеличивается итеративно, в отличие от PGSA, как показано на рисунке 55. Шаги алгоритма CPGSA повторяются в цикле. Решение, полученное на предыдущем шаге, является входными данными алгоритма на следующем шаге. Каждая итерация немного улучшает решение. Кривая, изображающая зависимость целевой функции (11) для CPGSA от шага, принимает форму гиперболы, что указывает на то, что решение приближается к оптимальному. Алгоритм выходит на плато после большого количества шагов, находя приближенное решение, которое он не может улучшить дальше.

Оптимальное количество шагов (итераций) составляет 3-5. Соответствующее ограничение на число шагов было встроено в программное обеспечение GraphHunter. Найденное решение считается наилучшим, если точность не улучшается на следующем шаге, и поиск останавливается.

Рассмотрим набор параметров, используемых на начальном и последующих шагах. Возможна тонкая настройка алгоритма отдельно на первом шаге, что существенно влияет на результат, и на последующих шагах, которые улучшают решение, полученное на предыдущих шагах.

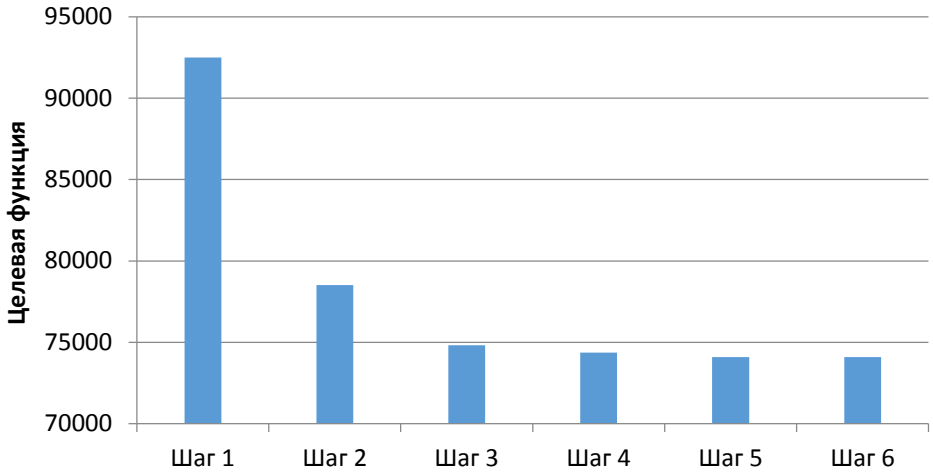

Рисунок 55. Зависимость функции потерь от шага (итерации) алгоритма CPGSA

Для решения проблемы «застревания» циклического алгоритма в локальном минимуме было предложено динамически изменять параметры используемых на каждом шаге алгоритмов отжига и генетического отбора. Подобный подход применялся в работе [225] на графах малой размерности, где в процессе поиска решения происходило понижение температуры отжига. В рамках циклического характера работы CPGSA на графах средней и большой размерности изменение температуры отжига не дало результатов. Изменение других параметров на каждом шаге также не принесло улучшений характеристик алгоритма. Был сделан вывод, что изменение только одного или нескольких параметров не влияет на точность найденного отображения. Было принято решение производить изменение сразу всех параметров алгоритма.

На каждом шаге алгоритма CPGSA К.А. Брагиным [224] было предложено применить т.н. модификаторы, которые представляют собой вещественные числа от 0 до 2. Например, начальное значение популяции в генетическом алгоритме равно 100. После первого шага, к этому значению применяется модификатор 0,8, и значение становится равным 80. В дальнейшем на каждом шаге размер популяции будет уменьшаться в соответствии с модификатором и составит, соответственно шагам: 64, 51, 41, 32. Таким образом, на каждом шаге работа алгоритма CPGSA несколько отличается от работы на предыдущем шаге. Однако возникает задача поиска таких модификаторов.

Было предложено автоматически находить значения модификаторов с помощью алгоритма имитации отжига. На каждой итерации CPGSA значение целевой функции (11) сравнивается со значением предыдущего шага. Если значение уменьшилось, модификаторы сохраняются. Если значение не уменьшилось, то случайным образом меняется один из модификаторов, и процесс повторяется. Со временем доступное окно изменения модификаторов сужается, что позволяет получать более точные решения.

Следует отметить, что процесс поиска модификаторов осуществляется только один раз. В следующий раз, при запуске на том же числе ядер, найденные модификаторы могут быть повторно применены для сокращения времени поиска отображения. При запуске алгоритма CPGSA на другом числе ядер необходимо определить новые значения модификаторов.

На рисунке 56 представлены результаты применения модификаторов для различных наборов графов. Для наборов tai75 и tai125 были найдены значения модификаторов, максимально приблизившие целевую функцию к оптимальному значению. Для графов небольших порядков поиск данных модификаторов не имеет значения, поскольку и CPGSA, и PGSA без модификаторов находят решение, близкое к оптимальному.

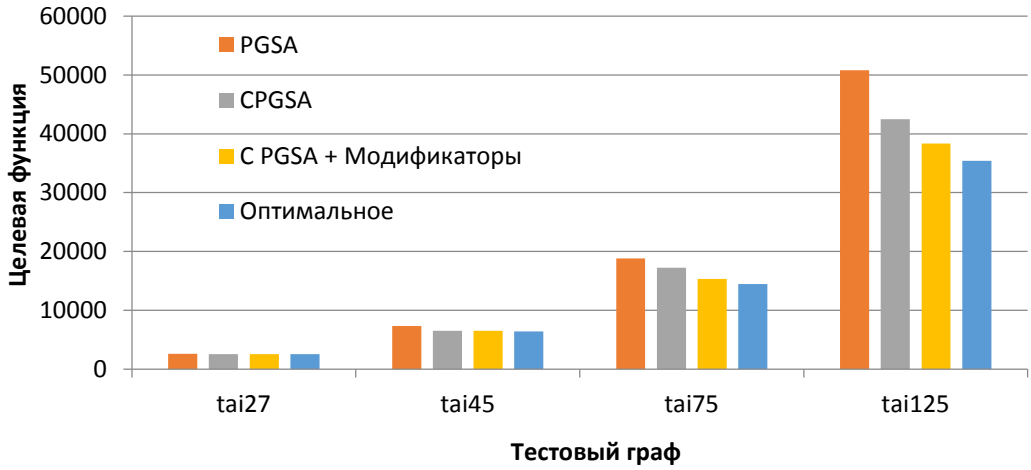

Рисунок 56. Значения целевой функции потерь для алгоритмов PGSA, CPGSA и CPGSA с применением модификаторов

Легко заметить значительное повышение точности по сравнению с алгоритмами без модификаторов. На примере tai75 точность CPGSA увеличилась на 19%, а точность PGSA увеличилась на 22%. На примере tai125 точность увеличилась на 10% по сравнению с CPGSA и на 32% по сравнению с алгоритмом PGSA.

На рисунке 57 показано время поиска решения в секундах для различных алгоритмов и наборов графов. Алгоритм CPGSA значительно быстрее, чем PGSA и CPGSA с использованием модификаторов. Стоит отметить, что отклонение точности от оптимального решения составляет 5% и 8% для наборов tai75, tai125 соответственно.

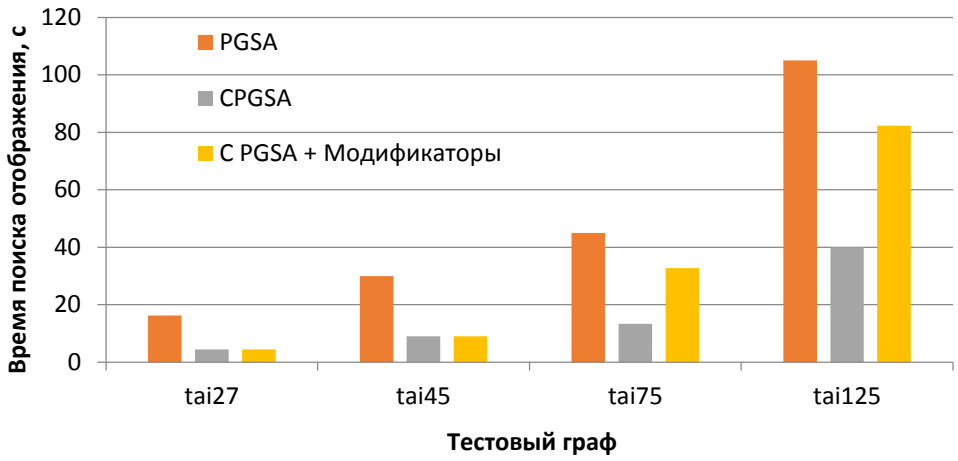

Рисунок 57. Время выполнения алгоритмов PGSA, CPGSA и CPGSA с применением модификаторов

Ускорение параллельной программы при оптимизации ее распределения по узлам суперкомпьютерной системы напрямую зависит от однородности связей между узлами. Оптимизация размещения процессов параллельной программы на узлах суперкомпьютера может увеличить производительность вычислений на 10-25%, как показано в п. 4.3.4 . Согласно статистике [114], среднее время выполнения задачи на суперкомпьютере МВС-10П ОП составляет 270 минут. Таким образом, для МВС-10П ОП процесс оптимизации размещения не должен превышать 10% от времени выполнения программы, что составляет 27 минут. Это время хорошо согласуется с системными таймаутами СУППЗ, которые обычно устанавливаются на уровне 10-15 минут. Поскольку график сходимости итеративного алгоритма

CPGSA имеет вид гиперболы, то его работу целесообразно прерывать, когда значение целевой функции (11) перестает уменьшаться в заданном диапазоне или когда истекает отведенное на процесс оптимизации время. Согласно статистическим данным, это время для суперкомпьютера МВС-10П ОП составляет 27 минут.

Выводы к главе 4

Одной из важных задач управления суперкомпьютерными ресурсами является распределение ветвей (процессов) параллельной программы из состава задания пользователя по вычислительным узлам. Эта задача также известна как задача поиска оптимального отображения информационного графа параллельной программы на граф коммуникационных связей ВУ суперкомпьютера. Оптимальное отображение позволяет снизить накладные расходы на информационные обмены между ветвями параллельной программы и тем самым повысить быстродействие расчетов. Задача поиска оптимального отображения является NP-полной, и точные методы ее решения требуют значительного времени расчетов. В то же время в условиях режима коллективного пользования необходимо получать близкое к оптимальному отображение за относительно небольшое время, сравнимое с длительностью системных таймаутов. При этом информационный граф программы и граф коммуникационных связей свободных ВУ не всегда заранее известны, поскольку первый задается произвольным образом пользователем, а второй определяется текущей мультипрограммной ситуацией. Кроме этого, при отображении на вычислительные узлы параллельной программы очередного задания необходимо учитывать наличие в системе других заданий, которым также необходимо обеспечить отображение, минимизирующее время выполнения.

Для поиска оптимального отображения за приемлемое для системы коллективного пользования время предложен двухэтапный метод. На первом этапе решается задача выделения для очередного задания подсистемы ВУ из множества свободных на момент запуска задания узлов. Эта задача сведена к разрезанию графа свободных ВУ на минимально связанные между собой подграфы, для чего

предложен алгоритм РГО, основанный на имитации отжига. На втором этапе производится поиск оптимального отображения программного графа на граф выделенных заданию ВУ при помощи предложенного автором параллельного алгоритма ПОГ имитации отжига в комбинации с генетическим алгоритмом. Для расчетов при этом используются выделенные для задания вычислительные узлы.

Алгоритмы РГО и ПОГ были реализованы в СУППЗ для суперкомпьютеров серии МВС-1000 и обеспечили рост быстродействия расчетов как для тестовых программ, так и для реальных приложений. Алгоритм ПОГ получил свое развитие в виде параллельного комбинированного алгоритма PGSA и его циклической модификации CPGSA. Алгоритм PGSA состоит из двух фаз: имитации отжига и генетического отбора, в алгоритме CPGSA эти фазы циклически повторяются. Указанные алгоритмы были реализованы в составе программного средства GraphHunter, с помощью которого было произведено экспериментальное исследование их характеристик.

Эксперименты показали, что алгоритм PGSA всегда даёт среднее лучшее решение, чем генетический алгоритм и имитационный отжиг, однако его быстродействие не превышает быстродействия генетического алгоритма. Циклическое повторение фаз отжига и генетического отбора в алгоритме CPGSA позволило добиться итерационного повышения точности найденного отображения и скорости его поиска. В результате удалось достичь более высокой точности отображения графа программы на граф ВУ по сравнению с известными алгоритмами при времени поиска отображения, сравнимом с системными таймаутами.

Глава 5. Организация параллельных вычислений с распараллеливанием по данным

5.1 Задача организации параллельных вычислений с распараллеливанием по данным

В рассмотренной в п. 1.7 архитектуре СУППЗ на втором уровне иерархии управления вычислительными ресурсами можно провести разграничение условных «зон ответственности» СУППЗ и параллельной программы пользователя. Фактически за управление вычислительными ресурсами на этом уровне отвечает сценарий выполнения задания, в большинстве случаев автоматически формируемый сценариями надстройки подготовки заданий. Сценарий выполнения задания запускается с правами пользователя на первом по списку из выделенных заданию ВУ и обеспечивает распределение процессов параллельной программы по выделенным узлам. Стоит отметить, что к этому моменту СУППЗ завершила выбор ВУ, произвела их диагностику, осуществила конфигурацию ВУ в соответствии с заданной пользователем программной платформой (средой выполнения программ) и открыла пользователю доступ на подготовленные для выполнения его задания ВУ.

За счет автоматизации подготовки паспортов заданий для стандартных программных сред с пользователя снимается значительный объем рутинной работы по корректному оформлению задания. При этом задача распараллеливания вычислений в прикладной программе остается зоной ответственности пользователя. Другими словами, после запуска сценария выполнения задания СУППЗ делегирует функции управления второго и первого уровня иерархии пользователю-программисту. С одной стороны, это предоставляет пользователю широкие возможности для непосредственного взаимодействия с аппаратурой ВУ и тонкой настройки своих приложений, с другой стороны, для того, чтобы в полной мере воспользоваться этими возможностями, пользователь должен иметь определенный опыт организации параллельных вычислений. Для большинства пользователей-исследователей научного суперкомпьютерного центра коллективного пользования приобретение такого опыта означает

непроизводительную трату усилий и времени, по сути отнятых от исследовательской работы в своей предметной области.

Таким образом, автоматизация процессов организации параллельных вычислений на первом и втором уровнях иерархии управления является актуальной научной задачей. Как показано в монографии [14], решение этой задачи в общем случае невозможно. Однако, возможно выделить отдельные классы задач высокопроизводительных расчетов, для которых подобная автоматизация осуществима. Одним из таких классов задач являются задачи с распараллеливанием по данным.

Распараллеливание по данным часто применяется для решения ряда важных прикладных вычислительных задач. В таких задачах одна и та же последовательность вычислений (прикладной алгоритм) выполняется над всеми элементами множества (пула) входных данных. Между процессами параллельной программы отсутствуют информационные обмены. К типичным задачам распараллеливания по данным относятся виртуальный скрининг, поиск простых чисел, проверка стойкости паролей и др. Обычно распараллеливание по данным применяется в распределенных средах добровольных вычислений [226]. Однако решение подобных задач часто осуществляется на отдельных суперкомпьютерах.

Пусть вычислительный алгоритм может быть реализован в виде **одной последовательной программы** (ОПП), для которой порция входных данных определяется значениями одного или нескольких параметров. Из таких порций вычислительной работы складывается пул входных данных – множество всех возможных значений параметров ОПП во всех их комбинациях, как показано на рисунке 58. Примером ОПП может служить известная программа подбора паролей HashCat [227].

При распараллеливании по данным на суперкомпьютере на каждом его узле выполняются один или несколько экземпляров ОПП с различными значениями входных параметров. Организация параллельных вычислений заключается в осуществлении запусков экземпляров ОПП на всем множестве доступных узлов, как показано на рисунке 59. Определим **технологию распараллеливания по дан-**

ным как совокупность методов, алгоритмов и средств, осуществляющих распределение экземпляров ОПП и порций данных для обработки по вычислительным узлам суперкомпьютера.

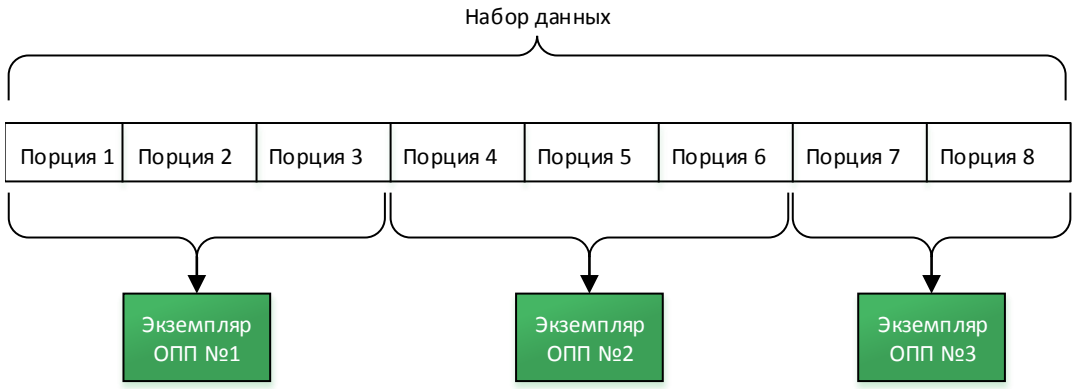

Рисунок 58. Схема параллельных вычислений на основе ОПП

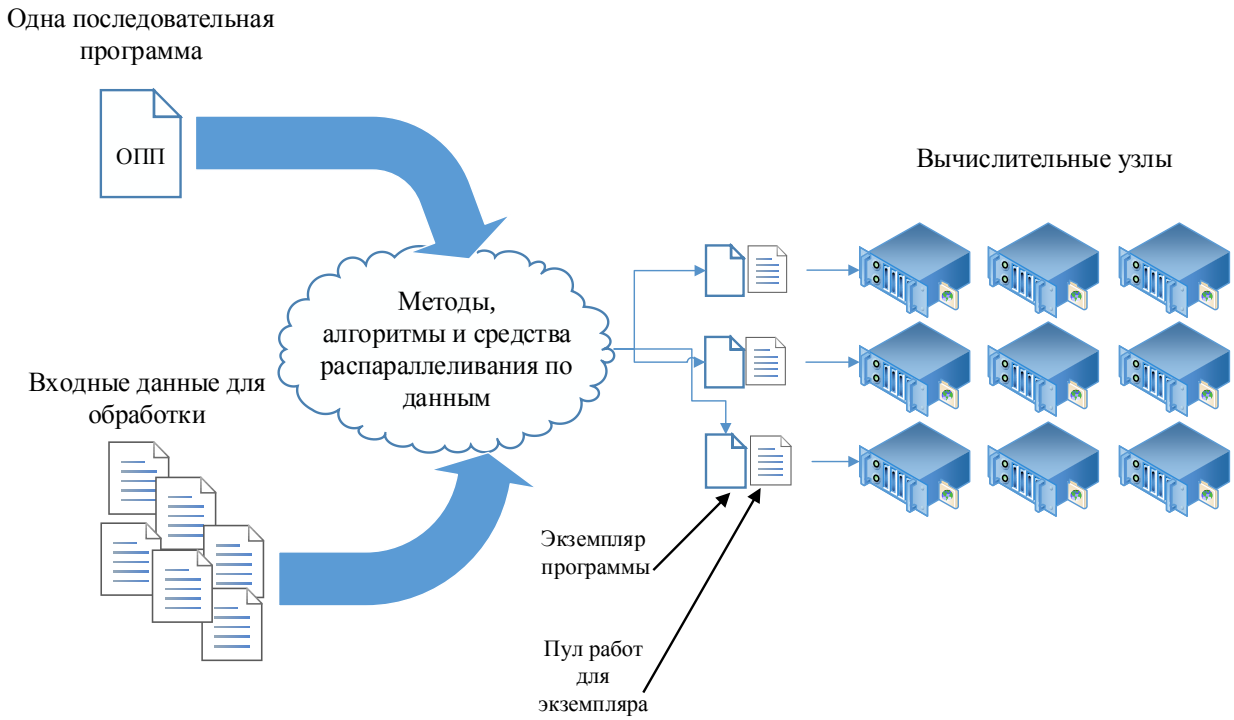

Рисунок 59. Организация параллельных вычислений с распараллеливанием по данным с применением ОПП

Оптимизация параллельных вычислений в модели ОПП связана с минимизацией доли накладных расходов на распараллеливание, которые напрямую определяют быстродействие расчетов. Пусть существует одна последовательная программа, обрабатывающая массив данных в течение некоторого времени. Если её запустить на p процессорах и массив данных равномерно распределить по p про-

цессорам, то обработка ускорится в p раз. Но в действительности этого не произойдёт, поскольку имеют место накладные расходы на организацию распараллеливания: затраты времени на передачу данных между вычислительными узлами, на обращения к сервисам (база данных, веб сервер, планировщик), задержки между получением данных и началом их обработки, между окончанием обработки и началом передачи результатов. Помимо этого, существенную долю накладных расходов может внести программист, неэффективно реализовав управляющую программу для запуска экземпляров ОПП на множестве вычислительных узлов.

Помимо накладных расходов, определяющих производительность, следует упомянуть о таких важных показателях качества, как отказоустойчивость, предел масштабируемости и реакция системы на дисбаланс вычислительной нагрузки. Отказоустойчивость подразумевает способность системы распараллеливания по данным как продолжать вычисления в случае выхода из строя части ВУ, так и возобновлять вычисления с автоматически сохраняемой контрольной точки в случае отказа всех ВУ или управляющей ЭВМ.

Предел масштабируемости показывает, какой объем данных, измеренный в порциях, можно обработать с помощью той или иной технологии распараллеливания по данным. Реакция системы на дисбаланс вычислительной нагрузки показывает, как изменяются накладные расходы и предел масштабируемости системы в случае, если в неоднородном решающем поле, используемом для расчётов, есть ВУ существенно различной производительности.

5.2 Технологии распараллеливания по данным

Для организации параллельных вычислений с распараллеливанием по данным в модели ОПП могут быть применены различные методы, алгоритмы и средства, которые в совокупности мы обозначили как технологии распараллеливания по данным. Рассмотрим наиболее известные технологии.

Компания Google в 2004 году представила стек технологий MapReduce [56], включающий модель параллельного программирования, параллельную файловую

систему Google File System (GFS), многомерную базу данных BigTable и высоконадежную службу синхронизации Chubby. Модель программирования MapReduce и связанная с ней реализация для обработки и генерации больших наборов данных применяется для широкого спектра реальных задач.

Технология MapReduce подразумевает выполнение двух обязательных шагов: Map и Reduce. На Map-шаге происходит предварительная обработка входных данных, заключающаяся в анализе набора входных данных и формировании на выходе множества пар «ключ-значение». Важно, что Map-шаг выполняется по схеме «мастер-рабочие». Процесс-мастер получает входные данные, разделяет их на части и передает рабочим для выполнения предварительной обработки. На Reduce-шаге происходит свертка полученных на Map-шаге результатов. Процесс-мастер получает результаты от рабочих и на их основе формирует итоговое решение исходной задачи. Такая структура процесса вычислений полностью соответствует модели распараллеливания по данным с ОПП.

Для организации вычислений пользователю достаточно разработать две функции – Map и Reduce. На вход Map-функции поступает элементарная порция данных, на выходе должна быть сформирована пара «ключ-значение». На вход Reduce-функции соответственно поступает набор пар «ключ-значение», функция выполняет заданную пользователем свертку (агрегацию) полученных пар. Реализация MapReduce автоматически распараллеливает вычисления по кластерам, состоящим из множества ВУ, обрабатывает сбои ВУ и планирует межузловое взаимодействие для эффективного использования сети и дисков. В случае выхода ВУ из строя, вычислительная нагрузка автоматически перераспределяется на исправные узлы. При подключении новых или восстановленных узлов на них автоматически назначается вычислительная работа.

Помимо реализации MapReduce от компании Google, существует большое число реализаций этой технологии от сторонних производителей, в том числе свободно распространяемых. Обзор реализаций MapReduce можно найти в публикации

[57]. Из известных свободно распространяемых реализаций MapReduce можно выделить Hadoop [57] и Disco [228].

Распараллеливание по данным в модели ОПП возможно также организовать при помощи известной платформы BOINC (Berkeley Open Infrastructure for Network Computing) [58, 59]. BOINC является стандартом де-факто при организации добровольных вычислений, под которыми понимается использование разнородных распределенных цифровых устройств (от университетских кластеров до персональных компьютеров и мобильных телефонов), добровольно предоставляемых их владельцами для высокопроизводительных научных вычислений. Все предоставляемые устройства соединяются через Интернет с сервером BOINC, который разбивает входные данные на порции и раздает эти порции для обработки подключившимся устройствам-клиентам. Обработка данных на клиентских устройствах обычно осуществляется во время их простоя и автоматически прерывается, когда владелец возобновляет пользование устройством. Суммарная вычислительная мощность устройств добровольцев может быть значительной, однако организация упорядоченного вычислительного процесса в такой системе сталкивается с массой проблем из-за неоднородности устройств, их ненадежности, сильной текучести состава добровольцев, необходимости учета обработанных и необработанных порций данных и др. BOINC, как система промежуточного программного обеспечения с открытым исходным кодом для добровольных вычислений, призвана решать эти проблемы.

Среди отечественных технологий распараллеливания по данным следует выделить программный комплекс (ПК) X-COM [60, 61], разработанный специалистами НИВЦ МГУ им. М.В. Ломоносова. ПК X-COM реализован на языке программирования Perl, что делает его одним из наименее ресурсоёмких средств распараллеливания по данным. Архитектурно в ПК X-COM можно выделить два основных компонента: сервер и вычислительные узлы. Сервер X-COM отвечает за разделение исходной задачи на блоки (задания), распределение заданий по узлам, координацию работ всех узлов, контроль целостности результата и сбор результата в единое целое. В качестве узла может выступить любая вычислительная единица (ра-

бочая станция, узел суперкомпьютера, виртуальная машина), имеющая возможность выполнить экземпляр прикладной программы. Узлы отвечают за расчёт блоков прикладной задачи (принятых от сервера заданий), запрос заданий от сервера, передачу результатов расчётов на сервер.

Платформы BOINC и X-COM предназначены для функционирования в распределённой среде, но могут быть установлены и запущены на отдельном суперкомпьютере. В этом случае один из ВУ принимает на себя функции сервера, раздающего порции вычислительной работы, а остальные выступают в роли клиентских устройств. Подобная организация параллельных вычислений также полностью соответствует модели ОПП распараллеливания по данным.

Другой отечественной разработкой является созданный при участии автора диссертации программный комплекс «Пирамида» [229, 230]. ПК «Пирамида» обеспечивает параллельное отказоустойчивое и масштабируемое выполнение множества копий последовательной программы с индивидуальными наборами значений входных параметров на вычислительном кластере. Значения параметров для каждой копии программы формируются автоматически из заданного пользователем множества значений. Отказоустойчивость вычислений обеспечивается записью контрольных точек и перезапуском копий программ после отказа на работоспособных вычислительных ресурсах. Масштабируемость вычислений обеспечивается динамическим запуском копий программы на всех доступных вычислительных ресурсах. ПК «Пирамида» способен обеспечить параллельные вычисления с распараллеливанием по данным в иерархически организованных гетерогенных вычислительных системах [231]. Развитием ПК «Пирамида» является созданный под руководством автора программный комплекс XP-COM [113]. В основе обоих комплексов лежит предложенный автором метод иерархического разделения данных. Этот метод, структура и алгоритмы работы программных комплексов «Пирамида» и XP-COM, сравнение этих ПК с технологиями распараллеливания MapReduce, BOINC и X-COM будут рассмотрены в последующих параграфах настоящей главы.

5.3 Метод иерархического разделения данных

Семантику вычислений в ПК «Пирамида» и XP-COM пользователь определяет в прикладной последовательной программе с учетом следующих особенностей. Экземпляры программы запускаются на выделенных заданию пользователя вычислительных узлах и выполняются независимо и асинхронно. Выполняемая программа, ее параметры и требования к вычислительным ресурсам задаются пользователем в паспорте задания.

Множество *D* обрабатываемых программой данных представляет собой декартово произведение множеств $D = P_1 \times P_2 \times ... \times P_k$, задаваемых параметрами программы. Множество значений параметра может задаваться непосредственно, в виде списка значений, либо косвенно, в виде диапазона значений или ссылки на файл, содержащий множество значений. В основе организации распараллеливания по данным лежит принцип рекурсивного формирования множества подмножеств данных:

$$D = \{D_{i_1} |\, i_1 = \overline{1, n_1}, D_{i_1} = \{D_{i_1 i_2} |\, i_2 = \overline{1, n_2}, D_{i_1 i_2} = \cdots\} \ldots \{D_{i_1 \ldots i_r}\}\} \ldots\}$$

Назовем элемент множества $D_{i_1 \ldots i_r}$ **слайсом.** Слайс задает элементарную работу (элементарную порцию данных), выполняемую экземпляром программы. Множество слайсов $\{D_{i_1 \ldots i_r}\}$, выделяемых для обработки экземпляру или группе экземпляров программы, называется **пулом работ.** Введем иерархическую систему объектов, обрабатывающих множество *D*, с числом уровней иерархии *r*. Для каждого уровня иерархии (рекурсии) $l \in \overline{1, r}$ может быть определена мощность n_l пула работ. Объект, обрабатывающий пул работ на уровне *l*, назовем менеджером M_l^i, где $i \in \overline{1, c_l}$ – это номер менеджера, а c_l – число менеджеров на уровне *l*. В подчинении менеджера M_l^i находятся менеджеры следующего уровня иерархии M_{l+1}^j, $j \in \overline{1, c_{l+1}}$.

Каждому слайсу $S \in D$ сопоставим номер – натуральное число $i, 1 \le i \le N$, где $N = |D|$, Слайсы, составляющее множество *D*, оказываются таким образом пронумерованными от *1* до *N*. Определим подмножество

$$D_i^j \subseteq D,\ i \in \overline{1, N},\ j \in \overline{1, N},\ i < j,\ \left|D_i^j\right| = j - i + 1$$

как множество слайсов с последовательными номерами от i до j. Очевидно, что $D = D_1^N$.

Пул работ D_m^n полностью определяется номерами m, n первого и последнего слайсов соответственно и имеет мощность $|D_m^n| = n - m + 1$. Определим функцию $Chip(D_m^n, k)$ разделения множества D_m^n на два подмножества D_m^{m+k-1} и D_{m+k}^n. Функция $Chip(D_m^n, k)$ отделяет от некоторого исходного множества упорядоченных слайсов D_m^n подмножество D_m^{m+k-1} упорядоченных слайсов мощностью k, получая при этом оставшееся подмножество D_{m+k}^n упорядоченных слайсов мощностью n-m-k+1.

Определим на множестве натуральных чисел $\overline{1,N}$ функцию $S(x)$ с областью значений D. Аргументом функции S является номер x некоторого слайса $d \in D$, а значением является сам слайс d. Аналогичным образом на множестве D определим обратную функцию $S'(d)$, аргументом которой является некоторый слайс $d \in D$, а значением – порядковый номер этого слайса x $\in \overline{1,N}$. Очевидно, что $S'\big(S(x)\big) = x$.

При помощи функции $Chip$ возможно организовать рекурсивное разделение исходного множества D_1^N на подмножества упорядоченных слайсов (пулы работ) мощностью n_l, где $l = \overline{1,r}$ – это уровень иерархии (рекурсии). Любой слайс из пула работ любого уровня рекурсии может быть получен при помощи функции S. С другой стороны, для любого слайса $d \in D$ при помощи функции $S'(d)$ можно однозначно определить его номер и подмножество $D_{i_1 \dots i_r}$, которому принадлежит этот слайс.

Метод иерархического разделения данных заключается в следующем. Пусть в качестве пула входных данных для некоторого менеджера M_l^m выступает подмножество D_1^n, где $n = |D_1^n| = n_l$. Менеджер M_l^m отделяет от своего пула бот D_1^n очередную порцию D_i

$$D_i = D_{1+i \cdot n_{l+1}}^{(i+1)n_{l+1}},\ i \in \overline{1,k_{l+1}},\ k_{l+1} = \left\lceil \frac{n_l}{n_{l+1}} \right\rceil,$$

которая передается для выполнения первому свободному менеджеру нижестоящего уровня M_{l+1}^j, не занятому в этот момент вычислительной работой. Если все

менеджеры нижестоящего уровня заняты работой, менеджер M_l^m переходит в состояние ожидания результатов.

Для каждого менеджера нижестоящего уровня M_{l+1}^j, выполнившего переданную ему порцию работ, вышестоящий менеджер M_l^m сохраняет результат обработки порции, отделяет от остатка пула входных данных очередную порцию D_i и передает её для выполнения освободившемуся менеджеру M_{l+1}^j. Если очередную порцию работы отделить нельзя по причине пустого остатка D_l, то запускается фаза перераспределения работы. Смысл этой фазы состоит в том, чтобы каждая еще не обработанная к текущему моменту порция работы D_i была бы вновь назначена (перераспределена) освободившимся, т.е. уже выполнившим свои порции менеджерам. Алгоритм перераспределения следующий.

1. Пусть в некоторый момент фазы перераспределения освободился менеджер M_{l+1}^j, выполнив очередную порцию работы. Менеджер M_l^m проверяет, была ли перераспределена порция работы освободившегося менеджера M_{l+1}^j.

2. Если порция работы освободившегося менеджера M_{l+1}^j была перераспределена некоторому менеджеру M_{l+1}^f, то менеджер M_{l+1}^f извещается о том, что перераспределенная порция работы уже выполнена и необходимо прекратить её обработку.

3. Назначается шаг перераспределения *t*, ему присваивается начальное значение 1.

4. Если $t > c_{l+1}$, то завершение алгоритма.

5. Если порция менеджера M_{l+1}^t была обработана или перераспределена, то увеличиваем *t* на 1 и переходим к шагу 4.

4. Менеджеру M_{l+1}^j перераспределяется порция работы менеджера M_{l+1}^t. Завершение алгоритма.

Алгоритм повторяется до тех пор, пока все выданные и перераспределенные порции не будут обработаны. После этого пул работ D_1^n менеджера M_l^m считается выполненным, и менеджер M_l^m возвращает результат работы менеджеру вышестоящего уровня *l-1*.

Метод разделения данных определил архитектуру вычислительной среды, создаваемой программным комплексом «Пирамида». На основе доступных ВУ суперкомпьютера ПК «Пирамида» формирует *r*-уровневую иерархию вычислителей, показанную на рисунке 60. Первый уровень иерархии соответствует вычислителю *C*, выполняющему обработку множества данных *D*. Каждый следующий уровень соответствует множеству вычислителей, обрабатывающих подмножества множества данных предыдущего уровня. Уровню *r* соответствуют n_r вычислителей (на рисунке 60 обозначены как c^t), выполняющих обработку пулов работ $\{D_{i_1 \dots i_r}\}$.

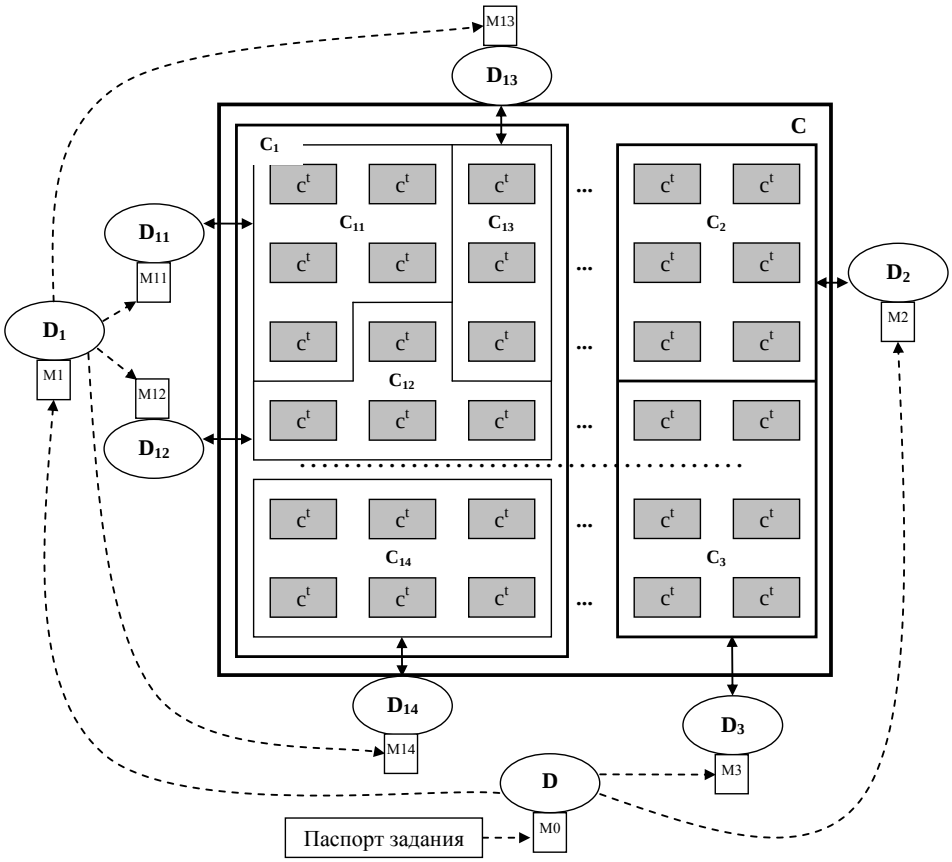

Рисунок 60. Пример структуры вычислительной среды, организованной на базе ВУ суперкомпьютера средствами ПК «Пирамида»

Нижний уровень иерархии *r* может соответствовать различным видам вычислителей: вычислительная система, вычислительный узел, микропроцессор, процессорное ядро.

Распределение данных для обработки вычислителями осуществляют менеджеры соответствующего уровня иерархии. Из пула работ, выделенных вычислите-

лю уровня l, менеджер M_l формирует пулы работ для вычислителей уровня l+1 с применением рассмотренного метода разделения работ.

В обозначении на рисунке 60 c^t (вычислителя уровня r) индекс t указывает его тип. $t \in \{t\}$, где $\{t\}$ – множество типов вычислителей. Пользователь должен предусмотреть подготовку вариантов исполняемых модулей своей последовательной программы для всех типов вычислителей, включаемых в состав вычислительной среды. Обозначив как $W^t(D_{i_1 \dots i_r})$ результат преобразования, выполняемого программой W^t над слайсом $\mathrm{D}_{i_1 \dots i_r}$ на вычислителе c^t, можно записать требование, предъявляемое к различным вариантам исполняемых модулей программы:

$$\forall t1, t2 \in \{t\}\ (W^{t1}(\mathrm{D}_{i_1 \dots i_r}) = W^{t2}(\mathrm{D}_{i_1 \dots i_r}))$$

Менеджеры, с учетом типа вычислителя t, осуществляют выбор варианта исполняемого модуля для запуска экземпляра программы, выполняющей выделенный ей пул работ $W^t(\{\mathrm{D}_{i_1 \dots i_r}\})$.

Отметим важные преимущества предложенного метода иерархического разделения данных.

Во-первых, контрольная точка на каждом уровне иерархии l в каждый момент времени будет определяться невыполненной на этот момент времени работой, которую составляют:

– текущий остаток невыполненной работы $D_l = D^n_{i \times n_{l+1}+1}$, который полностью определяется номерами граничных слайсов $i \times n_{l+1} + 1$ и n;

– множество из c_{l+1} порций работы, розданных менеджерам нижестоящего уровня M^j_{l+1}, причем каждая порция работы также полностью будет определяться номерами первого и последнего слайсов.

Таким образом, для записи контрольной точки достаточно зафиксировать границы текущего остатка и границы каждой порции розданной работы. Вся остальная вычислительная работа из пула D^n_1 в этот момент времени уже заведомо выполнена. Для восстановления менеджера M^m_l с контрольной точки достаточно восстановить

текущий остаток D_l и розданные менеджерам M_{l+1}^j порции работы, которые будет необходимо повторно передать этим менеджерам для выполнения.

Рассмотренный подход делает контрольную точку компактной, а ее фиксацию быстрой. При этом отпадает необходимость ведения базы данных учета обработанных и необработанных порций данных. На каждом уровне иерархии l в каждый момент времени мы имеем зафиксированную компактную контрольную точку, полностью определяющую оставшуюся для выполнения вычислительную работу. Этим обеспечивается минимизация накладных расходов на распараллеливание, что подтверждается данными экспериментов, приведенными ниже в настоящей главе.

Во-вторых, автоматически обеспечивается отказоустойчивость вычислений. При отсутствии ответа от любого менеджера нижестоящего уровня, его текущая порция вычислительной работы будет гарантированно выполнена на фазе перераспределения. Поэтому если какой-либо контролируемый этим менеджером вычислительный узел или вычислитель выходит из строя, вычисления будут выполнены другими вычислителями или узлами с некоторым снижением общей производительности.

В-третьих, аналогично отказоустойчивости обеспечивается масштабируемость вычислений. При появлении в решающем поле новых узлов или вычислителей достаточно добавить в систему менеджеров соответствующего уровня иерархии. Асинхронный механизм распределения работы в предложенном методе разделения данных обеспечит динамическое подключение новых подчиненных менеджеров, которым автоматически начнут выделяться порции вычислительной работы.

В-четвертых, обеспечивается возможность организации вычислений на гетерогенных вычислителях, вычислительные узлы которых имеют архитектурные различия (например, построены на программно-несовместимых микропроцессорах) или (и) состоят из вычислителей различного типа (например, вычислительные узлы содержат универсальные и графические процессоры). Разница в производительности ВУ будет автоматически компенсироваться асинхронным механизмом распределения работы, поскольку очередная порция будет передаваться

соответствующему менеджеру только после его освобождения. Чем быстрее будет производить расчеты соответствующий менеджеру вычислитель, тем больше порций работы получит этот менеджер.

5.4 Архитектура программного комплекса «Пирамида»

Вычислительная система, работающая под управлением комплекса «Пирамида», логически организована в иерархическую структуру вычислителей, содержащих отдельные ВУ или вычислители (ВЧ) нижнего уровня иерархии. Для каждого вычислителя назначается сервер управления (сервер вычислителя – СВЧ), а для вычислительной системы в целом – центральный сервер управления (сервер ВС), как показано на рисунке 61. Такая вычислительная система может быть отдельным суперкомпьютером, а может являться подсистемой ВУ, выделенных СУЗ для очередного задания. В последнем случае в качестве серверов ВС и серверов вычислителей могут выступать назначенные ВУ суперкомпьютера из состава подмножества ВУ, выделенных для задания.

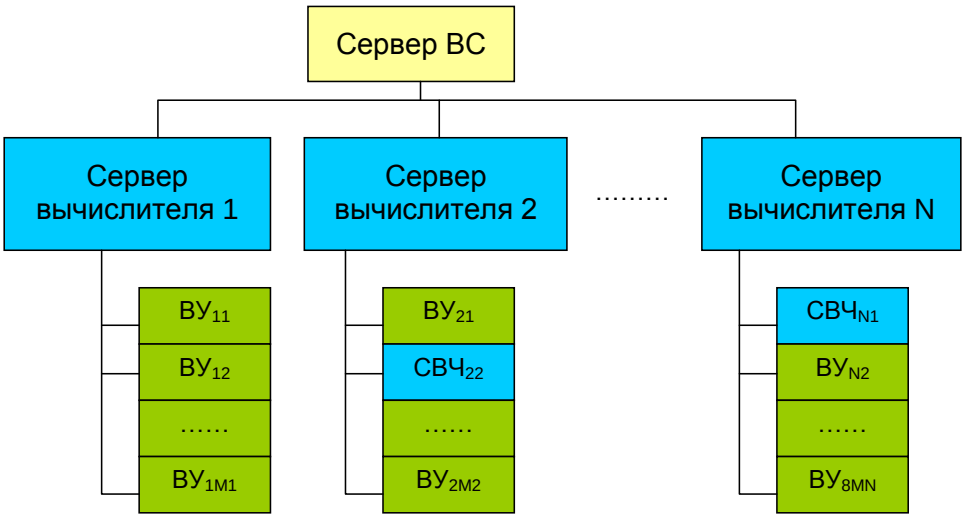

Рисунок 61. Структура иерархической массово-параллельной ВС

Управляющие процессы ПК «Пирамида» – менеджеры – выполняются на сервере ВС (центральный менеджер, он же менеджер задания), серверах вычислителей (менеджер вычислителя) и вычислительных узлах (менеджер ВУ – МВУ). Менеджеры комплекса «Пирамида», взаимодействуя друг с другом, осуществляют запуски ОПП на ВУ системы со всеми допустимыми экземплярами данных, формируемыми

на основе заданных пользователем значений параметров. Иерархическая структура системы менеджеров комплекса «Пирамида» приведена на рисунке 62.

Менеджер задания получает пул работ задания из паспорта и разделяет этот пул при помощи рассмотренного в п. 5.3 метода. Выделяемые менеджером задания порции данных становятся пулом работ для соответствующего вычислителя, менеджер которого повторяет алгоритм метода разделения данных на своем уровне иерархии. Наконец, менеджер ВУ, получая пул работ от менеджера вычислителя, повторяет алгоритм разделения на порции, при этом каждая такая порция определяет конкретные значения параметров ОПП. Таким образом, порция работы на уровне менеджера ВУ выполняется путем запуска на контролируемом узле одного или нескольких экземпляров ОПП. Каждый из запускаемых экземпляров получает на вход параметры, значения которых определяются соответствующей очередной порцией данных, отделенной менеджером ВУ от своего текущего пула работ. Исполняемый модуль ОПП при этом должен соответствовать архитектуре ВУ.

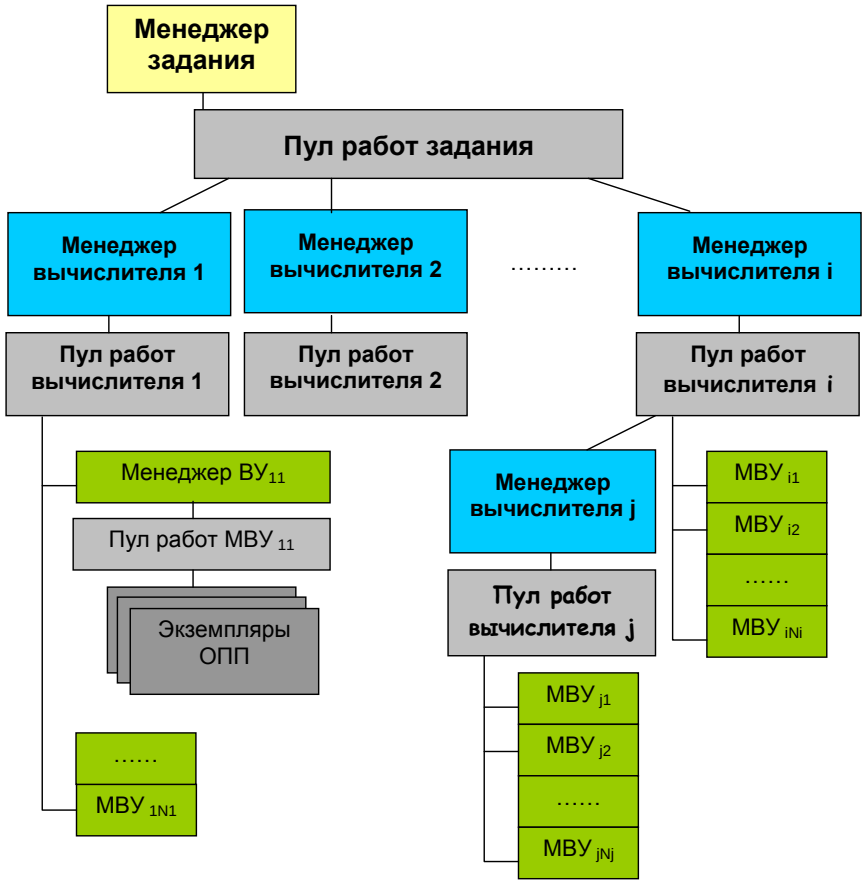

Рисунок 62. Архитектура ПК «Пирамида»

Менеджеры ПК «Пирамида» имеют схожую структуру, которая на примере менеджера вычислителя показана на рисунке 63. Менеджер вычислителя реализован как многопоточное приложение. Главный поток осуществляет прием вычислительной работы от центрального менеджера, сбор и отправку результатов. По одному потоку выделяется на каждый подчиненный менеджер ВУ или вычислителя.

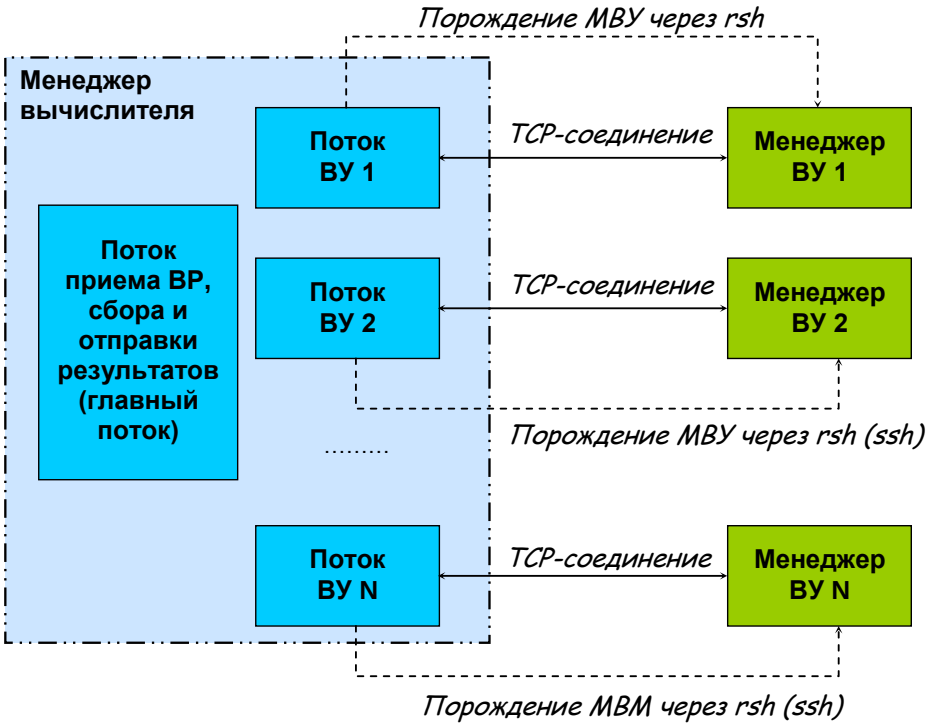

Рисунок 63. Структура менеджера вычислителя

Подчиненный менеджер выступает по отношению к породившему его потоку в роли сервера. После порождения подчиненного менеджера соответствующий поток менеджера вычислителя пытается соединиться с ним в режиме клиента. При успешном соединении поток менеджера вычислителя в соответствии с рассмотренном в п. 5.3 методом начинает передавать подчиненному менеджеру порции вычислительной работы и получать в ответ результаты. Заметим, что использование в качестве сервера именно подчиненного менеджера, а не менеджера вычислителя, позволяет избежать проблемы «узкого горла», когда большое число клиентов (подчиненных менеджеров) с различных ВУ одновременно пытаются соединиться с единственным сервером (менеджером вычислителя).

Целостность установленного соединения непрерывно контролируется. При разрыве соединения соответствующий поток ВУ менеджера вычислителя пред-

принимает меры по его восстановлению. При невозможности восстановления предпринимается попытка перезапуска подчиненного менеджера через процесс удаленного выполнения команд, например, rsh или ssh.

В случае выхода из строя произвольного числа вычислительных узлов или вычислителей выполнение задания не прерывается, и полученные ранее результаты не теряются. При восстановлении работоспособности вычислительных узлов производится их автоматическое подключение к процессу вычислений. Кроме этого, периодическое сохранение контрольных точек позволяет прервать и возобновить вычисления в произвольный момент времени на каждом уровне иерархии менеджеров.

5.5 Экспериментальное сравнение технологий распараллеливания по данным

5.5.1 Методика проведения экспериментов

Для оценки эффективности технологий распараллеливания по данным, используемых в разных программных комплексах, могут применяться различные методики. В работе [232] для оценки эффективности X-COM рассматриваются несколько вариантов методик, например, оценка эффективности расчетов по количеству избыточных вычислений и/или потерь, неизбежных при обеспечении надежности методом дублирования расчетов. В работе [233] производительность комплекса распараллеливания по данных предлагается определять как разницу между эталонным и реальным планами распределения ресурсов. В [234] для оценки производительности при применении MapReduce предлагается набор синтетических микротестов и реальных приложений Hadoop. При этом оценивается как время выполнения отдельного задания, так и пропускная способность – число заданий в минуту. В работе [235] оценивается время завершения программы MapReduce в зависимости от размера входного набора данных и заданных ресурсов кластера. В работах [236-238] для оценки производительности программных комплексов используется программа, вычисляющая частоту встречаемых в текстовом файле слов. В работе [237] автор применяет программу вычисления ча-

стоты используемых RGB-цветов в изображении и программу поиска заданной строки (подстроки) в файле. В [238] для оценки производительности используется программа вычисления рейтинга посещаемости Интернет-ресурсов.

В рамках настоящего исследования было проведено экспериментальное сравнение технологии распараллеливания по данным в ПК «Пирамида» с альтернативными технологиями, применяемыми в различном ПО. Сравнение производилось в два этапа, при этом методика экспериментов на втором этапе была усовершенствована по результатам первого.

На первом этапе в 2014 году ПК «Пирамида» сравнивался с параллельной программой, использующей коммуникационную библиотеку MPI, и технологией MapReduce в реализации Disco [239].

На втором этапе [240] в 2016 году ПК «Пирамида» сравнивался с программной платформой организации добровольных вычислений BOINC и программным комплексом X-COM.

В качестве экспериментального стенда была использована аппаратная платформа aTCA-8214 Series, состоящая из восьми вычислительных узлов, один из которых играл роль управляющей ЭВМ. Характеристики ВУ платформы aTCA-8214 Series приведены в таблице 19.

Таблица 19. Характеристики вычислительного узла платформы aTCA-8214 Series

Модель процессора	Intel Xeon L5408
Архитектура процессора	x64
Количество процессоров	2
Тактовая частота процессора	2,13 ГГц
Количество процессорных ядер	4
Количество потоков на ядро	1
Размер кэш-памяти второго уровня	12 Мб
Тип оперативной памяти	DDR2
Объём оперативной памяти	8 Гб

Эксперименты проводились путем запуска разных тестовых примеров, каждый из которых представлял собой одну последовательную программу (ОПП). На

первом этапе [239] исследований в качестве тестовых примеров были разработаны ОПП, получившие названия «Сочетания» и «Штурм». В программе «Сочетания» осуществляется перебор и вывод в стандартный вывод всех возможных комбинаций из трех диапазонов, переданных в качестве параметров. Программа производит случайную задержку в заданном интервале, моделирующую обработку входных параметров. Программа «Штурм» осуществляет подбор пароля пользователя ОС Linux. Программа генерирует пароли заданного алфавита в соответствии с переданным ей диапазоном, производит их хеширование и сравнение с эталонным хешэм, моделируя тем самым проверку системным администратором стойкости паролей пользователей.

Сравнение технологий распараллеливания по данным производилось по показателям производительности и отказоустойчивости.

Для оценки производительности определялось время, затраченное на обработку всех входных параметров. Для некоторых из сравниваемых технологий требовалась подготовка входных данных. Например, в технологии MapReduce, необходимо было произвести загрузку данных для расчетов в распределенную файловую систему. Если данные необходимо было загружать каждый раз при запуске ОПП, то это время обязательно учитывалось. Если же было возможно загрузить некоторые вспомогательные данные один раз и далее использовать их для всех запусков ОПП, то время загрузки в таком случае не учитывалось.

Эксперименты первого этапа проводились на аппаратной платформе aTCA-8214 Series с установленной средой виртуализации VMware vSphere, в которой были развернуты три виртуальных кластера, реализующие технологии MapReduce, MPI и ПК «Пирамида». Сравнение было проведено последовательно для каждого виртуального кластера. Количество процессорных ядер (56) и объем оперативной памяти (8 ГБ) были одинаковыми для всех сравниваемых технологий.

Для оценки производительности тестовые примеры запускались с несколькими наборами входных данных, каждый из наборов обеспечивал свой объем вычислений. Для каждого набора входных данных осуществлялось не менее десяти

запусков, после чего для оценки статистической значимости результатов рассчитывались математическое ожидание, дисперсия коэффициент вариации. При получении статистически значимых оценок определялось среднее значение для каждого варианта входных данных.

Для оценки отказоустойчивости сравниваемых технологий применялась следующая методика. Все потенциальные отказы можно разделить на три группы: связанные с перезагрузкой ВУ, связанные с аварийным отключением питания одного или нескольких ВУ и связанные с разрывом соединения в коммуникационной сети. Для имитации отказов первой группы во время вычислений осуществлялась перезагрузка одного из ВУ командой перезапуска виртуальной машины на панели инструментов гипервизора. Отказы второй группы имитировались выключением ВУ в процессе вычислений командой выключения виртуальной машины на панели инструментов гипервизора. Для имитации отказов третьей группы осуществлялось отключение виртуального коммутатора.

Результаты первого этапа экспериментов представлены в п. 5.5.2

На втором этапе [240] было принято решение оценить производительность и масштабируемость сравниваемых технологий путем определения зависимости величины накладных расходов на распараллеливание от числа используемых процессорных ядер и параметров ОПП. Накладные расходы на распараллеливание как доля времени, затрачиваемого каждым сравниваемым ПК на организацию вычислительного процесса, определялись следующим образом.

Введём понятие элементарной вычислительной работы, под которой будем понимать выполнение обработки неделимой (элементарной) порции входных данных. Элементарной работой могут быть обработка строки, сгенерированной на основе переданных параметров или считанной из файла, либо перебор вариантов значений в некотором минимальном диапазоне входных данных. Важно, что элементарная работа не может быть разделена на более мелкие части и, следовательно, не может быть распараллелена.

Пусть элементарная вычислительная работа выполняется за время τ, а весь объём входных данных составляют N элементарных порций. Следовательно, на одном процессорном ядре весь объём входных данных будет обработан за время $T_1 = N \cdot \tau$. При использовании p процессорных ядер идеальное время обработки T_p составит

$$T_p = \frac{T_1}{p} = \frac{N \cdot \tau}{p}$$

Пусть исследуемый ПК произвёл обработку входных данных на p процессорных ядрах за время $T_{экс}(p)$. Тогда доля накладных расходов μ, привносимых ПК, составит

$$\mu = 1 - \frac{T_p}{T_{экс}(p)} = 1 - \frac{N \cdot \tau}{p \cdot T_{экс}(p)} \qquad (14)$$

Как видим, доля накладных расходов зависит от параметров N, τ и p. Варьируя значения одного из параметров при фиксированных значениях двух других, можно оценить динамику изменения накладных расходов на организацию распараллеливания по данным для каждого из исследуемых ПК.

Для экспериментального определения накладных расходов были разработаны тестовые примеры ОПП, удовлетворяющие следующим требованиям.

1. Тестовая ОПП должна иметь возможность обработки от одной до произвольного числа элементарных порций входных данных, т.е. быть способной выполнить произвольное число N элементарных работ.

2. Для тестовой ОПП должна быть предусмотрена возможность задания времени τ выполнения элементарной работы.

3. Если тестовая ОПП выполняет N элементарных работ, за время τ каждую, то время выполнения ОПП на одном процессорном ядре должно составлять $N \cdot \tau$.

4. Обрабатываемая тестовой ОПП порция данных должна полностью определяться значениями её параметров.

5. Случай, когда обрабатываемая ОПП порция данных определяется значением (диапазоном значений) только одного параметра, встречается на практике

довольно часто, поэтому целесообразно при сравнении ПК рассмотреть этот случай отдельно с использованием соответствующего тестового примера.

6. Отдельно следует рассмотреть случай, когда обрабатываемая ОПП порция данных определяется комбинацией значений (диапазонов значений) нескольких параметров, что существенно повышает трудоёмкость организации параллельных вычислений. Заметим, что именно этот случай является типичным при практическом применении ПК «Пирамида».

7. Отдельно следует рассмотреть случай, когда обрабатываемая ОПП порция данных определяется строкой (диапазоном строк) файла входных данных. Этот случай является типичным при применении ПК X-COM и BOINC.

Приведенная методика проведения экспериментов на втором этапе хорошо согласуется в рассмотренной в [232] оценкой производительности как отношением времени, затраченного на организацию и поддержку расчетов, ко времени проведения самого расчета.

В качестве ОПП, удовлетворяющих выдвинутым требованиям, были разработаны три тестовых примера.

Тестовый пример Opp_one моделирует работу ОПП с одним входным параметром, значение которого задаёт диапазон входных данных в виде тройки чисел "a b s", где a – начало диапазона, b – конец диапазона, s – шаг. Например, тройка "10 20 3" задаёт последовательность перебираемых чисел 10, 13, 16, 19, каждое из которых определяет элементарную порцию данных. Для теста Opp_one есть возможность указания времени τ обработки одной элементарной порции данных. Например, если указать τ = 1 с, то время обработки порции данных, заданной значением входного параметра "10 20 3", составит 4 с.

Тестовый пример Opp_three моделирует работу ОПП с тремя входными параметрами. Значения первых двух параметров задают диапазоны входных данных в виде троек чисел "a b s", где a – начало диапазона, b – конец диапазона, s – шаг. Значение третьего параметра задаёт список перебираемых строк. Например, если значение первого параметра – "10 15 3", значение второго – "1 2 1", значение тре-

тьего – "str1 str2", то порцию входных данных будут определять 8 возможных комбинаций значений трёх параметров:

10 1 str1

10 1 str2

10 2 str1

10 2 str2

13 1 str1

13 1 str2

13 2 str1

13 2 str2

Так же как для теста Opp_one, для теста Opp_three предусмотрена возможность указания времени τ обработки одной комбинации значений входных параметров, т.е. времени обработки одной элементарной порции данных. Для нашего примера при $\tau = 1$ с время обработки 8 комбинаций составит 8 с.

Тестовый пример Opp_file моделирует обработку строк, считываемых из переданного ему в качестве параметра файла. Программа принимает на вход два параметра – имя файла со строками и время τ обработки одной строки из файла. Например, если в файле 20 строк, и время τ обработки одной строки равняется 2 с, то время работы тестового примера Opp_file на одном процессорном ядре составит 40 с.

На втором этапе экспериментов для оценки отказоустойчивости сравниваемых ПК была применена методика, аналогичная методике первого этапа. Дополнительно, помимо имитации отказа одного ВУ, была произведена имитация отказа всех ВУ решающего поля и управляющей машины. После выключения ВУ они запускались повторно, и осуществлялась попытка продолжить прерванные вычисления.

Результаты экспериментов второго этапа представлены в п. 5.5.3

5.5.2 Результаты сравнения ПК «Пирамида», MapReduce и MPI

При сравнении ПК «Пирамида», MapReduce и MPI возникла дополнительная сложность. Если для запуска тестовых примеров под управлением ПК «Пирамида» достаточно составить паспорт задания, то для MapReduce и MPI требуется разработка управляющих программ. От качества этих программ во многом зависит итоговая производительность вычислений. Чтобы обеспечить сравнительно одинаковое качество для разных технологий, было принято во внимание следующее предположение.

Как уже упоминалось, освобождение пользователя от рутинной работы по организации параллельных вычислений являлось одной из целей создания ПК «Пирамида». Другими словами, комплекс позволяет производить расчеты пользователям, которые не являются глубокими специалистами в параллельном программировании. Логичным выглядит использование для экспериментов управляющих программ для MapReduce и MPI, разработанных «начинающим» программистом. В качестве такого «начинающего» специалиста выступил соавтор работы [239] А.В. Алексеев, который на момент проведения экспериментального исследования являлся студентом выпускного курса. Его усилиями были разработаны соответствующие управляющие программы для MapReduce и MPI.

Рассмотрим организацию параллельных вычислений с использованием MapReduce-кластера Disco [228]. Основу кластера составляет распределенная файловая система Disco – DDFS, которая позволяет хранить объекты различного формата от текстовых данных системных журналов до больших файлов мультимедиа. DDFS представляет собой распределенное хранилище данных, представленных в формате «ключ-значение». DDFS оперирует понятиями тега и объекта. Тег является ключом, а объект, в свою очередь, значением ключа. Один тег может ссылаться на несколько объектов, и таким образом можно удобно организовать хранение различных объектов, характеризующих одну сущность.

Для работы ОПП «Штурм» в составе кластера необходимо запускать её экземпляры на каждом ВУ и передавать им соответствующие порции данных для пе-

ребора. После получения входного пула работ управляющая программа должна разбить его на порции, которые будет перебирать каждый экземпляр ОПП «Штурм». Затем необходимо поместить в DDFS объекты, количество которых будет равно числу порций. В каждом объекте будут записаны начальный и конечный номер пароля для перебора в конкретной порции.

Операция загрузки объектов с порциями занимает значительное время и поэтому каждый раз при решении задачи перебора в DDFS один раз размещалось большое количество объектов, содержащих числа от 0 до N, где N – большое число порядка 10^{12}. Управляющая программа в ходе вычислений, с помощью специально разработанной функции преобразует эти числа в соответствующие порции для перебора и передает их ОПП «Штурм» в качестве входных данных.

Для работы второго тестового примера, ОПП «Сочетания», в составе кластера Disco было необходимо реализовать разделение данных на поддиапазоны для обработки ВУ кластера. Разбиение на поддиапазоны производилось по значениям первого параметра, которым также сопоставлялись числа от 0 до N, содержащиеся в заранее записанных в DDFS объектах.

В управляющих программах с использованием MPI было применено разделение вычислительной работы по рангам MPI-процессов, значения которых определяли диапазоны порций вычислительной работы для тестовых примеров. Для ОПП «Штурм» каждый MPI-процесс осуществлял генерацию очередной порции данных для передачи ПС «Штурм», затем происходило ожидание завершения обработки первой порции всеми процессами, и осуществлялся сбор результатов. В противном случае происходили генерация новой порции и ее передача экземпляру ПС «Штурм».

Работа ОПП «Сочетания» с использованием MPI была организована путем разбиения данных на поддиапазоны по значениям первого параметра. Каждый MPI-процесс осуществлял обработку очередного значения первого параметра, перебирая значения остальных двух. После обработки всеми MPI-процессами своего очередного значения первого параметра осуществлялось сохранение контрольной

точки, после чего каждый MPI-процесс переходил к обработке следующего значения первого параметра. Сохранение контрольных точек позволило обеспечить отказоустойчивость вычислений с применением MPI.

Следует отдельно отметить, что перед началом основной серии экспериментов для каждой из сравниваемых технологий была проведена предварительная серия с целью подбора такого размера порции для обработки, при котором та или иная технология демонстрирует лучшую производительность. Другими словами, была произведена оптимизация конфигурации входных данных для каждой технологии распараллеливания.

Результаты экспериментов для тестового примера «Штурм» представлены в таблице 20. Для всех технологий для выполнения тестовых наборов данных использовались 56 процессорных ядер платформы aTCA-8214 Series.

Таблица 20. Время выполнения тестового примера «Штурм» при применении технологий ПК «Пирамида», MapReduce и MPI

Набор данных	Искомый пароль	Время выполнения теста, с		
		MapReduce	MPI	Пирамида
1	cwWqM	151	125	7,33
2	fUJIo	482	250	203,46
3	mGkhC	738	549	500,74
4	GY5Zi	1546	1237	1150,70

Результаты экспериментов для тестового примера «Сочетания» представлены в таблице 21. Для всех технологий для выполнения тестовых наборов данных использовались 56 процессорных ядер платформы aTCA-8214 Series.

Результаты экспериментов по сравнению производительности значительно различаются для двух тестовых примеров «Штурм» и «Сочетания». Для случая, когда в ОПП один динамический параметр (ОПП «Штурм»), все сравниваемые технологии распараллеливания показывают примерно одинаковое быстродействие с некоторым преимуществом ПК «Пирамида».

Таблица 21. Время выполнения тестового примера «Сочетания» при применении технологий ПК «Пирамида», MapReduce и MPI

Номер набора данных	Диапазон входных данных	Время выполнения теста, с		
		MapReduce	MPI	Пирамида
1	«0 10000» «0 5000» «str1 str2 str3»	40	37	8,92
2	«0 30000» «0 5000» «str1 str2 str3»	130	109	19,13
3	«0 60000» «0 5000» «str1 str2 str3»	267	218	35,24
4	«0 120000» «0 5000» «str1 str2 str3»	542	437	64,18

В случае перебора трех параметров (ОПП «Сочетания») наблюдается кратное преимущество ПК «Пирамида». Это логичный и ожидаемый результат, поскольку ПК «Пирамида» изначально рассчитан на организацию вычислений с перебором всех возможных комбинаций нескольких параметров ОПП при помощи рассмотренного в п. 5.3 метода иерархического разделения данных. При использовании ПК «Пирамида» прикладной пользователь имеет возможность гибкой настройки распределения вычислительной работы по уровням иерархии вычислителей, за счет чего минимизируется число запусков ОПП, и повышается быстродействие.

Технологии, основанные на моделях MapReduce и MPI, подобной возможностью не обладают, и работа по эффективной организации параллельных вычислений целиком ложится на пользователя-программиста. В случае, когда пользователь не является специалистом в области параллельных вычислений и сосредоточен на решении своей прикладной задачи, применение ПК «Пирамида» может дать кратный эффект по быстродействию при распараллеливании по данным.

Результаты оценки отказоустойчивости сравниваемых технологий для трех вариантов аварийных ситуаций собраны в таблице 22.

Сравнение технологий распараллеливания по отказоустойчивости показало ожидаемые результаты. Две технологии (ПК «Пирамида» и MapReduce), в которые разработчиками были заложены механизмы отказоустойчивости, продемон-

стрировали одинаковое поведение: при возникновении отказов вычислительная работа перераспределяется между рабочими ВУ. После восстановления вычислительных узлов им автоматически начинает выделяться вычислительная работа. При распараллеливании при помощи MPI-программы отказоустойчивость вычислений фактически целиком зависит от заложенных разработчиком в эту программу возможностей.

Таблица 22. Результаты оценки отказоустойчивости технологий ПК «Пирамида», MapReduce и MPI

Аварийная ситуация	Реакция программного средства		
	MapReduce	MPI	Пирамида
Перезагрузка вычислительного узла	После перезагрузки ВУ ему автоматически выделяется вычислительная работа. При отказе управляющей машины вычисления прекращаются.	Вычисления аварийно завершаются	После перезагрузки ВУ ему автоматически выделяется вычислительная работа. При отказе управляющей машины вычисления прекращаются.
Выключение вычислительного узла	Вычислительная работа перераспределяется между рабочими узлами. При отказе управляющей машины вычисления прекращаются.	Вычисления аварийно завершаются	Вычислительная работа перераспределяется между рабочими ВУ. При отказе управляющей машины вычисления прекращаются.
Сбой в работе сети	Управляющая машина пытается связаться с ВУ. После нескольких попыток, если соединение не было установлено, вычисления прекращаются.	Вычисления аварийно завершаются	Управляющая машина пытается связаться с ВУ. После нескольких попыток, если соединение не было установлено, вычисления прекращаются.

5.5.3 Результаты сравнения ПК «Пирамида», BOINC и X-COM

В ходе проведения экспериментов для каждого из сравниваемых комплексов были осуществлены замеры времени выполнения трёх тестовых примеров – Opp_one, Opp_file и Opp_three, после чего по формуле (14) были вычислены накладные расходы на организацию распараллеливания по данным. Измерения времён выполнения для каждого тестового примера были проведены в три этапа:

– при переменном числе процессорных ядер p и постоянных объёме вычислений в N элементарных порций и времени τ обработки элементарной порции данных;

– при переменном τ и постоянных p и N;

– при переменном N и постоянных p и τ.

На рисунке 64 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_one перебора значений одного параметра при переменном числе процессорных ядер, объёме данных в 10^5 элементарных порций, времени обработки элементарной порции 1 с. Отметим рост накладных расходов ПК BOINC с 6% до 36%. У ПК X-COM и «Пирамида» наблюдается медленный рост накладных расходов с 2% до 6-7%.

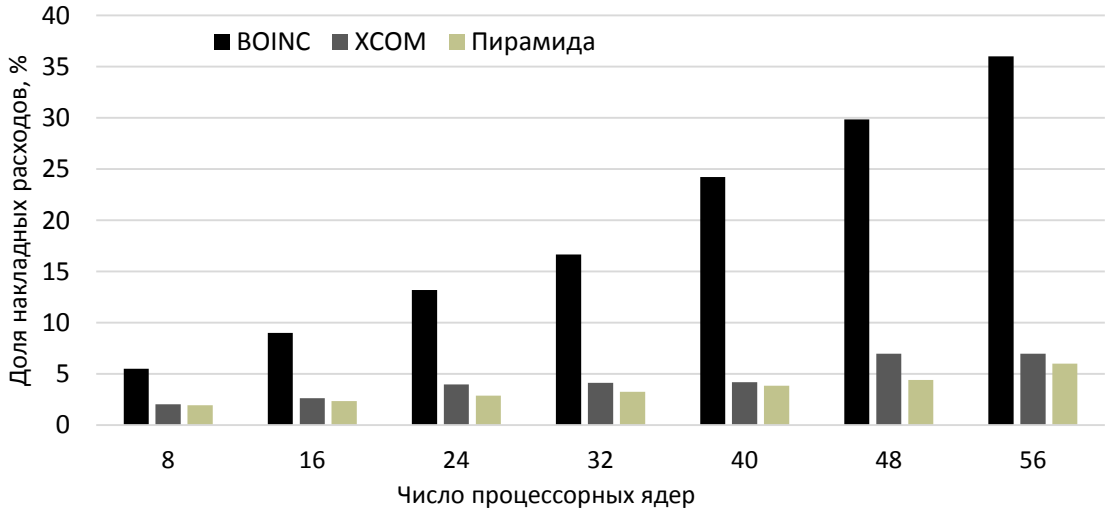

Рисунок 64. Накладные расходы для перебора значений одного параметра, $N = 10^5$ элем. порций, время обработки одной элем. порции $\tau = 1$ с

На рисунке 65 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_one перебора значений одного параметра при переменном значении времени обработки элементарной порции данных, объёме данных в 10^5 элементарных порций, числе процессорных ядер 56. При увеличении времени обработки элементарной порции у ПК BOINC отмечается снижение накладных расходов с 51% до 20%, после чего накладные расходы устанавливаются у отметки в 18-20%. У ПК X-COM и ПК «Пирамида» доля накладных расходов плавно снижается с 12% до 2%.

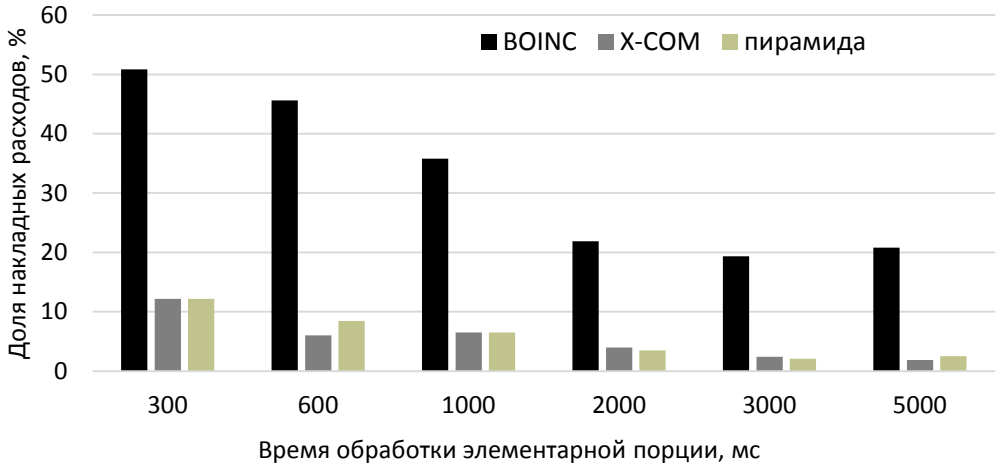

Рисунок 65. Накладные расходы для перебора значений одного параметра, $N = 10^5$ элем. порций, число ядер $p = 56$

На рисунке 66 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_one перебора значений одного параметра при переменном объёме данных, числе процессорных ядер 56, времени обработки элементарной порции 1 с. Наблюдается уменьшение накладных расходов ПК BOINC с 34% до 20%, после чего снижение останавливается у отметки в 20%. У ПК X-COM и ПК «Пирамида» накладные расходы плавно снижаются с 6% до 1%.

Представленные на рисунках 65 и 66 результаты свидетельствуют о повышении эффективности сравниваемых ПК как при укрупнении зерна параллелизма, так и при увеличении объёма входных данных.

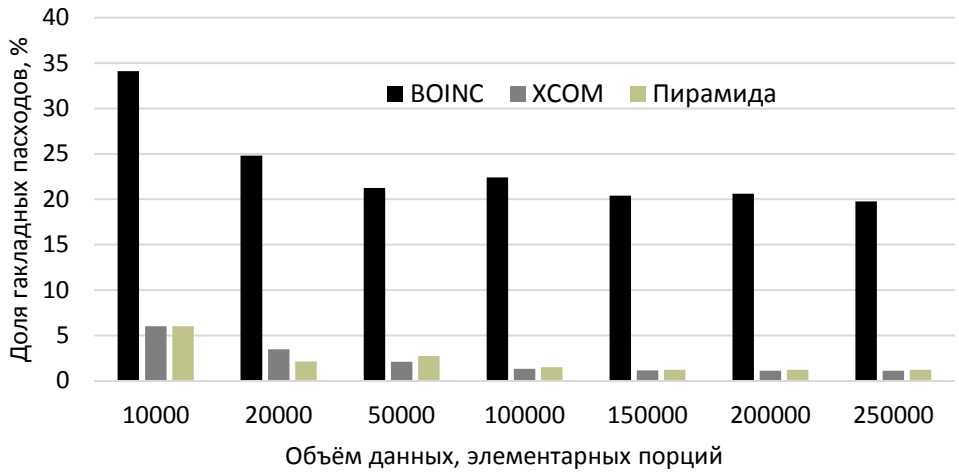

Рисунок 66. Накладные расходы для перебора значений одного параметра, число ядер p = 56, время обработки одной элементарной порции τ = 1 с

На рисунке 67 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_file перебора строк файла при переменном числе процессорных ядер. Объём данных во входном файле составил 10^5 строк, время обработки элементарной порции данных (одной строки файла) – 1 с. Наблюдается рост накладных расходов у всех сравниваемых ПК. У ПК X-COM доля накладных расходов увеличивается с 2% до 6%, у ПК «Пирамида» – с 7% до 14%, у ПК BOINC – с 5% до 36%.

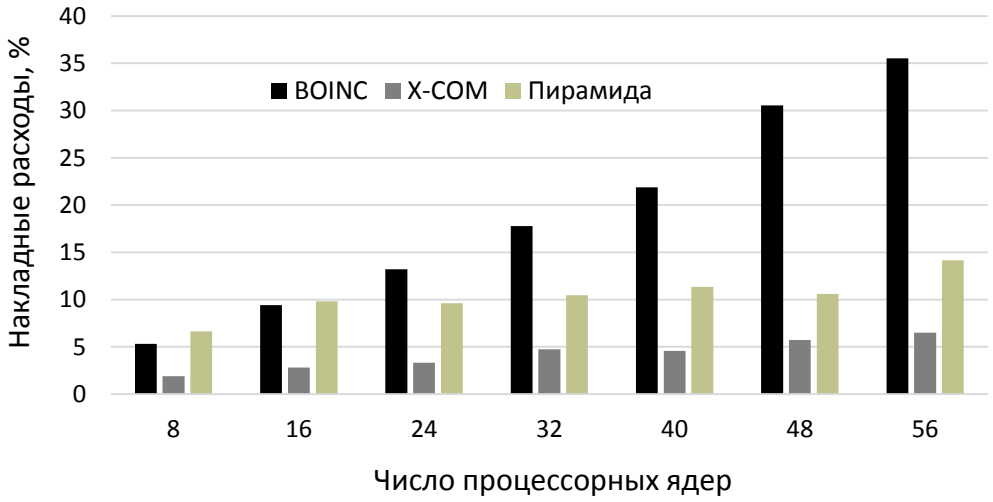

Рисунок 67. Накладные расходы для перебора строк файла, число строк $N = 10^5$, время обработки одной строки τ = 1 с

На рисунке 68 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_file перебора строк файла при переменном числе строк во входном файле, времени обработки одной строки 1 с, числе процессорных ядер 56. Наблюдается уменьшение накладных расходов ПК X-COM с 7% до 2%, при объёме файла 50 тыс. строк и более накладные расходы составляют 2%. Накладные расходы ПК «Пирамида» уменьшаются с 15% до 11% и при объёме файла 20 тыс. строк и более находятся в пределах 10-11%. Накладные расходы ПК BOINC сравнительно быстро снижаются при увеличении объёма файла до 100 тыс. строк, далее наблюдается незначительное изменение накладных расходов в пределах 20-22%.

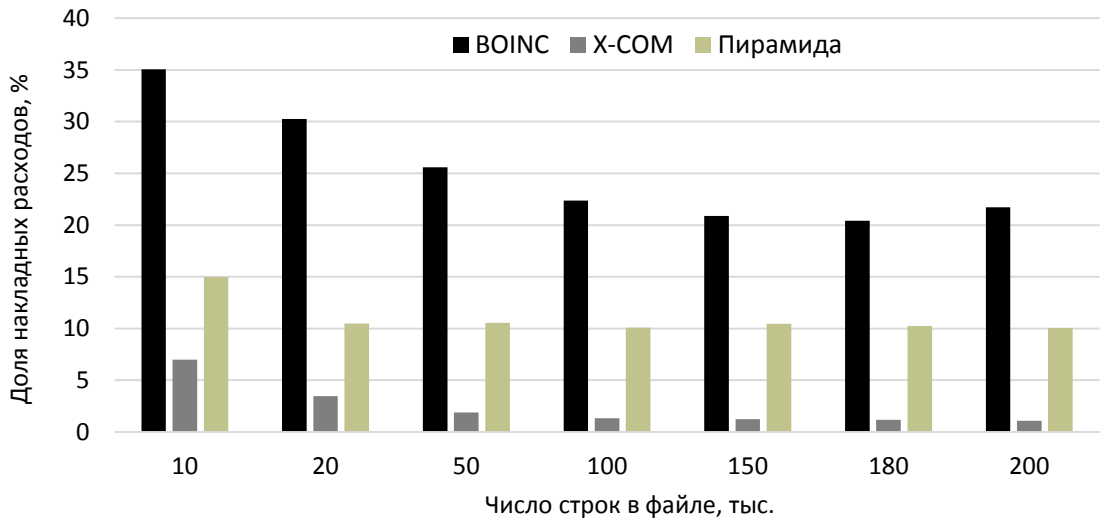

Рисунок 68. Накладные расходы для перебора строк файла,
число ядер p = 56, время обработки одной строки τ = 1 с

На рисунке 69 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_file перебора строк файла при переменном значении времени обработки одной строки, объёме данных 10^5 строк в файле и числе процессорных ядер 56. Наблюдается быстрое снижение накладных расходов ПК X-COM с 15% до 5%, далее они медленно снижаются до значения 2%. Доля накладных расходов ПК «Пирамида» медленно снижается с 16% до 6%. Накладные расходы ПК BOINC быстро снижаются при увеличении времени обработки элементарной

порции до 2000 мс, далее наблюдается незначительное изменение накладных расходов в пределах 20-22%.

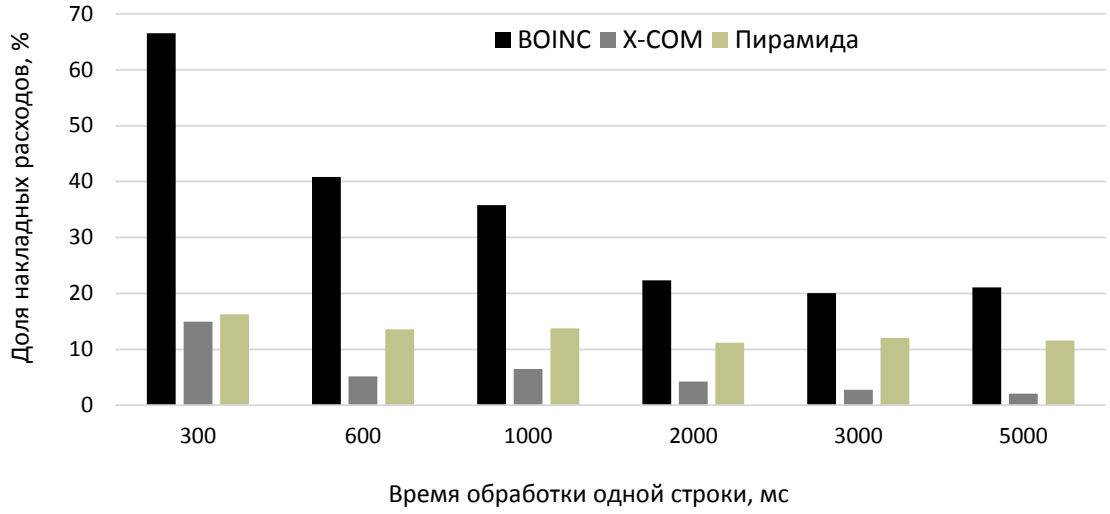

Рисунок 69. Накладные расходы для перебора строк файла при переменном времени обработки одной строки, $N = 10^5$ строк, число ядер $p = 56$

На рисунке 70 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_three перебора комбинаций значений трех параметров при переменном числе процессорных ядер, объёме данных в 10^5 элементарных порций (комбинаций значений), времени обработки элементарной порции (одной комбинации) 1 с. Наблюдается рост накладных расходов у ПК X-COM до значения 23%, затем наблюдается снижение до 13%. У ПК «Пирамида» наблюдается медленный рост величины накладных расходов с 2% до 5%. Накладные расходы ПК BOINC быстро растут с 11% до 33%.

На рисунке 71 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_three перебора комбинаций значений трех параметров при переменном значении времени обработки элементарной порции данных (одной комбинации), объёме данных в 10^5 комбинаций, числе процессорных ядер 56. Наблюдается снижение доли накладных расходов у ПК X-COM с 14% до 11%, у ПК «Пирамида» – с 9% до 1%, у ПК BOINC – с 54% до 24%.

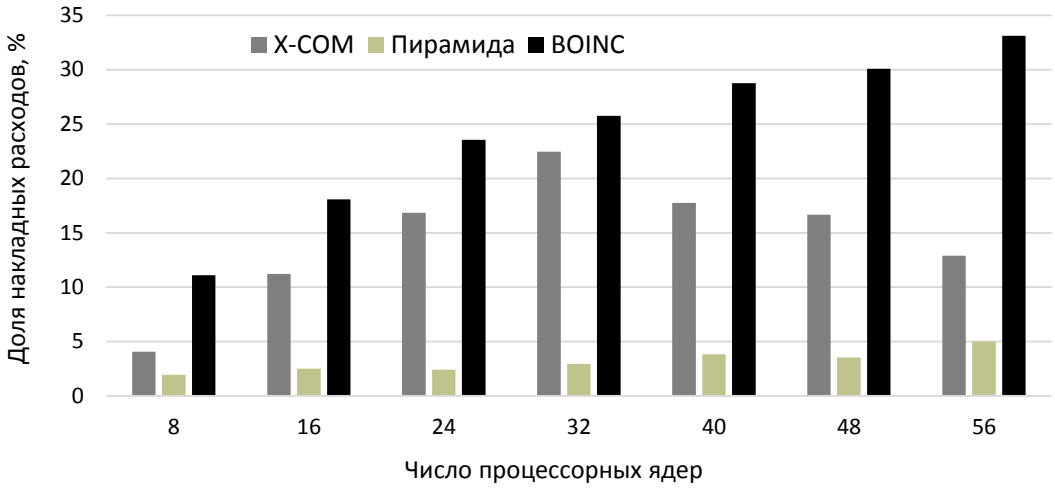

Рисунок 70. Накладные расходы для перебора комбинаций значений трех параметров, число комбинаций $N = 10^5$ элем. порций, время обработки одной комбинации $\tau = 1$ с

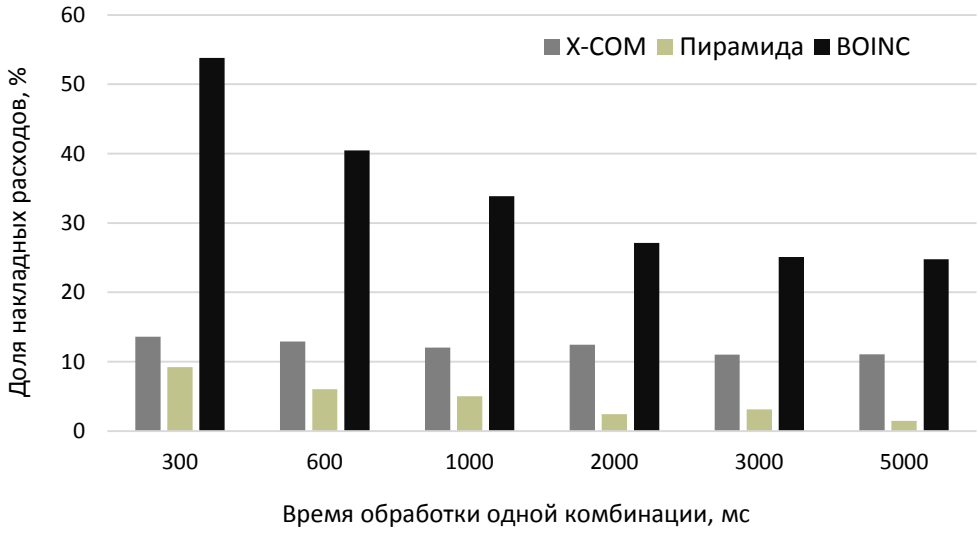

Рисунок 71. Накладные расходы для перебора комбинаций значений трех параметров, число ядер $p = 56$, число комбинаций $N = 10^5$ элем. порций

На рисунке 72 представлена динамика изменения накладных расходов на организацию распараллеливания ПК X-COM, BOINC и «Пирамида» для тестового примера Opp_three перебора комбинаций значений трех параметров при переменном объёме данных, числе процессорных ядер 56, времени обработки элементарной порции (одной комбинации) 1 с. Доля накладных расходов у ПК X-COM находится в диапазоне с 11% до 12%. Накладные расходы ПК «Пирамида» снижаются с 4% до 2%, ПК BOINC – с 34% до 20%.

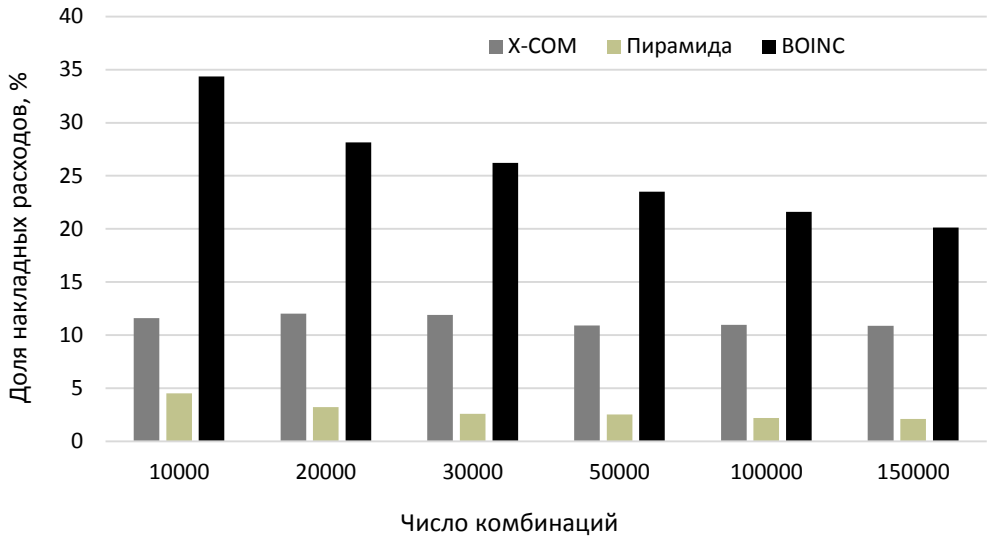

Рисунок 72. Накладные расходы для перебора комбинаций значений трех параметров, число ядер $p = 56$, время обработки одной комбинации $\tau = 1$ с

В ходе экспериментальных тестов на отказоустойчивость все сравниваемые ПК продемонстрировали одинаковую реакцию на все смоделированные согласно приведенной в п. 5.5.1 методике аварийные ситуации:

– после перезагрузки вычислительного узла, ему автоматически начинает выделяться вычислительная работа;

– при выключении ВУ вычислительная работа перераспределяется между рабочими узлами;

– при отказе управляющей машины вычисления прекращаются;

– при отключении сетевого интерфейса у управляющей машины она бесконечно ожидает соединения с вычислительными узлами;

– при отключении сетевого интерфейса у вычислительного узла вычислительная работа перераспределяется между рабочими узлами;

– при перезапуске всех ВУ, после попытки продолжить прерванные вычисления, вычисления продолжаются с момента сохранения последней контрольной точки.

Анализ результатов экспериментов позволяет сделать следующие выводы.

1. ПК BOINC показал во всех экспериментах кратно большие накладные расходы по сравнению с ПК «Пирамида» и X-COM. Связано это, прежде всего, со

сложной структурой комплекса, которую составляют множество компонентов, включая СУБД. ПК BOINC создавался для надёжного функционирования в распределённой среде, и его применение в рамках отдельных кластерных вычислительных систем нецелесообразно.

2. Накладные расходы всех ПК во всех тестовых примерах растут при увеличении числа процессорных ядер и, как правило, снижаются при увеличении объёма входных данных и времени обработки элементарной порции данных.

3. ПК X-COM показал кратно лучшие результаты при обработке строк, считываемых из входного файла, причём наивысшей эффективности (накладные расходы около 2%) комплекс достигает на больших файлах и при крупном зерне параллелизма.

4. ПК «Пирамида» показал кратно лучшие результаты при обработке нескольких параметров ОПП, что логично, поскольку этот ПК – единственный из исследуемых комплексов, обладающий встроенным механизмом перебора комбинаций значений нескольких параметров.

5. Все исследуемые ПК продемонстрировали одинаково высокие показатели отказоустойчивости.

5.6 Гибридный программный комплекс XP-COM

5.6.1 Архитектура и алгоритмы гибридного программного комплекса XP-COM

Рассмотренное в п. 5.5.3 экспериментальное сравнение программных комплексов X-COM и «Пирамида» показало кратно лучшие результаты ПК X-COM при обработке строк, считываемых из входного файла. Анализ возможностей ПК X-COM позволил сделать вывод, что этот комплекс обладает более совершенной транспортной подсистемой. В то же время ПК «Пирамида» показал кратно лучшие результаты при обработке комбинаций нескольких параметров ОПП. Это естественно, поскольку Пирамида обладает встроенным механизмом перебора комбинаций параметров, основанным на методе иерархического разделения данных. Для совмещения указанных достоинств ПК «Пирамида» и X-COM было

принято решение о разработке гибридного программного комплекса, получившего название XP-COM [113].

Для возможности запуска при помощи X-COM множества экземпляров ОПП необходимо разработать серверную и клиентскую управляющие программы. Серверная программа должна обеспечивать логику деления вычислительной работы на порции, а клиентская – запуск ОПП с параметрами, задающими очередную порцию работы. X-COM обеспечит передачу порций работы от сервера клиентам, а в обратном направлении – получение результатов. Сам X-COM не содержит механизмов деления работы и запуска ОПП, соответствующие управляющие программы должен разработать пользователь. Основная идея гибридного ПК XP-COM заключается в реализации в составе управляющих серверной и клиентских программ X-COM метода иерархического разделения данных и связанной с ними вычислительной работы, запуска множества экземпляров ОПП, а также автоматического сохранения контрольных точек.

Названные механизмы были заимствованы из исходных текстов ПК «Пирамида» и реализованы в виде следующих отдельных программных модулей.

1. Модуль деления вычислительной работы Divisio предназначен для выделения порций данных вычислительным клиентам по требованию сервера X-COM и сохранения контрольных точек через определенные промежутки времени. При выделении порций данных применяется рассмотренный в п. 5.3 метод.

2. Модуль запуска ОПП Divisio_client предназначен для преобразования полученной от сервера X-COM порции в параметры запуска ОПП и последующего запуска ОПП.

Внедрение указанных модулей в ПК X-COM было осуществлено через разработанные авторами [113] на языке Perl управляющие серверную и клиентскую программы, как показано на рисунке 73. Управляющие программы были разработаны в соответствии с правилами и руководством пользователя ПК X-COM [241].

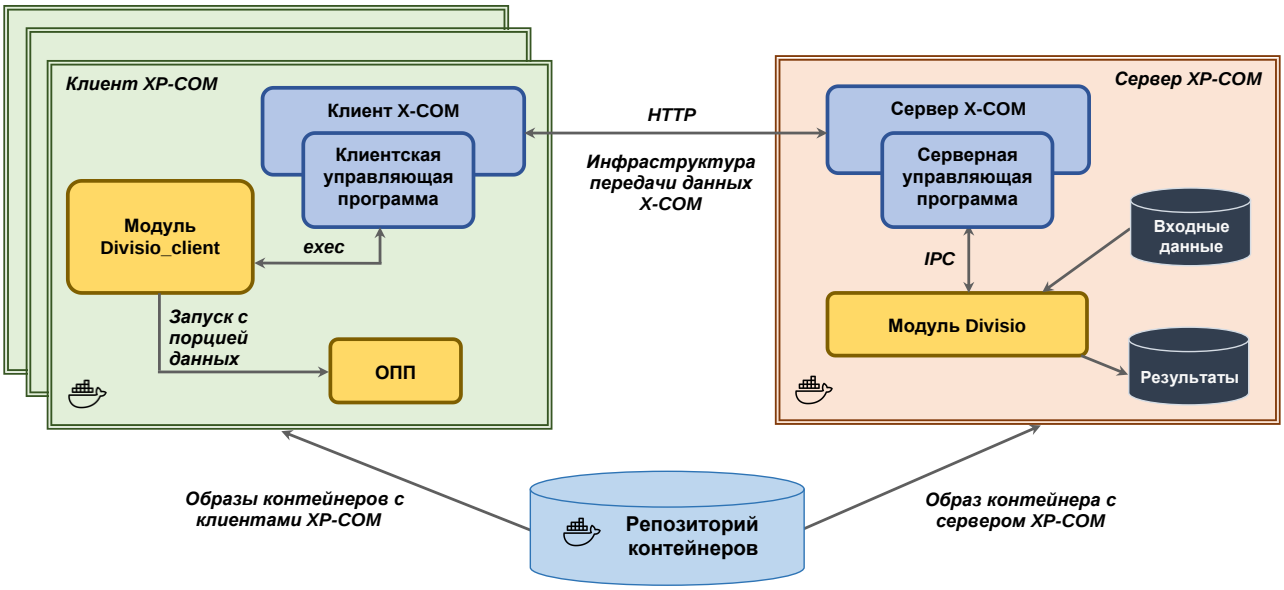

Рисунок 73. Архитектура гибридного ПК XP-COM

Взаимодействие клиентской и серверной частей гибридного ПК осуществляется по следующему алгоритму.

1. Запуск серверной части и модуля Divisio.

2. Запуск клиента X-COM.

3. Запрос порции данных клиентом у сервера.

4. Выделение порции данных сервером (взаимодействие с Divisio), если не получается выделить, то завершение работы.

5. Отправка порции данных клиенту.

6. Обработка клиентом полученной порции (запуск Divisio_client).

7. Отправка результатов обработки серверу.

8. Запись полученных от клиента результатов.

9. Переход к пункту 3.

Отметим важную деталь сопряжения ПК X-COM и «Пирамида». В ПК «Пирамида» распределение вычислительной работы и учет порций ведется по номерам менеджеров нижестоящего уровня иерархии и обслуживающих их клиентских потоков, как показано на рисунке 63. В ПК X-COM отсутствует нумерация клиентов. Для возможности сопряжения в Divisio был реализован механизм динамического выделения клиентских номеров. При обращении за очередной порцией работы клиенту автоматически назначается наименьший свободный номер.

Назначенный номер записывается в метаданные порции работы и с этого момента считается занятым. Получивший порцию клиент, выполнив работу, возвращает результат серверу. В метаданных результата указывается полученный клиентом номер. При получении результата сервером этот номер освобождается.

Для возможности запуска на суперкомпьютере в режиме коллективного пользования под управлением системы управления заданиями ПК XP-COM был реализован в виде контейнеров Docker. Было сформировано два образа контейнеров – для серверной и клиентской частей XP-COM. Образы помещены в репозиторий МСЦ РАН в соответствии с подходом, рассмотренным в п. 2.1.2 и публикации [75]. В качестве системы управления заданиями была применена СУППЗ. При запуске задания СУППЗ извлекает образы сервера и клиентов XP-COM, копирует их на выделенные для задания узлы суперкомпьютера, запускает на узлах экземпляры контейнеров, связывая их при этом в единую виртуальную сеть. После этого в контейнерах начинают функционировать процессы комплекса XP-COM.

5.6.2 Экспериментальная оценка производительности ПК XP-COM

Эксперименты производились в МСЦ РАН на разделе Broadwell суперкомпьютера МВС-10П ОП. Выполнение тестов происходило в режиме коллективного пользования под управлением СУППЗ. Характеристики раздела Broadwell суперкомпьютера МВС-10П ОП представлены в п.2.5 и работах [10, 66].

Для экспериментов была применена рассмотренная в п. 5.5.1 и использованная в п. 5.5.3 методика. Производительность ПК XP-COM выражалась через долю накладных расходов на распараллеливание, которая определялась в соответствии с формулой (14). В экспериментах использовались те же тестовые примеры ОПП, что применялись в п. 5.5.3 : тест Opp_one моделирует работу ОПП с одним входным параметром, тест Opp_three моделирует работу ОПП с тремя входными параметрами и перебирает все комбинации их значений, тест Opp_file моделирует обработку строк, считываемых из переданного ему в качестве параметра файла. В [240] вычислительная работа ОПП моделировалась вызовом функции sleep стан-

дартной библиотеки C, т.е. тестовая программа «засыпала» на заданное время, имитируя работу. В экспериментах с ПК XP-COM функция sleep была заменена на вычисление хеш-функции md5 заданное число раз.

На рисунке 74 представлена сравнительная диаграмма накладных расходов ПК XP-COM, X-COM и «Пирамида» для теста Opp_one перебора одного параметра при переменном значении числа вычислительных ядер, при объеме данных в 10^5 элементарных порций, при количестве подсчетов хеш-функции на одну элементарную порцию 1,3 млн. раз (~1 секунда). Как видно, гибридный XP-COM показывает лучшую производительность.

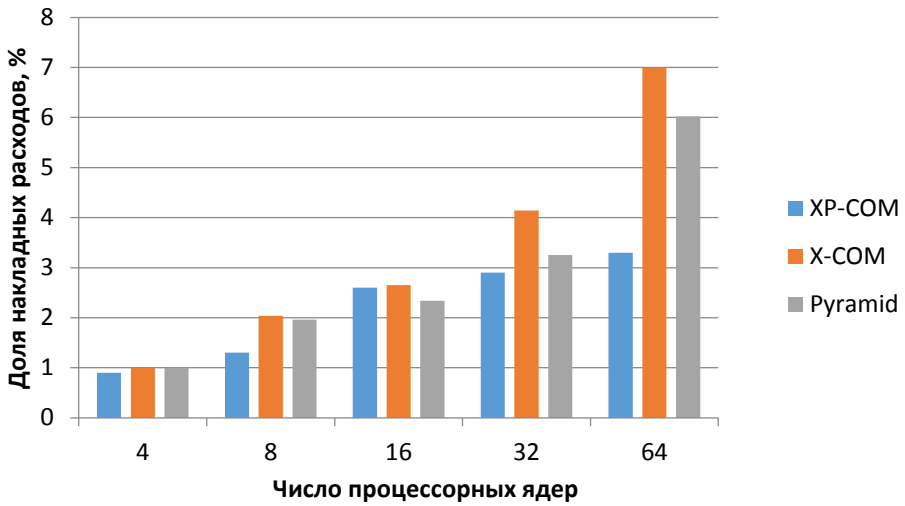

Рисунок 74. Накладные расходы ПК XP-COM, X-COM и «Пирамида» для перебора значений одного параметра

На рисунке 75 представлены графики зависимостей накладных расходов ПК XP-COM для теста Opp_one перебора одного параметра при переменном значении времени обработки элементарной порции данных (количества подсчетов хеш-функции), при объеме данных в 10^5 элементарных порций и размере разовой порции на одного клиента в 98 элементарных порций.

На рисунке 76 представлена диаграмма накладных расходов XP-COM, X-COM и «Пирамида» для теста Opp_file перебора строк файла при переменном значении числа ядер, объеме данных в 10^5 элементарных порций, числе подсчетов хеш-функции на одну элементарную порцию 1,3 млн. раз (~1 секунда). Видно, что гибридный XP-COM показывает наилучшую производительность.

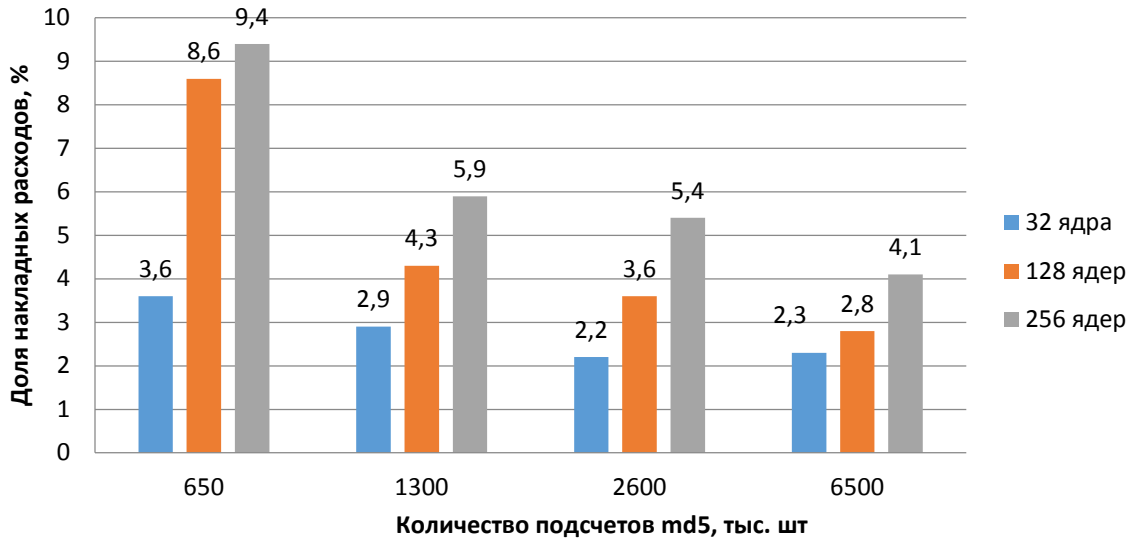

Рисунок 75. Накладные расходы XP-COM при изменении времени обработки одной элементарной порции для перебора одного параметра

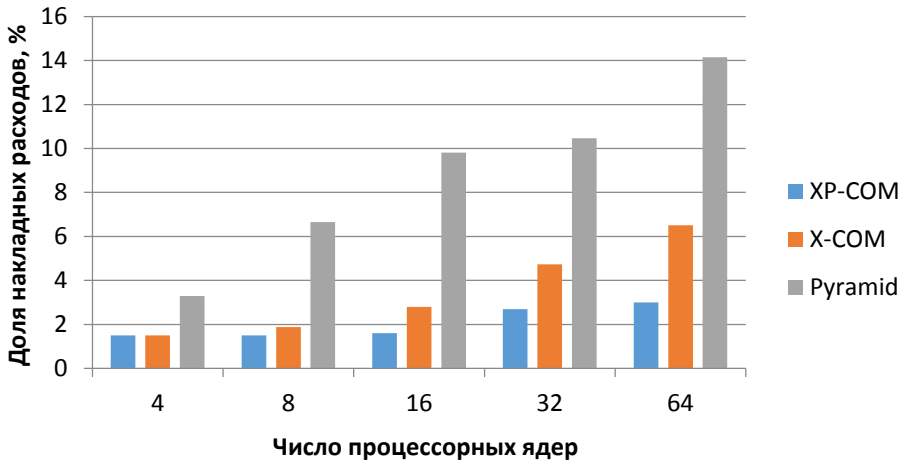

Рисунок 76. Накладные расходы для ПК XP-COM, X-COM и «Пирамида» для перебора строк файла

На рисунке 77 изображены графики зависимостей накладных расходов ПК XP-COM для теста Opp_file перебора строк файла при переменном значении времени обработки элементарной порции данных (количества подсчетов хеш-функции), при объеме данных в 10^5 элементарных порций и размере разовой порции на одного клиента в 98 элементарных порций. Из графиков рисунка 77 видно, что вне зависимости от количества вычислительных узлов, с увеличением времени обработки элементарной порции происходит снижение накладных расходов.

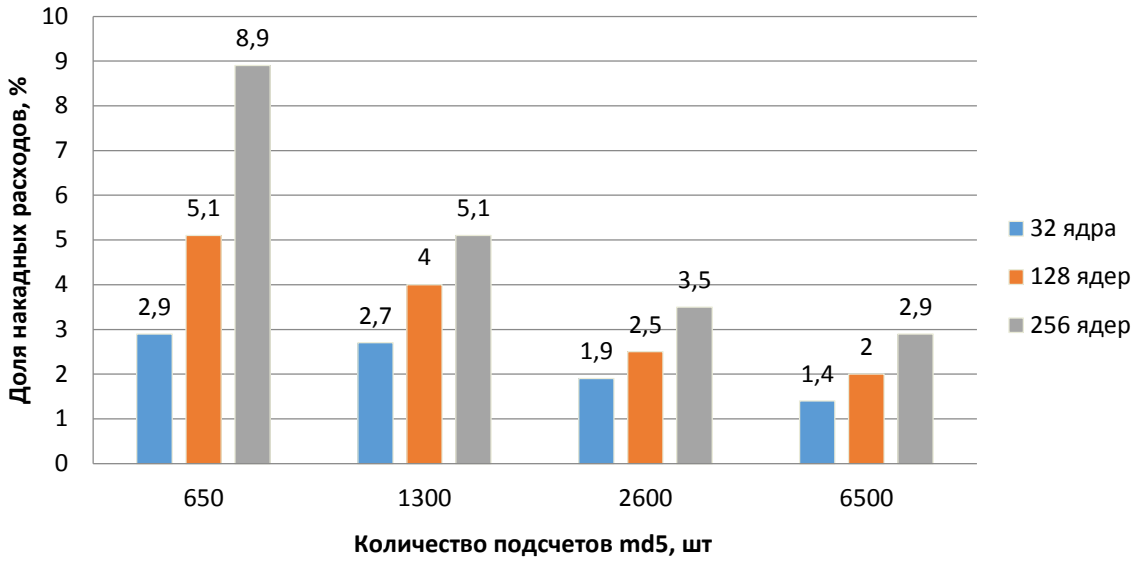

Рисунок 77. Накладные расходы для ПК XP-COM при изменении времени обработки одной элементарной порции для перебора строк файла

На рисунке 78 представлена диаграмма накладных расходов ПК XP-COM, X-COM и «Пирамида» для теста Opp_three перебора комбинаций значений трех параметров при переменном числе ядер, объеме данных в 10^5 элементарных порций, числе подсчетов хеш-функции на одну элементарную порцию 1,3 млн. раз (~1 секунда). Из диаграммы видно, что гибридный ПК показывает наилучшую производительность.

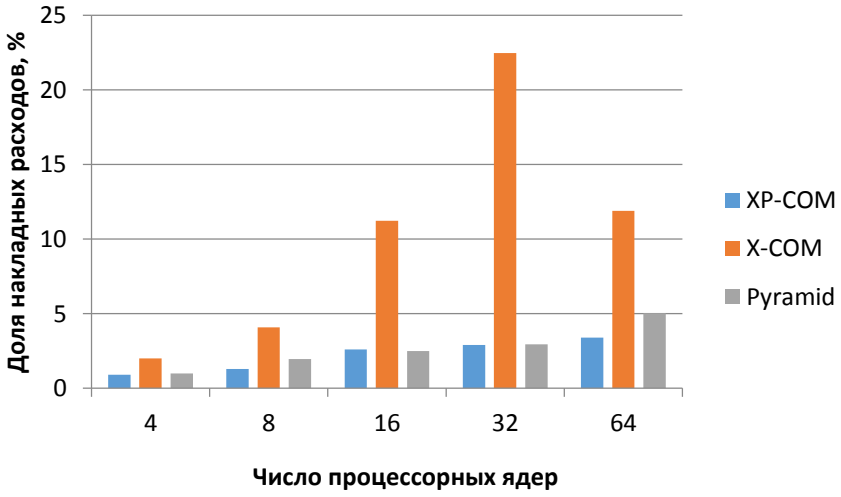

Рисунок 78. Накладные расходы ПК XP-COM, X-COM и «Пирамида» для перебора комбинаций значений трех параметров

На рисунке 79 изображены графики зависимостей накладных расходов ПК XP-COM для теста Opp_three перебора комбинаций значений трех параметров

при переменном значении времени обработки элементарной порции данных (количества подсчетов хеш-функции), объеме данных в 10^5 элементарных порций и размере разовой порции на одного клиента в 98 элементарных порций. Из графиков рисунка 79 видно, что вне зависимости от количества вычислительных узлов, с увеличением времени обработки элементарной порции происходит снижение накладных расходов.

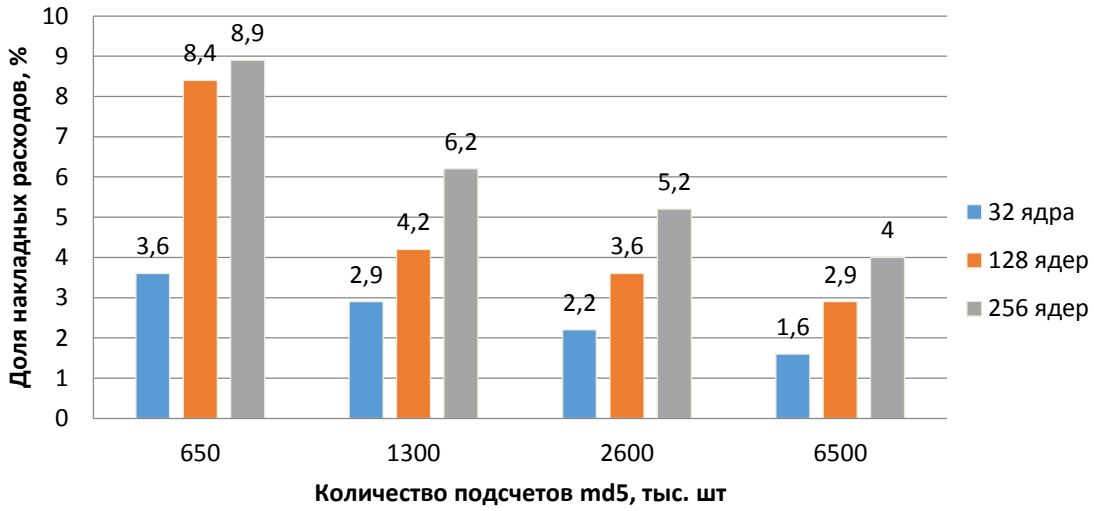

Рисунок 79. Накладные расходы ПК XP-COM при изменении времени обработки одной элементарной порции для перебора комбинаций значений трех параметров

Как показывают результаты экспериментов, гибрибный ПК XP-COM показывает меньшие накладные расходы по сравнению с ПК «Пирамида» и X-COM. Однако, во всех тестах наблюдалось резкое увеличение накладных расходов при количестве клиентов более 256. Проблема может быть связана с тем, что в XP-COM используется двуранговая структура (отсутствуют промежуточные серверы), и сервер X-COM не успевает обрабатывать все поступающие от клиентов запросы. Клиенты, не получившие в ответ на запрос очередную порцию данных, по истечению времени ожидания снова посылают запрос (во время ожидания клиент не выполняет вычислительной работы – простаивает).

Проблема решается за счет добавления промежуточных серверов. На выделенных вычислительных узлах размещаются промежуточные серверы, которые будут аккумулировать часть от общей работы и обслуживать нижестоящих клиентов. Такой подход прямо рекомендуется разработчиками X-COM [61]. Создание

древовидной структуры позволит снизить нагрузку на основной сервер, так как все запросы вычислительных клиентов будут равномерно распределены между несколькими серверами.

Выводы к главе 5

Среди различных классов задач, требующих для своего решения высокопроизводительных вычислительных ресурсов, можно выделить задачи с распараллеливанием по данным. Пул входных данных при решении таких задач разбивается на независимые порции, каждая из которых может быть обработана на отдельном вычислителе (процессорном ядре) при помощи последовательной программы, реализующей прикладной алгоритм. Организация параллельных вычислений с распараллеливанием по данным заключается в запуске множества экземпляров последовательной программы на доступных вычислительных ресурсах и распределении порций входных данных запущенным экземплярам.

Автором предложен метод иерархического (рекурсивного) разделения данных, основанный на организации вычислительного процесса в виде иерархической системы менеджеров, каждый из которых контролирует либо группу менеджеров нижестоящего уровня иерархии, либо запускаемые на вычислительном узле последовательные программы. Метод подразумевает деление на каждом уровне иерархии входного пула данных на порции заданного размера и асинхронное распределение этих порций между менеджерами нижестоящего уровня иерархии в качестве входных пулов. Пулы данных при этом представляются виде упорядоченного набора нумерованных слайсов – элементарных порций данных. Каждая порция (пул) данных при таком представлении однозначно определяется номерами первого и последнего слайсов порции. При разделении данных на каждом уровне иерархии фиксируется компактная контрольная точка в виде текущих выделенных порций и остатка входного пула данных, что позволяет обойтись без специализированной базы учета обработанных порций данных и одновременно свести к минимуму накладные расходы на распараллеливание. За счет асинхронного распределе-

ния и механизма перераспределения порций данных обеспечиваются отказоустойчивость вычислений и возможность применения для решения одной задачи вычислительных средств разной производительности и архитектуры.

Метод иерархического деления данных был реализован в составе программного комплекса распараллеливания по данным «Пирамида». Экспериментальное сравнение ПК «Пирамида» с программными средствами, реализованными на основе технологий MapReduce, MPI, BOINC, а также с разработанным в НИВЦ МГУ им. М.В. Ломоносова ПК X-COM, показало, что ПК «Пирамида» вносит существенно меньшие накладные расходы при распараллеливании, основанном на переборе комбинаций параметров. В ходе экспериментов было обнаружено, что ПК «Пирамида» уступает ПК X-COM при обработке пула входных данных, представленного в виде набора строк в файле.

Для устранения обнаруженного недостатка была предложена и реализована архитектура гибридного программного комплекса XP-COM, в котором были совмещены достоинства программных комплексов X-COM и «Пирамида». Основу гибридного комплекса составили транспортная инфраструктура ПК X-COM и программные модули, реализующие предложенный метод иерархического деления данных. Результаты экспериментов продемонстрировали, что созданный гибридный комплекс XP-COM опередил по производительности как ПК X-COM, так и ПК «Пирамида».

Заключение

В диссертационной работе обобщен многолетний опыт автора по разработке, созданию и практическому применению методов и средств управления вычислительными ресурсами суперкомпьютерных систем коллективного пользования. В результате представлен комплекс новых научно обоснованных архитектурных, технических и технологических решений, внедрение которых вносит значительный вклад в развитие научных суперкомпьютерных центров коллективного пользования, как неотъемлемой части исследовательской инфраструктуры страны.

Основные результаты диссертационной работы следующие.

1. Построена иерархическая пятиуровневая модель управления вычислительными ресурсами суперкомпьютерных систем коллективного пользования. На основе модели предложена архитектура системы управления заданиями, обеспечивающая выполнение таких требований, как универсальность, надежность, императивность управления, модульность.

Для каждого уровня построенной модели предложены технические и технологические решения по управлению вычислительными ресурсами этого уровня иерархии. На основе предложенной архитектуры, разработанных технических и технологических решений создана Система управления прохождением параллельных заданий (СУППЗ), обеспечивающая комплекс возможностей, качественных характеристик и количественных показателей, соответствующий мировому уровню.

Эффективность СУППЗ подтверждается ее успешным применением в наиболее мощных российских суперкомпьютерах, установленных в 1999-2024 гг. в МСЦ РАН, ИПМ им. М.В. Келдыша РАН и ряде других научных и образовательных организаций. Услугами СУППЗ воспользовались более 1240 пользователей-исследователей из 135 организаций, выполнивших свыше 3,7 млн. заданий при реализации более чем 530 научных проектов.

Объем и характеристики предоставляемых под управлением СУППЗ услуг по высокопроизводительным вычислениям позволяют говорить о формировании на основе СУППЗ цифровой экосистемы в виде информационно-вычислительной

среды суперкомпьютерного центра коллективного пользования, доступной практически для всех российских ученых. Сформированная цифровая экосистема высокопроизводительных вычислений обеспечила возможность проведения научных исследований, результаты которых опубликованы в более чем 6500 научных статьях в ведущих отечественных и зарубежных изданиях.

2. Предложен и реализован в составе СУППЗ метод планирования заданий, основанный на выделении во входном потоке классов отладочных, ординарных и фоновых заданий. Выделение отдельного класса фоновых заданий было предложено впервые и определяет научную новизну метода планирования. Кроме этого, для повышения загрузки суперкомпьютера метод применяет стратегию плавающего резерва, позволяющей при отсутствии в системе отладочных заданий использовать выделенные для них вычислительные узлы заданиям других классов. Эффективность предложенного метода планирования подтверждена статистикой суперкомпьютеров под управлением СУППЗ, показывающей кратное снижение коэффициента замедления для классов отладочных и фоновых заданий.

3. Предложен метод постпланирования, позволяющий совмещать потоки заданий с фиксированными параметрами и адаптивных заданий. Эффективность метода подтверждается результатами имитационного моделирования, которые на входных потоках разной интенсивности продемонстрировали максимизацию загрузки суперкомпьютера при незначительном увеличении коэффициента замедления основного потока заданий с фиксированными параметрами.

4. Предложены методы совмещения потоков заданий, поступающих от облачных платформ, и локальных заданий СУЗ, в том числе представленных в виде виртуальных машин и контейнеров. Эффективность методов подтверждена экспериментальной оценкой накладных расходов, которые сведены к минимуму для заданий, использующих типовой стек программного обеспечения.

5. Предложен двухэтапный метод и алгоритмы отображения параллельной программы на вычислительные узлы суперкомпьютера. Идея метода, определяющая его научную новизну, состоит в том, что на первом этапе решается задача

выделения для очередного задания подсистемы узлов из множества свободных на момент запуска задания. Эта задача сведена к разрезанию графа свободных узлов минимально связанные между собой подграфы. Для решения этой задачи автором предложен эвристический алгоритм, научная новизна которого состоит в том, что для разрезания графа на минимально связанные между собой подграфы впервые было применено сочетание алгоритма имитации отжига и алгоритма случайного перебора вершин.

На втором этапе производится поиск оптимального отображения программного графа на граф выделенных заданию узлов при помощи предложенного автором параллельного алгоритма, для работы которого используются выделенные для задания вычислительные узлы. Научная новизна предложенного параллельного алгоритма состоит в циклическом повторении фаз имитации отжига и генетического отбора, что позволило добиться лучших характеристик точности и скорости поиска отображения по сравнению с известными решениями.

Эффективность предложенных метода и алгоритмов подтверждена результатами экспериментов на суперкомпьютерах серии МВС.

6. Предложен и реализован в составе программных комплексов «Пирамида» и XP-COM метод иерархического разделения данных для организации параллельных вычислений с распараллеливанием по данным. Метод обеспечивает меньшие накладные расходы по сравнению с известными решениями, что подтверждается результатами экспериментов.

Основные результаты, выводы и рекомендации, изложенные в диссертации, получены и использовались при реализации следующих проектов:

− программ фундаментальных научных исследований государственных академий наук на 2008-2012 годы, на 2013-2020 годы и на 2021-2030 годы;

− программ фундаментальных исследований Президиума и Отделения математических наук РАН;

− проектов Российского фонда фундаментальных исследований.

Результаты диссертации изложены в 45 печатных работах [10, 67, 69, 70, 72-77, 82-84, 101, 102, 106, 109, 112, 113, 118, 121-123, 131, 134-137, 140, 155, 156, 169-171, 174, 188, 189, 211, 218, 219, 224, 230, 231, 239, 240], по теме диссертации получено 6 свидетельств [100, 103, 127, 128, 132, 229] о государственной регистрации программ для ЭВМ и баз данных.

Результаты диссертации внедрены в практическую деятельность Федерального научного центра Научно-исследовательский институт системных исследований Национального исследовательского центра «Курчатовский институт», Федерального исследовательского центра Институт прикладной математики им. М.В. Келдыша Российской академии наук, Научно-исследовательского института «Квант», Института математики и механики им. Н.Н. Красовского Уральского отделения Российской академии наук, а также использованы для проведения высокопроизводительных расчетов в практике Института биохимической физики им. Н.М. Эмануэля Российской академии наук и Физико-технического института им. А.Ф. Иоффе Российской академии наук.

Материалы диссертации были использованы в учебном процессе при подготовке специалистов по специальностям «Вычислительные машины, комплексы, системы и сети» и «Информационная безопасность автоматизированных систем», а также при создании учебника «Организация ЭВМ и систем» [242] для студентов высших учебных заведений, обучающихся по направлению «Информатика и вычислительная техника» специальности «Вычислительные машины, системы, комплексы и сети».

Результаты диссертационной работы могут быть использованы для исследований и разработки систем управления заданиями суперкомпьютеров, организации параллельных вычислений с распараллеливанием по данным, создания и развития суперкомпьютерных центров коллективного пользования.

Список литературы

1. Воеводин Вл. В., Попова Н.Н. Инфраструктура суперкомпьютерных технологий // Программирование, 2019, № 3, с. 6-13. doi: 10.1134/S0132347419030087

2. Ezell S., Atkinson R.D. The Vital Importance of High-Performance Computing to U.S. Competitiveness. Information Technology and Innovation Foundation report, 2016. URL: https://itif.org/publications/2016/04/28/vital-importance-high-performance-computing-us-competitiveness/ (дата обращения 31.05.2025)

3. Rivière C. High Performance Computing: A tool to foster European competitiveness // Ninth International Conference on Computer Science and Information Technologies Revised Selected Papers, 2013, pp. 1-2, doi: 10.1109/CSITechnol.2013.6710370

4. Указ Президента Российской Федерации от 28.02.2024 г. № 145 О Стратегии научно-технологического развития Российской Федерации. URL: http://www.kremlin.ru/acts/bank/50358 (дата обращения 21.04.2025).

5. Четверушкин Б.Н., Якобовский М.В. О перспективах развития в России высокопроизводительных вычислений и предсказательного моделирования в современных технологиях // Вестник Российской академии наук, 2021, Т.91, №12, с. 1108-1114. doi: 10.31857/S0869587321120057

6. Абрамов Н.С., Абрамов С.М. Ноябрь 2022: состояние и перспективы развития суперкомпьютерной отрасли в мире и в России // Программные системы: теория и приложения, 2023, Т.14, № 2(57), с. 49-93. doi: 10.25209/2079-3316-2023-14-2-49-93

7. Voevodin V.V., Antonov A.S., Nikitenko D.A. [et al.] Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community / // Supercomputing Frontiers and Innovations, 2019, vol. 6, No.2, pp. 4-11. doi: 10.14529/jsfi190201

8. Велихов В.Е., Климентов А.А., Машинистов Р.Ю. [и др.] Интеграция гетерогенных вычислительных мощностей НИЦ «Курчатовский институт» для

проведения масштабных научных вычислений // Суперкомпьютерные технологии (СКТ-2016): Материалы 4-й Всероссийской научно-технической конференции. Дивноморское, Геленджик: Южный федеральный университет, 2016, т.2, с.. 17-21.

9. Якобовский М.В., Корнилина М.А. Развитие суперкомпьютерных технологий в ИММ РАН и ИПМ им. М.В. Келдыша РАН // Computational Mathematics and Information Technologies, 2024, т. 8, №1, с. 12–28. doi: 10.23947/2587-8999-2024-8-1-12-28

10. Savin G.I., Shabanov B.M., Telegin P.N., Baranov A.V. Joint Supercomputer Center of the Russian Academy of Sciences: Present and Future // Lobachevskii J Math, 2019, vol. 40, pp. 1853-1862. doi: 10.1134/S1995080219110271

11. Baginyan A., Balandin A., Dolbilov A. [et al.] JINR Grid Infrastructure: Status and Plans / // Physics of Particles and Nuclei, 2024, vol.55, no.3, pp. 355-359. doi: 10.1134/s1063779624030079

12. Reuther A. et al. Scalable system scheduling for HPC and big data // Journal of Parallel and Distributed Computing, 2018, vol. 111, pp. 76–92. doi: 10.1016/j.jpdc.2017.06.009

13. Henderson R.L. Job scheduling under the Portable Batch System // Lecture Notes in Computer Science, 1995, vol. 949, pp. 279–294. doi: 10.1007/3-540-60153-8_34

14. Лацис А.О. Как построить и использовать суперкомпьютер. М.: Бестселлер, 2003. 240 с.

15. Hussain H. et al. A survey on resource allocation in high performance distributed computing systems // Parallel Computing, 2013, Volume 39, Issue 11, pp. 709-736. doi: 10.1016/j.parco.2013.09.009

16. Sterling T., Anderson N., Brodowicz M. The Essential SLURM: Resource Management. In High Performance Computing (Second Edition), 2025, pp. 119-152. doi: 10.1016/B978-0-12-823035-0.00007-9

17. Quintero D., Black M., Hussein A.Y., McMillan B.S., Samu G., Welch J.S. IBM Spectrum LSF Suite: Installation Best Practices Guide, 2020, IBM Redbooks. ISBN: 9780738458571

18. Feitelson D. Metrics for parallel job scheduling and their convergence // Lect. Notes Comput. Sci., 2001, vol. 2221, pp. 188–205. doi 10.1007/3-540-45540-x_11

19. Yalim J. Toward Dynamically Controlling Slurm's Classic Fairshare Algorithm // In Practice and Experience in Advanced Research Computing (PEARC '20), 2020, pp. 538–542. doi: 10.1145/3311790.3399628.

20. Cox, R., Morrison, L.: Fair tree: fairshare algorithm for slurm // Proceedings of the Slurm User Group Meeting, 2014. URL: https://slurm.schedmd.com/SC14/BYU_Fair_Tree.pdf (дата обращения: 04.04.2025)

21. Lelong J., Reis V., Trystram D. Tuning EASY-Backfilling Queues // Lecture Notes in Computer Science, 2018, vol. 10773, pp. 43–61. doi:10.1007/978-3-319-77398-8_3

22. Carastan-Santos D., Camargo R., Trystram D., Zrigui S. One Can Only Gain by Replacing EASY Backfilling: A Simple Scheduling Policies Case Study, 2019, pp. 1-10. doi: 10.1109/CCGRID.2019.00010.

23. Wong A.K.L., Goscinski A.M. Evaluating the EASY-backfill job scheduling of static workloads on clusters // IEEE International Conference on Cluster Computing, 2007, pp. 64-73. doi: 10.1109/CLUSTR.2007.4629218

24. Le Hai T.H., Duy K.N., Manh T.N. et al. Deviation Backfilling: A Robust Backfilling Scheme for Improving the Efficiency of Job Scheduling on High Performance Computing Systems // International Conference on Advanced Computing and Analytics (ACOMPA), 2023, pp. 32-37. doi: 10.1109/ACOMPA61072.2023.00015.

25. Игнатьев А.О., Калинин А.А., Мокшин С.Ю. Реализация функций управления задачами и ресурсами высокопроизводительной вычислительной системы в «СПО Супер-ЭВМ» // Труды Института системного программирования РАН, 2022, т. 34, №2, с. 159–178. doi: 10.15514/ISPRAS-2022-34(2)-13.

26. Minami Sh., Endo T., Nomura A. Effectiveness of the Oversubscribing Scheduling on Supercomputer Systems // Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia '23). Association for Computing Machinery, 2023, pp. 18–28. doi: 10.1145/3578178.3578221

27. Klusacek D., Chlumsky V. Evaluating the impact of soft walltimes on job scheduling performance // Lect. NotesComput. Sci., 2019, vol. 11332, pp. 15–38. doi: 10.1007/978-3-030-10632-4_2

28. Samu G., Tang T. et al. Utilizing IBM Spectrum LSF Simulator to Understand the Impacts of Adding AI Workloads to Capability Supercomputing. Oak Ridge National Laboratory, 2019. doi: 10.2172/2205452

29. Liu Z., Kettimuthu R., Papka M., Foster I. FreeTrain: A Framework to Utilize Unused Supercomputer Nodes for Training Neural Networks // 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2023, pp. 299-310. doi: 10.1109/CCGrid57682.2023.00036

30. Polyakov S., Dubenskaya J. Improving the Effective Utilization of Supercomputer Resources by Adding Low-Priority Containerized Jobs // CEUR Workshop Proceedings, 2019, vol. 2406, pp. 43-53. doi: 10.48550/arXiv.1909.00394

31. Олейник Д.А. Методика и программное обеспечение для подключения суперкомпьютеров к распределенной системе обработки данных эксперимента ATLAS. Диссертация на соискание ученой степени кандидата технических наук. Дубна: ОИЯИ, 2021.

32. Кудрявцев А.О., Кошелев В.К., Избышев А.О., Аветисян А.И. Высокопроизводительные вычисления как облачный сервис: ключевые проблемы // Международная научная конференция «Параллельные вычислительные технологии 2013 (ПаВТ'2013)», Россия, Челябинск, 2013, с. 432–438.

33. Balashov N., Kuprikov I., Kutovskiy N. [et al.] Changes and Challenges at the JINR and Its Member States Cloud Infrastructures // Physics of Particles and Nuclei., 2024, Vol.55, No.3, pp. 366-370. doi: 10.1134/s1063779624030092

34. Smirnov S., Sukhoroslov O., Voloshinov V. Using resources of supercomputing centers with Everest platform // Communications in Computer and Information Science, 2019, Vol.965, pp. 687-698. doi: 10.1007/978-3-030-05807-4_59

35. Sukhoroslov O. Integration of Everest platform with BOINC-based desktop grids // CEUR Workshop Proceedings, Petrozavodsk, 2017, Vol.1973, pp. 102-107.

36. Ciaschini V., Dal Pra S., dell'Agnello L. Dynamic partitioning as a way to exploit new computing paradigms: the cloud use case // Journal of Physics: Conference Series, 2015, Vol.664. doi: 10.1088/1742-6596/664/2/022014

37. Rad P. et al. Benchmarking Bare Metal Cloud Servers for HPC Applications // IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2015, pp. 153-159. doi: 10.1109/CCEM.2015.13

38. Кудрявцев А.О., Кошелев В.К., Избышев А.О. [и др.] Разработка и реализация облачной системы для решения высокопроизводительных задач // Труды Института системного программирования РАН, 2013, Т. 24, с. 13-34.

39. Younge A.J., Pedretti K., Grant R.E., Gaines B.L., Brightwell R. Enabling Diverse Software Stacks on Supercomputers Using High Performance Virtual Clusters // IEEE International Conference on Cluster Computing (CLUSTER), 2017, pp. 310-321. doi: 10.1109/CLUSTER.2017.92

40. Монахов О.Г., Гросбейн Е.Б., Мусин А.Р. Алгоритмы отображения для параллельных систем на основе моделирования эволюции и моделирования отжига // Труды 6-го международного семинара «Распределенная обработка информации». Под ред. В.Г. Хорошевского. Новосибирск: СО РАН, 1998, с.142.

41. Sahni S., Gonzalez T. P-Complete Approximation Problems // J. ACM, 1976, vol. 23(3), pp. 555–565. doi: 10.1145/321958.321975

42. Gupta M., Bhargava L., Indu S. Mapping techniques in multicore processors: Current and future trends // J. Supercomput., 2021, vol. 77, pp. 9308–9363. doi: 10.1007/s11227-021-03650-6

43. Леушкин А.Д., Неймарк Е.А. Квадратичная задача о назначении. Обзор методов, генерация тестовых задач с априорно известным оптимумом // Труды НГТУ им. Р.Е. Алексеева, 2020, № 4(131), с. 26-34. doi: 10.46960/1816-210X_2020_4_26

44. Braun T.D. et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems // J. Parallel Distrib. Comput., 2001, vol. 61, pp. 810–837. doi: 10.1006/jpdc.2000.1714

45. Orsila H., Salminen E., Hamalainen T.D. Recommendations for using simulated annealing in task mapping // Autom. Embed. Syst., 2013, vol. 17, pp. 53–85. doi: 10.1007/s10617-013-9119-0

46. de A. Rocha H.M.G. et al. A routing based genetic algorithm for task mapping on MPSoC // Proceedings of the 10th Brazilian Symposium on Computing Systems Engineering, 2020, pp. 1–8. doi: 10.1109/SBESC51047.2020.9277843

47. Khalilov M.R., Timofeev A.V. Optimization of MPI-Process Mapping for Clusters with Angara Interconnect // Lobachevskii J Math, 2018, vol. 39, pp. 1188–1198. doi: 10.1134/S1995080218090111

48. Полупанова Е.Е., Нигодин Е.А. Гибридный алгоритм решения квадратичной задачи о назначениях // Современные информационные технологии и ИТ-образование, 2021, Т. 17, №2, с. 315-323. doi: 10.25559/SITITO.17.202102.315-323

49. Alfaifi H., Daadaa Y. Parallel Improved Genetic Algorithm for the Quadratic Assignment Problem // International Journal of Advanced Computer Science and Applications, 2022, no. 13(5). doi: 10.14569/IJACSA.2022.0130568

50. Predari M., Tzovas C., Schulz C., Meyerhenke H. An MPI-based Algorithm for Mapping Complex Networks onto Hierarchical Architectures // Lecture Notes in Computer Science, 2021, vol. 12820, pp. 167-182. doi: 10.1007/978-3-030-85665-6_11

51. Misevičius A., Verenė D. A Hybrid Genetic-Hierarchical Algorithm for the Quadratic Assignment Problem // Entropy, 2021, vol. 23(1), 108. doi: 10.3390/e23010108

52. Galea F., Sirdey R. A parallel simulated annealing approach for the mapping of large process networks // Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, 2012, pp. 1787–1792. doi: 10.1109/IPDPSW.2012.221

53. Lakhdar L., Mehdi M., Melab N., Talbi E.-G. Parallel hybrid genetic algorithms for solving Q3AP on computational grid // Int. J. Found. Comput. Sci., 2012, vol. 23, pp. 483–500. doi: 10.1142/S0129054112400242

54. Türkkahraman Ş.M., Dindar Öz An Improved Hybrid Genetic Algorithm for the Quadratic Assignment Problem // 6th International Conference on Computer Science and Engineering (UBMK), 2021, pp. 86-91, doi: 10.1109/UBMK52708.2021.9558978

55. Zhong L., Sheng J., Jing M., Yu Z., Zeng X., Zhou D. An optimized mapping algorithm based on simulated annealing for regular NoC architecture // Proceedings of the 9th IEEE International Conference on ASIC, 2011, pp. 389–392. doi: 10.1109/ASICON.2011.6157203

56. Dean O., Ghemawat S. MapReduce: simplified data processing on large clusters // Communications of the ACM, vol. 51, no. 1, pp. 107–113. doi: 10.1145/1327452.1327492

57. Jin H., Ibrahim S., Qi L., Cao H., Wu S., Shi X. The MapReduce Programming Model and Implementations // Cloud Computing, 2011, pp. 373–390. doi: 10.1002/9780470940105.ch14

58. Anderson D.P. BOINC: A Platform for Volunteer Computing // Journal of Grid Computing, 2019, 18(1), pp. 99–122. doi: 10.1007/s10723-019-09497-9

59. Тищенко В.И. BOINC: парадигма и модели // Труды Института системного анализа Российской академии наук, 2024, Т.74, №1, с. 79-89. doi: 10.14357/20790279240110

60. Филамофитский М.П. Система поддержки метакомпьютерных расчетов X-Com: архитектура и технология работы // Вычислительные методы и программирование, 2004, Т.5, №2, с. 1-9.

61. Соболев С.И. Иерархические методы улучшения масштабируемости и эффективности распределенных расчетов в системе метакомпьютинга X-Com // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование, 2010, № 35(211), с. 113-120.

62. Duncan R. A survey of parallel computer architectures // Computer, vol. 23, no. 2, pp. 5-16. doi: 10.1109/2.44900

63. Zabrodin A.V., Levin V.K., Korneev V.V. The massively parallel computer system MBC-100 // Lecture Notes in Computer Science, 1995, vol. 964, pp. 341-355. doi: 10.1007/3-540-60222-4_124

64. Забродин А.В., Левин В.К. Опыт разработки параллельных вычислительных технологий. Создание и развитие семейства МВС // Труды Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения», г. Черноголовка, 2000, с. 3-8. URL: https://parallel.ru/conferences/chg2000works.html (дата обращения 21.04.2025).

65. Gavrilovska A. Attaining High Performance Communications: A Vertical Approach (1st ed.). Chapman and Hall/CRC, 2010. doi: 10.1201/b10249

66. Шабанов Б.М. Методы и способы построения, выбора и применения высокопроизводительных вычислительных систем для выполнения научных и технических задач. Диссертация на соискание ученой степени доктора технических наук, 2019. URL: https://www.frccsc.ru/sites/default/files/docs/ds/002-073-02/diss/08-shabanov/ds02-08-shabanov_main.pdf (дата обращения 21.04.2025).

67. Baranov A.V., Lyakhovets D.S., Konstantinov P.A. A Method for Combining Heterogeneous Workflows in HPC Systems // Lobachevskii Journal of Mathematics, 2024, vol. 45, No. 10, pp. 5111-5125. doi: 10.1134/S1995080224606131

68. Cirne W., Berman F. A model for moldable supercomputer jobs // Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001, 8 p. doi: 10.1109/IPDPS.2001.925004

69. Baranov A.V. HPC Scheduling Method Based on Debug and Background Job Classes Division // Lobachevskii Journal of Mathematics, 2024, vol. 45, no. 10, pp. 4899-4911. doi: 10.1134/S1995080224606118

70. Баранов А.В., Ляховец Д.С. Влияние пакетирования на эффективность планирования параллельных заданий // Программные системы: теория и приложения, 2017, Т. 8, № 1(32), с. 193-208. doi: 10.25209/2079-3316-2017-8-1-193-208

71. Левин И.И., Дордопуло А.И., Каляев И.А., Доронченко Ю.И., Раскладкин М.К. Современные и перспективные высокопроизводительные

вычислительные системы с реконфигурируемой архитектурой // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика, 2015, Т.4, №3, с. 24-39. doi: 10.14529/cmse150303

72. Баранов А.В., Николаев Д.С. Использование контейнерной виртуализации в организации высокопроизводительных вычислений // Программные системы: теория и приложения, 2016, Т.7, №1(28), с. 117-134. doi: 10.25209/2079-3316-2016-7-1-117-134

73. Аладышев О. С., Баранов А. В., Ионин Р. П., Киселёв Е. А., Орлов В. А. Сравнительный анализ вариантов развертывания программных платформ для высокопроизводительных вычислений // Вестник УГАТУ, 2014, Т.18, №3(64), с. 295-300.

74. Lyakhovets D.S., Baranov A.V. Efficiency Thresholds of Group Based Job Scheduling in HPC Systems // Lobachevskii Journal of Mathematics, 2022, vol. 43, no. 10, pp. 2863-2876. doi: 10.1134/S1995080222130261

75. Baranov A., Savin G., Shabanov B. et al. Methods of Jobs Containerization for Supercomputer Workload Managers // Lobachevskii Journal of Mathematics, 2019, vol. 40, no. 5, pp. 525-534. doi: 10.1134/S1995080219050020

76. Шабанов Б.М., Овсянников А.П., Баранов А.В. [и др.] Методы управления параллельными заданиями суперкомпьютера, требующими развёртывания отдельных программных платформ и виртуализации сетей // Суперкомпьютерные дни в России: Труды международной конференции, Москва, 2017, с. 616-627.

77. Баранов А.В., Тихомиров А.И. Планирование заданий в территориально распределенной системе с абсолютными приоритетами // Вычислительные технологии, 2017, Т. 22, № S1, с. 4-12.

78. Миренков Н.Н. Параллельное программирование для многомодульных вычислительных систем. М.: Радио и связь, 1989. 320 с.

79. Мукосей А.В., Семенов А.С., Симонов А.С. Оптимизация утилизации при выделении ресурсов для высокопроизводительных вычислительных систем с

сетью Ангара // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2019, т. 8, №1, с. 5-19. doi: 10.14529/cmse190101.

80. Nikitenko D., Zhumatiy S., Paokin A., Voevodin V. Evolution of the Octoshell HPC Center Management System // Communications in Computer and Information Science, 2019, vol. 1063, pp. 19–33. doi: 10.1007/978-3-030-28163-2_2.

81. Костенецкий П.С., Шамсутдинов А.Б., Чулкевич Р.А., Козырев В.И. HPC TaskMaster – система мониторинга эффективности задач суперкомпьютера // Суперкомпьютерные дни в России: Труды международной конференции, 2021, с. 18–25. doi: 10.29003/m2454.RussianSCDays2021

82. Баранов А.В., Киселев Е.А., Кормилицин Е.С. и др. Модернизация подсистемы сбора и обработки статистики центра коллективного пользования вычислительными ресурсами МСЦ РАН / // Труды научно-исследовательского института системных исследований Российской академии наук, 2018, Т. 8, № 4, с. 136-144. doi: 10.25682/NIISI.2018.4.0016

83. Баранов А.В., Лацис А.О., Храмцов М.Ю. Организация многопользовательского режима работы многопроцессорных вычислительных систем // Труды Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения», г. Черноголовка, 2000, с. 67-69. URL: https://parallel.ru/conferences/chg2000works.html (дата обращения 21.04.2025).

84. Баранов А.В., Киселев А.В., Старичков В.В. и др. Сравнение систем пакетной обработки с точки зрения организации промышленного счета // Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции, 2012, с. 506–508.

85. Гольдштейн М.Л., Самофалов В.В. Формирование регионального суперкомпьютерного центра на базе ЭВМ семейства МВС-100, МВС-1000 // Труды Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения», г. Черноголовка, 2000, с. 30-32. URL: https://parallel.ru/conferences/chg2000works.html (дата обращения 21.04.2025).

86. Staples G. TORQUE resource manager // Proc. of the 2006 ACM/IEEE conference on Supercomputing (SC '06), 2006, pp. 8-es. doi: 10.1145/1188455.1188464

87. Joshi P., Babu M.R. Openlava: An open source scheduler for high performance computing // Proc. of the International Conference on Research Advances in Integrated Navigation Systems (RAINS), 2016, pp. 1-3. doi: 10.1109/RAINS.2016.7764375

88. Jackson D.B. Maui Scheduler: A Multifunction Cluster Scheduler. In Beowulf Cluster Computing with Linux, The MIT Press, 2001. doi: 10.7551/mitpress/1556.003.0020

89. Moab HPC Suite. URL: https://adaptivecomputing.com/wp-content/uploads/2022/05/Moab-HPC-Suite_datasheet_05062022.pdf (дата обращения 21.04.2025).

90. Gentzsch W. Sun Grid Engine: towards creating a compute power grid // Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001, pp. 35-36. doi: 10.1109/CCGRID.2001.923173

91. Yang Y., Chen Y. Sun Grid Engine (SGE) and its application // International Symposium on Computers & Informatics, 2015, pp. 975–982. doi: 10.2991/isci-15.2015.129

92. Gruber D. Gridware Cluster Scheduler. URL: http://gridengine.eu/index.php/gridengineprojects/264-gridware-cluster-scheduler (дата обращения 21.04.2025).

93. Sridutt Bhalachandra, Sripriya Thothadri, and Pradeep Rao. Enabling high performance computing using Microsoft HPC server // Proceedings of the 5th ACM COMPUTE Conference: Intelligent & scalable system technologies (COMPUTE '12). Association for Computing Machinery, 2012, Article 12, pp. 1–6. doi: 10.1145/2459118.2459130

94. Jette M.A., Wickberg T. Architecture of the Slurm Workload Manager // Lecture Notes in Computer Science, 2023, vol. 14283, pp. 3–23. doi: 10.1007/978-3-031-43943-8_1

95. Воеводин Вл.В., Жуматий С.А. Вычислительное дело и кластерные системы. М.: Изд-во МГУ, 2007. - 150 с. ISBN 978-5-211-05440-0

96. Kostenetskiy P.S., Chulkevich R.A., Kozyrev V.I. HPC Resources of the Higher School of Economics // Journal of Physics: Conference Series, 2021, p. 012050. doi: 10.1088/1742-6596/1740/1/012050.

97. Savin G., Shabanov B., Rybakov A., Shumilin S. Vectorization of Flat Loops of Arbitrary Structure Using Instructions AVX-512 // Lobachevskii Journal of Mathematics, 2020, vol. 41, No. 12, pp. 2575–2592. doi: 10.1134/S1995080220120331.

98. Nemirovsky M., Tullsen D.M. Simultaneous Multithreading. In: Multithreading Architecture. Synthesis Lectures on Computer Architecture, 2013, pp. 33-40. doi: 10.1007/978-3-031-01738-4_5.

99. Hsu K.Ch., Tseng H.W. Simultaneous and Heterogenous Multithreading: Exploiting Simultaneous and Heterogeneous Parallelism in Accelerator-Rich Architectures // IEEE Micro, 2024, vol. 44, no. 4, pp. 11–19. doi: 10.1109/mm.2024.3414941

100. Аладышев О.С., Баранов А.В., Храмцов М.Ю., Лацис А.О., Дбар С.А., Шарф С.В. Система управления прохождением параллельных заданий: Свид. о регистр. ПрЭВМ № 2012614853. Рос. Федерация, 2012.

101. Баранов А.В. Построение системы управления заданиями пользователей суперкомпьютера на основе иерархической модели // Программные продукты и системы, 2025, №2, с. 345-360. doi: 10.15827/0236-235X.150.345-360

102. Баранов А.В., Киселев Е.А. Интеграция систем управления заданиями SLURM и СУППЗ // Труды научно-исследовательского института системных исследований Российской академии наук, 2019, т. 9, № 5, с. 29–35.

103. Шабанов Б.М., Баранов А.В., Киселев Е.А., Телегин П.Н. Программа управления модулем сопряжения систем управления заданиями СУППЗ и SLURM. Свидетельство о государственной регистрации программы для ЭВМ № 202268164, Российская Федерация: опубл. 16.11.2022.

104. VxWorks, Version 5.2 [New Products] // IEEE Design & Test of Computers, 1995, vol. 12, no. 3, p. 104. doi: 10.1109/MDT.1995.466392

105. Лацис А.О. Инструкция прикладному программисту по работе на МВС-100 в среде Router. Авторское описание системы программирования, 1995. URL: http://parallel.imm.uran.ru/archives/mvs100-doc/progdevl.txt (дата обращения 21.04.2025).

106. Savin G.I., Shabanov B.M., Fedorov R.S., Baranov A.V., Telegin P.N. Checkpointing Tools in a Supercomputer Center // Lobachevskii Journal of Mathematics, 2020, vol. 41, no. 12, pp. 2603-2613. doi: 10.1134/S1995080220120355

107. Tošić A. Run-time Application Migration using Checkpoint/Restore In Userspace // Journal of Web Engineering, 2024, no. 23(05), pp. 735–748. doi: 10.13052/jwe1540-9589.2357

108. Ansel J., Arya K., Cooperman G. DMTCP: Transparent checkpointing for cluster computations and the desktop // IEEE International Symposium on Parallel & Distributed Processing, 2009, pp. 1-12, doi: 10.1109/IPDPS.2009.5161063

109. Aladyshev O.S., Baranov A.V., Ionin R.P., Kiselev E.A., Shabanov B.M. Variants of deployment the high performance computing in clouds // IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)", Russia, Moscow, 2018, 1453–1457. DOI: 10.1109/EIConRus.2018.8317371

110. Щапов В.А., Латыпов С.Р. Способы запуска задач на суперкомпьютере в изолированных окружениях с применением технологии контейнерной виртуализации Docker // Научно-технический вестник Поволжья, 2017, № 5, с. 172–177.

111. Николаев Д.С., Корнеев В.В. Использование механизмов контейнерной виртуализации в высокопроизводительных вычислительных комплексах с системой планирования заданий Slurm // Программная инженерия, 2017, Т. 8, № 4, с. 157–160.

112. Баранов А.В., Долгов Б.В., Федотов А.В. Контейнеризация пользовательских заданий в суперкомпьютерной системе коллективного

пользования // Труды НИИСИ РАН, 2019, Т. 9, № 6, с. 123-131. doi: 10.25682/NIISI.2019.6.0016

113. Baranov A., Aladyshev O., Kiselev E. et al. XP-COM hybrid software package for parallelization by data // Communications in Computer and Information Science, 2020, vol. 1263, pp. 3-15. doi: 10.1007/978-3-030-55326-5_1

114. Nagaraju Islavath. The Power of Docker: Containerization for Efficient Software Development and Deployment // International Journal of Science and Research (IJSR), 2020, vol. 9, no. 11, pp. 1748-1751. doi: 10.21275/SR201226085354

115. Орлов С. Самый мощный в Европе // Журнал сетевых решений LAN, 2013, № 4, с. 4-18e.

116. Sodani A. Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor // 2015 IEEE Hot Chips 27 Symposium (HCS), 2015, pp. 1-24. doi: 10.1109/HOTCHIPS.2015.7477467

117. Intel Knights Landing (KNL). User and Administrator Guide. URL: https://slurm.schedmd.com/intel_knl.html (дата обращения: 30.03.2025)

118. Баранов А.В., Смирнов С.В., Храмцов М.Ю., Шарф С.В. Модернизация СУПЗ МВС-1000 // Научный сервис в сети Интернет: решение больших задач: Труды Всероссийской научной конференции, 2008, с. 226-227.

119. Moab Workload Manager. Administrator Guide 9.0.1. Released: March 2016; Revised: May 16, 2016. http://docs.adaptivecomputing.com/9-0-1/MWM/Moab-9.0.1.pdf (дата обращения: 01.04.2025)

120. Using goal-oriented SLA scheduling. https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_admin/goal_oriented_sla_sched.html (дата обращения: 01.04.2025)

121. Аладышев О.С., Баранов А.В., Дербышев Д.Ю. Методы и алгоритмы обеспечения прохождения пользовательских заданий с заданным уровнем обслуживания // Труды научно-исследовательского института системных исследований Российской академии наук, 2019, Т. 9, № 5, с. 15-22.

122. Баранов А.В., Голинка Д.М. Исследование возможности использования планировщика Maui в составе СУПЗ МВС-1000 // Научный сервис в сети Интернет: решение больших задач: Труды Всероссийской научной конференции, 2008, с. 223-225

123. Баранов А.В., Голинка Д.М. Система управления прохождением задач и планировщик Maui для МВС-100К // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции, 2009, с. 314–315.

124. Wiki Interface Specification, version 1.1. URL: https://docs.adaptivecomputing.com/maui/wikiinterface.php (дата обращения 21.04.2025).

125. Гергель В.П., Кустикова В.Д., Сенин А.В. Интеграция системы управления интегрированной средой высокопроизводительных вычислений Метакластер с подсистемой планирования MAUI // Вестник Нижегородского университета им. Н.И. Лобачевского, 2011, № 3-2, с. 276-284.

126. Савин Г.И., Четверушкин Б.Н., Горобец А.В. [и др.] Моделирование задач газовой динамики и аэроакустики с использованием ресурсов суперкомпьютера МВС-100К // Доклады Академии наук, 2008, Т. 423, № 3, с. 312-315.

127. Баранов А.В., Дбар С.А. Статистика Системы управления прохождением параллельных заданий. Свидетельство о государственной регистрации базы данных № 2016621669 от 15 декабря 2016 года.

128. Аладышев О.С., Баранов А.В., Киселёв Е.А., Гришин Р.И. Система сбора и обработки статистики «МСЦ-МСЦ-КроСтат». Свидетельство о государственной регистрации программы для ЭВМ № 2016663606 от 13 декабря 2016 года.

129. Foster I., Kesselman C., Tsudik G., Tuecke S. A security architecture for computational grids // Proceedings of the 5th ACM conference on Computer and communications security (CCS '98), 1998, pp. 83–92. doi: 10.1145/288090.288111

130. Foster I., Kesselman C., Tuecke S. The Anatomy of the Grid. In Grid Computing, 2003. doi: 10.1002/0470867167.ch6

131. Корнеев В.В., Киселев А.В., Баранов А.В. [и др.] Опыт практической реализации сетевой среды распределенных вычислений // Научный сервис в сети Интернет: технологии параллельного программирования: Труды Всероссийской научной конференции, 2006, с. 148-149.

132. Баранов А.В., Киселев А.В., Киселев Е.А. [и др.]. Программный комплекс «Градиент». Свидетельство о государственной регистрации программы для ЭВМ №2016617767, Российская Федерация: опубл. 14.07.2016.

133. Савин Г.И., Шабанов Б.М., Телегин П.Н. [и др.] Инфраструктура грид для суперкомпьютерных приложений // Известия высших учебных заведений. Электроника, 2011, № 1(87), с. 51-56.

134. Шабанов Б.М., Овсянников А.П., Баранов А.В. [и др.] Проект распределенной сети суперкомпьютерных центров коллективного пользования // Программные системы: теория и приложения, 2017, Т. 8, № 4(35), с. 245-262.

135. Савин Г.И., Шабанов Б.М., Баранов А.В. [и др.] Об использовании федеральной научной телекоммуникационной инфраструктуры для суперкомпьютерных вычислений // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2020, Т.9, №1, с. 20-35. doi: 10.14529/cmse200102

136. Baranov A., Telegin P., Tikhomirov A. Comparison of auction methods for job scheduling with absolute priorities // Lecture Notes in Computer Science, 2017, vol. 10421, pp. 387-395. doi: 10.1007/978-3-319-62932-2_37

137. Lyakhovets D.S., Baranov A.V. Group Based Job Scheduling to Increase the High-Performance Computing Efficiency // Lobachevskii Journal of Mathematics, 2020, vol. 41, no. 12, pp. 2558-2565. doi: 10.1134/S1995080220120264

138. Kiselev E., Baranov A., Telegin P., Kuznetsov E. System for Collecting Statistics on Power Consumption of Supercomputer Applications // Lecture Notes in Computer Science, 2022, vol. 13708, pp. 548-561. DOI: 10.1007/978-3-031-22941-1_40

139. Киселёв Е.А., Баранов А.В., Аладышев О.И., Яровой А.В. Программные инструменты энергоэффективного планирования суперкомпьютерных заданий // Труды НИИСИ РАН, 2021, т.11, №4, с. 48-55

140. Баранов А.В., Ляховец Д.С. Сравнение качества планирования заданий в системах пакетной обработки SLURM и СУППЗ // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции, 2013, с. 410–414.

141. Lucero A. Simulation of batch scheduling using real production-ready software tools // Proceedings of the 5th IBERGRID, 2011, T.21.

142. Фортов В., Левин В.К., Савин Г.И. и др. Суперкомпьютер МВС-1000М и перспективы его применения // Наука и промышленность России, 2001, Т. 55, №11, с. 49.

143. Баранов А.В., Лацис А.О., Сажин С.В., Храмцов М.Ю. Руководство пользователя системы МВС-1000/16, 2002. URL: http://parallel.imm.uran.ru/MVS1000-16/user_guide.htm (дата обращения 21.04.2025).

144. Хлопков Ю.И. Создание центра параллельных вычислений на многопроцессорной ЭВМ МВС-1000/16 для решения фундаментальных проблем вычислительной аэродинамики. НИР: грант № 02-07-90475. Российский фонд фундаментальных исследований, 2002.

145. Основы работы с многопроцессорной вычислительной системой МВС-1000/16 в режиме удаленного доступа: Методические указания. Иваново: Ивановский государственный энергетический университет им. В.И. Ленина, 2003. 28 с.

146. Глинский Б.М., Черных И.Г., Кучин Н.В. [и др.] Управление вычислительными ресурсами Сибирского Суперкомпьютерного Центра // Суперкомпьютерные дни в России: Труды международной конференции, 2015, с. 667-674.

147. Исаев С.В., Малышев А.В., Шайдуров В.В. Развитие Красноярского центра параллельных вычислений // Вычислительные технологии, 2006, Т. 11, № S8, с. 27-33.

148. Ханчук А.И., Сорокин А.А., Наумова В.В. [и др.] Корпоративная сеть Дальневосточного отделения РАН // Вестник Дальневосточного отделения Российской академии наук, 2007, № 1(131), с. 3-19.

149. Дбар С.А., Басс Л.П., Лацис А.О. [и др.] Опыт эксплуатации суперкомпъютера К-100 в Институте прикладной математики им. М.В. Келдыша РАН // Информационные технологии и вычислительные системы, 2016, № 2, с. 5-12.

150. Давыдов А.А., Лацис А.О., Луцкий А.Е. [и др.] Многопроцессорная вычислительная система гибридной архитектуры МВС-Экспресс // Доклады Академии наук, 2010, т. 434, № 4, с. 459–463.

151. Каленов О.Е. Цифровые экосистемы организаций // Вестник Российского экономического университета имени Г.В. Плеханова, 2022, т. 19, № 1(121), с. 139–147. doi: 10.21686/2413-2829-2022-1-139-147.

152. Шамакина А.В. Обзор технологий распределенных вычислений // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2014, Т. 3, № 3, с. 51-85

153. Топорков В.В., Емельянов Д.М. Модели, методы и алгоритмы планирования в грид и облачных вычислениях // Вестник Московского энергетического института, 2018, №6, с. 75-86. doi: 10.24160/1993-6982-2018-6-75-86

154. Lifka D.A. The ANL/IBM SP scheduling system // Lecture Notes in Computer Science, 1995, vol. 949, pp. 295–303. doi: 10.1007/3-540-60153-8_35

155. Shabanov B., Baranov A., Telegin P., Tikhomirov A. Influence of Execution Time Forecast Accuracy on the Efficiency of Scheduling Jobs in a Distributed Network of Supercomputers // Lecture Notes in Computer Science, 2021, vol. 12942, pp. 338-347. doi: 10.1007/978-3-030-86359-3_25

156. Savin G.I., Lyakhovets D.S., Baranov A.V. Influence of Job Runtime Prediction on Scheduling Quality // Lobachevskii J Math, 2021, vol. 42, pp. 2562–2570. doi: 10.1134/S1995080221110196

157. Hou Z., Shen H., Feng Q. et al. Optimizing job scheduling by using broad learning to predict execution times on HPC clusters // CCF Trans. HPC, 2024, vol. 6, pp. 365-377. doi: 10.1007/s42514-023-00137-z

158. Toporkov V., Yemelyanov D., Grigorenko M. Optimization of Resources Allocation in High Performance Computing Under Utilization Uncertainty // Lecture Notes in Computer Science, 2021, Vol. 12747, pp. 540-553. doi: 10.1007/978-3-030-77980-1_41

159. Леоненков С.Н. Целевая оптимизация структуры потока задач суперкомпьютеров // Вычислительные методы и программирование, 2019, Т. 20, № 3, с. 199-210. doi: 10.26089/NumMet.v20r319

160. Huber D., Streubel M., Compres U., Isaias A., Schulz M., Schreiber M., Pritchard H. Towards Dynamic Resource Management with MPI Sessions and PMIx. // EuroMPI/USA '22: Proceedings of the 29th European MPI Users' Group Meeting, 2022, pp. 57-67. doi: 10.1145/3555819.3555856

161. Hall J., Lathi A., Lowenthal D.K., Patki T. Evaluating the Potential of Coscheduling on High-Performance Computing Systems. // Lecture Notes in Computer Science, 2023, vol. 14283, 155-172. doi: 10.1007/978-3-031-43943-8_8

162. Maeno T. Harvester: an edge service harvesting heterogeneous resources for ATLAS // EPJ Web of Conferences, EDP Sciences, 2023, vol. 214, no. 03030. doi: 10.1051/epjconf/201921403030

163. Taylor R.P., Albert J.R., Megino F. A grid site reimagined: Building a fully cloud-native ATLAS Tier 2 on Kubernetes // EPJ Web of Conferences, 2024, vol. 295, no. 07001. doi: 10.1051/epjconf/202429507001

164. Megino F. Accelerating science: The usage of commercial clouds in ATLAS Distributed Computing // EPJ Web of Conferences, 2024, vol. 295, no. 07002. doi: 10.1051/epjconf/202429507002

165. Cascajo A., Arbe A., Garcia-Blas J., Carretero J., Singh D.E. Malleable Techniques and Resource Scheduling to Improve Energy Efficiency in Parallel Applications // Lecture Notes in Computer Science, 2023, vol. 13999, pp. 16-27. doi: 10.1007/978-3-031-40843-4_2

166. Besnard J. B. et al. Towards Smarter Schedulers: Molding Jobs into the Right Shape via Monitoring and Modeling // Lecture Notes in Computer Science, 2023, vol. 13999, pp. 68-81. doi: 10.1007/978-3-031-40843-4_6

167. Wood C. et al. Artemis: Automatic Runtime Tuning of Parallel Execution Parameters Using Machine Learning // Lecture Notes in Computer Science, 2021, vol. 12728, pp. 453-472. doi: 10.1007/978-3-030-78713-4_24

168. D'Amico M., Jokanovic A., Corbalan J. Holistic Slowdown Driven Scheduling and Resource Management for Malleable Jobs // Proceedings of the 48th International Conference on Parallel Processing (ICPP '19), Association for Computing Machinery. 2019, vol. 31, pp. 1-10. doi: 10.1145/3337821.3337909

169. Баранов А.В., Киселёв Е.А., Ляховец Д.С. Квазипланировщик для использования простаивающих вычислительных модулей многопроцессорной вычислительной системы под управлением СУППЗ // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, 2014, №3(4), с. 75-84. doi: 10.14529/cmse140405

170.Баранов А., Николаев Д. Применение машинного обучения для прогнозирования времени выполнения суперкомпьютерных заданий // Программные продукты и системы, 2020, №2, 218-228. doi: 10.15827/0236-235X.130.218-228

171. Баранов А., Ляховец Д. Методы и средства моделирования системы управления суперкомпьютерными заданиями // Программные продукты и системы, 2019, №4, с. 581-594. doi: 10.15827/0236-235X.128.581-594

172. Adufu T., Choi J., Kim Y. Is container-based technology a winner for high performance scientific applications? // 17th Asia-Pacific Network Operations and Management Symposium (APNOMS), 2015, pp. 507-510, doi: 10.1109/APNOMS.2015.7275379

173. Дудина И.А., Кудрявцев А.О., Гайсарян С.С. Разработка и реализация облачного планировщика, учитывающего топологию коммуникационной среды

при высокопроизводительных вычислениях // Труды Института системного программирования РАН, 2013, Т.24, с. 35-48.

174. Баранов А.В., Зонов А.А. Вариант организации облачного сервиса для высокопроизводительных вычислений // Программные системы: теория и приложения, 2016, №7:3(30), с. 3-23. doi: 10.25209/2079-3316-2016-7-3-3-23

175. Гнеденко Б.В., Даниелян Э.А., Димитров Б.Н., Климов Г.П., Матвеев В.Ф. Приоритетные системы обслуживания. М.: МГУ, 1973.- 448 с.

176. Липаев В.В., Яшков С.Ф. Эффективность методов организации вычислительного процесса в АСУ. М.: Статистика, 1975, 256 с.

177. Балыбердин В.А. Методы анализа мультипрограммных систем. М.: Радио и связь, 1982, 152 с.

178. Балыбердин В.А. Оценка и оптимизация характеристик систем обработки данных. М.: Радио и связь, 1987, 176 с.

179. Костогрызов А.И. Исследование условий эффективного применения пакетной обработки заявок в приоритетных вычислительных системах с ограничением на время ожидания в очереди // Автоматика и телемеханика, 1987, № 12, с. 158-164.

180. Костогрызов А.И. Исследование эффективности комбинации различных дисциплин приоритетного обслуживания заявок в вычислительных системах // Кибернетика и системный анализ, 1992, Т.28, №1, с. 128-138.

181. Pechinknin A.V., Chaplygin V.V. Stationary Characteristics of the SM/MSP/n/r Queuing System // Automation and Remote Control, 2004, vol. 65, pp. 1429-1443. doi: 10.1023/B:AURC.0000041421.62689.a8

182. Морозов Е.В., Румянцев А.С. Вероятностные модели многопроцессорных систем: стационарность и моментные свойства // Информатика и ее применения, 2012, Т. 6, № 3, с. 99-106.

183. Zayats O.I., Baksheev V.E., Zaborovsky V.S., Muliukha V.A. Model of a supercomputer cluster in the form of a queueing system with a random limit on the

execution time of applied tasks // Computing, Telecommunications and Control, 2024, vol. 17, no. 3, pp. 71–83. doi: 10.18721/JCSTCS.17307

184. Rumyantsev A., Morozov E. Stability criterion of a multiserver model with simultaneous service // Annals of Operations Research, 2017, Vol. 252, No. 1, pp. 29-39. doi: 10.1007/s10479-015-1917-2

185. Разумчик Р.В., Румянцев А.С., Гаримелла Р.М. Вероятностная модель для оценки основных характеристик производительности марковской модели суперкомпьютера // Информатика и ее применения, 2023, Т.17, № 2, с. 62-70. doi: 10.14357/19922264230209

186. Вишневский В.М., Ефросинин Д.В. Теория очередей и машинное обучение. М.: ИНФРА-М, 2025, 370 с.

187. Lublin U., Feitelson D. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Job // J. of Parallel and Distributed Computing Archive, 2003, no. 63(11), pp. 542-546. doi: 10.1016/S0743-7315(03)00108-4

188. Баранов А.В., Аладышев О.С., Овсянников А.П. [и др.] Методы и средства совмещения потоков заданий от облачных платформ и менеджеров управления ресурсами суперкомпьютера // Программные продукты, системы и алгоритмы, 2018, № 4, с. 8.

189. Баранов А.В., Киселев Е.А. Облачные сервисы для научных высокопроизводительных вычислений на базе платформы Proxmox // Вычислительные технологии, 2019, Т.24, №6, с. 5-12. doi: 10.25743/ICT.2019.24.6.002

190.Vazquez A. libvirt Virtual Machine Management. In: LPIC-3 Virtualization and Containerization Study Guide. Certification Study Companion Series. Apress, Berkeley, 2024, pp. 147-226. doi: 10.1007/979-8-8688-1080-0_4

191. Proxmox Virtual Environment. URL: https://pve.proxmox.com/wiki/Main_Page (дата обращения: 25.04.2025)

192. Drezner Z. The Quadratic Assignment Problem. In: Laporte, G., Nickel, S., Saldanha da Gama, F. (eds) Location Science, 2015, pp. 345–363. doi: 10.1007/978-3-319-13111-5_13

193. Ferrandi F., Lanzi P.L., Pilato C., Sciuto D., Tumeo A. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems // IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 2010, vol. 29, pp. 911–924. doi: 10.1109/TCAD.2010.2048354

194. Orsila H., Kangas T., Salminen E., Hamalainen T.D. Parameterizing simulated annealing for distributing task graphs on multiprocessor SoCs // Proceedings of the 2006 International Symposium on System-on-Chip, 2006, pp. 1–4. doi: 10.1109/ISSOC.2006.321971

195. Azketa E., Uribe J.P., Marcos M., Almeida L., Gutierrez J.J. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems // Proceedings of the 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, pp. 958–965. doi: 10.1109/TrustCom.2011.132

196. Kohmoto K., Katayama K., Narihisa H. Performance of a genetic algorithm for the graph partitioning problem // Math. Comput. Model., 2003, vol. 38, pp. 1325–1332. doi: 10.1016/S0895-7177(03)90134-8

197. Kernighan B.W., Lin S. An efficient heuristic procedure for partitioning graphs // Bell Syst. Tech. J., 1970, vol. 49, pp. 291–307. doi: 10.1002/j.1538-7305.1970.tb01770.x

198. von Laszewski G., Muhlenbein H. Partitioning a graph with a parallel genetic algorithm // Lect. Notes Comput. Sci., 1991, vol. 496, pp. 165–169. doi: 10.1007/BFb0029748

199. Talbi E.-G., Bessiere P. A parallel genetic algorithm for the graph partitioning problem // Proceedings of the 5th International Conference on Supercomputing. Assoc. Comput. Mach., 1991, pp. 312–320. doi: 10.1145/109025.109102

200. Lei T., Kumar S. A two-step genetic algorithm for mapping task graphs to a network on chip architecture // Proceedings of the Euromicro Symposium on Digital System Design, 2003, pp. 180–187. doi: 10.1109/DSD.2003.1231923

201. Rivera W. Scalable parallel genetic algorithms // Artif. Intell. Rev., 2001, vol. 16, pp. 153–168. doi: 10.1023/A:1011614231837

202. Alba E. ParallelMetaheuristic: A New Class Of Algorithms. Wiley, NJ, 2005. doi: 10.1002/0471739383

203. Luong T.V., Taillard É.D. Unsupervised Machine Learning for the Quadratic Assignment Problem // Lecture Notes in Computer Science, 2023, vol. 13838, pp. 118-132. doi: 10.1007/978-3-031-26504-4_9

204. Kirkpatrick S. et al. Optimization by Simulated Annealing // Science, 1983, vol. 220, pp. 671-680. doi: 10.1126/science.220.4598.671

205. Пантелеев А.В. Метаэвристические алгоритмы поиска глобального экстремума. М.: Издательство МАИ-Принт, 2009, 160 с.

206. Коробков В.П. Методы разрезания графа на минимально связные подграфы и их использование в задачах адаптивной обработки информации. В кн.: Адаптация в вычислительных системах. Рига: Зинатне. 1978.

207. Горинштейн Л.Л. О разрезании графов. Изв. АН СССР, сер. Техническая Кибернетика. № 1. 1969. С. 79-85.

208. Рыжков А.П. Алгоритм разбиения графа на минимально связные подграфы. Изв. АН СССР, сер. Техническая Кибернетика, № 6. 1975. С. 122-128.

209. Фещенко В.П., Матюшков Л.П. Итерационный алгоритм разрезания графа на К подграфов. В сб.: Автоматизация проектирования сложных систем (Вычислительная техника в машиностроении). Минск. 1976. Вып. 2. С. 74-77.

210. Коробков В.П. Растригин Л.А. Рандоминизированные алгоритмы агрегации графов. В кн.: Адаптация в вычислительных системах. Рига: Зинатне. 1978.

211. Баранов А.В. Метод и алгоритмы осуществления оптимального отображения параллельной программы на структуру многопроцессорного

вычислителя // Материалы конференции «Высокопроизводительные вычисления и их приложения», Черноголовка, 2000, с. 65-67.

212. NAS Parallel Benchmark. URL: http://www.nas.nasa.gov/NAS/NPB/ (дата обращения: 12.04.2025)

213. Козлов Н.Н., Кугушев Е.И., Энеев Т.М. Параллельные вычисления при решении некоторых задач астрофизики и молекулярной биологии // Матем. моделирование, 2000, №12:7, с. 65–70.

214. Pellegrini F. Scotch and PT-Scotch Graph Partitioning Software: An Overview. In Naumann, U., Schenk, O. (Eds.). Combinatorial Scientific Computing (1st ed.), 2012, pp. 373-406. DOI:10.1201/b11644

215. Hoefler T., Snir M. Generic topology mapping strategies for large-scale parallel architectures // Proceedings of the international conference on Supercomputing (ICS '11). Association for Computing Machinery, 2011, pp. 75-84. doi: 10.1145/1995896.1995909

216. Chen H., Chen W., Huang J., Robert B, Kuhn H. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters // Proceedings of the 20th annual international conference on Supercomputing (ICS '06). Association for Computing Machinery, 2006, pp. 353-360. doi: 10.1145/1183401.1183451

217. Sanchez S. Metaheuristica – Practica 3.a. URL: https://raw.githubusercontent.com/salvacorts/UGR-Metaheuristics/P3/doc/memoria/memoria.pdf (дата обращения: 10 апреля 2025 г.)

218. Баранов А.В., Киселёв Е.А., Телегин П.Н., Сорокин А.А. Программное средство GraphHunter поиска отображения параллельной программы на структуру суперкомпьютерной системы // Программные продукты и системы, 2022, №4, с. 583-597. doi: 10.15827/0236-235X.140.583-597

219. Baranov A.V., Kiselev E.A., Shabanov B.M., Sorokin A.A., Telegin P.N. Comparison of Three Job Mapping Algorithms for Supercomputer Resource

Managers // Lobachevskii Journal of Mathematics, 2022, no 43 (10), pp. 121-133. doi: 10.1134/S199508022213008X

220. Quadratic Assignment Instances. URL: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html (дата обращения: 10.04.25)

221. Drezner Z., Hahn P.M., Taillard E.D. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods // Ann. Oper. Res., 2005, vol. 139, pp. 65–94. doi: 10.1007/s10479-005-3444-z

222. Summary of best solution values known. URL: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/summary_bvk.txt (дата обращения: 10.04.25)

223. Mihic K., Ryan K., Wood A. Randomized decomposition solver with the quadratic assignment problem as a case study // INFORMS J. Comput., 2018, no 30, pp. 295–308. doi: 10.1287/ijoc.2017.0781

224. Baranov A., Aladyshev O., Bragin K. Job Mapping Cyclic Composite Algorithm for Supercomputer Resource Manager// Lecture Notes in Computer Science, vol. 15406, pp. 377–389, 2025. doi: 10.1007/978-3-031-78459-0_27

225. Bokhari S. On the Mapping Problem. // IEEE Transactions on Computers, vol. C-30, no. 3, pp. 207-214, March 1981, doi: 10.1109/TC.1981.1675756

226. Tessema M.M., Che D. Survey and Taxonomy of Volunteer Computing // ACM Comput. Surv, 2019, vol. 52, no. 3, article 59. doi: 10.1145/3320073

227. Binnie C. Password Cracking with Hashcat. In Linux Server Security, C. Binnie (Ed.), 2016. doi:10.1002/9781119283096.ch9

228. Mundkur P., Tuulos V., Flatow J. Disco: a computing platform for large-scale data analytics // Proceedings of the 10th ACM SIGPLAN workshop on Erlang. Association for Computing Machinery, 2011, pp. 84–89. doi: 10.1145/2034654.2034670

229. Баранов А.В., Киселев А.В., Киселев Е.А., Семенов Д.В. Программный комплекс «Пирамида». Свидетельство о государственной

регистрации программы для ЭВМ № 2016617813, Российская Федерация, 14.07.2016.

230. Баранов А.В., Киселёв А.В., Киселёв Е.А., Корнеев В.В, Семёнов Д.В. Программный комплекс «Пирамида» организации параллельных вычислений с распараллеливанием по данным // Труды Международной суперкомпьютерной конференции «Научный сервис в сети интернет: суперкомпьютерные центры и задачи». М: Изд-во МГУ, 2010, с. 299-302.

231. Баранов А.В., Киселёв А.В., Киселёв Е.А. [и др.] Применение программного комплекса «Пирамида» для SPMD-вычислений на гетерогенных массово-параллельных ВС // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции, Новороссийск, 2013, с. 308-310.

232. Воеводин В.В., Соболев С.И. Технология распределенных вычислений X-COM: возможности, задачи, направления развития // Механика, управление и информатика, 2011, №5, с. 183-191.

233. Хританков А.С. Анализ производительности распределенных вычислительных комплексов на примере системы X-COM // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции, 2009, с. 46-52.

234. Huang S., Huang J., Dai J., Xie T., Huang B. The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis // Lecture Notes in Business Information Processing, 2011, vol. 74, pp. 209-228. doi: 10.1007/978-3-642-19294-4_9

235. Zhang Z., Cherkasova L., Loo B.T. Parameterizable benchmarking framework for designing a MapReduce performance model // Concurrency Computat.: Pract. Exper., 2014, vol. 26; pp. 2005– 2026. doi: 10.1002/cpe.3229

236. Costa F., Silva L., Dahlin M. Volunteer Cloud Computing: MapReduce over the Internet // IEEE International Symposium on Parallel and Distributed

Processing Workshops and Phd Forum, 2011, pp. 1855-1862. doi: 10.1109/ipdps.2011.345

237. Naegele T. MapReduce Framework Performance Comparison, 2013. URL: http://www.cs.ru.nl/bachelorscripties/2013/Thomas_Naegele___4031253___MapReduce_Framework_Performance_Comparison.pdf (дата обращения: 17.04.2025)

238. Pavlo A., Paulson E. et al. A comparison of approaches to large-scale data analysis // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (SIGMOD '09). Association for Computing Machinery, 2009, pp. 165–178. doi: 10.1145/1559845.1559865

239. Алексеев А.В., Баранов А.В., Киселёв А.В., Киселёв Е.А. Экспериментальное сравнение технологий распараллеливания по данным Пирамида, MapReduce и MPI // Суперкомпьютерные технологии (СКТ-2014): Материалы 3-й Всероссийской научно-технической конференции. Ростов-на-Дону: Издательство ЮФУ, 2014, Т.1, с. 77-80.

240. Baranov A., Kiselev E., Chernyaev D. Experimental comparison of performance and fault tolerance of software packages pyramid, X-COM, and BOINC // Communications in Computer and Information Science, 2016, vol. 687, pp. 279-290. doi: 10.1007/978-3-319-55669-7_22

241. X-Com: Distributed Computing Software // International Supercomputer Conference, 2010. URL: http://x-com.parallel.ru/download/X-Com_tutorial.pdf (дата обращения: 18.04.2025)

242. Тихонов В.А., Баранов А.В. Организация ЭВМ и систем: учебник для студентов высших учебных заведений, обучающихся по направлению 230100 «Информатика и вычислительная техника», специальности 230101 «Вычислительные машины, системы, комплексы и сети». М.: Гелиос АРВ, 2008. 383 с. ISBN 978-5-85438-179-6.