

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»
(ФГАОУ ВО СПбПУ)

УДК 004.89:.[004.451.42:004.382.2]
Номер гос.регистрации 122112100006-0

УТВЕРЖДАЮ

Проректор по научно-
организационной деятельности



Ю.С. Клочков

30 декабря 2022 г.

ОТЧЕТ
О ПРИКЛАДНОЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Разработка мультиагентного диспетчера управления ресурсами гетерогенной
суперкомпьютерной платформой с использованием методов машинного
обучения и искусственного интеллекта

по теме:

ИССЛЕДОВАНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ МЕТОДОВ
МУЛЬТИАГЕНТНОГО УПРАВЛЕНИЯ РЕСУРСАМИ ГЕТЕРОГЕННОЙ
СУПЕРКОМПЬЮТЕРНОЙ ПЛАТФОРМЫ ИНТЕЛЛЕКТА
(промежуточный, этап 1)

Тема государственного задания FSEG-2022-0001

Руководитель проекта,
профессор, д.т.н.


30 декабря 2022 г.

В.С. Заборовский

Санкт-Петербург 2022

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель проекта,
профессор ВШИИ,
профессор, д-р.техн.наук


30 декабря 2022 г.

В.С. Заборовский
(введение,
заключение)

Отв. исполнитель:
ст. науч. сотр.,
канд.техн.наук


30 декабря 2022 г.

В.А. Мулюха
(разделы 1,2,4,7)


Исполнители:

Ведущий программист


30 декабря 2022 г.

А.В. Андриюшина
(разделы 8,9)

ст. науч. сотр., доцент,
канд.техн.наук.


30 декабря 2022 г.

А.П. Антонов
(раздел 9)

лаборант,
студент


30 декабря 2022 г.

М.Ю. Асташов
(разделы 4,8)

доцент,
канд.техн.наук


30 декабря 2022 г.

Н.В. Воинов
(раздел 4)

старший преподаватель


30 декабря 2022 г.

А.В. Востров
(раздел 1)

доцент,
канд.техн.наук


30 декабря 2022 г.

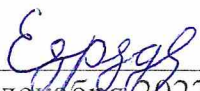
В.В. Глазунов
(раздел 8)

ст. науч. сотр., доцент,
канд.техн.наук


30 декабря 2022 г.


П.Д. Дробинцев
(раздел 2)

Инженер-исследователь


30 декабря 2022 г.

Д.Ю. Еременко
(раздел 6)

ст. науч. сотр.,
канд. физ.-мат. наук


30 декабря 2022 г.

А.С. Ильяшенко
(раздел 6)

ведущий науч. сотр.,
д-р техн. наук


30 декабря 2022 г.


А.И. Каляев
(разделы 3,5)

глав. науч. сотр.,
академик РАН,
д-р техн. наук


30 декабря 2022 г.


И.А. Каляев
(введение, раздел 3)

Старший лаборант, аспирант


30 декабря 2022 г.


И.О. Киселев
(раздел 9)

Инженер, аспирант


30 декабря 2022 г.

А.В. Константинов
(раздел 6)

ст. науч. сотр.,
канд. техн. наук


30 декабря 2022 г.

А.А. Лукашин
(раздел 7)

вед. науч. сотр, доцент,
канд. физ.-мат. наук.


30 декабря 2022 г.

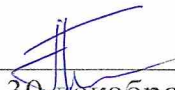
С.В. Малов
(разделы 4)

ассистент, канд. техн. наук


30 декабря 2022 г.

Д.Е. Моторин
(раздел 1)

зам. начальника управления


30 декабря 2022 г.


Е.П. Петухов
(раздел 2)

доцент, доцент,
канд.техн.наук


30 декабря 2022 г.


С.Г. Попов
(раздел 10)

науч. сотр.


30 декабря 2022 г.

З.С. Савельева
(раздел 10)

науч. сотр.


30 декабря 2022 г.


С.А. Семенистый
(разделы 3,5)

доцент, канд.техн.наук


30 декабря 2022 г.

А.В. Силиненко
(раздел 2)

ведущий инженер


30 декабря 2022 г.

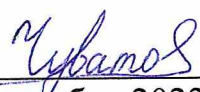
К.В. Трусова
(разделы 2,7)

вед.науч.сотр, профессор,
д-р.техн.наук


30 декабря 2022 г.

Л.В. Уткин
(разделы 6,7)

ассистент


30 декабря 2022 г.

М.В. Чуватов
(раздел 8)

РЕФЕРАТ

Отчет 282 с., 92 рис., 16 табл., 154 источн.

ИНТЕЛЛЕКТУАЛЬНОЕ УПРАВЛЕНИЕ СУПЕРКОМПЬЮТЕРНЫМИ
РЕСУРСАМИ; МАШИННОЕ ОБУЧЕНИЕ; ИСКУССТВЕННЫЙ
ИНТЕЛЛЕКТ; ГЕТЕРОГЕННАЯ ПЛАТФОРМА; МУЛЬТИАГЕНТНЫЙ
ДИСПЕТЧЕР

Объектом исследования являются модели диспетчеризации ресурсов высокопроизводительного вычислителя суперкомпьютерного центра «Политехнический»

Цель работы - разработка методов повышения реальной производительности гетерогенных суперкомпьютерных платформ на основе применения технологий искусственного интеллекта, мультиагентного управления и машинного обучения.

Основные прикладные результаты, полученные в 2022 г.:

- новые методы управления ресурсами высокопроизводительных распределенных вычислительных систем на основе мультиагентного подхода;
- экспериментальные исследования факторов, определяющих эффективность управления различными классами потоков прикладных задания пользователей гетерогенного суперкомпьютерного кластера, работающего в режиме центра коллективного пользования;
- архитектура интеллектуальной системы диспетчерского управления ресурсами гетерогенной суперкомпьютерной системы, в состав которой входят специализированные векторные ускорители и высокопроизводительные отечественные реконфигурируемые вычислительные узлы.
- разработка и ввод в эксплуатацию первой в стране аппаратно-программной гетерогенной реконфигурируемой кластерной платформы, для управление ресурсами которой используются технологии интеллектуальной диспетчеризации.

СОДЕРЖАНИЕ

Введение.....	11
1 Анализ возможностей применения мультиагентного подхода и методов машинного обучения для диспетчеризации ресурсов гетерогенных суперкомпьютерных кластеров и реконфигурируемых вычислительных платформ	14
1.1 Обзор задачи и методов решения диспетчеризации ресурсов гетерогенных суперкомпьютерных кластеров и реконфигурируемых вычислительных платформ	14
1.2 Методы машинного обучения подход к диспетчеризации ресурсов	17
1.3 Мультиагентный подход к диспетчеризации ресурсов	22
1.4 Диспетчеризация ресурсов в суперкомпьютерных системах и облачных вычислениях.....	27
1.5 Мультиагентный подхода и методы машинного обучения для диспетчеризации ресурсов	36
2 Анализ технологий координации взаимодействия и динамического управления вычислительными ресурсами гетерогенных суперкомпьютерных кластеров	38
2.1 Анализ проблем оптимизации работы суперкомпьютерной системы.....	38
2.2 Описание суперкомпьютерной системы	46
2.3 Обзор исследований в области планирования задач и диспетчеризации суперкомпьютерных ресурсов	54
3 Архитектура мультиагентного диспетчера для многоуровневого адаптивного планирования ресурсов гетерогенного кластера	69
3.1 Задача оптимального диспетчирования.....	69
3.2 Мультиагентный подход к диспетчированию	73
3.3 Архитектура диспетчера потока заданий в ГСК	76

3.4 Метод мультиагентного диспетчирования для обеспечения многоуровневого адаптивного распределения ресурсов гетерогенного кластера	82
Основы информационного взаимодействия в мультиагентных диспетчерах.....	85
4Разработка статистической модели функционального управления распределением ресурсов гетерогенного кластера.....	87
4.1 Разработка протокола сбора статистических данных	87
4.2 Разработка и реализация алгоритмов анализа результатов работы суперкомпьютера.	89
4.3 Разведочный статистический анализ результатов работы СКЦ «Политехнический»	91
4.4 Разработка модели предсказания времени выполнения задач.....	104
5Алгоритм мультиагентного диспетчерского управления вычислительными ресурсами гетерогенного суперкомпьютерного кластера с использованием методов искусственного интеллекта	131
6Разработка методов кластеризации, как механизма управления заданиями, исполняемыми с использованием вычислительных ресурсов гетерогенного кластера.....	139
6.1 Анализ данных.	139
6.2 Выбор методов на основании анализа данных	141
6.3 Полученные результаты моделирования.....	146
7Анализ протоколов взаимодействия пользователей, размещаемых задач и ресурсов гетерогенного кластера при решении прикладных задач в предметных областях.....	160
7.1 Постановка исследовательской задачи.....	160
7.2 Анализ ресурсов, выделяемых пользователям	165
7.3 Оценка ресурсов, запрашиваемых пользователями	181
7.4 Анализ предполагаемых зависимостей между различными факторами и целевой величиной.....	193

8Разработка аппаратно-программного макетного образца мультиагентного диспетчера ресурсов гетерогенного вычислительного кластера	198
8.1. Разработка лабораторного стенда вычислительного кластера.....	198
8.2. Разработка протокола сбора метаданных о вычислительных процессах.....	221
8.3. Разработка базы данных хранения метаданных о вычислительных процессах.....	224
9Разработка «высокоуровневой» архитектуры аппаратных компонент мультиагентного диспетчера, использующего каналы информационного обмена между агентами	230
9.1 Проблемы повышения производительности вычислителя с фиксированной архитектурой.....	230
9.2 Влияние процессов мультиагентного планирования SLURM на реальную производительность гетерогенных вычислительных кластеров ..	231
9.3 Использование технологии аппаратной реконфигурации для повышения реальной производительности гетерогенного кластера.....	233
9.4 Направления развития гетерогенной распределенной аппаратно реконфигурируемой суперкомпьютерной вычислительной системы СКЦ Политехнический и интеллектуального диспетчера	246
10Разработка протоколов агрегации информационно-вычислительных и коммуникационных ресурсов агентов, участвующих в диспетчерском управлении ресурсами гетерогенного суперкомпьютерного кластера	248
10.1 Концепция организации данных информационно-вычислительных ресурсов агентов в интересах системы машинного обучения средств диспетчеризации.....	248
10.2 Протоколы организации взаимодействия агентов в интересах системы машинного обучения средств диспетчеризации	251
10.3. Управление потоком задач непрерывной подготовки данных для совершенствования технологий машинного обучения	257

Заключение	262
Список использованных источников	266

ТЕРМИНЫ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

БД	–	База данных
ВРВК	–	Высокопроизводительный реконфигурируемый вычислительный кластер
ГСК	–	Гетерогенный суперкомпьютерный кластер
ПО	–	Программное обеспечение
РВБ	–	Реконфигурируемый вычислительный блок
СК	–	Суперкомпьютер
СКЦ	–	Суперкомпьютерный центр
СПбПУ	–	Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Введение

Целью настоящей работы является разработка методов повышения реальной производительности гетерогенных суперкомпьютерных платформ на основе применения технологий искусственного интеллекта, мультиагентного управления и машинного обучения.

Для распределения заданий на суперкомпьютерах (СК) необходимо для каждого из них иметь оценку его потребности в ресурсах: процессорное время, объем памяти (оперативной, внешнего хранилища), пропускная способность подсистемы ввода-вывода, библиотеки и т.д. На основании этой информации задания назначаются на те или иные процессоры вычислителя, а также формируется расписание выполнения заданий.

При традиционном подходе оценка потребности заданий в ресурсах задается пользователем и поступает в СК вместе с заданием. Пользовательские оценки формируются экспертным путем и могут быть существенно неточными. Предлагается использовать методы машинного обучения для прогнозирования потребности заданий в ресурсах на основе имеющейся истории выполнения задания на СК (том же, или другом). При прогнозировании потребности в ресурсах предлагается использовать в т.ч. числовые параметра задания, характеризующие его масштаб (например, размерность решаемой системы уравнений), при условии, что эти параметры сохраняются в истории.

С учетом того, что энергопотребление (мощность) СК измеряется сотнями кВт и даже мегаваттами, актуальна также задача прогнозирования энергетической «стоимости» выполнения заданий, чтобы оценивать нагрузку на линии электропитания, а также решать задачи энергоэффективного планирования вычислений. Делать такой прогноз можно методами с машинным обучением на основе анализа потребленных ресурсов при предшествующих прогонах задания. Для этого необходимо оснастить вычислитель средствами сбора и анализа данных об истории вычислений, инициированных заданием.

Время выполнения задания на одном и том же вычислителе и со схожими (вплоть до совпадения) входными данными может заметно различаться в связи с т.н. аномалиями, влияющими на характеристики производительности вычислителя. Примерами аномалий являются: частичное разделение ресурсов вычислителя процессами предыдущего задания (они продолжают потреблять ресурсы), снижение частоты процессора из-за перегрева, утечки памяти, перегрузки каналов ввода-вывода. Перечисленные явления, помимо собственно замедления выполнения задания, находят отражение в системных индикаторах и протоколах («логах») работы СК.

Современные суперкомпьютеры представляют собой гетерогенные среды: сочетание многоядерных процессоров общего назначения, графических процессоров (GPU) с многими сотнями и даже тысячами ядер и другими видами сопроцессоров - ускорителей вычислений (FPGA, фотонные, квантовые). В таких условиях выделение заданию среднего масштаба целого CPU и GPU может привести к существенному недоиспользованию ресурсов вычислителя. Методы машинного обучения на основе истории прогонов программ позволяют довести точность прогноза до одного узла или даже до ядра процессора.

В настоящем проекте при помощи методов машинного обучения и мультиагентного подхода осуществляется решение задачи диспетчеризации с анализом истории и текущей динамики системных журналов для выявления изменений параметров исполняемых задач с целью прогнозирования наиболее вероятного времени выполнения задач и снижения времени ожидания заданий в очереди.

В рамках проекта разрабатывается перспективная структура интеллектуальной мультиагентной системы диспетчеризации ресурсов суперкомпьютера, позволяющая осуществлять более эффективное планирование выполнения вычислительных задачи с использованием переназначения параметров заданий в суперкомпьютерной системе.

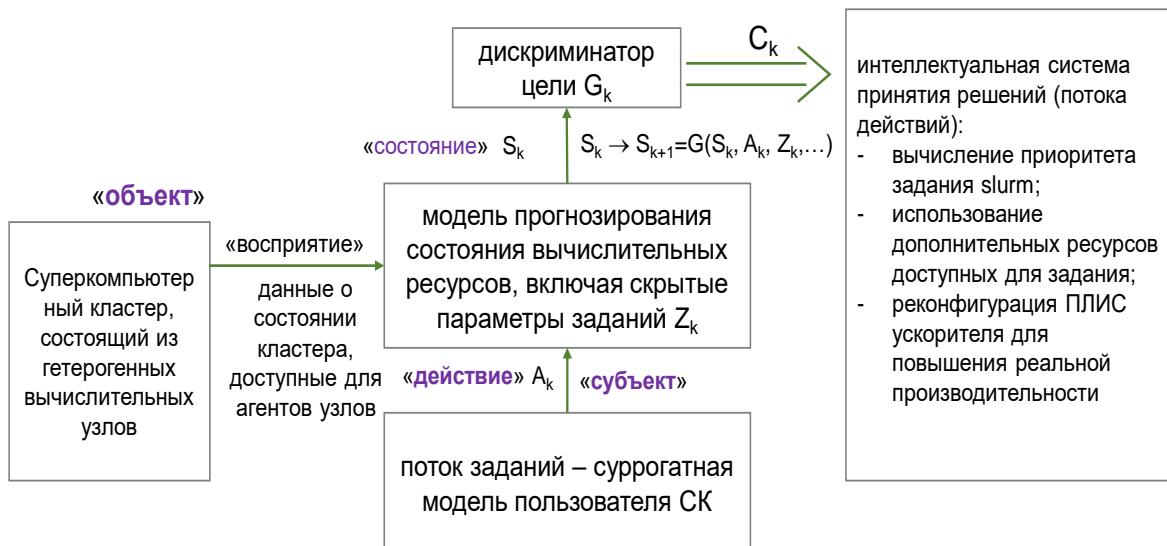


Рисунок – структура интеллектуальной мультиагентной системы диспетчеризации ресурсов суперкомпьютера

1 Анализ возможностей применения мультиагентного подхода и методов машинного обучения для диспетчеризации ресурсов гетерогенных суперкомпьютерных кластеров и реконфигурируемых вычислительных платформ

1.1 Обзор задачи и методов решения диспетчеризации ресурсов гетерогенных суперкомпьютерных кластеров и реконфигурируемых вычислительных платформ

Задачи диспетчеризации ресурсов, планирования и управления очередями в компьютерных сетях являются вычислительно сложными, а также содержат в себе большую степень неопределенности. Дополнительная сложность возникает при формировании классификации вычислительных задач по сложности их решения. Такие классификации опираются на архитектуру ПО, которая имеет большую степень разнообразия, что влечет за собой проблему анализа вычислительной сложности и специфики индивидуально для каждого типа ПО, и проектировать логику для разных групп.

Доказательство NP-полноты при поиске оптимального решения задачи управления очередями рассматривается в статье [1], в которой представлены два случая с ограничениями: все решаемые задачи требуют одной единицы времени; все задачи требуют одной или двух единиц времени, и существует два процессора для вычислений. Рассмотренная постановка имеет вид: $S = \{J_1, \dots, J_n\}$ – множество задач, частичная упорядоченность $<$ на S , весовая функция W от S рассчитывает количество единиц времени для каждой задачи, k количество процессоров. Представленные математические доказательства позволяют разделить разрешимые за полиномиальное время и неразрешимые случаи задачи планирования.

Одним из первых исследований диспетчеризации суперкомпьютеров с массовым параллелизмом является система ASCI Option Red Supercomputer фирмы Intel состоящая из 4500 нод, 9200 процессоров Pentium Pro, 594

Гигабайт RAM и двух независимых 1 Терабайтных диска [2]. Эта машина первой преодолела рубеж пиковой производительности в 1 Терафлопс.

Системы обеспечивающие высокопроизводительные вычисления сложных задач, разделяемые на подзадачи, могут быть распределенным кластером из множества персональных компьютеров. Пример такой грид-системы рассматривается в работе [3]. Сеть определяется как гибкое, безопасное, координируемое, совместное использование ресурсов виртуальной организацией. Представлена открытая грид-архитектура с протоколами, службами и интерфейсами прикладного программирования обеспечивают совместное использование ресурсов. Такие системы выполняют диспетчеризацию на основе выделенных вычислительных узлов самой системы.

Параллельного вычисление на суперкомпьютере расширяет возможности планирования и управления ресурсами при выполнении задач. Фейтельсон [4] представил широкий обзор теоретических и практических подходов по распараллеливанию вычислений. Рассмотрены наиболее распространенные показатели планирования работ: пропускная способность, использованные ресурсы и время отклика. Авторами предложены комбинированные метрики позволяющие более эффективно работать с очередями, состоящими из коротких задач.

Одной из основных проблем построения плана при загрузке ресурсов суперкомпьютера является существенная переоценка времени выполнения задачи пользователем. Данный вопрос рассматривается в работе [5] в системе IBM SP2 и MPP с распределенной памятью. Используемый подход к выделению ресурсов по принципу FCFS (first-come first-serve) обладает высокой предсказуемостью, но может приводить к низкой загрузке. Разработанный планировщик EASY (the Extensible Argonne Scheduling sYstem), который реализует стратегию жадного обратного заполнения: короткие задания перемещаются вперед и заполняют пробелы в расписании, если они не задерживают первое задание в очереди. В работе показано что, при

такой стратегии пользовательская переоценка времени выполнения задачи делает систему планирования более гибкой. Аналогичный подход использован [6] при решении задачи планирования, поддерживающего приоритизацию для пользователя и администратора.

Событийно-ориентированный подход к стратегиям планирования исследуется в работе [7]. В качестве рабочей машины использовалась Intrepid — это система Blue Gene/P 557 TF с 40 стойками. Система состоит из 40960 вычислительных узлов с более чем 160000 ядер и связанных узлов ввода-вывода, серверов хранения и сети ввода-вывода. Исследуемый механизм выбора заданий на основе служебных программ позволяет пользователям настраивать стратегии планирования для конкретных ограничений без изменения кода планировщика. Разработанный симулятор планирования и управления событиями Qsim может быть использован напрямую с менеджером ресурсов Cobalt.

Грид-архитектура обладает спецификой, которую необходимо учитывать в задачах планирования. В работе [8] рассматриваются вычислительные модели для планирования и решения задач в грид-системах с использованием эвристического и метаэвристического подходов. Рассмотрены критерии планирования, такие как статическая и динамическая среда, многообъектность, адаптивность и т.д. В ходе исследования выявлена сложность планирования в грид-системах по сравнению с классическими параллельными и распределенными системами, а также показана полезность эвристических и метаэвристических подходов для разработки эффективных грид-планировщиков.

Реконфигурируемые вычисления представляют собой архитектуру, сочетающая гибкость программного обеспечения с высокой производительностью аппаратного обеспечения за счет обработки с гибких высокоскоростных вычислительных структур, таких как программируемые вентильные матрицы. В работе [9] представлена реконфигурируемая вычислительная платформа, в которой используется эвристический подход к

загрузке контекста, время его выполнения может быть локально минимизировано за счет параллельного выполнения пользовательских блоков без зависимостей. Время реконфигурации сводится к минимуму за счет предварительной выборки пользовательских блоков следующего уровня во время выполнения пользовательских блоков текущего уровня.

Концепция реконфигурируемой высокопроизводительной инфраструктуры как услуги (Hi-IaaS) и ее реализация в системе представлена в работе [10]. Система состоит из программного обеспечения для перекрестного управления заданиями и ресурсами с реконфигурируемым оборудованием, которое может создавать компьютерное оборудование из пула ресурсов путем подключения/отсоединения компьютеров и устройств на уровне периферийных компонентов Interconnect Express (PCI Express) в соответствии с запросом пользователя. Он также оснащен промежуточной/программной платформой для высокопроизводительного анализа данных, которая используется в качестве высокопроизводительных вычислений. Результаты моделирования показали, что эффективность реконфигурируемой платформы для коэффициента полезности ресурсов увеличилась с 20% до 47%, а время выполнения заданий сократилось на 42% в 64-узловой системе. Система может исключить длительные ожидания, зафиксированные в реальных данных, собранных в полугодовом отчете о реальной работе вычислительного центра.

1.2 Методы машинного обучения подход к диспетчеризации ресурсов

Развитие машинного обучения как группы методов решения сложно формализуемых задач нашло свое применение в задачах диспетчеризации вычислительных ресурсов. Современные кластеры суперкомпьютеров требуют сложного математического описания и быстрой адаптивности к возможным изменениям структуры аппаратной части, что приводит к низкой

эффективности классических подходов распределения ресурсов в таких системах.

Использование методов обучения с подкреплением рассмотрено в работе [11], где исследуется стратегия представлена аппроксимирующей функцией, не зависимой от функции ценностей и обновляемой согласно градиенту ожидаемого вознаграждения. Главным результатом является представление градиента в форме оценки на основе опыта с помощью приближенной функции действия-вознаграждения. Это позволило математически доказать, что итеративная стратегия с произвольной аппроксимацией дифференцируемой функцией сходится и является локально оптимальной.

В статье [12] исследуется диспетчеризация в системах с неидентичными вычислительными машинами используя подход дополненной нейронной сети AugNN (augmented-neural-network). Предложено 12 базовых эвристик, которые по сути являются комбинацией четырех правил приоритета задачи и трех правил приоритета машины. Правила приоритета задачи: HLF (Highest-Level-First), HTRPTF (Highest-Total-Remaining-Processing-Time-First), SLFTF (Smallest-Latest-Finish-Time-First) и MSF (Minimum-Slack-First). Жадное правило приоритета машины: FAMF (Fastest-Available-Machine-First); нежадные правила: FAMF-CW-1 (Fastest-Available-Machine-First-With-Conditional-Wait-1) и FMF-CW-2 (Fastest-Available-Machine-First-With-Conditional-Wait-2). Исследования проводились на 100 случайно сгенерированных задачах. Использование предложенных AugNN относительно однопроходной эвристики повысило эффективность планирования от 24,4% до 50%, при этом наилучший результат дает комбинация HTRPTF/FAMF-CW-1, далее HLF/FAMF-CW-2 и HTRPTF/FAMF-CW-2.

Приложения для обработки больших данных и глубокие нейронные сети, требуют высокопроизводительных вычислений, что увеличивает спрос на ресурсы. Управление вычислительными ресурсами и их выделение для

таких приложений требуют технологий реконфигурации и распределенных вычислений. В работе [13] представлен кластерный подход с четырьмя формами для управления аппаратными ресурсами. Подход основан на двух четырехъядерных алгоритмах кластеризации ресурсов и разделения приложений, которые решают задачу отображения иерархическим образом. Создание кластеризованных ресурсов от верхнего к нижнему уровню шаг за шагом формируют глобальную структуру конфигурации сопоставления и упрощает пространство поиска для нахождения окончательного решения. Управление кластером нуждается в точной функции оценки стоимости, основанной на расстоянии между ресурсами.

В статье [14] представлен широкий обзор применения искусственных нейронных сетей в задачах производственного планирования, которые заключаются в распределении ограниченных ресурсов с течением времени и определение последовательности операций, с учетом ограничений и критериев оптимизации. Анализ разделяет работы по четырем используемым архитектурам: нейронные сети Хопфилда, многослойный перцептрон, конкурентные сети и гибридные сети. Сети Хопфилда хорошо применимы для задач комбинаторной оптимизации, целевая функция и ограничения задачи формируют подходящую энергетическую функцию, а состояние сети изменяется чтобы минимизировать эту функцию. Многослойные сети обучаются с помощью алгоритма обратного распределения и нуждаются в обучающем наборе для решения конкретной задачи. Многослойные сети эффективно применяются в задачах обобщения. Конкурентные сети состоят из входного слоя, все узлы которого связаны с обрабатывающими узлами конкурентного слоя. Сумма произведения весовых коэффициентов входов для узла сравнивается с суммами значений соседних узлов, в результате только один нейрон в конкурентной группе будет иметь ненулевой выходной сигнал. Эти сети лучше всего применимы к задачам оптимизации и классификации. Гибридные сети представляют собой комбинацию подходов, один из которых является главным и решает поставленную задачу, а второй ему помогает.

В работе [15] рассматриваются показатели параллельного планирования выполнения задач на суперкомпьютерах и их сходимость. Процесс поступления и параллельного перераспределения заданий вызывает скачкообразные колебания нагрузки системы, что приводит к задержкам сходимости симуляций, используемых для оценки плана. Данные собирались на суперкомпьютерах с соответствующими метриками: LANL-CM5: The Los Alamos National Lab 1024-node CM-5 (201387 jobs, 10/1994 to 9/1996); SDSC-Par: The San-Diego Supercomputer Center 416-node Intel Paragon (115595 jobs, 1/1995 to 12/1996); CTC-SP2: The Cornell theory Center 512-node IBM SP2 (79296 jobs, 7/1996 to 5/1997); KTH-SP2: The Swedish Royal Institute of Technology 100-node IBM SP2 (28490 jobs, 10/1996 to 8/1997); SDSC-SP2: The San-Diego Supercomputer Center 128-node IBM SP2 (67665 jobs, 4/1998 to 4/2000); LANL-O2K: The Los Alamos National Lab 2048- node Origin 2000 cluster (122233 jobs, 12/1999 to 4/2000). Рассмотренные различные метрики и подходы к планированию показали сильную зависимость от типа нагрузки и изучаемой системы.

В современных высокопроизводительных системах преобладают пакетные задания, и их эффективное планирование является решающим для эффективности системы в целом. Широко распространенные эвристические функции приоритета после настройки и развертывания зачастую не способны к адаптации при изменении рабочих нагрузок, целей оптимизации или системных настроек, что снижает эффективность. Подход к решению данной задачи исследуется в работе [16]. Представленный автоматизированный планировщик пакетных заданий, основанный на обучении с подкреплением, RLScheduler, может обучаться стратегиям планирования посредством непрерывного использования метода «проб и ошибок». Исследования проводились на Intel Xeon Silver 4109T CPU и 32GB DDR4 DRAM, при этом сортировка кратчайших работ производится за 0.71ms, а принятие решений RLScheduler для 128 ожидающих за 0.30ms. Кроме того, временные затраты не

будут расти от количества заданий, так как количество заданий ограничивается на начальном этапе.

К задаче распределение ресурсов применяются методы обучения с подкреплением. В статье [17] предложена система DeepRM, основанная на глубоком обучении с подкреплением, которая переводит многокритериальную задачу упаковки ресурсов в задачу обучения. DeepRM является нейронной сетью с полностью связанным скрытым слоем из 20 нейронов и 89451 параметром, при обучении использовалось 100 различных наборов, в каждой итерации 20 симуляций Монте-Карло параллельно, параметры сети обновлялись алгоритмом rmsprop со скоростью обучения 0,001. Разработанная система адаптируется к нагрузкам и условиям, быстро сходится и обучается стратегиям на ретроспективных данных. Оценка эффективности работы алгоритма проводилась путем сравнения стандартных тестов со следующими агентами: Shortest Job First (SJF), который распределяет задачи по возрастанию временных затрат, Packer, который распределяет задания по возрастанию соответствия приоритета задания и доступностью ресурсов, и Tetris, который уравнивает приоритет коротких заданий для упаковки.

Динамическое планирование задач высокопроизводительных вычислений обычно выполняется с применением специфических эвристик, основанных на параметрах задачи и стратегии обратного заполнения. В статье [18] предложена и исследована методология определения эвристик, которые эффективно работают на разнообразных типах задач и архитектур платформ, основанную на моделировании и машинном обучении, для формирования стратегии динамического планирования. Модели симуляции и генерации рабочих нагрузок определяют характеристики задач и снижают среднее время в очереди выполнения. Входными данными практических исследований использованы открытые данные из Parallel Workloads Archive [19], а для оценки полученных нелинейных функций выбран набор конфигураций высокопроизводительных машин в диапазоне от 338 до 163 840 ядер, средней

нагрузкой от 59,6% до 85,2% и измерениями с 1997 по 2011 год. В результате исследуемая стратегия сократила медианное время в 12,5 раза по сравнению с наиболее эффективной ad-hoc стратегией.

Методы машинного обучения в задачах диспетчеризации ресурсов суперкомпьютерных систем показывают хорошую адаптивность к изменению нагрузки, реконфигурации платформы, а результаты планирования последовательности выполнения работ и распределения вычислительных ресурсов учитывают ошибки и особенности пользователей на прошлых итерациях планирования.

1.3 Мультиагентный подход к диспетчеризации ресурсов

Применение мультиагентных методов широко распространено в распределенных системах, таких как интернет вещей (IoT), роботизированные группы, вычислительные кластеры, телекоммуникации и т.д. Агенты представляют собой аппаратные или программные автономные сущности, которые могут функционировать как индивидуальной, так и в составе группы, а также формировать кооперативные и конкурентные стратегии. Агенты способны действовать согласно поставленной целевой функции и обладают интеллектуальной компонентой.

Развертывание приложений в IoT требует управления большим разнообразием устройств, обеспечения их связи через различные протоколы связи, устранения несовместимости между базовыми сетями даже во время выполнения. В статье [20] представлена агентная платформа Sol, которая предоставляет решение для поддержки неоднородности устройств и связи, для создания гибкой инфраструктуры связи и эффективной групповой передачи данных. Платформа определяет точку доступа Sol (т. е. инфраструктуру агентов), которая облегчает (1) связь и взаимодействие агентов, работающих на разном наборе устройств (таких как датчики SunSPOT, устройства на базе Android и мобильные телефоны) и с использованием разных протоколов связи; (2) групповая доставка сообщений (связь «один ко многим») эффективным

образом. Точка доступа Sol поддерживает собственные протоколы связи каждого устройства (ZigBee, WiFi) и действует как шлюз, выполняя определенные функции для обеспечения совместимости. На уровне агента предоставляется индивидуальная настройка внутренней архитектуры, необходимая для использования протоколов связи, с учетом контекста и потребностей приложения. Эта гибкость внутренней структуры агента также упрощает одновременное использование различных механизмов распространения сообщений. Работоспособность платформы проиллюстрирована несколькими сценариями в Интеллектуальном музее, а также продемонстрирована осуществимость и преимущества подхода с точки зрения времени отклика реконфигурации и беспроводного обмена данными, столь важного в IoT.

В работе [21] представлен обзор кооперативных многоагентных систем и распределение ролей группы агентов в концепциях проектирования. Роль определяется в самом общем смысле для обозначения функций и обязанностей агента. Важность ролей для многоагентных систем выражается в: концепции разработки, разработчики описывают задачи с точки зрения ролей, необходимых команде для выполнения каждой задачи; возможности агентам сосредоточиться на небольшом подмножестве моделей поведения, что приводит к специализации и повышению общей производительности; сокращении как физических, так и виртуальных конфликтов за выполнение задачи или доступ к ресурсам. Распределение ролей рассматривается в контекстах биологии, социальных наук, робототехники и разработки программного обеспечения. Модели распределения ролей требуют компромисса между универсальностью и вычислительной сложностью. Методы построения моделей делятся на две категории: методы, в которых реакция агентов на задачи определяет их роли, и методы, в которых агенты изучают новое поведение, которое можно описать с помощью ролевой терминологии. Децентрализованный многоагентный подход к решению задач может обеспечить более надежные, более масштабируемые и более дешевые

решения, чем централизованные подходы. Однако многоагентные системы подвержены патологическому поведению, а их распределенная природа часто не позволяет агентам вычислять глобально оптимальные решения.

Настройка и развертка распределенной сети приложений в интеллектуальных средах, учитывающих неоднородность устройств и динамику, и конфиденциальность ресурсов может решаться по средствам применения многоагентных платформ, пример такого решения представлен в [22]. Ключевыми аспектами многоагентной платформы для развертки эмбиентных систем являются: (1) онтология позволяющая описывать среду (аппаратные устройства и приложения) с помощью графов, причем развертывание приложения формализовано с помощью гомоморфизма графа между шаблонами приложений и графа инфраструктуры; (2) распределенный алгоритм сопоставления графов, который анализирует в графе распределенной инфраструктуры аппаратные объекты, соответствующие ограничениям и находит подходящие для развертывания приложений; (3) многоагентная система, состоящая из четырех классов целевых агентов для обеспечения четкого разделения между аппаратным и программным уровнем, и конфиденциальности ресурсов во внешних системах, то есть разделение прикладного и инфраструктурного уровня.

Системы принятия решений могут применяться для решений задач распределения ресурсов. Классической задачей в этой сфере является – повторяющаяся дилемма заключенного, она идеально подходит в качестве модели для анализа взаимодействия агентов в сложных сетях. В работе [23] предложена адаптивная стратегия на основе обучения с подкреплением. Агент с адаптивной стратегией может принимать решения с учетом долгосрочного вознаграждения. Оценка эффективности проводилась на основе трех турниров в двух средах. Каждый из теоретико-игровых турниров может быть представлен в виде кортежа $G(N, A, F)$, где N — количество агентов, $A = \{C, D\}$ — множество действий агентов, C и D — взаимодействие. и дефект, а F — функция выигрыша для каждого действия. Сеть с квадратной решеткой и

безмасштабная сеть вводится в качестве среды моделирования. Результаты моделирования показали, что адаптивные агенты могут сотрудничать со своими противниками, не теряя конкурентоспособности. Предложенная стратегия может добиться взаимного сотрудничества, что не только долгосрочно выгодно для команды, но и повышает отказоустойчивость безмасштабной сети.

В статье [24] рассматривается задача разработки адаптивного алгоритма обучения для поиска равновесия по Нэшу в игре с ограниченным обменом ресурсами между агентами с неполной информацией. Каждый агент использует схему обучения для генерации распределения вероятности действия на основе собственной закрытой информации для максимизации индивидуальной усредненной полезности. Если одна из допустимых смешанных стратегий сходится к равновесию по Нэшу с вероятностью единица, то усредненная полезность и объем торговли почти наверняка сходятся к ожидаемым. В работе доказывалось, что в силу строгой диагональной вогнутости функции полезности мы также обнаружили, что равновесие по Нэшу регуляризованной функции Лагранжа для повторяющейся некооперативной игры уникальна и все допустимые оптимальные стратегии будут сходиться к этой единственной точке равновесия.

Разработанная [25] платформа MAgent – это исследовательская система, предназначенная для поддержки многоагентного обучения с подкреплением, фокусируется на поддержке задач и приложений, работающих с сотнями и миллионами агентов. Взаимодействие популяций агентов обеспечивает обучение оптимальных стратегий, а также отслеживать поведение отдельных агентов их социальные связи. Высокая масштабируемость платформы обеспечивает гибкие конфигурации для разработки специализированных сред и агентов.

Конкурентные и кооперативные среды, в которых действуют агенты могут существенно влиять на выбор стратегии и результат работы системы. В

статье [26] изучаются методы глубокого обучения с подкреплением для многоагентных доменов. Проведенный анализ сложности традиционных алгоритмов обучения группы агентов представляет следующие результаты: Q-обучение ограничено в связи с не стационарностью среды; градиентная стратегия показывает сильное расхождение при увеличении числа агентов. Предложена адаптация метода актор-критик (actor-critic), учитывающая стратегии других агентов и способна обучаться стратегиям, требующим сложной кооперации нескольких агентов. Разработанный авторами подход превосходит традиционные алгоритмы обучения с подкреплением в конкурентных и кооперативных средах.

Многоагентные системы могут обеспечивать интеграцию распределенных вычислительных систем, устройств IoT или робототехнических групп. В работе [27] рассматривается платформа BEMOSS (Building Energy Management Open Source Software), предназначенная для управления энергопотреблением зданий, обнаружения и контроля систем отопления, вентиляции и кондиционирования воздуха, освещения и контроллеров нагрузки в малых и средних коммерческих зданиях. Основными задачами, выполняемыми платформой, являются повышение энергоэффективности, снижение энергопотребления и содействие внедрению системы реагирования в зданиях с протоколом OpenADR. Функциональные возможности программной платформы BEMOSS и ее интеграция с устройствами IoT продемонстрированы на лабораторном стенде, расположенном в Технологическом институте перспективных исследований штата Вирджиния.

Современные темпы развития вычислительной техники позволяют использовать мультиагентный подход к распределенным системам, которые состоят из множества ресурсов, выполняющих индивидуальные функции, и каналов связи, обеспечивающих взаимодействия элементов для выполнения функционального задания. Координация работы ресурсов таких распределенных систем, при выполнении пользовательских запросов,

занимается автоматизированный диспетчер. Если требуется выполнять только одно задание, то диспетчеризация может быть проведена единожды и заранее. А в случае множества разнообразных заданий, поступающих в не определенный момент времени, необходимо решать задачу распределения ресурсов в реальном времени, с учетом специфики задачи и текущим состоянием ресурсов вычислительной системы. Задача автоматизации процесса диспетчеризации является вычислительно сложной для ее решения могут быть использованы методы теории массового обслуживания.

1.4 Диспетчеризация ресурсов в суперкомпьютерных системах и облачных вычислениях

Парадигма облачных вычислений, которая основана на удаленных высокопроизводительных вычислительных средах, позволяет решать широкий спектр задач на запрашиваемом пользователем объеме ресурсов. Такой подход существенно усложняет алгоритмы диспетчеризации. Разработанные эвристические и метаэвристические методы применяются для оптимизации энергетических и временных затрат. Двухкритериальный гибридный алгоритм планирования рабочего процесса в облачных вычислениях представлен в работе [28]. Предлагаемый гетерогенный алгоритм гравитационного поиска (HGSA) представляет собой гибрид популярной метаэвристики, алгоритма гравитационного поиска (GSA) и столь же популярной эвристики, гетерогенного самого раннего времени завершения (HEFT) для планирования приложений рабочего процесса. Результаты показывают, что HGSA работает лучше, чем гибридный генетический алгоритм на 14%, GSA на 20% и HEFT на 35% в отношении значения пригодности. Результаты получены с помощью статистического теста дисперсионного анализа (ANOVA).

Планирование совокупности рабочих процессов в распределенных вычислительных системах применяется в облаках «инфраструктура как услуга» (IaaS) и является NP-полной задачей. В статье [29] предложен

многокритериальный оптимизационный подход к решению задачи планирования рабочих процессов, основанный на динамической модели теории игр, направленный на сокращение времени создания рабочего процесса, снижение общей стоимости и оптимизации распределения рабочей нагрузки системы между гетерогенными облачными виртуальными машинами.

Спрос на «программное обеспечение как услуга» (SaaS) в облачных средах растет в геометрической прогрессии и требует обеспечения доступа к вычислительным ресурсам пользователям со всего мира. Централизованная архитектура ограничивает возможности широкого доступа и усложняет диспетчеризацию. Для решения этой задачи требуются кластеры крупномасштабных распределенных серверов. В статье [30] предлагается двухуровневый регионально-локальный алгоритм управления распределенными ресурсами, допусками запросов и балансировки нагрузки. Алгоритм может быть масштабирован серверами приложений и прокси-серверами, поскольку каждый компонент системы выполняет задачу управления независимо. Алгоритм балансировки нагрузки, основанный на теории игр, представленный в этом исследовании, способен адаптироваться к изменениям ресурсов и новым условиям путем постоянного выделения или освобождения облачных ресурсов. При развертывании этого алгоритма средний процент отклоненных запросов и значение коэффициента балансировки нагрузки улучшаются по сравнению с конкурирующими алгоритмами примерно на 15% и 4,5% соответственно.

Задача оптимизации планирования работы крупных приложений с параллельными процессами в гетерогенных вычислительных системах, таких как гибридные облака является NP-полной и является определяющей для требований QoS (Quality of Service). В статье [31] рассматривается постановка задачи оптимизации большого количества однородных параллельных пакетов задач, которые являются узким местом. Задача планирования решается алгоритмом кооперативной игры с двухкритериальной оптимизацией по времени

выполнения и экономическим затратам и с двумя ограничениями – по пропускной способности сети и объему памяти. Предложенный алгоритм показал лучшие результаты времени выполнения вычислительных задач по сравнению с такими жадными подходами как G- Min-min, G-Max-min или G-Sufferage, причем эффективность растет с простым размера инфраструктуры.

Динамический теоретико-игровой подход для решения задач планирования выполнения научных вычислений в облачных системах рассмотрен в исследовании [32]. Предложена много целевая структура планирования рабочего процесса направлена на сокращение периодов создания и стоимости облака при максимизации распределения нагрузки между гетерогенными облачными виртуальными машинами. Проверка работоспособности разработанной системы проводилась на случайно сгенерированных шаблонах научных рабочих процессов и на данных сторонних коммерческих облаков.

Для решения задач многокритериальной оптимизации при наличии нескольких целевых функций, таких как планирование операций мультиагентных систем, может быть использован метод оптимизации роя частиц. Расширение этого подхода с доминированием по Парето и использование внешнего хранилища частиц представлено в работе [33]. Разработанный авторами алгоритм использует оператор мутации, который меняется со в ходе работы, что облегчает настройку весов. Результаты проверены с использованием стандартной методологией эволюционной многокритериальной оптимизации.

Планирование многоцелевого рабочего процесса в гетерогенных облачных средах IaaS (infrastructure-as-a-service) в работе [34] реализовано на основе стохастических марковских игр и разработанной децентрализованной структуры многоагентного обучения с подкреплением MARL (Multi-agent Reinforcement Learning) с использованием DOQ (deep-Q-network), которая способна рассчитывать коррелированные равновесные решения планирования рабочего процесса. Исследование структуры проведенное на Amazon EC2 и

шаблонах научных процессов показало превосходство над базовыми алгоритмами, такими как NSGA-II, MOPSO и GTBGA.

Эволюционные и роевые алгоритмы используются в задачах планирования m работ на n ресурсов для облачных вычислений и позволяют получать субоптимальные решения. В работе [35] предложен алгоритм MOBFOA (multi-objective bacteria foraging optimization algorithm), который является модификацией алгоритма BFOA для решения многокритериальных задач планирования с Парето оптимальным фронтом. Модификация заключается в выборе позиций бактерий из доминирующих и недоминирующих фронтов для получения разнообразия решений, что увеличивает точность и скорость сходимости за счет введения адаптивного размера хемотаксического шага. Сравнение производительности предложенного алгоритма производилось по скорости сходимости к Парето оптимальному фронту и распределения решений в пространстве, с NSGA-II и OMOPSO.

Планирование расписаний по критериям энергопотребления и надежности в гетерогенных вычислительных системах с приоритезацией рассматривается в работе [36]. Предложен двухкритериальный генетический алгоритм, обеспечивающий низкое энергопотребление и высокую надежность системы. Для оценки производительности используется три типа нагрузок: алгоритм Гаусса-Жордана, параллельная декомпозиция LU и дискретное преобразование Лапласа, а также реальные приложения на графах. Алгоритм показал значительное превосходство по показателям поиска компромиссных решений MOHEFT (multi-objective heterogeneous earliest finish time) и MODE (multi-objective differential evolution).

В статье [37] анализируется MOHEFT, где планирование списка основано на Парето эвристике, которая предоставляет пользователю набор компромиссных решений на выбор. Критериями оптимизации выбраны время и затраты вычислительных ресурсов. Рассматриваемый метод проверяется на коммерческом облаке Amazon EC2, и сравнивался с HEFT и SPEA2*. Во всех

экспериментах MOHEFT рассчитывал расписание с теми же затратами времени, что и существующие аналоги, но с меньшими затратами ресурсов.

Технология IaaS позволяет удаленно масштабировать ресурсы и обеспечить их доступность с оплатой по факту использования. Пользователи, запрашивающие доступ к таким сервисам, задают ограничения по времени и ресурсам, которые зачастую противоречат друг другу. К решению этой двухкритериальной задачи с конфликтующими целевыми функциями предложен алгоритм гибридной оптимизации роя частиц (HPSO) на основе сортировки без доминирования в исследовании [38]. Гибрид состоит из двух алгоритмов BDHEFT (Budget and Deadline constrained Heterogeneous Earliest Finish Time) и PSO (Particle Swarm Optimization). Результатом работы алгоритма является набор Парето-оптимальных решений, из которых пользователь должен выбрать наиболее подходящее решение. Производительность предложенной эвристики сравнивается с современными многоцелевыми метаэвристиками, такими как NSGA-II, MOPSO и ϵ -FDPSO, и показывает большую эффективность по показателям сходимости к истинному оптимальному фронту Парето и равномерно распределенным решениям с небольшими затратами на вычисления.

Двухкритериальная оптимизация затрат и сроков планирования рабочих процессов в облаках IaaS рассматривается в работе [39]. Разработанный алгоритм FDHEFT (fuzzy dominance sort based heterogeneous earliest-finish-time) с интегрированным механизмом нечетким доминированием и эвристикой планирования списка HEFT. Исследование проводилось на синтезированных и реальных данных, отношение затраченных ресурсов и временем планирования системы FDHEFT достигает двух порядков по сравнению с SPEA2*, NSPSO, MOHEFT, и ϵ -Fuzzy PSO.

Обеспечение QoS в сервисно-ориентированных вычислительных системах состоит из двух ключевых требований – времени выполнения задачи и стоимости. В статье [40] предложен алгоритм гетерогенного планирования с ограниченным бюджетом HBCS (Heterogeneous Budget Constrained

Scheduling), который минимизирует время выполнения пользовательского приложения с учетом указанного ограниченного бюджета. Сравнение предложенного алгоритма показало, что HBCS показывает более низкое время выполнения (до 30% при том же бюджете) для всех бюджетных ограничений в случаях с более высокой гетерогенностью. Коэффициент успеха планирования (PSR) для HBCS в некоторых случаях до 2,7 раз выше других алгоритмов. Результаты, полученные для двух реальных приложений, LIGO и Epigenomics, согласуются со случайно сгенерированными рабочими процессами для всех метрик. HBCS имеет ту же временную сложность, что и GreedyTimeCD и BHEFT, при постоянном времени работы для всех диапазонов ограничений бюджета для данного рабочего процесса и платформы. Алгоритмы LOSS имеют шаг, основанный на поиске, и поэтому время работы зависит от количества итераций поиска решения, где для низких бюджетов количество итераций значительно выше.

Точное прогнозирование времени выполнения заданий в сложной высокопроизводительной динамической вычислительной среде – сложная задача, зависящая от большого количества ограничений и параметров. Одна единственная стратегия не подходит для решения всех видов гетерогенных задач. Для решения этой проблемы в [41] предложена модель совместного прогнозирования с использованием нескольких стратегий MSCPM (multi-strategy collaborative prediction mode) для выполнения онлайн задач в режиме рантайм. Оценка точности прогнозов проводилась на введенной авторами концепции «Обеспечение точности прогнозов». Экспериментальные результаты, основанные на кластерной вычислительной среде, показывают, что прогнозы модели совместного прогнозирования согласуются с оптимальными прогнозами в рамках результатов, обеспечиваемых одиночными стратегиями, а MSCPM обеспечивает повышенную точность прогнозирования времени выполнения онлайн-задач.

Приложения с параллельным выполнением состоит из стохастических приоритетизированных задач, причем время обработки задач и внутреннего

взаимодействия между задачами – случайные величины, подчиняющиеся определенным распределениям вероятностей. Планирование таких стохастических задач в гетерогенной кластерной системе с переменными вычислительными ресурсами исследуется в работе [42]. Авторы доказали, что интеграция ожидаемых значений и дисперсии случайных величин является функцией, которая влияет на производительность алгоритм стохастического планирования. Разработан стохастический алгоритм планирования динамического уровня SDLS, который использует стохастический нижний уровень и стохастический динамический уровень для создания расписаний высокого качества. Производительность алгоритма SDLS сравнивается с тремя существующими алгоритмами планирования: Rob-HEFT, SHEFT и HEFT. Результаты моделирования показывают, что детерминированные алгоритмы планирования, такие как HEFT и DLS, не подходят для стохастического планирования задач. Алгоритм стохастического планирования SDLS показывает большую производительность с точки зрения временных затрат, ускорения стандартного отклонения времени изготовления в кластерных системах по сравнению с Rob-HEFT и SHEFT.

В статье [43] предложен алгоритм IPEFT (Improved Predict Earliest Finish Time) для статистического планирования задач в гетерогенной вычислительной среде. В отличие от существующих алгоритмов, основанных на раннем времени окончания, IPEFT вычисляет ранг приоритета на основе пессимистичной таблицы стоимости (PCT) на этапе определения приоритетов задачи, а таблица PCT вычисляется до планирования. Каждая пара задача-процессор с таблицей предоставляет максимальное время обработки самого длинного пути от непосредственных преемников текущей задачи к выходной задаче, причем каждая задача назначается самому слабому процессору. Приоритет в PCT вычисляется предварительным планированием задач, которые генерируют наихудшие исходы. Алгоритм учитывает родительские и критические задачи таким образом, чтобы гарантировать кратчайшее время выполнения критических задач на следующем этапе. Новый алгоритм имеет

ту же временную сложность, что и алгоритмы HEFT и PEFT, но обеспечивает более эффективную длину расписаний в случайных графах при количестве задач от 10 до 400, имеет большую надежность и частоту лучших результатов.

Планирование направленного ациклического графа (DAG) в сетевых системах должно максимизировать параллелизм и минимизировать межпроцессорное взаимодействие для минимизации времени сквозного отклика в наихудшем случае (WCRT). Для решения этой задачи в работе [44] исследуется сквозное синхронизированное планирование, основанное на неоднородности и ограничениями приоритета во встроенных сетевых системах. Для оптимизации алгоритма планирования списка задач используется HPRV (Heterogeneous Priority Rank Value) для сортировки и HSV (Heterogeneous Selection Value) для выбора процессора. Предложен алгоритм HSV_CC (Heterogeneous Selection Value on Communication Content Conference), основанный на модели конкуренции связи, для учета неоднородности сети и синхронизации задач и сообщений в средах с конфликтом связи. Эффективность алгоритмов проверялась на стандартизированных тестах.

Алгоритмы планирования DAG могут генерировать расписания требующие чрезмерно большого числа процессоров. Решение этой задачи предложено в статье [45], рассматривающей алгоритм сжатия расписаний (SC) для минимизации требований к процессору любого допустимого расписания. SC сохраняет длину исходного расписания и уменьшает количество процессоров за счет объединения расписаний процессоров и удаления избыточных, повторяющихся задач. В среднем SC снизила требования к процессору на 91, 82 и 72 процента для расписаний, созданных с помощью алгоритмов PLW, TCSD и CPFD, соответственно. Алгоритм SC имеет низкую сложность $O(|N|^3)$ по сравнению с большинством алгоритмов, основанных на дублировании. Отделения задачи оптимизации числа процессоров от минимизации длины расписания исследован алгоритм SDS (sub-DAG based scheduling), имеющий ту же временную сложность что и SC. Длина расписаний, сгенерированных SDS, на 3% больше, чем у алгоритма CPFD,

одного из лучших алгоритмов в данной области. Поскольку временная сложность SDS меньше, чем CPFD на $O(|N|)$, можно заключить, что сочетание SDS и SC обеспечит оптимальную комбинацию алгоритмов с точки зрения минимизации длины расписания, низкой временной сложности и низких требований к числу процессоров.

В работе [46] предложен эвристический алгоритм оптимизации на основе биогеографии (BBO) и алгоритм гибридной миграции BBO (HMBBO), который является комбинацией стратегии миграции с оптимизацией роя частиц (PSO). Методы применяются для решения задач планирования ациклических графов DAG в среде облачных вычислений. Основная идея подхода состоит в том, чтобы использовать преимущества алгоритмов PSO и BBO, избегая при этом их недостатков. В HMBBO стратегия в структуре миграции BBO гибридизована для ускорения скорости поиска, а HEFT_D используется для оценки сортировки задач. Экспериментальные результаты показывают, что HMBBO наследует сильную способность к оптимальности от BBO и имеет преимущества простой реализации, отличной производительности и быстрой скорости сходимости с различными тестами по сравнению с BBO, PSO и HEFT, что доказывает, что HMBBO можно применять для решения задач планирования в средах облачных вычислений. На основе WorkflowSim проводится сравнительный эксперимент с использованием времени планирования задач в качестве целевой функции.

Рост производительности суперкомпьютерных систем сопряжен с ростом потребления электроэнергии. Балансировка потребления энергии и производительности вычислительных ресурсов рассматривается в статье [47]. Авторами предложен новый алгоритм планирования на основе дублирования с учетом энергии – NEADS (novel energy-aware duplication-based scheduling). В процессе дублирования учитываются как FP (favorite predecessor), так и следующий FP. Если FP не соответствует условию дублирования, предшественник FP не будет проверяться. Затем оценивается второй FP. Что позволяет избежать репликации косвенных задач на один и тот же процессор.

Соответственно, дополнительные затраты на связь и вычисления могут быть уменьшены. Результаты экспериментов показывают, что для синтетических приложений NEADS эффективно снижает потребление энергии.

1.5 Мультиагентный подхода и методы машинного обучения для диспетчеризации ресурсов

Рассмотренные выше теоретические и практические аспекты применения мультиагентного подхода и методов машинного обучения в суперкомпьютерных системах показывают: повышение эффективности использования вычислительных ресурсов, сокращение энергетических затрат, оптимизация системы связи и обмена информации между узлами, сокращение длины расписаний, адаптивность систем к изменению нагрузки и программно-аппаратной частей. Использование указанных методов и подходов позволяет сокращать расходы на содержание и создание кластерных суперкомпьютерных систем. Новые математические и программные методы диспетчеризации ресурсов являются ключевыми факторами, расширяющими вычислительные возможности современных систем.

Одной из главных проблем сложных вычислений является прогнозирование соответствия временных и ресурсных ограничений, предложенных пользователем, и реальных затрат при выполнении задач на вычислительных машинах. Представленные исследования рассматривают такую задачу, но не предлагают общего решения, не зависящего от специфики платформы и/или узкоспециализированных пользовательских запросов. Дальнейшие заявленные авторами работы в этой области нацелены на увеличение точности, области применения и адаптивности к изменениям пользовательских запросов.

На текущем этапе исследования многокритериальной оптимизации планирования и распределения ресурсов учитывают не более двух критериев, что объясняется сложностью задачи. Значительное повышение эффективности выполнения задач и распределения ресурсов на текущем этапе говорит о

перспективности данного направления и необходимости расширения пула учитываемых в оптимизации критериев.

Реконфигурируемые платформы применяется в моделях IaaS, PaaS и SaaS для современных высокопроизводительных приложений. Дальнейшие исследования и разработка таких программно-аппаратных сред позволит расширить возможности коммерческого и научного-исследовательского применения суперкомпьютерных кластерных систем.

2 Анализ технологий координации взаимодействия и динамического управления вычислительными ресурсами гетерогенных суперкомпьютерных кластеров

2.1 Анализ проблем оптимизации работы суперкомпьютерной системы

2.1.1 Показатели и критерии оптимизации

Оптимизация работы суперкомпьютерной системы сводится к улучшению показателей, приведённых в таблице 2.1.

Таблица 2.1 Показатели и критерии оптимизации

Показатель	Описание	Критерий оптимизации
Использование процессора	Поддержание в режиме занятости максимально возможный период времени	Максимизация
Пропускная способность системы	(Среднее) число процессов, завершающих свое выполнение за единицу времени	Максимизация
Время обработки процесса	Время, необходимое для исполнения какого-либо процесса	Минимизация
Время ожидания	Время, которое процесс ждет в очереди процессов, готовых к выполнению	Минимизация
Время ответа	Время, требуемое от момента формирования запроса пользователя до получения ответа	Минимизация

К показателям оптимизации вычислений на суперкомпьютере можно также отнести:

– эффективность ресурсов – использование ресурсов, таких как процессоры, память и дисковое пространство, для задачи;

- распараллеливание – способность задачи или приложения использовать несколько процессоров или узлов кластера одновременно;
- скалярная производительность – количество операций, выполненных за единицу времени;
- параллелизм памяти – способность задачи или приложения использовать большое количество памяти для обработки больших данных;
- энергоэффективность – отношение мощности вычислений к затраченной энергии;
- надежность – защита от сбоев.

2.1.2 Поиск оптимального решения

Оптимизация работы суперкомпьютерной системы обычно требует нахождения баланса между различными критериями. Например, увеличение производительности может привести к большему потреблению ресурсов, и наоборот, снижение потребления ресурсов может уменьшить производительность. Идеальное решение может быть недостижимым, так как критерии оптимизации одних показателей могут противоречить критериям оптимизации других. В конечном итоге, оптимизация зависит от конкретных целей и ограничений каждой задачи и приложения, и может потребовать компромисса между различными критериями для достижения оптимального результата.

Нахождение компромисса в оптимизации вычислений на суперкомпьютере означает создание баланса между различными критериями, чтобы достичь оптимального результата. Это может включать в себя такие шаги как:

- определение ключевых критериев, которые являются наиболее важными для достижения конкретных целей и ограничений;
- оценка различных вариантов оптимизации и их соответствия ключевым критериям;

- выбор оптимального решения, которое лучше всего соответствует ключевым критериям и достигает наилучшего баланса между ними.

Поиск оптимального решения может зависеть от различных факторов:

- изменения в задаче или приложении – изменения в структуре или размере данных, или изменения в алгоритме могут потребовать изменений в компромиссе;

- изменения в аппаратной инфраструктуре – наличие новых технологий или оборудования может изменить оптимальный компромисс;

- изменения в распределении ресурсов – изменения в доступности ресурсов, таких как процессоры и память, могут потребовать изменения в компромиссе;

- изменения в ограничениях и целях.

Важно отметить, что нахождение компромисса является процессом и может потребовать некоторое количество итераций для достижения оптимального решения. Иногда баланс может быть достигнут путем снижения одного или нескольких критериев до определенного уровня, чтобы улучшить другие критерии.

2.1.3 Возможности планировщика задач Slurm

С точки зрения достижения критериев оптимизации вычислений можно отметить следующие преимущества и недостатки Slurm – планировщика задач, используемого в СКЦ «Политехнический»:

- использование процессора: Slurm позволяет распределять задачи между узлами кластера и оптимизировать использование процессора, что позволяет достигать высокой производительности вычислений. Однако при этом могут возникнуть трудности в оптимизации использования процессора в случае работы с гетерогенными узлами кластера;

- пропускная способность системы: поддерживает систему приоритезации задач, которая позволяет распределять ресурсы между различными проектами и группами пользователей, что повышает пропускную

способность системы. В то же время администраторы системы могут испытывать сложности с настройкой приоритетов задач для достижения максимальной пропускной способности системы, ввиду широкого разнообразия типов вычислений, что является помехой в поиске оптимальной схемы для всех групп пользователей одновременно;

- время обработки процесса: планировщик позволяет назначать задачи на определенные узлы в зависимости от их возможностей, что сокращает время обработки процесса. Тем не менее недостаточная информация о характеристиках приложений способна создать трудности,

- время ожидания: использует алгоритм планирования задач, который снижает время ожидания для задач пользователей, что является одним из основных достоинств Slurm. Однако высокая нагрузка на систему на систему и недостаток информации о приоритетах задач могут помешать диспетчеру,

- время ответа: предоставляет интерфейс для мониторинга состояния ресурсов и задач, который позволяет быстро идентифицировать и решать возможные проблемы, что снижает время ответа на запросы пользователей.

2.1.4 Планировщик Slurm в гетерогенных суперкомпьютерных системах

Slurm может быть использован для управления гетерогенными вычислительными кластерами, то есть кластерами, которые состоят из узлов с различными типами аппаратной конфигурации или различными операционными системами.

Планировщик позволяет различать узлы по типу и конфигурации, и назначать задачи на определенные узлы в зависимости от их возможностей. Это крайне необходимо для оптимизации использования ресурсов и повышения эффективности вычислений.

Также Slurm поддерживает систему приоритезации задач, которая позволяет распределять ресурсы между различными проектами и группами пользователей, обеспечивая справедливое распределение ресурсов.

Одновременно с этим использование планировщика в гетерогенных кластерах может привести к дополнительным сложностям. Например, может быть трудно обеспечить совместимость между различными типами узлов и операционными системами, а также может возникнуть проблема с обслуживанием и поддержкой различных типов узлов.

Следует отметить главные преимущества использования Slurm в гетерогенных кластерах:

- возможность распределять задачи между узлами кластера в зависимости от их потребностей, что позволяет максимально использовать имеющиеся ресурсы;
- возможность работы с гетерогенными ПО и библиотеками, как на уровне операционной системы, так и на уровне приложений;
- поддержание системы приоритезации задач, которая позволяет распределять ресурсы между различными проектами и группами пользователей, что повышает пропускную способность системы.

К недостаткам применения планировщика в гетерогенных кластерах можно отнести:

- сложности настройки и обслуживания системы, особенно в случае необходимости работы с различными версиями ПО и библиотек;
- трудности в оптимизации использования процессора и памяти в случае работы с гетерогенными узлами кластера;
- необходимость иметь детальное понимание характеристик приложений и ресурсов для оптимальной настройки приоритетов задач и распределения ресурсов.

В целом использование Slurm для управления гетерогенными вычислительными кластерами предоставляет мощные возможности для

оптимизации ресурсов, но может так же столкнуться с некоторыми трудностями в настройке и обслуживании системы.

2.1.5 Проблема несоответствия между запрошенными и выделенными ресурсами

Планировщик Slurm – это компонента операционной системы, которая имеет демонов или агентов в узлах кластера, и на основе априорных данных, указанных в заявке пользователя на размещение задачи, и данных о текущей загрузке вычислительных узлов, выделяет вычислительные ресурсы на то время, которое было указано в заявке. Трудности возникают, если время, запрашиваемое пользователем для задачи, не совпадает с временем, которое фактически было потрачено на выполнение задачи. Это может привести к неэффективному использованию ресурсов и неравномерному энергопотреблению, задержкам в выполнении других задач и перегрузке системы:

- если задача завершается раньше, чем предполагалось, то ресурсы, выделенные для нее, остаются неиспользованными, что приводит к неэффективной работе системы;
- если задача занимает больше ресурсов, чем предполагалось, то это может потребовать переназначения ресурсов для других задач в системе;
- если задача завершается позже, чем предполагалось, то это может привести к задержкам в выполнении других задач, которые зависят от ее результатов.

Ошибочное определение пользователями времени, необходимого для выполнения задач, может быть вызвано различными причинами:

- недостаточный опыт – пользователи могут не иметь достаточного опыта или знаний о приложении или вычислительной системе, что может приводить к недооценке или переоценке времени, необходимого для выполнения задачи;

- недостаточная информация – пользователи могут не иметь достаточной информации о требуемых ресурсах или ожидаемых результатах;
- ошибки в данных или модели – пользователи могут использовать некорректные данные или модели для определения времени, необходимого для выполнения задачи.

Чтобы избежать этих проблем, на первый взгляд необходимо, чтобы пользователи точнее указывали время, необходимое для выполнения задачи, и обновляли информацию о них, если она меняется. Администраторы кластера могут использовать статистические методы для определения наиболее точных прогнозов времени, необходимого для выполнения задач. Однако достаточно сложно обучить пользователей более точной оценке необходимого времени, поскольку это требует решения большого количества сопутствующих вопросов, связанных с обозначенными выше причинами, а статистические методы необходимо будет адаптировать под меняющиеся данные. Таким образом, эффективнее использовать методы машинного обучения для решения проблемы несоответствия запрошенного пользователем времени и времени, которое выполнялась задача, поскольку эти методы предназначены для работы с меняющимися данными.

2.1.6 Применение методов машинного обучения

Внедрение методов машинного обучения в Slurm может способствовать более эффективному использованию ресурсов и улучшению производительности вычислений. В то же время такой подход требует дополнительной разработки и настройки, а также требуется иметь опыт в работе с методами машинного обучения.

Внедрение методов машинного обучения в систему управления ресурсами Slurm может принести следующие выгоды:

- повышение точности определения времени, необходимого для выполнения задач: использование методов машинного обучения для анализа исторических данных о задачах способно помочь системе управления

ресурсами более точно предсказывать время выполнения новых задач, что позволит эффективнее распределять ресурсы;

- повышение эффективности управления ресурсами: применение машинного обучения для анализа динамики использования ресурсов поможет системе управления ресурсами более эффективно реагировать на изменения нагрузки, адаптируя распределение ресурсов в реальном времени;

- автоматизация рутинных задач, таких, как планирование и оптимизация распределения ресурсов, а также снижение нагрузки на администраторов;

- предоставление более эффективного и адаптивного сервиса для пользователей, увеличивающего скорость выполнения задач и снижающего задержки.

В целом внедрение методов машинного обучения в систему управления ресурсами Slurm позволит значительно улучшить эффективность использования суперкомпьютера и предоставить более высококачественный сервис для пользователей. Однако следует отметить, что потребуются дополнительные ресурсы и время для разработки и настройки моделей, а также может потребоваться дополнительное обучение для администраторов и разработчиков.

Внедрение методов машинного обучения в систему управления ресурсами Slurm возможно осуществить с помощью нескольких подходов:

- оптимизация планирования задач: модели машинного обучения могут быть использованы для прогнозирования нагрузки на ресурсы и оптимизации распределения задач между узлами,

- мониторинг и диагностика ресурсов: анализ данных с кластера и предсказание потенциальных проблем с ресурсами,

- адаптивное управление ресурсами: адаптивная реконфигурация ресурсов в зависимости от текущей нагрузки и потребностей приложений.

2.2 Описание суперкомпьютерной системы

2.2.1 Описание СКЦ «Политехнический»

СКЦ «Политехнический» располагает следующими вычислительными ресурсами:

- 612 узлов кластера "Политехник - РСК Торнадо" (далее узлы tornado):
 - 2 x Intel Xeon CPU E5-2697 v3 @ 2.60GHz
 - 64G RAM
- 56 узлов кластера "Политехник - РСК Торнадо" с ускорителями NVIDIA K-40 (далее узлы tornado-k40):
 - 2 x Intel Xeon CPU E5-2697 v3 @ 2.60GHz ○ 64G RAM
 - 2 x Nvidia Tesla K40x 12G GDDR
- 81 узел кластера "Политехник - РСК Cascade":
 - Intel Xeon Platinum 8268/Xeon Gold 6248R
 - 192G RAM

Все доступные узлы используют сеть Infiniband FDR 56Gbps в качестве интерконнекта.

Также, на всех узлах доступна параллельная файловая система Lustre объёмом около 1 ПБ.

Зарегистрированные пользователи получают доступ к вычислительным ресурсам с машины `login1.hpc.spbstu.ru`. Вход осуществляется с использованием протокола SSH (терминального клиента).

Управление ресурсами кластера осуществляется с помощью программного пакета Slurm. Принцип его работы можно описать следующим образом: пользователь запрашивает некоторый ресурс (процессорные ядра, память и т.п.), размещая свою задачу в очереди; система, основываясь на приоритетах пользователя и текущем заполнении очереди, выбирает момент запуска задачи. Под очередью понимается последовательность задач, которая должна решаться на определенном вычислительном ресурсе (группе узлов).

На данный момент доступны три очереди для узлов: `tornado`, `tornado-k40`, `cascade`. При этом, каждый узел в текущий момент времени может быть занят только одной задачей, одного пользователя. Таким образом узел отводится в монопольное использование размещенной на нем задачи, и другие задачи на занятом узле выполняться не будут. У большинства узлов, которые входят в очереди `tornado` и `tornado-k40` по 28 физических процессоров, с учетом многопоточности – 56 логических. Узлы из очереди `cascade` имеют по 48 физических или 96 логических процессоров.

2.2.2 Описание планировщика задач Slurm

Slurm – это система планирования задач и управления суперкомпьютерным кластером. Система выполняет три основные функции:

- распределяет доступ к ресурсам – вычислительным узлам – для пользователей;
- предоставляет среду для запуска, выполнения и мониторинга задач на выделенных узлах;
- управляет очередью задач.

Архитектура Slurm состоит из нескольких служб/сервисов или агентов, называемых демонами (`daemons`) в Linux-системах (рисунок 2.1). Пользователи взаимодействуют с Slurm посредством набора команд через командную строку.

Центральный демон управления `slurmctld` организывает действия Slurm:

- постановку задач в очередь;
- мониторинг состояний узлов;
- распределение узлов задачам.

Агенты или демоны `slurmd` исполняются на каждом узле, иницируют задачи пользователей и управляют ими.

Процесс `slurmdbd` предоставляет интерфейс к базам данных и архивирует записи заданий в БД.

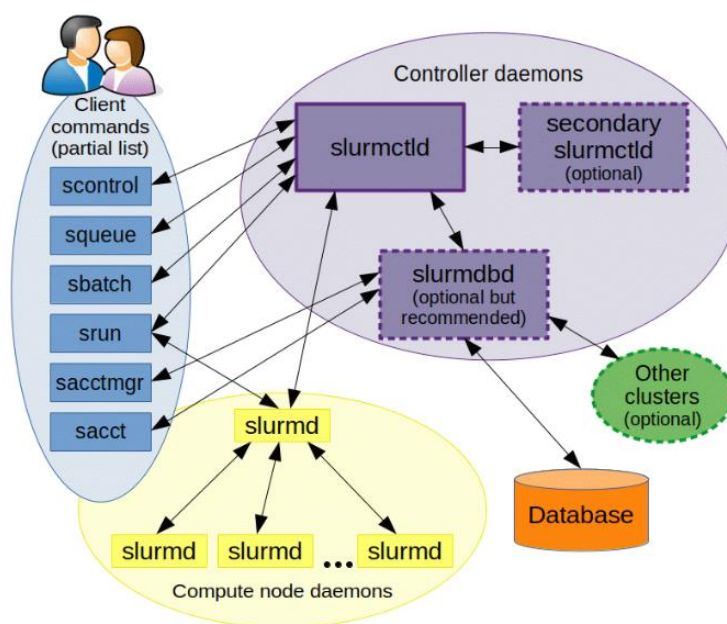


Рисунок 2.1 Архитектура Slurm

Пользователь может взаимодействовать с планировщиком посредством следующих основных команд:

- sinfo – просмотр информации об очередях и вычислительных узлах;
- salloc – запрос на выделение ресурсов для интерактивной работы;
- srun – запуск параллельных задач или шагов задач;
- sbatch – запуск задач в пакетном режиме;
- scancel – отмена заданий;
- scontrol – просмотр и изменение конфигурации планировщика (административная команда);
- sacct – просмотр данных системы использования ресурсов о завершённых или активных задачах;
- squeue – просмотр состояния очередей.

Вычислительные узлы группируются в партии (partitions) или очереди – логические объединения вычислительных ресурсов, в которые Slurm распределяет задачи пользователей (пример на рисунке 2.2). Задача

пользователя может состоять из одного или нескольких шагов (job step). В одной задаче может быть несколько подзадач.

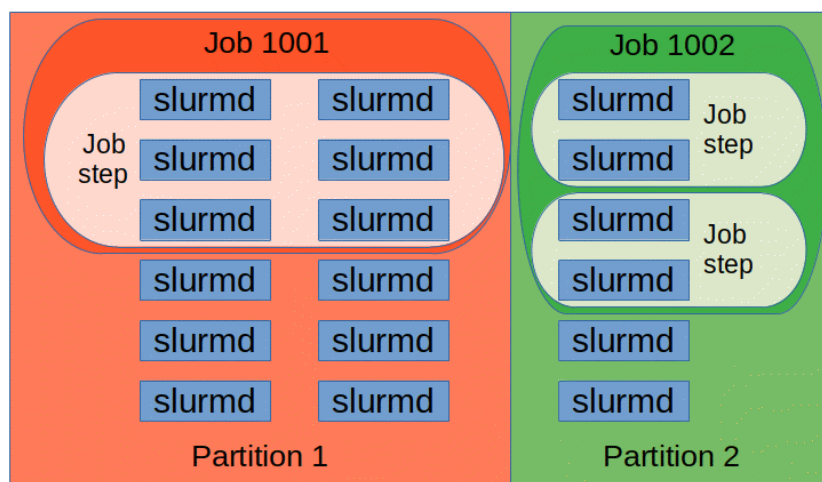


Рисунок 2.2 Партиции вычислительных узлов

Пользователь может размещать задачи в двух режимах посредством командной строки – интерактивном и пакетном.

В интерактивном режиме пользователь может действовать по двум алгоритмам. В первом случае запрашиваются ресурсы, запускается необходимое для работы приложение на головном узле СКЦ, и далее пользователь работает в приложении. Во втором случае выполняется подключение к выбранному узлу, и остальные действия выполняются на этом узле.

В пакетном режиме пользователь готовит скрипт задания в командном окне: определяет параметры запуска задачи, настраивает рабочее окружение необходимое для работы приложения, определяет последовательность запуска задачи. После чего задача помещается в соответствующую очередь.

Основными запрашиваемыми пользователем для выполнения задачи ресурсами и их параметрами являются вычислительные узлы (количество) и время их использования.

Дополнительно могут быть указаны количество процессоров на подзадачу, количество памяти в узле, количество памяти в процессоре, число подзадач в задаче, время запуска задачи и другие. Пользователь имеет

возможность выбора множества опций для подробного описания, как должна выполняться задача.

Задачи могут находиться в следующих состояниях:

- CANCELLED – отменена пользователем или администратором;
- COMPLETED – завершена успешно;
- FAILED – завершена неуспешно;
- NODE_FAIL – прекращена из-за сбоя одного или нескольких узлов;
- PENDING – ожидает ресурсов для выполнения;
- RUNNING – выполняется;
- TIMEOUT – прекращена из-за превышения временного лимита, заданного пользователем.

К основным достоинствам Slurm можно отнести открытый исходный код, распределённую архитектуру и поддержку множества видов задач:

- является открытым программным обеспечением, что позволяет легко модифицировать и настраивать его для конкретных потребностей;
- способен работать на большом количестве вычислительных узлов, позволяя распределять задачи и управлять ресурсами для достижения максимальной эффективности вычислений;
- поддерживает множество различных видов задач, включая пакетные задачи, задачи, вычисляемые на одном и на нескольких узлах.

Недостатками Slurm являются относительная сложность настройки, отсутствие интеграции с другими инструментами, ограничения в масштабировании и отсутствие поддержки облачных ресурсов:

- может быть сложной для настройки и поддержки, особенно для новых пользователей, которые не имеют опыта работы с системами управления ресурсами;
- не имеет встроенной интеграции с другими инструментами для анализа и визуализации данных, которые могут быть необходимы для некоторых приложений;

- в некоторых случаях Slurm имеет ограничения в масштабировании и распределении задач при работе с очень большими кластерами;
- не имеет встроенной поддержки для использования облачных ресурсов и не может напрямую использовать ресурсы, предоставленные в облаке и управлять ими.

2.2.3 Описание жизненного цикла задачи

Жизненный цикл задачи, создаваемой пользователем в СКЦ, состоит из четырёх этапов (рисунок 2.3):

- подключения пользователя к консоли управляющего узла СКЦ через клиент (например, PuTTY, для ОС Windows, или встроенный SSH-клиент для ОС Linux);
- создания и запуска задачи в консоли в интерактивном или пакетном режиме;
- выполнение задачи;
- завершение задачи.

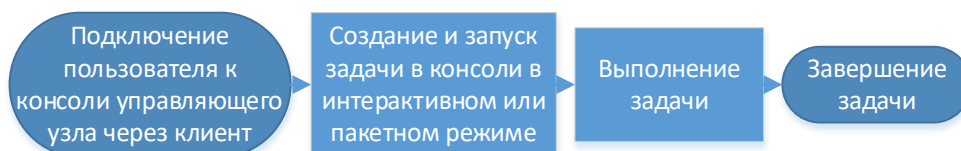


Рисунок 2.3 Этапы жизненного цикла задачи

На **первом этапе** пользователь осуществляет удалённое подключение к консоли управляющего узла СКЦ посредством клиента, установленного на компьютере пользователя, и вводит адрес управляющего узла (сервера). Доступ осуществляется по SSH-ключу и данным пользователя. Далее пользователь работает в командной строке.

На **втором этапе** пользователь готовит свою задачу, выбирая способ её постановки – интерактивный или пакетный режим (рис. 2.4).

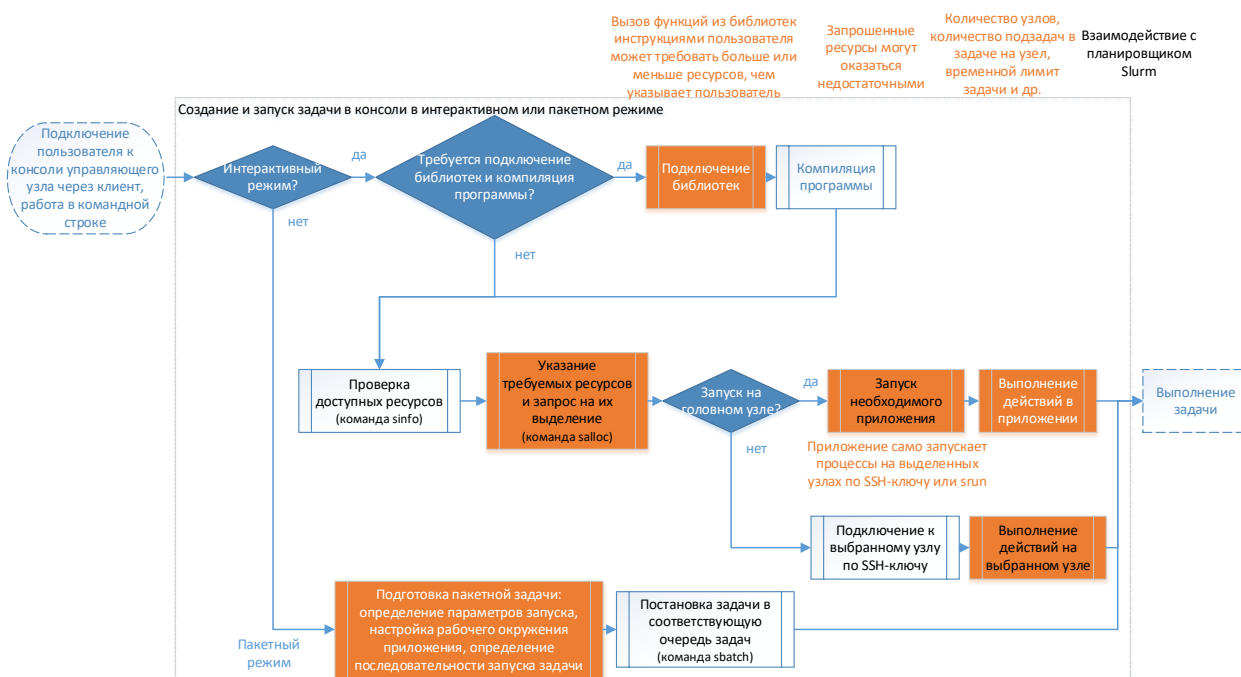


Рисунок 2.4 Создание и запуск задачи

В *пакетном* режиме пользователь выполняет подготовку пакетной задачи и создаёт исполняемый сценарий задачи, в котором указывает:

- параметры запуска задачи (время выполнения, количество узлов, количество подзадач на узел и др.);
- параметры рабочего окружения приложения;
- последовательность запуска задачи.

Затем сценарий передается утилите `sbatch` в качестве параметра. Сценарий попадает в одну из очередей задач, сформированных планировщиком Slurm. Он будет запущен на первом из выделенных вычислительных узлов.

В *интерактивном* режиме пользователь:

- если необходимо: подключает библиотеки и выполняет компиляцию программы;
- проверяет доступные в системе ресурсы с помощью команды `sinfo`;
- указывает требуемые ресурсы и запрашивает их с помощью утилиты `salloc`;

– выполняет запуск приложения на головном узле, которое само запускает процессы на выделенных узлах, и выполняет действия в приложении;

или

– подключается к выбранному узлу по SSH-ключу и выполняет действия на выбранном узле.

На рисунке 2.4 оранжевым цветом отмечены операции, которые оказывают влияние на время выполнения задачи, а чёрным шрифтом – те операции, в которых задействован планировщик Slurm. Функции библиотек, вызываемых пользователем, могут не совпадать с теми ресурсами, которые изначально запросил пользователь.

На **третьем этапе** осуществляется выполнение задачи.

В ходе выполнения могут возникнуть следующие ситуации, описанные на рисунке 2.5:

- отмена задачи;
- истечение временного лимита, заданного пользователем (времени выполнения);
- завершение выполнения программы пользователя;
- появление событий, вызывающих приостановку (таких, как, например, сбой в работе вычислительных узлов и перераспределение ресурсов) и последующее возобновление выполнения задачи.



Рисунок 2.5 Выполнение задачи

На **последнем этапе** задача завершается. Результат выполнения будет сохранён в виде файла в директории, из которой производился запуск задачи. При этом результат зависит от того, каким образом была запущена задача – в пакетном или интерактивном режиме. Если задача завершена успешно, то возвращается нулевой код выхода (ExitCode) и сигнал о завершении, если неуспешно, то возвращается ненулевой код выхода с причиной «NonZeroExitCode» и сигнал о завершении (рис. 2.6).

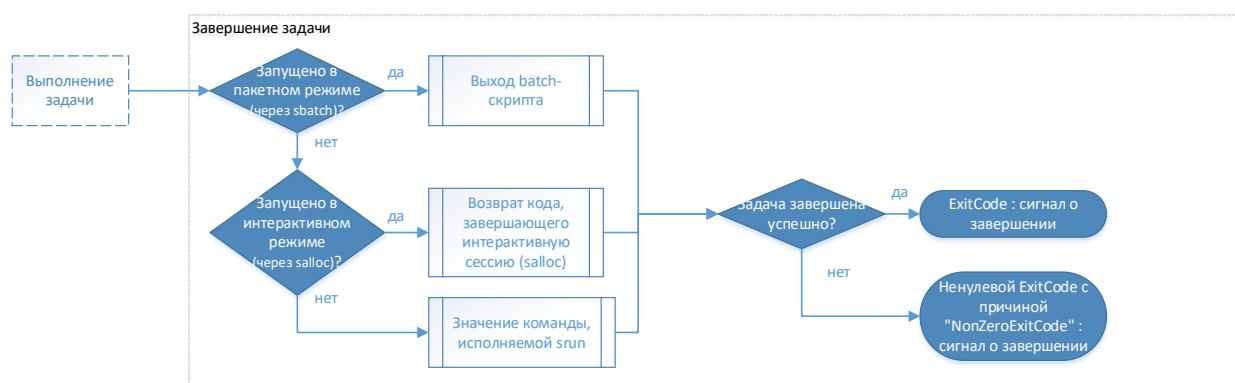


Рисунок 2.6 Завершение задачи

2.3 Обзор исследований в области планирования задач и диспетчеризации суперкомпьютерных ресурсов

Как и в начале любого исследования, целесообразно было посмотреть на работы, сделанные другими исследователями по теме планирования задач в суперкомпьютерных системах, и узнать, как они подходили к решению похожих исследовательских задач, какие методы использовали, какие результаты получили и как оценивали результаты применения выбранных методов на своих наборах данных.

Спектр задач, решаемых исследователями в рамках обозначенной темы, оказался широк: предсказание времени выполнения задания, анализ данных в суперкомпьютерных системах (характеристик задач, пользователей, загрузки системы и др.), мультиагентные подходы к планированию заданий, а также подходы на основе обучения с подкреплением (reinforcement learning).

В своих публикациях исследователи рассуждают об эффективности используемых ими методов. Для оценки эффективности каждая группа выбирала такую метрику, которая была наиболее важна для каждой поставленной исследовательской задачи и наиболее полно отражала действие выбранного метода на систему.

2.3.1 Работы, посвящённые предсказанию времени выполнения

В [48] исследователи решали задачу планирования очередей заданий на основе предсказания времени их выполнения с помощью обучения с подкреплением (reinforcement learning). Исследования проводились в суперкомпьютерном центре Университета Науки и Технологии Китая (University of Science and Technology of China) на 28-х ядерных вычислительных процессорах Intel(R) Xeon(R) Gold 5118 CPU @ 2.30 GHz и четырёх графических процессорах GeForce GTX-1080 Ti 12 GB. Были взяты задания пользователей одного прикладного программного пакета для квантовых вычислений VASP. Таким образом, в качестве атрибутов выступили специфические атрибуты заданий пакета VASP. Предсказательная модель – RLSchert – состояла из рекуррентной нейронной сети, декодирующей временные ряды атрибутов, и двухслойной полносвязной нейронной сети с 64-ю нейронами в каждом слое для предсказания времени выполнения. Модель использовалась планировщиком заданий, и эффективность её работы сравнивалась с эвристическими алгоритмами планирования:

- случайным, когда планировщик выбирает задачу из очереди случайным образом и выделяет ресурсы для её выполнения;
- сначала короткие задачи;
- сначала те задачи, которые требуют меньше ресурсов;
- сначала те задачи, чьи требуемые ресурсы наименьшим образом отличаются от доступных в системе;
- сначала те задачи из числа коротких, чьи требуемые ресурсы наименьшим образом отличаются от доступных в системе.

Для оценки качества модели использовались метрики mean absolute percentage error (MAPE) – средняя абсолютная ошибка в процентах, средняя задержка и среднее время завершения. По результатам исследования MAPE предсказания времени выполнения предложенной моделью составила 10% против 40% алгоритма XGBoost, использованного для сравнения. Время предсказывалось после получения входных атрибутов и до запуска задачи. RLSchert показала наименьшие значения среди всех рассмотренных эвристических алгоритмов планирования по средней задержке и среднему времени завершения задачи. Чем больше нагрузка на кластер, тем выше оказалась эффективность работы, предложенной исследователями модели.

В этой работе исследователи построили модель на основе специфических атрибутов пакета для квантовых вычислений VASP, что делает её применение ограниченным. В публикации упоминается, что для построения независимой от каких-либо программных пакетов модели необходимо оперировать атрибутами пользователей, описывающими их поведенческие особенности (ID пользователя, предыдущие завершённые задания пользователя, ID очереди, ID группы и т.д.).

Исследовательская задача предсказания времени выполнения задачи пользователя решалась в [49] методами иерархической классификации. Обеспечена более высокая точность предсказания, чем оценочное время, указываемое пользователями. Модель построена на таких атрибутах, как требования к ресурсам и приоритет задачи.

Проблема ошибки пользователя при оценивании времени выполнения в большую (переоценка) или меньшую сторону (недооценка) рассмотрена в [50]. Улучшение точности предсказания времени выполнения задачи достигнуто за счёт тобит модели. Предложенное решение может использоваться в качестве плагина, подключаемого к планировщику. Исследователи работали с двумя суперкомпьютерными системами Argonne Leadership Computing Facility:

- Mira, IBM Blue Gene/Q, 786,432 процессоров, 768 Тбайт памяти;
- Intrepid, IBM Blue Gene/P, 40,960 4-х ядерных узлов, 163,849 ядер.

Данные были взяты за периоды с января по декабрь 2014 для Mira и с января по сентябрь 2009 для Intrepid. В список рассмотренных атрибутов вошли:

- атрибуты постановки задачи: номер задачи, имя пользователя, имя проекта, размер задачи, оценочное время выполнения и т.д.;
- время выполнения последней задачи;
- время выполнения предпоследней задачи;
- число узлов, запрошенных пользователем;
- средняя точность предыдущих задач;
- максимальная точность предыдущих задач;
- максимальная длительность задачи среди предыдущих задач;
- максимальная длительность задачи последних 10-ти задач;
- среднее время выполнения среди предыдущих задач;
- среднее время выполнения последних 10-ти задач;
- время выполнения 25-го перцентиля среди предыдущих задач.

Алгоритм извлекал атрибуты задачи, которую отправил пользователь (имя пользователя, проект, имя задачи). Затем сравнивал извлечённые атрибуты с атрибутами, хранящимися в репозитории задач. Если количество хранящихся задач было меньше порога, то оценочное время выполнения, заданное пользователем, отправлялось в планировщик. В противном случае, алгоритм проверял, удовлетворяло ли оценочное время средней точности времени выполнения предыдущих задач. Если удовлетворяло, то оценочное время отправлялось в планировщик. Если же нет, то тобит модель выполняла переоценку времени, введённого пользователем, и новое значение поступало в планировщик.

Решение сравнивалось с такими моделями машинного обучения, как метод опорных векторов, случайный лес и Last-2.

В качестве метрик оценки эффективности решения рассмотрены степень недооценки времени выполнения, средняя точность, среднее время ожидания задачи, средняя задержка и загрузка системы. Предлагаемая модель снизила

степень недооценки времени выполнения на 5-8% в то время, как Last-2, метод опорных векторов и случайный лес увеличили степень недооценки на 47%, 29% и 31% соответственно. Все методы показали значительно более высокую среднюю точность, чем оценка времени выполнения задачи, задаваемая пользователем, 75%-80% против 55%-58%. Предложенное решение сбалансировало загрузку системы до 20%. Другие методы оказали незначительное или негативное влияние по этой метрике.

2.3.2 Работы, посвящённые исследованию данных

Анализ загрузки, поведенческих атрибутов пользователей, временных атрибутов задач и зависимостей между ними был рассмотрен в [51] авторами публикации [48]. Были собраны записи двух суперкомпьютерных систем TC4600 и SJTUл с января 2018 по декабрь 2019 и с января 2019 по декабрь 2019 соответственно. Построенные распределения загрузки ресурсов – количества ядер, времени выполнения задачи, количества ядерных часов и времени ожидания задачи в очереди – показали, что большинство задач

- (97%) расходовали менее 64-х ядер, больше половины – менее 4-х;
- выполнялись не более 1 часа;
- потребляли менее 10 ядерных часов;
- ожидали в очереди менее 1 часа.

Исследователями было введено определение портфеля задач (Bag-of-Task) – набора независимых задач, отправляемых одним и тем же пользователем с интервалом менее 60 с. Были исследованы время выполнения задачи в портфеле задач, время ожидания задания, время выполнения всего портфеля задач, время между постановками задач одного и того же пользователя, время между постановкой новой задачи и окончанием предыдущей от одного и того же пользователя.

Анализ времени между задачами показал, что большинство задач было поставлено в пределах 10000 с после завершения предыдущего. Когда промежутки между задачами пользователя составляли менее 10000 с,

наблюдалась зависимость времени выполнения задач от этих промежутков: чем меньше промежутки, тем меньше время выполнения. Большинство портфелей задач пользователей не превышало 10 задач. При этом, когда размер портфеля задач увеличивался, время выполнения задач в нём уменьшалось. Исследователи также выяснили, что более длительные задачи дольше ожидают в очереди.

Таким образом, согласно полученным результатам, задача будет выполняться быстрее, если 1) между постановками задач от одного пользователя пройдёт меньше времени, 2) в портфель задач поместить больше задач, 3) задача будет короткой.

Возможно, на выбранные исследователями данные оказывала влияние политика планирования, установленная на суперкомпьютерных системах.

Анализ атрибутов задач и потребления ресурсов был проведён в [52] авторами исследования [50]. Данные суперкомпьютерных систем Intrepid и Mira были взяты за периоды с 2009 по 2013 и с 2014 по 2018 соответственно. Рассмотрены атрибуты: ID пользователя, ID проекта, ID задачи, очередь, время постановки в очередь, время начала и окончания, оценочное время выполнения, фактическое время выполнения, число узлов, число ядер и т.д. Обе системы использовались программными инструментами из самых разных научных областей: материаловедение (QMCPACK и CPMD), биология (NAMD), транспортная инфраструктура, энергетика (MADNESS и MPQC), ядерное горение (FLASH and GFMC), космология (HACC), квантовая физика (MILC, Chroma и CPS) и физика плазмы (GTC и GTS), химия и климатология (HIRAM and CM4+).

Изучение загрузки системы показало, что использование ресурсов значительно возросло. Задачи стали больше и длиннее, и медианное число ядерных часов на задачу увеличилось. Длительные и среднего размера задачи потребляли большинство ядерных часов. Время ожидания задачи в очереди стало больше. Пользователи продолжали переоценивать время выполнения и запрашивали больше расчётного времени.

Анализ поведения пользователей позволил отметить следующие наблюдения:

- пользователи ставили задачи с очень похожими характеристиками, требующими похожие ресурсы. Их задачи могли быть разделены на группы, где задачи в каждой группе запускались на таком же количестве узлов и выполнялись схожее количество времени. Однако время ожидания для задач от пользователей с похожими ресурсами могло быть различным;

- корреляция между потребляемыми ресурсами и пользовательскими характеристиками задачи оказалась слабой, в особенности для активных потребителей ресурсов (тех, кто потреблял больше остальных). Некоторые пользователи могли сталкиваться с «непреднамеренной несправедливостью» – большим временем ожидания задачи в очереди;

- пользовательские характеристики (идентификаторы пользователя, проекта, очереди) могут быть использованы для оценки расчётного времени выполнения и необходимого количества ядер до постановки задачи в очередь с медианным значением ошибки оценки менее 15 %. Таким образом, по принадлежности заявки к какой-либо группе можно предсказать время выполнения задачи по среднему значению для этой группы.

Следует отметить, что в проекте могут быть разные задачи, как короткие, так и длинные, проект может закрыться, открыться новый, и среднее время выполнения задачи в проекте может оказаться ненадёжным показателем.

Подробный анализ данных суперкомпьютерных систем необходим для выбора и построения модели предсказания времени выполнения задачи, адекватной оценки и интерпретации результатов её работы.

2.3.3 Работы, посвящённые планированию на основе мультиагентного подхода

Мультиагентный подход к распределению задач рассмотрен в [53]. В описываемой системе планировщик отсутствует (децентрализованная система). Агентами являются узлы. Каждый узел обновляет свой вектор состояния, сканирует очередь, общается запросами с другими узлами, чтобы сформировать кластер для выполнения задачи. Оптимизируя время сбора кластера, исследователи проводят аналогию с работой иммунной системы человека.

Эксперименты проводились на инструментах MASON – мультиагентном симуляторе. Была задана конфигурация в 100 узлов, 1000 задач, выбранных случайным образом по нормальному закону распределения, количество процессоров от 1 до 5. Подход сравнивался с эвристикой first-in first-out (FIFO). В качестве метрик сравнения были выбраны время, требуемое для завершения всех задач в очереди, и среднее время ожидания задач в очереди. Предложенный подход, названный dRAP, уменьшил упомянутые показатели на 20 и 25% соответственно по сравнению с FIFO. Эффективность использования ресурсов кластера составила 100% против 56%. dRAP использовал 90-95% узлов для большинства симуляций в то время, как FIFO использовал 70-75%.

В [54] мультиагентный подход к планированию параллельных задач сравнивался с адаптивным факторингом (adaptive factoring), основанном на вероятностном анализе. Исследования проводились на специально собранном кластере Института Компьютерных Наук Университета Филиппин (the Institute of Computer Science, University of the Philippines Los Banos) с 64-ю узлами, 64-ю процессорами и памятью 32 Гб, процессорами Pentium III, Pentium IV и AMD Athlon с частотой от 800 МГц до 2.4 МГц, взаимодействующими через Ethernet переключатели 10 Мбит/с и Ethernet-соединение 100 Мбит/с.

Wave Packet Dynamics (метод квантовой траектории) использовался для симуляции задач (волновой пакет из 501 псевдочастицы был симулирован за 10000 временных отсчётов). Исследователи рассматривали количество процессоров и время выполнения в качестве атрибутов задач. Процессоры выступили агентами, в остальном подход схож с [53].

Параллельные затраты (с ростом числа процессоров) были выбраны в качестве метрики оценки эффективности подхода. Адаптивный факторинг показал меньшие затраты, чем мультиагентный подход, до 32-х процессоров. При количестве процессоров больше 32-х мультиагентный подход работал лучше.

2.3.4 Работы, посвящённые планированию с применением обучения с подкреплением (Reinforcement Learning)

Эффективность использования различных алгоритмов, основанных на обучении с подкреплением, активно исследуется в задачах планирования.

В [55, 56] предложено обучение агентов планирования с помощью алгоритмов q-обучения (q-learning) и policy gradient, которые бы превзошли традиционные политики планирования (эвристики).

Планирование связанных подзадач, составляющих задачу, и представленных в виде направленного ациклического графа (DAG) рассмотрено в [57]. Решение построено на алгоритмах q-обучения и State-Action-Reward-State-Action (SARSA). Задача собрана из 100 подзадач, запущенных многократно на гетерогенном кластере из 4 узлов.

Модель описана тремя уровнями сложности. На первом уровне учитывалась связь между подзадачами и вычислительная мощность узлов, чтобы агент поставил больше подзадач в очередь более производительного узла, но при этом учёл и связи между подзадачами. При этом предполагалось, что все подзадачи имели одинаковую структуру, и за один раз запускалась только одна подзадача. Второй уровень дополнительно рассматривал

состояние кластера (загрузку узлов). Таким образом, запускалась уже не одна подзадача. На третьем уровне добавлялись атрибуты внутренней структуры подзадач, то есть вводились различия между подзадачами.

Исследователи отметили, что обучение с подкреплением даёт хороший выигрыш (суммарное вознаграждение) только на большом количестве итераций. Предложенная модель комбинирует характеристики всех узлов кластера, получая в результате огромное пространство состояний, которое не может быть должным образом применено агентом. Чем больше количество узлов в распределённой системе, тем сложнее агенту выбрать оптимальную политику планирования. Рекомендуется использовать подобные алгоритмы совместно с другими эвристиками и стратегиями для достижения хорошей эффективности.

Схожую задачу решали исследователи в [58], минимизируя среднее время прохождения всей цепочки подзадач внутри задачи. Был сконструирован экспериментальный кластер, представленный двумя типами ресурсов (процессор и память), который содержал 48 слотов ресурсов каждого типа.

Конфигурация загрузки была представлена тремя одинаковыми по структуре и тремя различными задачами в двух экспериментах. Каждая из задач состояла из 100 подзадач. Требования к ресурсам и время выполнения генерировались случайным образом для каждой подзадачи в задаче. Для обучения и тестирования нейронных сетей были сгенерированы по 300 задач.

На каждой итерации одна задача выполнялась единожды. Агент наблюдал состояние системы в дискретные моменты времени (состояние кластера и профили подзадач) и принимал решение о планировании через политику нейронной сети. Конфигурация сети состояла из входного слоя, двух полносвязных скрытых слоёв с нелинейной функцией активации ReLU6 и функции softmax на выходном слое.

Предложенный алгоритм, названный GoDAG, сравнивался с тремя эвристиками – 1) сначала те подзадачи, чьи требуемые ресурсы наименьшим

образом отличаются от доступных в системе, 2) сначала короткие подзадачи, 3) сначала критичные подзадачи. Среднее время прохождения всей цепочки подзадач для задач одинаковой структуры составило 99,35, 108,15 и 99,15 – от одной до нескольких единиц меньше, чем у эвристик; для задач различной структуры – 102,53, что на несколько единиц меньше, чем у эвристик. GoDAG превзошёл эвристики после 300 итераций и установился после 1000.

Объединение алгоритмов графовых свёрточных сетей (Graph Convolutional Networks, GCN) и Actor-Critic (A2C) использовано в [59] для решения задачи динамического планирования вычислений, представленных в виде направленного ациклического графа.

В [60] авторы исследовали механизм внимания (attention mechanism) для планирования задач с учётом их взаимозависимости и гетерогенности узлов. Механизм внимания обучался захватывать меняющиеся со временем независимые атрибуты задач и узлов и использовал их для уточнения состояния кластера.

Был создан симулятор мульти-ресурсного гетерогенного кластера, который взаимодействовал с планировщиком задач. Случайным образом сгенерировано 1000 последовательностей задач. Последовательность соответствовала временному ряду, в котором степень прибытия задач соответствовала закону распределения Пуассона. В качестве атрибутов задач и узлов выступили категориальные признаки: требования к ресурсам, время завершения задачи, дедлайн – для задач, доступность ресурсов, доступная вычислительная мощность – для узлов (различные типы ресурсов – процессор, память, ввод/вывод, графический процессор, сеть).

Таким образом, конструкция системы состояла из кластера, планировщика и транслятора внимания. Обучение с подкреплением характеризовалось тремя аспектами – состоянием (state), действием (action) и вознаграждением (reward). Каждая задача пребывала в кластер в интерактивном или в пакетном режиме (batch). Поступающие данные трансформировались в масштабируемое состояние системы через механизм

внимания. Алгоритм агрегировал набор локальных векторов в один вектор, представляющий состояния всех задач или узлов. Понижение сложности достигалось за счёт преобразования комбинаторного решения о независимых задачах и узлах к взвешенной паре действия в структуре обучения с подкреплением, что позволяло моделям обучаться политикам планирования онлайн на всей длине меняющегося массива состояний. Модели обучались по методу *policy gradient*.

Предложенный подход, названный SCARL, сравнивался с тремя эвристическими методами – сначала короткие задачи, сначала задачи с коротким дедлайном, сначала задачи с наименьшим временем ожидания, а также с модифицированным вариантом самой рассматриваемой модели – RL-E SCARL, отличие которой состояло в исключении механизма внимания из процедуры агрегации состояния кластера. В качестве метрик оценки эффективности методов выступили задержка и время завершения задачи. При высокой гетерогенности SCARL превзошёл по задержке остальные методы на 10.7 % (разница между SCARL и эвристикой, в которой короткие задачи выполняются первыми). С увеличением количества поступающих задач, SCARL также показал лучший результат по задержке с превосходством в 9.2 %. Предложенная модель оказалась устойчивой к различным видам пакетных заданий (батчей). Однако при обработке дедлайнов задач SCARL, RL-E SCARL и эвристики (кроме – короткие задания первыми) показали схожие результаты.

2.4.6 Результаты обзора литературных источников

Проведённый анализ публикаций позволил отметить несколько важных для данного исследования аспектов:

- наибольший интерес представляют [50-52], поскольку в них собраны данные из реальных суперкомпьютерных систем, подробным образом описаны атрибуты задач и их предполагаемое влияние на время выполнения задачи. Атрибуты постановки задачи (номер задачи, имя пользователя, имя

проекта, размер задачи, оценочное время выполнения и т.д.), среднее время выполнения предыдущих задач и средняя точность для времени выполнения предыдущих задач были использованы в [50] для построения тобит модели (регрессинной модели) и определения степени недооценки пользователем времени выполнения задачи. В [51] были обозначены время выполнения задачи (чем короче, тем быстрее она выполняется), время между постановками задач от одного пользователя (чем меньше, тем быстрее выполняются задачи пользователя), количество подзадач в задаче (чем больше подзадач, тем быстрее выполняется задача), время ожидания задачи в очереди. Здесь следует отметить, что такие выводы могли быть обусловлены политикой планирования, установленной в суперкомпьютерной системе, о которой не говорится в исследовании. В [52] авторы рекомендовали использовать идентификаторы пользователя, проекта и очереди для оценки расчётного времени выполнения, и необходимого количества ядер до постановки задачи в очередь, поскольку получили медианное значение ошибки оценки менее 15 %, обнаружив, что пользователи часто ставили задачи с очень похожими характеристиками, требующими похожие ресурсы (время выполнения, количество узлов/ процессоров/ ядер);

- в большинстве работ авторы проводили эксперименты на симуляторах кластеров, оставляя вопрос тестирования на реальных кластерах открытым;

- предложенные подходы сравнивались преимущественно с эвристическими методами планирования (например, сначала короткие задачи / сначала задачи с коротким дедлайном / сначала те задачи, чьи требуемые ресурсы ближе к доступным в системе и др.). В [50] тобит модель сравнивалась с методом опорных векторов, случайным лесом и моделью Last-2;

- эффективность используемых методов оценивалась по разным метрикам в зависимости от задачи, решаемой исследователями. В задаче планирования заданий на основе предсказания времени их выполнения с помощью обучения с подкреплением (reinforcement learning) [48], а также для планирования задач пользователей с учётом их взаимозависимости и

гетерогенности узлов за счёт механизма внимания (attention mechanism) [60] в качестве метрик использовали среднюю абсолютную ошибку в процентах, среднюю задержку и среднее время завершения задачи. Степень недооценки времени выполнения, средняя точность, среднее время ожидания задачи, средняя задержка и загрузка системы были рассмотрены в исследовании проблемы ошибки пользователя при оценивании времени выполнения в большую (переоценка) или меньшую сторону (недооценка) [50]. Время, требуемое для завершения всех задач в очереди, и среднее время ожидания задач в очереди оценивались при использовании мультиагентного подхода к распределению задач [53]. При планировании связанных подзадач, составляющих задачу, и представленных в виде направленного ациклического графа (DAG) вычисляли среднее время прохождения всей цепочки подзадач для задач [58];

- модели обучения с подкреплением на текущий момент требуют большого количества итераций, чтобы обеспечить сходимость лучше, чем эвристические методы;

- влияние различных программных инструментов, используемых пользователями для выполнения наукоёмких задач, мало анализировалось исследователями и упомянуто всего в двух публикациях – [48] и [52].

В заключение следует отметить, что, исходя из результатов анализа рассмотренных работ, исследователи использовали суперкомпьютерные системы с различными аппаратными возможностями, по-разному собирали данные для экспериментов, использовали критерии оценки результатов, подходящие под определённый рассматриваемый случай. Таким образом, на данный момент представляется затруднительным выработать единый подход к решению задачи планирования заданий пользователя. Следует отталкиваться от выводов, сделанных из результатов анализа аналогичных исследовательских работ и тщательно исследовать имеющиеся данные суперкомпьютерной системы.

Заключение

В разделе проведён анализ проблем оптимизации работы суперкомпьютерной системы с помощью планировщика Slurm, приведено описание СКЦ «Политехнический», планировщика задач Slurm и жизненного цикла задачи пользователя, а также выполнен обзор аналогичных исследований.

В заключение стоит отметить несколько важных выводов:

- потенциал Slurm позволяет достигать высокой производительности вычислений, повышать пропускную способность системы, сокращать время обработки процесса, снижать время ожидания для задач пользователей и время ответа на запросы пользователей;
- использование Slurm для управления гетерогенными вычислительными кластерами предоставляет мощные возможности для оптимизации ресурсов, но также требует формирования подходов к настройке и обслуживанию системы;
- существует проблема несоответствия между ресурсами, запрошенными пользователем для выполнения задачи, и ресурсами, выделенными планировщиком, когда время, запрашиваемое пользователем для задачи, не совпадает с временем, которое фактически было потрачено на выполнение задачи, что приводит к неэффективному использованию ресурсов и неравномерному энергопотреблению и задержкам в выполнении других задач;
- внедрение мультиагентного подхода и методов машинного обучения в систему управления ресурсами Slurm позволит значительно улучшить эффективность использования суперкомпьютера и предоставить более высококачественный сервис для пользователей, решив проблему несоответствия между запрошенными и выделенными ресурсами;
- среди аналогичных исследований единый эффективный подход к решению задачи планирования заданий пользователя на текущий момент отсутствует.

3 Архитектура мультиагентного диспетчера для многоуровневого адаптивного планирования ресурсов гетерогенного кластера

3.1 Задача оптимального диспетчирования

Основной особенностью гетерогенного суперкомпьютерного кластера (ГСК) является наличие различных типов оборудования в рамках одной вычислительной системы. Это с одной стороны является серьезным преимуществом, так как входящие в сеть вычислительные ресурсы в силу различий архитектуры обладают различной производительностью при решении различных типов заданий, и это открывает возможность повышения эффективности выполнения заданий за счет использования тех вычислительных ресурсов ГСК, которые обеспечивают максимальную реальную производительность при их решении, но с другой стороны такая архитектура порождает проблему оптимального распределения поступающих заданий по разнотипным вычислительным ресурсам, входящим в состав ГСК, так как неудачный выбор ресурса может привести как увеличению времени выполнения заданий, дополнительной существенной загрузке вычислительных ресурсов и источников питания оборудования ГСК.

Решение данной задачи возлагается, как правило, на специально выделенный служебный узел, функции которого заключаются в диспетчировании (координации) работы отдельных ресурсов ГСК при выполнении сложных заданий. Такой выделенный служебный узел принято называть диспетчером, а решаемую им задачу – диспетчированием ресурсов. Соответственно учет всех параметров ресурсов и заданий, а также организация взаимодействия между узлами ложится на диспетчер, соответственно, при прочих равных именно от диспетчера будет зависеть время выполнения заданий в ГСК и равномерность загрузки ресурсов ГСК, и, как следствие, эффективность использования ГСК в целом [61,62].

Решение задачи распределения ресурсов ГСК между поступающими заданиями классическими методами, например, методами линейного или динамического программирования, затруднено, во-первых, вследствие того, что число вычислительных ресурсов ГСК и различных заданий может быть велико, что ведет к экспоненциальному росту пространства перебора, а во-вторых, поскольку задания поступают в заранее неизвестные моменты времени, невозможно заранее сформировать временной график их исполнения, соответственно, необходимо динамически распределять задания между ресурсами ГСК с учетом их возможностей и производительности при выполнении того или иного задания.

В настоящее время существует несколько подходов к организации диспетчера ГСК, позволяющего решать указанную задачу. В простейшем случае диспетчирование ресурсов ГСК может осуществляться из единого центра, т.е. с помощью единственного (центрального) диспетчера. При этом центральный диспетчер должен получать от пользователей задания и распределять их между ресурсами ГСК на основании информации о предыдущей работе над отправленными ранее пользовательскими экземплярами такого типа заданий. К преимуществам такого подхода следует отнести простоту организации диспетчера. Однако, при этом резко повышаются требования к быстродействию центрального диспетчера ГСК, что, в свою очередь, ведет к его усложнению, поскольку он должен обеспечивать возможность решения задачи распределения ресурсов ГСК в реальном времени поступления заданий, что крайне проблематично осуществить в случае большого количества различных ресурсов ГСК и сложных заданий.

Частично указанные недостатки устраняются путем применения диспетчера ГСК с иерархической архитектурой. Для этого выделяется диспетчер, отвечающий за организацию работы всей ГСК и получение заданий, – главный диспетчер, а также ряд дочерних локальных диспетчеров, отвечающих за координацию работы некоторого подмножества ресурсов ГСК.

Преимуществом такой иерархической организации диспетчера ГСК является снижение сложности задач диспетчирования ресурсов, возлагаемых на локальные диспетчеры, поскольку каждый из них отвечает за распределение относительно небольшого количества ресурсов ГСК. Однако, в такой схеме организации возникает сложная задача распределения множества поступающих заданий на подмножества, которые должны контролироваться отдельными локальными диспетчерами, а также организации взаимодействия локальных диспетчеров друг с другом при обмене данными.

Описанные выше подходы к организации диспетчера позволяют создавать ГСК, состоящие из ограниченного количества и типов элементов. В то же время более сложные ГСК, состоящие из разнородных узлов и предполагающие возможность поступления заданий от пользователей в произвольные моменты времени потребуют для своей реализации дальнейшее усложнение иерархии диспетчеров, что, в свою очередь, приведет к усложнению взаимодействия между ними. Таким образом, для расширения круга выполняемых в ГСК заданий необходимо дальнейшее изменение архитектуры диспетчера ГСК.

В то же время можно рассмотреть примеры современных способов взаимодействия групп людей при выполнении различных типов общих заданий. В прошлом практически все крупномасштабные задания решались организованными группами людей, где управление осуществлял начальник или иерархия начальников. Однако развитие технологий позволило создать в последние годы совершенно иные подходы к совместной работе, предполагающие применение средств автоматизации и самостоятельного выбора человеком направлений деятельности и заказчика. За счет высокой скорости передачи информации и повсеместной связи человек может моментально найти себе работу, при этом он будет взаимодействовать с неизвестными заранее людьми. Главными особенностями, позволяющими реализовать такой подход, являются возможность обмена информацией и активность членов групп, для решения задания применяется самоорганизация:

группа людей распределяет между участниками части общего задания, и каждый человек заинтересован в выполнении своей части работы, являясь по сути представителем своей группы [20, 63, 64].

Договорное распределение обязанностей широко распространено в социальных системах, например, при выполнении группами людей строительных работ, на сборочном производстве, при осуществлении боевых действий и т.п. Во всех перечисленных выше примерах каждый ресурс системы (например, станок в многостаночном производстве) обладает «своим» агентом [65, 66] – человеком, отвечающим за данный ресурс (станок), а распределение ресурсов при выполнении заданий осуществляется путем взаимодействия (договоренности) агентов (людей) друг с другом о том, кто, какое задание и в какой последовательности выполняет [67, 68]. Фактически в процессе выполнения задания необходимо общаться для достижения договоренностей, то есть осуществлять социальное поведение. Внедрение подобного подхода при организации диспетчеров позволит изменить парадигму диспетчирования сверху, что позволит применять новые методы управления, базирующиеся на самоорганизации. Это, в свою очередь, даст возможность осуществлять распределение и перераспределение большого количества одновременно поступающих в систему заданий, так как оптимизация распределения частей заданий происходит параллельно на множестве агентов.

Развитие сетевых технологий позволяют в настоящее время перейти к созданию диспетчеров принципиально нового типа, а именно мультиагентных диспетчеров. При этом предполагается, что каждый ресурс, входящий в состав системы, должен обладать неким программным агентом, представляющим «его интересы» при реализации диспетчирования [67, 69, 70]. Агенты, представляющие различные ресурсы, могут «договариваться» друг с другом с целью распределения между собой поступающих заданий посредством некоторого информационного канала связи. Такая мультиагентная организация диспетчера обеспечивает возможность создания

самоорганизующихся систем, в которых активные ресурсы будут самоорганизоваться при распределении заданий.

Постановка задачи оптимального распределения заданий в мультиагентном ГСК звучит следующим образом:

Необходимо таким образом осуществить распределение множества поступающих в ГСК экземпляров заданий Z типов P между вычислителями C , включенными в состав ГСК, чтобы в каждый момент времени работы системы суммарное значение выбранной целевой функции минимально возможным.

Таким образом, основная цель работы агентов состоит в взаимодействии с целью получения такого распределения поступающих в ГСК заданий, которое позволило бы обеспечить минимизацию значения выбранной целевой функции.

Представленная постановка задачи оптимального распределения заданий в ГСК является достаточно комплексной, что позволяет производить качественную оптимизацию процесса выполнения заданий в ГСК, но на практике реализация процесса распределения является сложной задачей, так как возникает необходимость мониторинга большого количества параметров каждого вычислителя в ГСК, осуществлять для каждого из них интегральную оценку оптимальности выбора задания для решения в условиях большого количества неизвестных, в частности степень загрузки аппаратуры вычислителей при решении определенных поступающих заданий может варьироваться в зависимости от типа применяемой аппаратуры, входных данных для задания, также от параметров оборудования и заданий может зависеть время их решения.

3.2 Мультиагентный подход к диспетчированию

Для мультиагентной реализации поставленной задачи оптимального распределения заданий в ГСК в рамках каждого вычислителя создается специальная виртуальная или программно-аппаратная сущность – агент, выполняющий роль представителя данного вычислителя в ГСК. Основная

особенность агента – его проактивность, заключающаяся в возможности самостоятельной оценки эффективности решения поступающего в ГСК задания на конкретном вычислителе путем вычисления возможного значения целевой функции и обмена с другими агентами полученными значениями с целью выбора вычислителя, позволяющего достичь минимально возможного в рамках всей ГСК значения целевой функции.

Основная задача агента вычислителя – поиск такого подмножества заданий ГСК для выполнения в рамках данного вычислителя, которое позволит минимизировать предложенная ранее целевая функция.

Второстепенные задачи агента – постоянный мониторинг актуальных значений параметров своего вычислителя и обмен полученными данными с другими агентами ГСК для возможности нахождения наиболее актуального значения целевой функции, а также сбор ретроспективной статистики по выполнению заданий с сохранением всех необходимых параметров вычислителя и заданий, позволяющих осуществлять обучение ГСК для возможности оценки загрузки вычислителя и времени работы при решении определенных заданий.

Для выполнения своего задания с интересующими его входными данными в ГСК каждый пользователь системы формирует свой экземпляр задания Z (задание типа P с необходимыми входными данными и служебной информацией) и направляет его в ГСК.

Поэтому в общем виде в ГСК поступает некоторое множество экземпляров заданий Z_a $a = 1, m$ типа P_n из заранее заданного множества типов заданий $\{P_i\}$, $i = 1, n$.

В состав ГСК входит множество вычислителей $\{C_i\}$, $i=1, m$, каждый из которых включает в свой состав некоторое множество различных типов вычислительных ресурсов, а также сеть передачи данных между ними (см. рисунок 3.1).

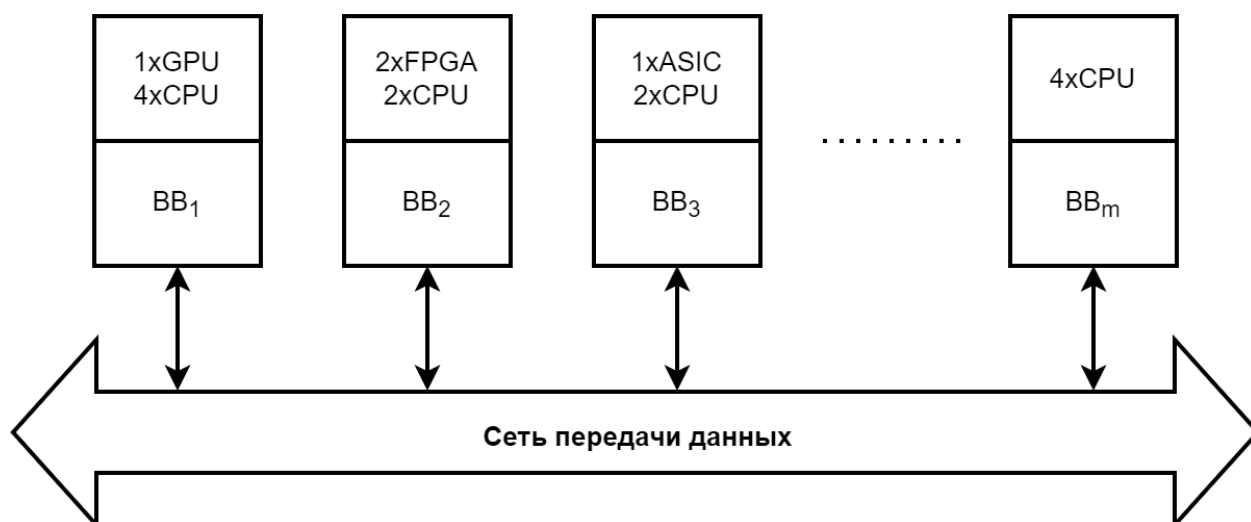


Рисунок 3.1 Структура ГСК

С учетом предложенной ранее целевой функции, задача диспетчирования для достижения требуемого оптимального распределения заданий в ГСК представляет собой достижение такого распределения множества поступающих в ГСК экземпляров заданий Z типов P между вычислителями C , включенными в состав ГСК, чтобы в каждый момент времени работы системы значение выбранной целевой функции для ГСК в целом было минимально возможным, при этом применяемые методы должны оставаться работоспособными при добавлении в состав ГСК нового типа вычислителей.

Соответственно, для достижения указанной задачи необходимо, чтобы диспетчер ГСК мог обеспечить требуемое распределение. Для этого в ходе настоящей работы необходимо разработать новые методы функционирования диспетчера потока заданий в ГСК. При этом, как было сказано ранее, для возможности эффективного диспетчирования большого количества гетерогенного оборудования при выполнении потока разнообразных заданий необходимо в первую очередь определить подходящую архитектуру диспетчера.

3.3 Архитектура диспетчера потока заданий в ГСК

Положим, что в состав ГСК входит множество вычислителей $\{C_i\}$, $i=1,m$, каждый из которых включает в свой состав некоторое множество различных типов вычислительных ресурсов, а также сеть передачи данных между ними. При этом важной особенностью таких вычислителей является то, что из-за наличия в составе каждого вычислителя нескольких видов различных ресурсов, обладающих определенной вычислительной архитектурой (универсальные процессоры, ПЛИС, графические ускорители и т.п.), в случае каждый из них может быть использован для решения своих заданий независимо. Поэтому при разработке архитектуры диспетчера потока заданий в ГСК было принято, что каждый из вычислительных ресурсов в рамках вычислителей должен иметь возможность быть загруженным заданиями независимо.

Итак, положим что в рамках каждого вычислителя $\{C_i\}$, $i=1,m$, входящего в состав ГСК, есть некоторый набор вычислительных ресурсов R_i , тогда фактически получаем, что в состав ГСК входит множество вычислительных ресурсов $R = \langle R_1, R_2, \dots, R_N \rangle$. При этом, под вычислительным ресурсом в дальнейшем будет пониматься некоторое виртуальное вычислительное устройство, опосредованно подключенное к инфраструктуре ГСК.

Как было сказано выше, множество вычислительных ресурсов $R = \langle R_1, R_2, \dots, R_N \rangle$ в рамках ГСК предназначено для выполнения некоторого множества пользовательских экземпляров заданий $Z = \langle Z_1, Z_2, \dots, Z_L \rangle$, которые могут поступать в неизвестные заранее моменты времени (см. рисунок 3.1), причем в ГСК может выполняться множество заданий одновременно. Для каждого задания $Z_l \in Z$ пользователь может установить дедлайн исполнения - момент времени T_{max}^l , к которому он хочет получить результат его выполнения [71].

Исходя из приведенных выше соображений цель диспетчера ГСК можно сформулировать в следующем виде: необходимо обеспечить такое распределение заданий $Z_l \in Z$, поступающих в произвольные моменты времени в множестве вычислительных ресурсов $R = \langle R_1, R_2, \dots, R_N \rangle$ в ГСК, которое позволило бы с одной стороны минимизировать предложенную целевую функцию, а с другой стороны выполнить как можно большее количество пользовательских заданий к требуемым моментам времени.

В процессе обработки поступившего экземпляра задания в ГСК диспетчер может сделать вывод о возможности решения задания в срок, о возможности решения задания не в срок (ориентировочно к моменту времени, выходящему за рамки таймаута) с соответствующим запросом к пользователю об актуальности решения задания к указанному времени, в зависимости от ответа на который выполнение задания может быть отменено или продолжено.

Исходя из приведенных выше соображений рассмотрим более подробно различные подходы к построению диспетчера распределенной системы, включающей в свой состав N ресурсов R_1, R_2, \dots, R_N .

Как было сказано выше, в простейшем случае диспетчер ГСК можно реализовывать по классической централизованной схеме (см. рисунок 3.2). При этом каждый ресурс R_i ($i = 1, 2, \dots, N$), входящий в состав ГСК, обладает некоторым простейшим пассивным локальным устройством управления $УУ_i$, а координация их взаимодействия при распределении и выполнении заданий должна осуществляться с помощью центрального диспетчера. При такой организации функции центрального диспетчера заключаются в приеме очередного задания Z_l , нахождение для его выполнения ресурса, входящего в состав ГСК, и организации их взаимодействия в процессе выполнения задания, в то время как локальные устройства управления отдельных ресурсов должны обеспечивать выполнение заданий, назначенных им диспетчером.

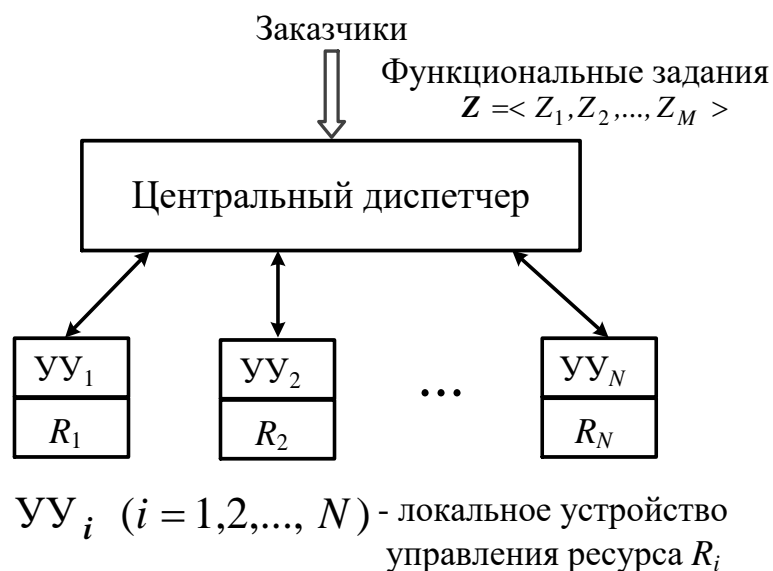


Рисунок 3.2 ГСК с централизованным диспетчером

К преимуществам такой схемы следует отнести простоту организации диспетчера, а также возможности использования классических методов и алгоритмов теории расписаний для решения задачи диспетчирования ГСК. Однако, поскольку сформулированная выше задача диспетчирования (распределения ресурсов) в ГСК предполагает использование гетерогенных ресурсов и выполнение разнообразных заданий, ее вычислительная трудоемкость будет экспоненциально расти при увеличении количества переменных (в данном случае – числа ресурсов в ГСК, а также количества поступающих заданий), то данная схема организации диспетчера возможна только в случае, когда число N ресурсов, входящих в ГСК, относительно невелико, а новые задания поступают на ГСК достаточно редко, кроме того при необходимости масштабирования системы (наращивания количества ресурсов) потребуются кардинальная переделка алгоритмов работы диспетчера ГСК. Помимо этого, существует важная проблема при использовании централизованного диспетчера, возникающая при введении в состав ГСК новых типов вычислительных ресурсов: необходимо каждый раз перестраивать реализованные алгоритмы оптимизации с учетом появления новых оптимизируемых параметров, что может привести к существенному

усложнению процесса оптимизации даже при добавлении нескольких узлов с новым видом вычислителей.

Частично указанные недостатки можно устранить за счет применения иерархической схемы организации диспетчера ГСК (см. рисунок 3.3). При этом диспетчер ГСК строится в виде иерархического дерева, в котором диспетчеры нижнего уровня отвечают за организацию работы некоторого подмножества ресурсов ГСК, а функции диспетчера верхнего уровня заключаются в приеме заданий и «спуске» в зоны ответственности диспетчеров нижнего уровня. При этом, поскольку количество ресурсов, за которые «отвечает» каждый из диспетчеров нижнего уровня, будет относительно невелико, то, соответственно, и временная сложность решаемой диспетчером нижнего уровня задачи диспетчирования также будет невелика. Это позволяет использовать при организации работы диспетчеров нижнего уровня классические методы и алгоритмы теории расписаний, реализация которых при небольшом количестве диспетчируемых ресурсов не потребуют больших вычислительных и временных затрат.

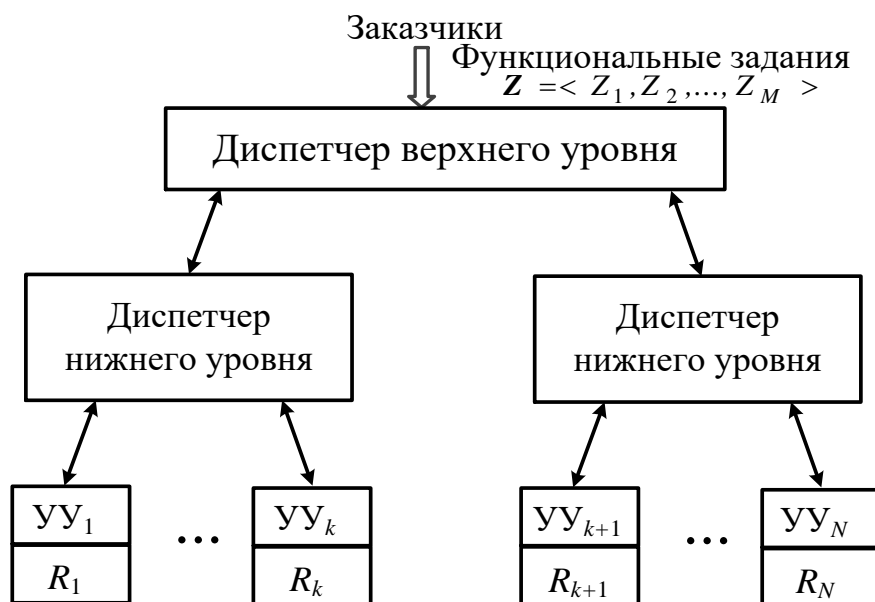


Рисунок 3.3 ГСК с иерархическим диспетчером

По сравнению с полностью централизованной схемой организации диспетчера при использовании иерархического диспетчера существенно

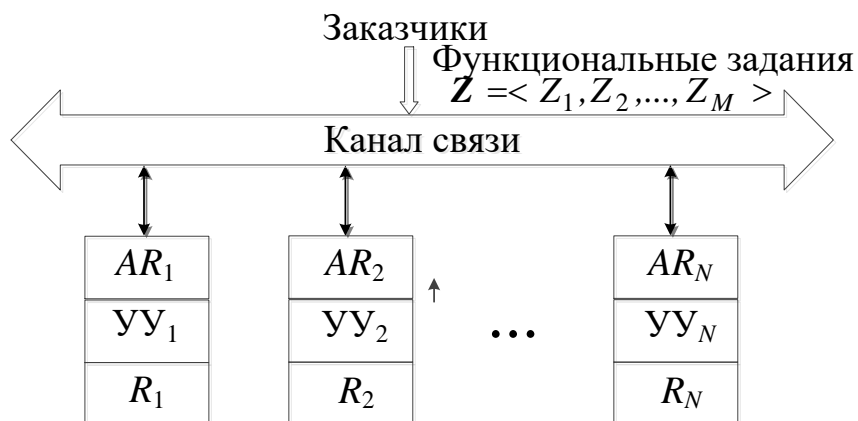
улучшаются возможности быстрого масштабирования, хотя это все равно требует введения дополнительных физических вычислительных узлов, что не всегда возможно быстро сделать. Однако, иерархическая схема организации диспетчера ГСК не решает проблему, возникающую при введении в состав ГСК новых типов вычислительных ресурсов.

В последние годы появилось и активно развивается направление в теории управления, предполагающее принятие решений, построенных за счет взаимодействия множества проактивных агентов, позволяющее осуществлять распределение заданий с применением коллективного интеллекта – теория мультиагентных систем [21, 22, 27, 72-82]. При этом каждый ресурс $R_i \in R$ должен представляться при диспетчировании ГСК некоторым программным агентом AR_i , которые «умеют договариваться» друг с другом о распределении между собой поступающих заданий таким образом, чтобы равномерно распределить нагрузку при выполнении заданий. Иными словами, задача диспетчирования в данном случае должна решаться в результате взаимодействия агентов, представляющих различные ресурсы ГСК, друг с другом. Выше в настоящем отчете приведен обзор актуальных алгоритмов и методов оптимизации распределения заданий в ГСК.

При этом в большинстве случаев в настоящее время применяется следующий подход к мультиагентной оптимизации: для оптимизируемой системы создаётся виртуальный двойник, в котором каждый из узлов представляется виртуализированным агентом, в рамках виртуального двойника они взаимодействуют, находят решения и эти решения переносятся в реальность. При этом фактически вычислительная сложность работы такого диспетчера становится даже выше, чем при применении классических методов. Поэтому в рамках настоящей работы для решения задачи диспетчирования в гетерогенной системе предлагается отказаться от концепции применения виртуализированных агентов.

Перспективной архитектурой построения вычислительных систем на базе мультиагентной схемы организации (см рисунок 3.4), которая позволит

минимизировать влияние гетерогенности системы на сложность процесса диспетчирования, является схема, в которой как таковой выделенный физический диспетчер отсутствует, а все ресурсы R_1, R_2, \dots, R_N , входящие в состав ГСК, включают в свой состав множество физически разнесенных работающих в составе вычислительных ресурсов программных (или программно-аппаратных) агентов AR_i , которые объединены информационным каналом связи, посредством которого они могут обмениваться информацией друг с другом. Главным отличием такой архитектуры от существующих в настоящее время методов, предполагающих оптимизацию с применением виртуализированных агентов, является ее распределенность: вычислительная нагрузка может быть равномерно распределена по множеству вычислителей.



AR_i ($i = 1, 2, \dots, N$) - программный агент ресурса R_i

Рисунок 3.4 Мультиагентная организация диспетчера ГСК

Такая мультиагентная организация диспетчера ГСК имеет и другие преимущества:

- программный агент AR_i ресурса R_i будет иметь достоверную и полную информацию о специализации и текущей производительности «своего» ресурса при выполнении тех или иных заданий, что позволяет повысить качество решения задачи диспетчирования;

- в процессе диспетчирования вычислительная и, соответственно, временная трудность задания, выполняемого программным агентом, существенно снижается, поскольку агент «отвечает» только за один «свой» ресурс, входящий в состав ГСК, что позволяет снизить время решения задачи диспетчирования;
- для масштабирования ГСК (увеличения числа ресурсов) нет необходимости менять какие-либо алгоритмы работы диспетчеров, а достаточно просто подключить новый ресурс (обладающий соответствующим программным агентом) к общему каналу информационного обмена, и агент сам возьмет на себя всю новую нагрузку;
- система будет обладать максимальной отказоустойчивостью, поскольку в ней отсутствуют части, выход из строя которых приводит к полной потере работоспособности ГСК в целом;

Однако, несмотря на очевидные преимущества, рассмотренная выше мультиагентная схема организации диспетчера практически не используется в современных суперкомпьютерных системах. Это вызвано, в первую очередь, отсутствием методов мультиагентного диспетчирования ресурсов при выполнении множества заданий, а также методов и алгоритмов работы программных агентов, позволяющих использовать предложенную распределенную архитектуру и обеспечивающих высокую скорость решения задачи диспетчирования при сохранении допустимого уровня качества распределения.

3.4 Метод мультиагентного диспетчирования для обеспечения многоуровневого адаптивного распределения ресурсов гетерогенного кластера

Можно заметить, что подобный распределенный мультиагентный подход к диспетчированию ресурсов широко распространен [83-90]. Приведем лишь несколько примеров.

1. Игра в футбол.

Футбольная команда – это пример гетерогенной (так как у игроков есть различные возможности на поле) социальной ГСК с мультиагентным диспетчером, в которой каждый ресурс (игрок) самостоятельно принимает решение о своих текущих действиях на поле для достижения общекomандной цели – забить гол противнику и не пропустить в свои ворота, координируя при этом взаимодействия с другими ресурсами (игроками) посредством визуального информационного канала.

2. Автомобильный трафик.

Автомобильный трафик – это «человеко-машинная» ГСК, включающая в свой состав тысячи и даже десятки тысяч ресурсов (автомобилей), каждым из которых управляет свой агент (шофер). В системе отсутствует какой-либо центральное устройство управления (если не считать дорожные знаки и светофоры), а взаимодействие агентов (шоферов) осуществляется посредством визуального информационного канала. Несмотря на сложность задачи управления автомобилем при плотном трафике, отдельный агент-шофер легко справляется с ее решением, что в целом обеспечивает достижение общей целевой функции всей распределенной системы – безаварийное достижение каждым из автомобилей заданного пункта назначения.

3. Боевые подразделения.

Боевые подразделения – это пример гетерогенных распределенных систем, включающих в свой состав множество ресурсов с различной специализацией, таких как отдельные бойцы-пехотинцы или боевые машины, управляемые экипажем. Общим заданием, стоящим перед такой системой, является нанесение максимального ущерба противнику при минимизации своих собственных потерь [91]. При этом в условиях скоротечного боя практически невозможно осуществлять управление каждым из этих ресурсов из единого центра управления (командного пункта), а каждый из них должен принимать самостоятельные решения, направленные на достижение общей цели. Поэтому, в данном случае, управление может осуществляться только

путем мультиагентного взаимодействия отдельных агентов-людей, представляющих различные ресурсы системы.

Во всех перечисленных выше социальных системах решение задачи диспетчирования ресурсов осуществляется с помощью множества агентов (людей), каждый из которых представляет свой ресурс (например, обрабатывающий станок) и решает относительно простую задачу загрузки «своего» ресурса поступающими заданиями, в то время как скоординированная работа всей системы, направленная на выполнение поступающего потока заданий, осуществляется путем информационного взаимодействия агентов [92-95]. Такое взаимодействие в сообществе людей, направленное на достижение некоторой общей цели, называется самоорганизацией [96-131].

В общем случае стратегии самоорганизации при выполнении некоторого общего задания, могут быть различными. В идеале все агенты должны четко и однозначно понимать стоящие перед ГСК задания и стараться решить их оптимальным образом с точки зрения ГСК в целом, что позволяет сделать вывод о том, что каждый агент должен обладать набором стратегий и алгоритмов, которые наиболее подходят для реализации диспетчера ГСК [132-146].

Таким образом, методология построения самоорганизующихся диспетчеров ГСК должна базироваться на следующих суждениях:

Во-первых, перспективным способом построения ГСК, позволяющим добиться самоорганизации процесса ее диспетчирования, является мультиагентная организация, в которой каждый ресурс, входящий в состав ГСК, обладает некоторым находящимся в составе вычислительного ресурса программным или программно-аппаратным агентом, участвующим во взаимодействии с программными агентами других ресурсов в решении задачи диспетчирования ГСК при выполнении множества заданий.

Во-вторых, при создании мультиагентных методов и алгоритмов диспетчирования ресурсов ГСК целесообразно использовать стратегии, методы и алгоритмы, предполагающие распределенную самоорганизацию.

Однако при реализации предлагаемого распределенного подхода к самоорганизации необходимо возникает ряд новых проблем:

- Необходимо разработать подходы к информационному взаимодействию;
- Необходимо разработать методы и алгоритмы взаимодействия агентов в составе ГСК при распределении и выполнении экземпляров пользовательских заданий.

Основы информационного взаимодействия в мультиагентных диспетчерах

Как было сказано ранее, взаимодействие пользователей с ГСК осуществляется через специально выделенные узлы $\{E_i\}$, $i = 1, k$: пользователи отправляют в ГСК свои экземпляры заданий Z_a типа P_n из заранее заданного множества типов заданий $\{P_i\}$, $i = 1, n$ с пользовательскими наборами входных данных.

При этом при классическом подходе к диспетчированию фактически диспетчер обычно реализуется на множестве выделенных контролирующей систему узлов. Однако, в связи с распределенностью предлагаемой архитектуры диспетчера, предполагающей проактивность программных или программно-аппаратных агентов, управляющие узлы не могут передавать информацию о поступающих заданиях всему множеству агентов в ГСК.

В качестве подхода к обмену информацией в распределенном диспетчере ГСК в рамках предлагаемой архитектуры предлагается использовать иной способ организации взаимодействия – применение служебных серверов, реализующих необходимый минимум функций по получению и хранению экземпляров пользовательских заданий, и арбитража, что позволит свести к минимуму вычислительную нагрузку этих узлов и не приведет к их превращению в бутылочное горлышко. Эти серверы фактически

будут выполнять роль «досок объявлений», соответственно, в дальнейшем такие серверы будем называть доски объявлений (ДО) [147]. Схема работы подобной системы представлена на рисунке 3.5.

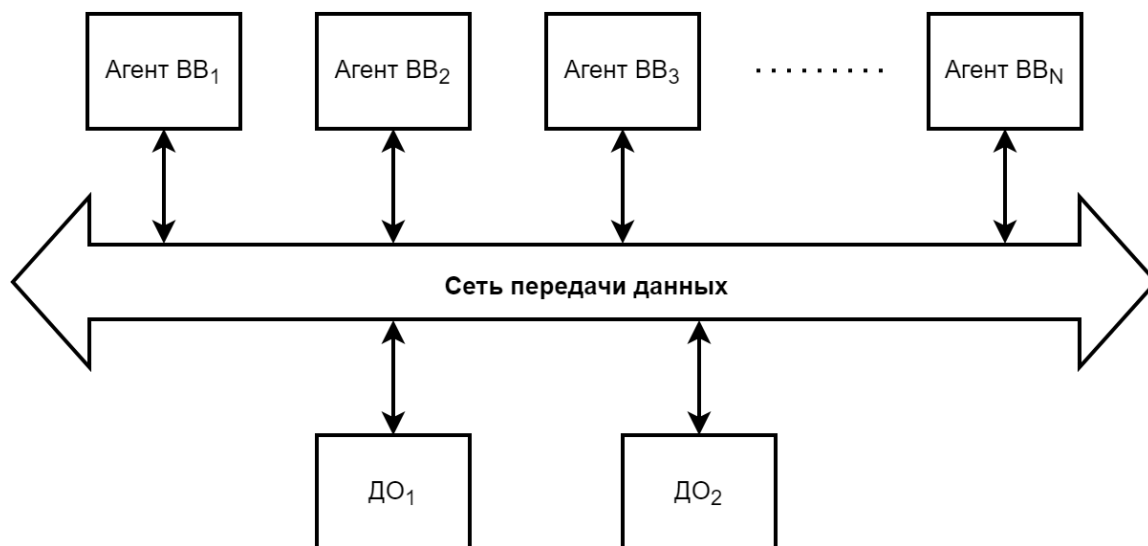


Рисунок 3.5 Архитектура распределенного диспетчера ГСК при взаимодействии с помощью досок объявлений

При этом предполагается, что функции непосредственно диспетчирования реализуются на уровне множества программных или программно-аппаратных агентов. Количество досок объявлений, используемых в системе, ничем не ограничено, в общем виде они даже могут быть предоставлены пользователями для своих собственных нужд.

4 Разработка статистической модели функционального управления распределением ресурсов гетерогенного кластера

4.1 Разработка протокола сбора статистических данных

В процессе эксплуатации суперкомпьютера и накопления данных о результатах выполнения вычислительных задач появляется уникальная возможность для исследования статистических характеристик текущих процессов с целью повышения эффективности работы суперкомпьютера. Правильная организация сбора статистических данных важна, поскольку беспорядочный сбор данных может привести к существенному ограничению возможностей статистического анализа, а также к ошибочным выводам, что обесценит его результаты.

При подготовке запуска задачи исследователь формирует заявку, в которую входят:

- тип вычислительных ресурсов;
- объем вычислительных ресурсов (количество вычислительных узлов кластера, требуемое количество процессоров, объем памяти, и др.);
- время выполнения задачи;
- скрипт запуска задачи.

Заявка отправляется в очередь, а затем поступает в работу. На момент снятия показаний поступает статистическая информация о проведенных вычислениях. Более подробное описание жизненного цикла задач приведено в разделе 2.2.3 настоящего отчета.

Единица статистической информации включает в себя априорную информацию пользователя и апостериорную информацию о текущем (на момент получения статистической информации) состоянии задачи, а также ресурсов суперкомпьютера, использованных для ее исполнения, записанную в два файла. Первичным носителем статистической информации является файл с апостериорными показателями работы суперкомпьютера. Заявка пользователя необходима для установления связи запрашиваемых

пользователем и реально использованных ресурсов суперкомпьютера в процессе исполнения задачи.

Статистические данные должны включать запуски, полученные за определенный период времени относительно момента времени подачи заявки пользователя, принятых к исполнению суперкомпьютером. Период времени желательно выбирать с учетом сезонного цикла. Использование частичных (отфильтрованных по тому или иному показателю) данных и данных, полученных в различные промежутки времени, в рамках одного статистического исследования не допускается. Запись статистических показателей в файлы, предназначенные для статистического анализа, должна производиться автоматически. Для установления связи заявленных пользователем и реально использованных ресурсов суперкомпьютера требуется использовать переменные идентификации задачи. Ввиду наличия сложных и пакетных запусков, переменная идентификации задачи в файле апостериорной информации может состоять из нескольких частей, первая часть которой должна соответствовать идентификационному номеру задачи в заявке пользователя. Предполагается, что все задачи, присутствующие в файле апостериорной информации, должны присутствовать и в файле заявок пользователей.

Для обозначения различных параметров, характеризующих данные заявки в работе используются следующие понятия и определения:

Необходимое время исполнения задачи – промежуток времени решения задачи на суперкомпьютере, необходимый для успешного завершения вычислений.

Затраченное время – реальное время исполнения задачи на суперкомпьютере.

Необходимое процессорное время – совокупное время решения задачи с учетом числа ядер суперкомпьютера, требуемое для решения задачи, равное времени исполнения задачи, умноженному на число центральных процессорных ядер, задействованных в процессе решения.

Затраченное процессорное время – реальное время исполнения задачи, умноженное на число центральных процессорных ядер суперкомпьютера, задействованных в процессе вычислений.

Успешный запуск – запуск задачи пользователя, завершающийся успешным ее выполнением.

Эффективность работы суперкомпьютера – тройка характеристик, характеризующих долю успешных запусков от общего числа запусков, а также отношения затраченного времени на выполнение успешных запусков к общему затраченному времени и процессорного времени, затраченного на выполнение успешных запусков, к общему затраченному процессорному времени. Последняя характеристика является наиболее показательной с точки зрения оценивания эффективности работы суперкомпьютера.

Время сбора данных – период времени с момента получения информации о наиболее раннем запуске до момента получения статистических данных.

4.2 Разработка и реализация алгоритмов анализа результатов работы суперкомпьютера.

Важным фактором эффективности работы суперкомпьютера является процент успешно завершенных задач, а также доля времени и процессорного времени работы суперкомпьютера, затраченного на успешное выполнение задач. В рамках разведочного анализа предполагается эмпирическое изучение этих характеристик и их зависимости от области знаний, к которой относится задача. Главной целью разведочного анализа является эмпирическое изучение распределения необходимого времени (процессорного времени) исполнения задачи пользователя, и его зависимости от набора наблюдаемых сопутствующих факторов. Каждое наблюдение (или единица статистической информации) представляет собой совокупность характеристик запуска задачи на суперкомпьютере, включающих априорные данные пользователя, совокупность выделяемых мощностей суперкомпьютера, затраченное время

выполнения задачи и показатель успешного завершения задачи. Следует отметить, что необходимое и затраченное время исполнения задачи в общем случае отличаются. Вычисления в рамках поставленной задачи могут быть завершены досрочно, ввиду сбоя работы системы ошибки программирования или нехватки выделенных ресурсов, а, в некоторых случаях, на момент получения статистических данных задача может оставаться в процессе выполнения. Известно, что исключение незавершенных задач, как и использование затраченного времени вычислений вместо необходимого ведет к недооценке значения изучаемой характеристики (систематической ошибке оценивания в меньшую сторону). Для корректного оценивания распределения времени исполнения задачи пользователя следует использовать методы анализа неполных данных типа времени жизни с правым цензурированием.

Непараметрическая оценка Каплана-Мейера функции отказа частично наблюдаемого необходимого времени исполнения задачи пользователя используется для оценивания его распределения при выполнении условия независимости цензурирования при наличии однородных данных с правым цензурированием. Разведочный анализ предполагает построение оценок Каплана-Мейера по объединенным данным работы суперкомпьютера и с учетом группировки задач по области знаний, числу потребляемых мощностей суперкомпьютера (число процессорных ядер). Дальнейший статистический анализ подразумевает исследование статистической значимости отличий распределений необходимого времени исполнения задачи в различных областях знаний. Обнаружение статистически значимых отличий позволит говорить о неслучайности полученных выводов с определенным уровнем доверия. В перспективе, для проверки значимости различий распределений необходимого времени исполнения задачи в различных областях знаний предполагается использовать критерий Жеана-Уилкоксона, применяемый для сравнения распределений частично наблюдаемых времен отказов по цензурированным справа данным. В случае обнаружения статистической значимости различий будет проведена проверка уточненных гипотез о

стохастическом порядке пар распределений. Для этого будут реализованы методы сравнения функций распределений в ряде точек (предполагается использовать октанты распределения по комбинированным данным) и построение совместных доверительных множеств для выбранных сравнений значений функций распределений в выбранных точках. Ввиду большого числа выдвигаемых гипотез проверка гипотез о стохастическом порядке распределений подразумевает наличие поправки, а следовательно, для получения достаточной мощности критерия потребуется большой объем данных. Предварительно будет проведено исследование мощности разрабатываемых методов проверки гипотез и выдвинуты лишь те гипотезы, которые позволят обеспечить достаточную мощность статистического критерия на базе имеющихся данных. На конец отчетного периода имеется реализация бета-версии критерия типа хи-квадрат для цензурированных справа данных, на базе которого предполагается реализовать методы множественного сравнения. В рамках дальнейшего исследования предполагается развитие адаптивного подхода к формированию задач статистического анализа, а также разработка и применение параметрических и семипараметрических регрессионных моделей анализа зависимости распределения времени исполнения задачи от сопутствующих факторов.

4.3 Разведочный статистический анализ результатов работы СКЦ «Политехнический»

Основные цели разведочного анализа – изучение структуры статистических данных, определение показателей эффективности работы суперкомпьютера при решении задач в различных областях знаний и получение первичного представления о распределении времени исполнения задачи и его зависимости от области знаний, к которой принадлежит решаемая задача.

4.3.1 Подготовка данных для анализа

В результате анализа структуры исходных данных результатов работы суперкомпьютера СКЦ «Политехнический» были выявлены полностью идентичные записи (дубликаты), записи, связанные с подключением пакетов. Кроме того, в течение времени сбора данных были произведены 38 сложных запусков, заключающихся в том, что пользователь в рамках одной задачи производит несколько запусков. Последовательные этапы подготовки данных отражены в таблице 4.1. В файле для проведения анализа осталось 440164, каждый из которых соответствует одному запуску задачи пользователя на суперкомпьютере.

Таблица 4.1. Последовательные этапы подготовки данных

Этап	Удалено записей	Осталось записей
Исходный файл	-	1010377
Удаление дубликатов	5452	1004925
Удаление запусков пакетов	457367	547558
Удаление сложных (многоуровневых) записей	2955	544603
Удаление запусков, продолжительность которых не превышает 5 секунд	104439	440164

4.3.2 Основные показатели, используемые в рамках разведочного анализа

Описание переменных, используемых в рамках разведочного анализа:

– 'ElapsedRaw' - Реальное время (затраченное время), за которое вычислялась задача, в секундах.

- 'CPUTimeRAW' - Затраченное процессорное время (реальное время, умноженное на число процессорных ядер), потребленное задачей, в процессоро-секундах.
- 'GroupID' - Идентификатор группы, от имени которой задача была поставлена в очередь.
- 'JobID' - Идентификационный номер задачи или шага задачи.
- 'NCPUs' - Общее число процессоров, выделенных задаче.
- 'NumNodes' - Число узлов, запрошенных или выделенных на задачу или шаг задачи.
- 'UserID' - Идентификатор пользователя, от имени которого была запущена задача.
- 'State' - Статус задачи после окончания вычисления.
- По ним дополнительно построены переменные:
- 'Industry' – Область знаний в которой проводятся расчеты, отраслевая принадлежность задачи.
- 'RunType' – Способ запуска задачи: вручную или с помощью автоматического ПО.

4.3.3 Результаты исследования эффективности работы суперкомпьютера

Эффективность работы суперкомпьютера измеряется тремя показателями: процентом успешно завершенных задач от общего числа задач, отношением затраченного времени в процессе исполнения успешно завершенных задач к общему времени исполнения задач пользователей и отношением затраченного процессорного времени в процессе исполнения успешно завершенных задач к общему затраченному процессорному времени. В данное исследование включены только запуски задач пользователя, произведенные в период с 19.06.2022 по 17.11.2022, продолжительность которых более 5 секунд. Запуски, время исполнения которых не превышает 5 секунд, составляют 19.2% от общего числа запусков, но на их исполнение приходится лишь 0.0006% затраченного времени и 0.0003% затраченного

процессорного времени. Также из рассмотрения исключаются запуски задач пользователя, находящиеся в процессе исполнения на момент получения статистических данных, в результате чего общее число запусков задач пользователя сокращается до 440013. Имеются два типа задач пользователя в зависимости от способа формирования задания суперкомпьютеру (заявки пользователя): автоматический и вручную. Автоматический способ формирования задания подразумевает формирование параметров задачи для планировщика с помощью сторонней программы или сервиса. Характерным примером такой программы является программный продукт «Geovation», предназначенный для обработки и интерпретации сейсмических данных. В качестве другого примера можно привести CML-Bench – систему класса SPDM (англ., simulation process and data management), которая осуществляет автоматический запуск задач в области вычислительной механики и решения оптимизационных задач. В остальных случаях пользователь составляет заявку вручную, в которой указываются основные параметры запуска задачи – кол-во и тип вычислительных ресурсов (при этом часто не указывается запрашиваемое время выполнения и в этом случае используется значение по умолчанию – двое суток). Задачи, запущенные вручную, составляют лишь (11.5%) от общего числа задач пользователей, но занимают 66.4% суммарного времени исполнения задач и 84.1% процессорного времени суперкомпьютера (см. рисунок 4.1).

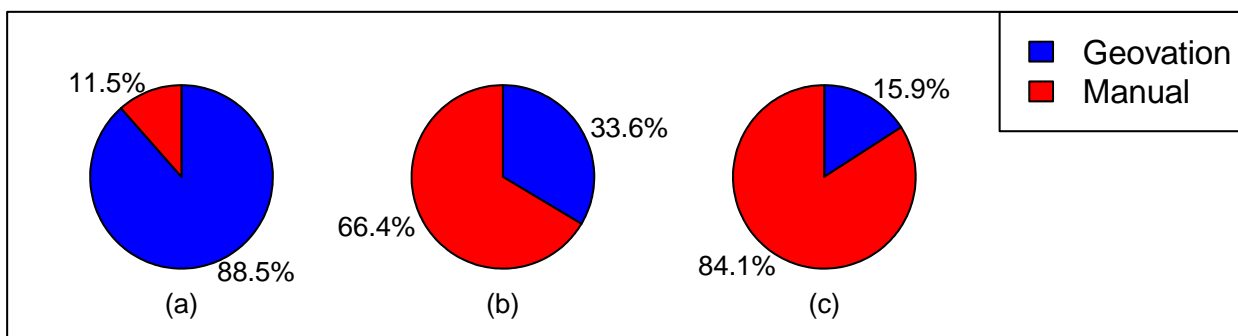


Рисунок 4.1 Распределение типов запуска (автоматический или вручную) для трех основных показателей эффективности: (а) число задач пользователя; (b) затраченное время; (с) затраченное процессорное время.

Отметим достаточно высокие показатели эффективности использования суперкомпьютера при выполнении задач, запущенных с использованием программного комплекса «Geovation» (99.9% выполненных задач, 97.6% затраченного времени и 97.8% затраченного процессорного времени на их выполнение), но низкие – в случае запуска задач вручную (см. рисунки 4.2 и 4.3). Отдельно необходимо отметить, что при запуске задач вручную 55.3% процессорного времени суперкомпьютера используется впустую.

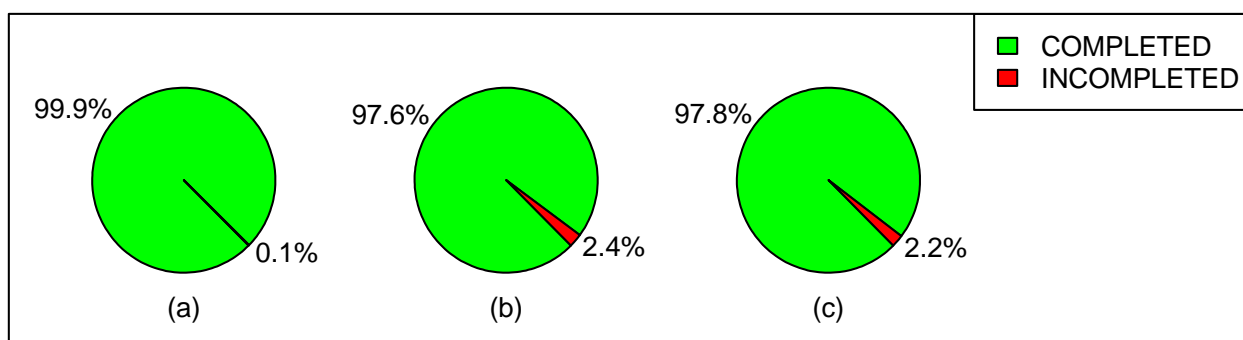


Рисунок 4.2 Эффективность использования суперкомпьютера при выполнении задач, запущенных с использованием «Geovation»: (а) число задач; (b) затраченное время; (с) затраченное процессорное время.

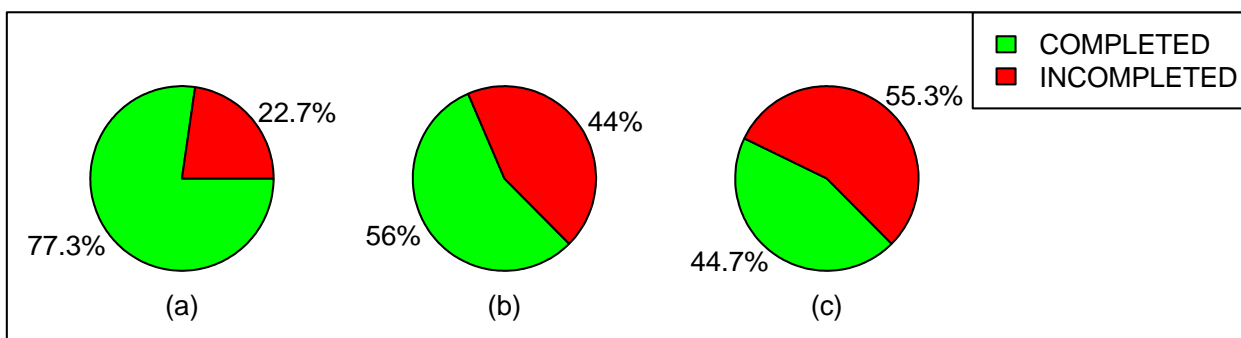


Рисунок 4.3 Эффективность использования суперкомпьютера при выполнении задач, запущенных вручную: (a) число задач; (b) затраченное время; (c) затраченное процессорное время.

Уточненные результаты эффективности использования суперкомпьютера получены с использованием классификации задач по областям знаний. Поскольку все задачи, запущенные с использованием «Geovation», относятся к геофизике, уточненное исследование проведено только для задач, запущенных вручную. Распределение показателей эффективности и эффективность использования суперкомпьютера при решении задач в 10 областях знаний приведены в таблице 4.2.

Необходимо отметить крайне низкую эффективность использования суперкомпьютера при решении задач в областях механики, геофизики и астрофизики, тогда как при решении задач в областях энергетики, биоинформатики и информационных технологий эффективность использования суперкомпьютера достаточно высокая, но гораздо ниже, чем при запусках задач с использованием «Geovation». При этом осуществлять прямое сравнение не совсем корректно, так как система «Geovation» формирует поток определенных задач одного класса, а пользователи, в большинстве случаев, решают различные типы задач.

Таблица 4.2 Эффективность использования суперкомпьютера при решении задач в различных областях знаний.

Область знаний	Число задач пользователей			Доля от общего значения			Эффективность		
	Завешено	Снято	Всего	Задачи	Время	Проц. время	Задачи	Время	Проц. время
Астрофизика	189	118	307	0.6%	0.9%	7.8%	61.6%	35.3%	29.2%
Биоинформатика	17888	313	18201	35.8%	1.3%	0.5%	98.3%	87.7%	80 %
Биофизика	4747	755	5502	10.8%	12 %	5.8%	86.3%	67.5%	47.3%
Энергетика	2918	1130	4048	8 %	1.9%	6.8%	72.1%	81.6%	90.4%
Геофизика	458	1139	1597	3.1%	7.8%	8.6%	28.7%	12.4%	10.8%
Инф. технологии	605	413	1018	2 %	0.9%	0.6%	59.4%	77.2%	77 %
Механическая инженерия	4246	1447	5693	11.2%	10.6%	23.7%	74.6%	69.3%	59.1%
Механика	551	1261	1812	3.6%	11.2%	21 %	30.4%	7.2%	6.5%
Физика	7377	4935	12312	24.2%	52.6%	24.6%	59.9%	65.4%	65.1%
Радиофизика	340	42	382	0.8%	0.8%	0.7%	89 %	89.5%	59.7%

4.3.4 Результаты исследования распределения времени исполнения задач

Для оценки распределения необходимого времени и процессорного времени исполнения задач в различных областях знаний использованы оценки Каплана-Мейера (рис. 4.4 и 4.5 соответственно). Визуальный анализ полученных оценок функций выживания в различных областях знаний позволяет установить, что задачи в областях геофизики и механики требуют стохастически больше времени для успешного завершения, чем задачи из других областей знаний, а наиболее короткие по времени использования суперкомпьютера и по процессорному времени задачи характерны для биоинформатики. Границы частных доверительных интервалов оцененных квартилей распределений времен и процессорных времен исполнения задачи (см. таблицы 4.3. и 4.4 соответственно) показывают достаточную точность каждой из большинства оценок, хотя некоторые квартили и не могут быть оценены ввиду большого процента неудачно завершенных задач. По

продолжительности необходимого времени (процессорного времени) исполнения задачи были классифицированы по 6 группам: от 5 секунд до 3 минут, от 3 до 10 минут, от 3 минут до 1 часа, от 1 часа до 1 суток, от 1 до 3 суток и более 3 суток (*56 ЦП каждый интервал процессорного времени соответственно). Такой подход удобно использовать в задачах предсказания. Распределение длительности времени исполнения задач, полученные на базе оценок Каплана-Мейера, приведены в таблице 4.5 (для процессорного времени – таблице 4.6 соответственно). Отметим существенные различия полученных распределений. В частности, продолжительность необходимого времени (процессорного времени) исполнения более половины задач биоинформатики от 3 до 10 минут, а продолжительность необходимого времени (процессорного времени) большей части задач механики и геофизики наиболее часто превышает 3 суток.

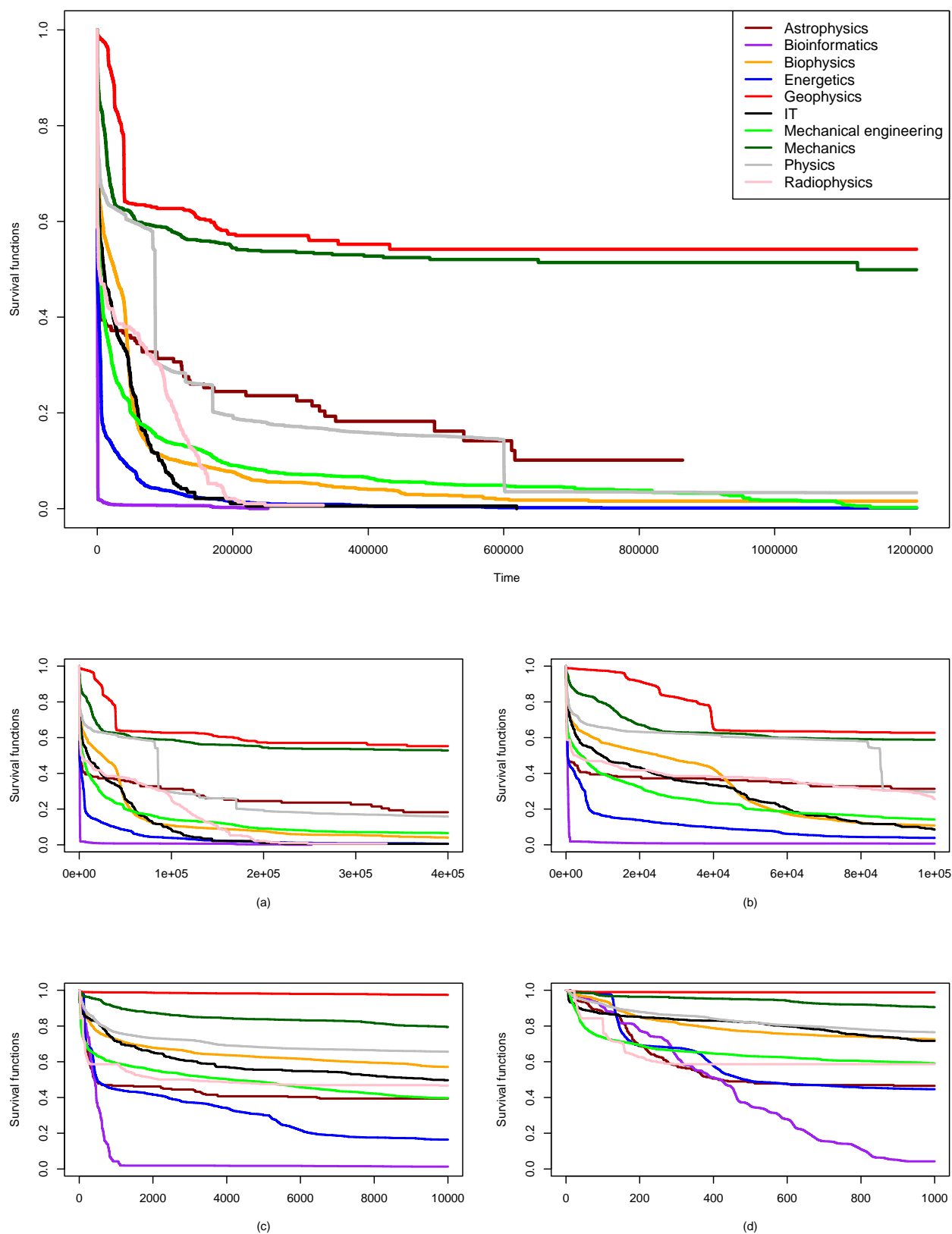


Рисунок 4.4 Оценки Каплана-Мейера распределений времени исполнения задач в различных областях знаний с детализацией: (а) область времен исполнения задач, не превышающих $4 \cdot 10^5$ сек.; (b) не превышающих $1 \cdot 10^5$ сек.; (с) не превышающих $1 \cdot 10^4$ сек.; (d) не превышающих $1 \cdot 10^3$ сек.

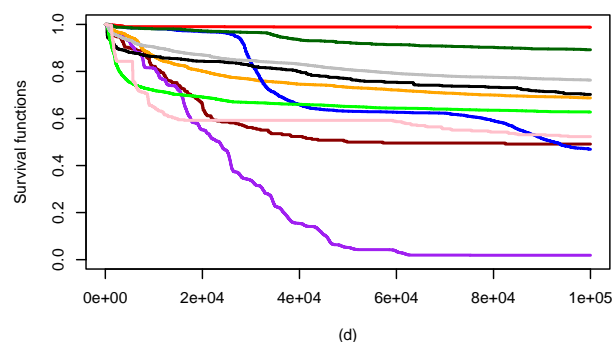
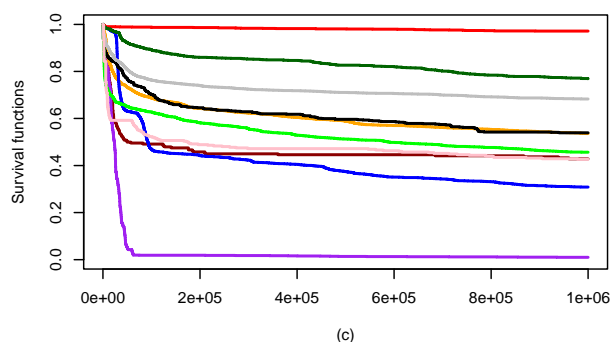
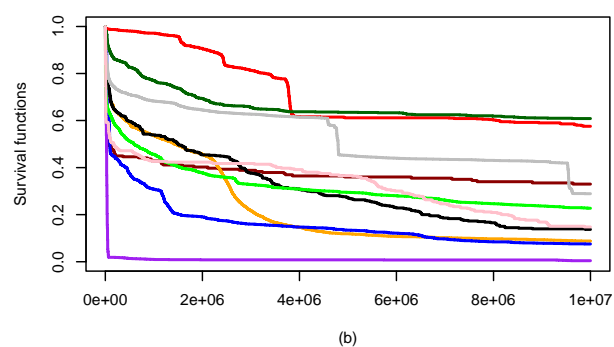
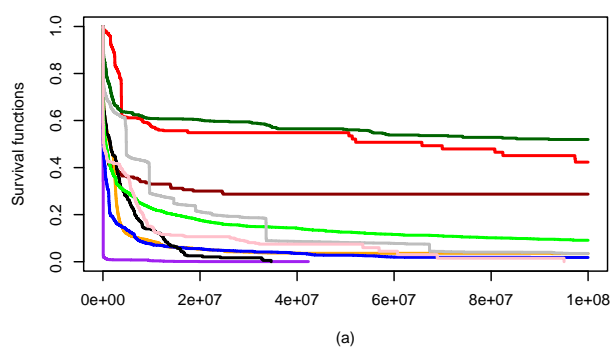
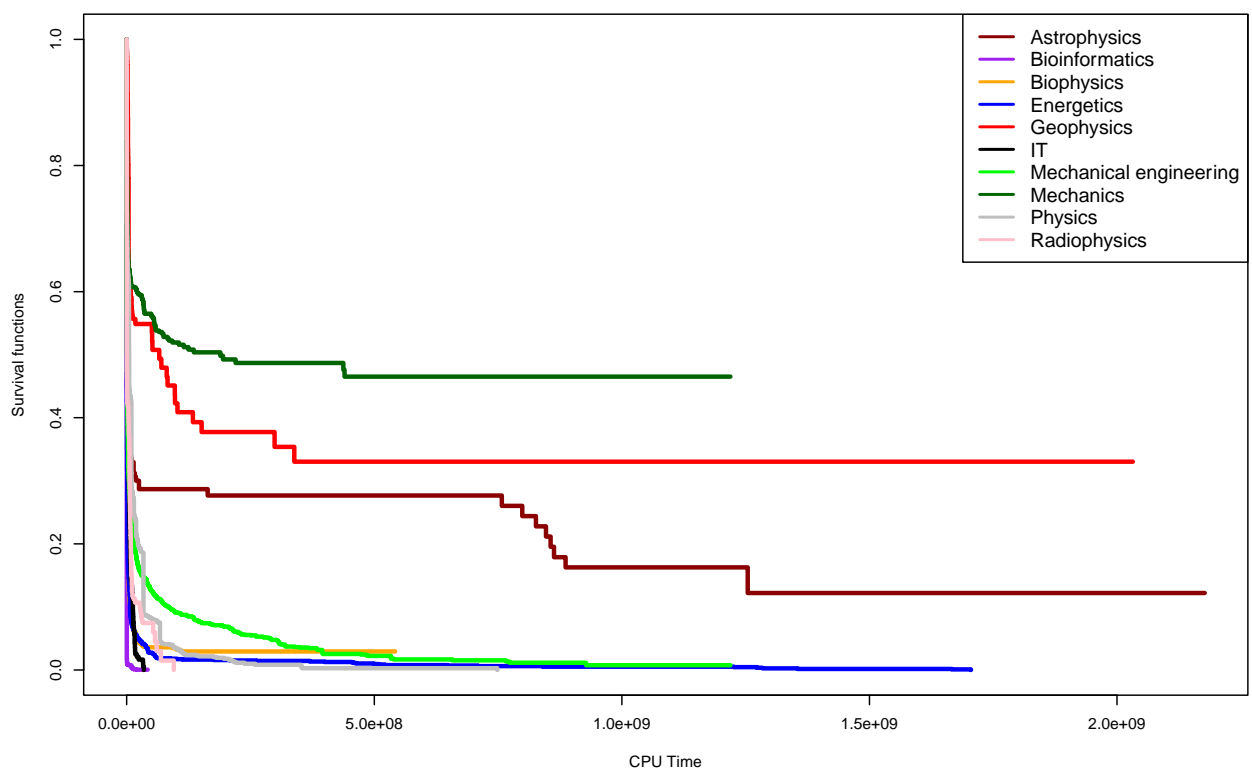


Рисунок 4.5 Оценки Каплана-Мейера распределений процессорного времени исполнения задач в различных областях знаний с детализацией: (а) область процессорного времени, не превышающих $1 \cdot 10^8$ сек.*чцп; (б) не превышающих $1 \cdot 10^7$; (с) не превышающих $1 \cdot 10^6$; (д) не превышающих $1 \cdot 10^5$

Таблица 4.3 Оцененные квартили распределений времен исполнения задач и границы частных доверительных интервалов уровня доверия 95%

Область знаний	Всего запусков	1 квартиль			Медиана			3 квартиль		
		Q25%	Нижняя граница	Верхняя граница	Q50%	Нижняя граница	Верхняя граница	Q75%	Нижняя граница	Верхняя граница
Астрофизика	307	178	162	197	414	342	3144	172134	123877	497767
Биоинформатика	18201	225	223	229	419	405	424	616	614	618
Биофизика	5502	668	557	838	24944	22427	28069	49449	48471	50606
Энергетика	4048	157	152	164	486	457	561	5467	5378	5777
Геофизика	1597	39249	38888	39396	-	-	-	-	-	-
Инф. технологии	1018	799	702	954	9571	7290	13521	51596	48285	59117
Механическая инженерия	5693	82	72	97	4117	3452	4995	35293	29688	38584
Механика	1812	13387	12077	14582	1122188	347034	-	-	-	-
Физика	12312	1356	1143	1606	85220	85193	85236	170167	153091	170227
Радиофизика	382	109	101	155	2969	1271	13042	100281	94058	115560

Таблица 4.4 Оцененные квартили распределений процессорных времен исполнения задач и границы частных доверительных интервалов уровня доверия 95%

Область знаний	Всего запусков	1 квартиль			Медиана			3 квартиль		
		Q 25%	Нижняя граница	Верхняя граница	Q 50%	Нижняя граница	Верхняя граница	Q 75%	Нижняя граница	Верхняя граница
Астрофизика	307	14784	11480	18144	49784	31136	938560	798948360	18576880	-
Биоинформатика	18201	12600	12488	12824	23464	22680	23744	34496	34384	34608
Биофизика	5502	37912	31360	47152	1428616	1278872	1589056	2793616	2729776	2859752
Энергетика	4048	32480	32032	33152	92512	89824	94976	1245888	1211168	1307264
Геофизика	1597	3761088	3710688	3779616	65729664	50668688	102560640	-	338453584	-
Инф. технологии	1018	63448	45920	87696	1502256	1132208	1941912	5586616	4490192	6457752
Механическая инженерия	5693	5488	4592	6664	589736	489720	700672	7663824	6973680	8964984
Механика	1812	1346352	964768	1596896	189107520	74159680	-	-	-	-
Физика	12312	148848	113568	194488	4796736	4795000	4798360	14599648	14592256	17754240
Радиофизика	382	6104	5656	8680	175840	75432	730352	6885424	6134184	8140104

Таблица 4.5 Оцененные распределения длительности необходимого времени исполнения задач в различных областях знаний, построенные на базе оценок Каплана-Мейера.

Область знаний	Всего запусков	Необходимое время выполнения задачи					
		От 5 секунд до 3 минут	От 3 до 10 минут	От 10 минут до 1 часа	От 1 часа до 1 суток	От 1 до 3 суток	Более 3 суток
Астрофизика	307	0.2767	0.2497	$5.83 \cdot 10^{-2}$	$8.81 \cdot 10^{-2}$	$9.13 \cdot 10^{-2}$	0.236
Биоинформатика	18201	0.18858	0.5341	0.2591	$1.12 \cdot 10^{-2}$	$6.95 \cdot 10^{-3}$	$2.32 \cdot 10^{-4}$
Биофизика	5502	0.1376	0.1069	0.1137	0.5261	$6 \cdot 10^{-2}$	$5.57 \cdot 10^{-2}$
Энергетика	4048	0.2960	0.2282	0.1183	0.3147	$3.33 \cdot 10^{-2}$	$9.55 \cdot 10^{-3}$
Геофизика	1597	$9.64 \cdot 10^{-3}$	$1.34 \cdot 10^{-3}$	$5.59 \cdot 10^{-3}$	0.3536	$5.93 \cdot 10^{-2}$	0.5705
Инф. технологии	1018	0.1467	$5.44 \cdot 10^{-2}$	0.2103	0.4723	0.1109	$5.34 \cdot 10^{-3}$
Механическая инженерия	5693	0.2982	$7.76 \cdot 10^{-2}$	0.1147	0.3578	$7.54 \cdot 10^{-2}$	$7.62 \cdot 10^{-2}$
Механика	1812	$3.65 \cdot 10^{-2}$	$2.25 \cdot 10^{-2}$	$9.38 \cdot 10^{-2}$	0.2578	$5.19 \cdot 10^{-2}$	0.5373
Физика	12312	0.1143	$8.1 \cdot 10^{-2}$	0.1022	0.3966	0.1301	0.1758
Радиофизика	382	0.3579	$5.55 \cdot 10^{-2}$	$9.73 \cdot 10^{-2}$	0.1868	0.2956	$6.98 \cdot 10^{-3}$

Таблица 4.6 Оцененные распределения длительности необходимого процессорного времени исполнения задач в различных областях знаний, построенные на базе оценок Каплана-Мейера

Область знаний	Всего запусков	Необходимое время выполнения задачи					
		От 5 секунд до 3 минут (*56 ЦП)	От 3 до 10 минут (*56 ЦП)	От 10 минут до 1 часа (*56 ЦП)	От 1 часа до 1 суток (*56 ЦП)	От 1 до 3 суток (*56 ЦП)	Более 3 суток (*56 ЦП)
Астрофизика	307	0.1788	0.2783	$8.47 \cdot 10^{-2}$	$9.32 \cdot 10^{-2}$	$5.27 \cdot 10^{-2}$	0.3122
Биоинформатика	18201	0.1882	0.5342	0.259	$1.07 \cdot 10^{-2}$	$6.36 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$
Биофизика	5502	0.1372	0.1058	0.1145	0.5223	$5.89 \cdot 10^{-2}$	$6.12 \cdot 10^{-2}$
Энергетика	4048	$1.81 \cdot 10^{-2}$	0.2594	0.281	0.3041	$7.44 \cdot 10^{-2}$	$6.31 \cdot 10^{-2}$

Продолжение таблицы 4.6

Область знаний	Всего запусков	Необходимое время выполнения задачи					
		От 5 секунд до 3 минут (*56 ЦП)	От 3 до 10 минут (*56 ЦП)	От 10 минут до 1 часа (*56 ЦП)	От 1 часа до 1 суток (*56 ЦП)	От 1 до 3 суток (*56 ЦП)	Более 3 суток (*56 ЦП)
Геофизика	1597	$9.68 \cdot 10^{-3}$	$6.74 \cdot 10^{-4}$	$3.45 \cdot 10^{-3}$	0.3716	$5.79 \cdot 10^{-2}$	0.5567
Инф. технологии	1018	0.1362	$4.86 \cdot 10^{-2}$	0.1698	0.3668	0.2111	$6.74 \cdot 10^{-2}$
Механическая инженерия	5693	0.2806	$5.5 \cdot 10^{-2}$	$8.34 \cdot 10^{-2}$	0.2854	$9.47 \cdot 10^{-2}$	0.2008
Механика	1812	$1.9 \cdot 10^{-2}$	$1.99 \cdot 10^{-2}$	0.1013	0.2246	$2.78 \cdot 10^{-2}$	0.6073
Физика	12312	$9.62 \cdot 10^{-2}$	$6.38 \cdot 10^{-2}$	0.1012	0.2851	0.1847	0.2689
Радиофизика	382	0.3526	$5.55 \cdot 10^{-2}$	0.103	0.1229	0.2534	0.1125

4.3.5 Статистические выводы.

Проведенный разведочный анализ показал высокую эффективность использования суперкомпьютера при использовании «Geovation» для автоматического запуска задач, но достаточно низкую эффективность в случае формирования задания пользователем вручную. Число задач, запущенных с использованием «Geovation», составляет 88.5%, тогда как доля процессорного времени (времени), затраченного на решение задач, запущенных с использованием «Geovation», составляет лишь 15.9% (33.6%) от общего затраченного процессорного времени. Особо низкая эффективность наблюдается при запуске задач в областях механики, геофизики и астрофизики, для которых характерны наиболее длительные времена исполнения задач. В областях энергетики, биоинформатики и информационных технологий, для которых характерны более короткие необходимые времена исполнения задач, эффективность использования суперкомпьютера выше, хотя все равно уступает эффективности использования суперкомпьютера при решении задач, запущенных с использованием «Geovation».

4.4 Разработка модели предсказания времени выполнения задач

4.4.1 Постановка задачи

Задачи на суперкомпьютере под управлением системы диспетчеризации Slurm запускаются пользователями на исполнение с указанием ими планируемого времени выполнения задач. На основании указанного пользователями времени выполнения задач Slurm строит расписание выполнения задач.

Фактическое время выполнения задач зависит от множества факторов и всегда отличается от планируемого. Если задача выполняется ранее планируемого времени, ресурсы кластера становятся доступными для выполнения других задач, и Slurm перепланирует расписание таким образом, чтобы последующие задачи раньше поступили на расчёт. С другой стороны, если выполнение задачи не завершается в запланированный пользователем срок, Slurm снимает задачу с вычисления, не дожидаясь её завершения, чтобы освободить ресурсы кластера для выполнения последующих задач в запланированное время. Это приводит к тому, что пользователи значительно завышают прогнозируемое время выполнения задач при постановке их на выполнение, чтобы гарантировать их успешное завершение. Расписание выполнения задач таким образом становится неоптимальным.

Оптимизация расписания выполнения задач на суперкомпьютере под управлением Slurm таким образом может быть достигнута при нахождении способа более точно предсказывать реальное время выполнения задач на этапе постановки их на выполнение. Прогнозируемое пользователем время при этом может быть заменено расчётным, которое, как правило, будет отличаться от заданного пользователем в меньшую сторону, за счёт чего расписание будет более оптимальным.

Интеграцию решения в существующую логику отправки задач на выполнение на суперкомпьютере реализуется следующим образом: при постановке задачи в очередь пользователем программа вычислит прогнозируемое время выполнения и заменит время, указанное

пользователем. Таким образом, система работает без вмешательства пользователя.

Вычисление расчётного времени выполнения задач должно производиться на основе накопленной на суперкомпьютере статистики выполнения ранее обработанных задач. В основе метода - поиск корреляции между параметрами задачи, известными до момента запуска её на выполнение, и фактическим временем её выполнения. Эти параметры могут как указываться пользователем, так и рассчитываться Slurm автоматически. Большую часть составляют автоматически рассчитанные параметры. Они должны использоваться для расчёта прогнозируемого времени выполнения задачи, чтобы не полагаться на решение пользователя.

Предсказание времени выполнения задач должно осуществляться с использованием накопленной статистики с применением методов машинного обучения.

4.4.2 Описание решения

Решение представляет собой программное обеспечение, реализующее логику предсказания времени выполнения задач на суперкомпьютере с применением методов машинного обучения. ПО реализовано на языке программирования Python с применением внешней библиотеки *scikit-learn*, которая позволяет работать с множеством гибко настраиваемых моделей машинного обучения. Также применяется библиотека *pandas* для загрузки статистики из файлов CSV и облегчения обработки табличных данных большого объёма при подготовке данных обучения. С полным списком библиотек можно ознакомиться в соответствующем разделе.

4.5.2.1. Используемые библиотеки

NumPy – библиотека для работы с массивами данных, предоставляет эффективную реализацию многомерных массивов и математических операций над ними.

Pandas – библиотека для создания и анализа высокоэффективных неоднородных структур данных. Значительно упрощает и ускоряет обработку, чтение и визуализацию данных.

Scikit-learn – библиотека, содержащая основные алгоритмы машинного обучения и интеллектуального анализа. Предоставляет интерфейс к их использованию настройке.

Matplotlib – библиотека, предоставляющая пользовательский интерфейс для построения и встраивания многомерных графиков в приложения. Используется для визуализации статистики и результатов предсказания модели в виде 2D-фигур.

Scipy – библиотека, содержащая модули эффективных математических процедур, таких как оптимизация, интерполяция, линейная алгебра и статистика. Основывается на библиотеке *NumPy*.

Pickle – библиотека, реализующая алгоритм сериализации и десериализации объектов языка Python. Используется для передачи обученной модели машинного обучения из модуля настройки и обучения в модуль предсказания.

4.4.3 Подробный алгоритм реализации

Реализация алгоритма состоит из трех скриптов-модулей:

- модуль импорта и обработки данных *datapreprocess.py*;
- модуль настройки и обучения модели предсказания *main.py*;
- модуль использования модели предсказания *timepredict.py*.

4.4.3.1. Модуль импорта и обработки данных

В качестве источника данных для построения входных данных обучения принимается собранная статистика выполнения задач на вычислительных кластерах суперкомпьютера СКЦ Санкт-Петербургского политехнического университета Петра Великого (СПбПУ).

Статистика вычислений задач собирается путём выполнения на суперкомпьютере следующих консольных команд планировщика Slurm:

– Команда *scontrol* позволяет получить конфигурацию задачи, которую задал пользователь при постановке ее в очередь на вычисление.

– Команда *sacct* предоставляет доступ к базе данных логирования результатов вычислений, проведенных под контролем планировщика Slurm.

На текущий момент была накоплена статистика с 19.06.2022 по 17.11.2022:

– *Scontrol*: 448 425 записей, 62 атрибута, объем данных - 296 МБ;

– *Sacct*: 547 558 записей, 108 атрибутов, объем данных - 913 МБ.

Данная статистика, сохраняется в формате *csv* при запросе с сервера СКЦ. нуждается в факторном анализе для очистки данных от наименее полезных для обучения атрибутов. По этой причине перед началом работы было проведено исследование задач по времени исполнения (реального и процессорного) в совокупности с анализом выживаемости задач, соответствующие графики представлены на рисунках 4.6 и 4.7, а также их отраслевой принадлежности на рис. 4.8.

Распределение задач по интервалам реального (Real) времени

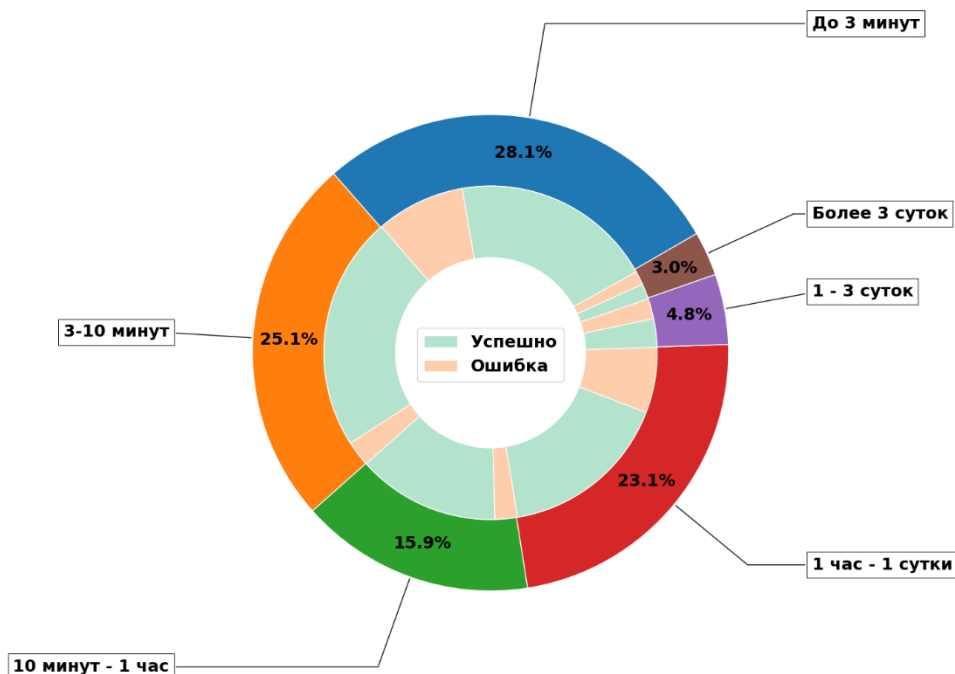


Рисунок 4.6 Распределение задач по интервалам реального времени. Каждый сектор круговой диаграммы разбит на два подсектора, показывающих доли успешно завершенных задач

Распределение задач по интервалам процессорного (CPU) времени

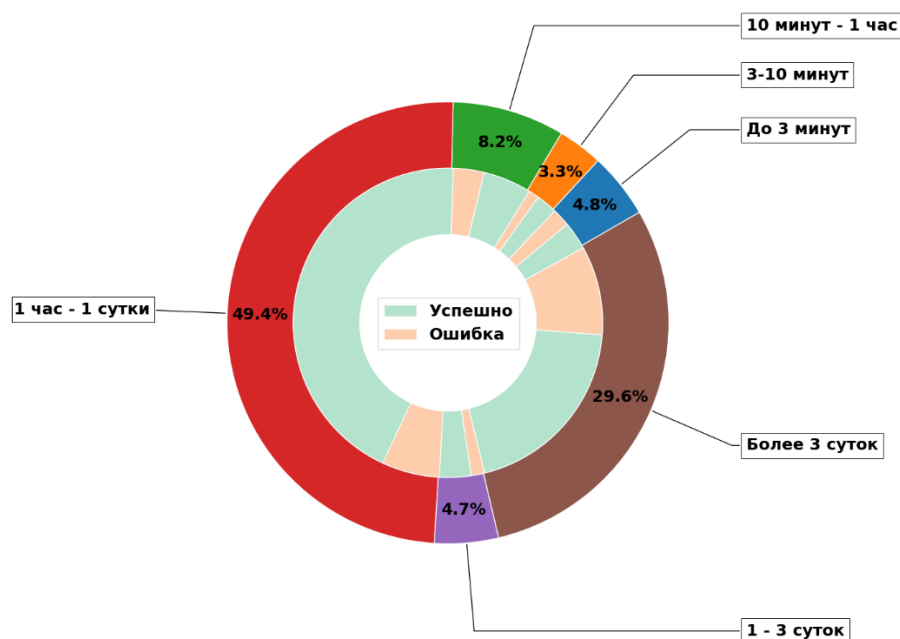


Рисунок 4.7 Распределение задач по интервалам процессорного времени. Каждый сектор круговой диаграммы разбит на два подсектора, показывающих доли успешно завершенных задач

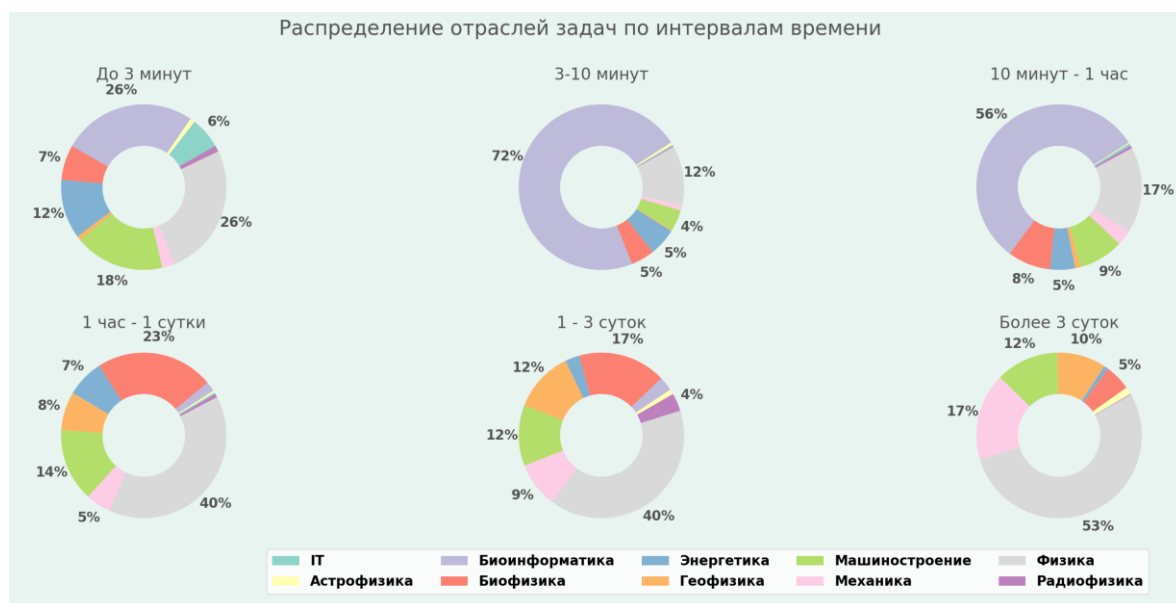


Рисунок 4.8 Распределение отраслей задач по интервалам времени

Также проведен анализ распределения задач по пользователям, составлены суррогатные модели пользователей, часть которых представлена на рис. 4.9.

UserID	Total	Completed	Failed	Cancelled	Timeout	Other	CompletedRate	FailedRate	CancelledRate	TimeoutRate	OtherRate	MeanTime
50121	27	2	25	0	0	0	0,07	0,93	0,00	0,00	0,00	1:26:17
50109	499	422	56	14	0	7	0,85	0,11	0,03	0,00	0,01	29:8:22
50430	789	687	93	9	0	0	0,87	0,12	0,01	0,00	0,00	3:37:34
50523	504	396	71	30	0	7	0,79	0,14	0,06	0,00	0,01	9:17:51
50100	38	27	0	11	0	0	0,71	0,00	0,29	0,00	0,00	21:16:47
50189	4	4	0	0	0	0	1,00	0,00	0,00	0,00	0,00	0:6:10
50190	241	164	52	6	15	4	0,68	0,22	0,02	0,06	0,02	11:40:8
50631	31	30	1	0	0	0	0,97	0,03	0,00	0,00	0,00	2:1:11
50703	5	3	2	0	0	0	0,60	0,40	0,00	0,00	0,00	0:0:2
50671	2	0	1	1	0	0	0,00	0,50	0,50	0,00	0,00	0:0:26
50119	81	17	23	40	0	1	0,21	0,28	0,49	0,00	0,01	47:41:11
50656	28	12	7	9	0	0	0,43	0,25	0,32	0,00	0,00	22:28:56
50672	2	2	0	0	0	0	1,00	0,00	0,00	0,00	0,00	0:1:0
50549	32731	32692	1	21	2	15	1,00	0,00	0,00	0,00	0,00	1:9:26
50594	147	94	20	30	0	3	0,64	0,14	0,20	0,00	0,02	7:16:31
50700	3672	1871	710	1055	36	0	0,51	0,19	0,29	0,01	0,00	0:41:18
50654	26	19	2	3	0	2	0,73	0,08	0,12	0,00	0,08	71:8:23
50262	11	11	0	0	0	0	1,00	0,00	0,00	0,00	0,00	7:38:29
50262	6	2	3	1	0	0	0,33	0,50	0,17	0,00	0,00	0:53:17
50369	52	33	19	0	0	0	0,63	0,37	0,00	0,00	0,00	0:39:35
50555	49	15	7	16	8	3	0,31	0,14	0,33	0,16	0,06	16:1:55
50123	61	0	40	17	0	4	0,00	0,66	0,28	0,00	0,07	66:24:51
50628	9	6	3	0	0	0	0,67	0,33	0,00	0,00	0,00	22:24:11
50614	55	55	0	0	0	0	1,00	0,00	0,00	0,00	0,00	0:2:33
50502	15	11	0	4	0	0	0,73	0,00	0,27	0,00	0,00	0:36:14
50701	28895	28854	3	37	1	0	1,00	0,00	0,00	0,00	0,00	0:6:31
50147	58	15	41	2	0	0	0,26	0,71	0,03	0,00	0,00	2:52:49
50539	81	63	10	7	0	1	0,78	0,12	0,09	0,00	0,01	11:10:15
50665	38	21	6	10	0	1	0,55	0,16	0,26	0,00	0,03	13:31:44
50626	1	1	0	0	0	0	1,00	0,00	0,00	0,00	0,00	43:55:16
50501	232	29	152	37	4	10	0,13	0,66	0,16	0,02	0,04	16:37:44
50600	63	46	14	3	0	0	0,73	0,22	0,05	0,00	0,00	3:35:7
50174	7	3	2	1	0	1	0,43	0,29	0,14	0,00	0,14	13:14:30
50687	1112	1071	17	23	0	1	0,96	0,02	0,02	0,00	0,00	0:52:0
50619	122756	122271	9	474	0	2	1,00	0,00	0,00	0,00	0,00	0:27:10
50696	1	1	0	0	0	0	1,00	0,00	0,00	0,00	0,00	5:33:30

Рисунок 4.9 Суррогатные модели пользователей, статистика распределения задач по пользователям

4.4.3.1.1. Важность атрибутов обучения

Для создания качественной модели машинного обучения необходимо понимание данных, а именно какие из множества атрибутов могут быть наиболее релевантными для предсказания времени выполнения задач.

Расчет коэффициентов важности дает представление о конкретной модели и о том, какие столбцы являются наиболее и наименее важными при прогнозировании. Также используя оценки важности, можно выбрать те атрибуты, которые нужно удалить (обладателей самых низких коэффициентов). Это может упростить моделирование, ускорить процесс обучения, проведения экспериментов и, в некоторых случаях, повысить производительность модели. Эта концепция называется важностью признаков (feature importance).

Таким образом, проведен ряд экспериментов выявления важности признаков с помощью метода *feature permutation*, который заключается в проведении серии случайных переупорядочиваний и взаимоисключений различных атрибутов обучения. С его использованием удалось уменьшить размерность данных, путем отбрасывания из обучающей выборки атрибутов, имеющих коэффициент важности, приближенный к нулю. На рисунке 4.10 представлен график, визуализирующий распределение важности атрибутов после отсеивания малозначимых атрибутов.

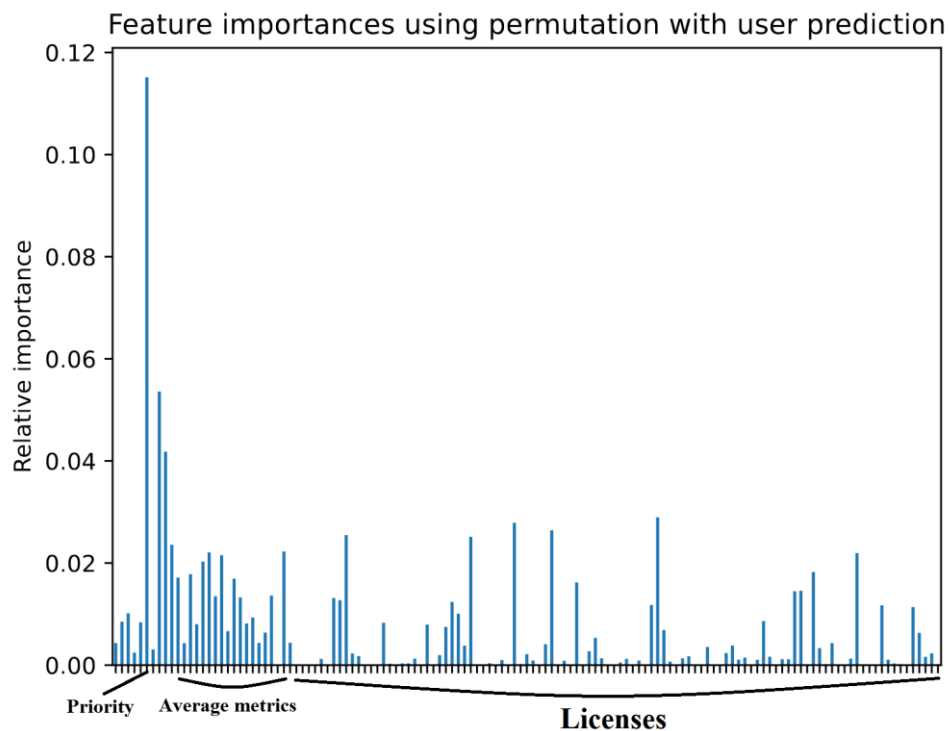


Рисунок 4.10 Распределение важности атрибутов после предобработки исходных данных.

Далее представлен список атрибутов, используемых в текущей реализации программы после вычисления важности атрибутов.

Атрибуты, используемые для машинного обучения по собранной по статистике *sacct* – данные из заявки пользователя и планировщика Slurm при постановке задач в очередь:

- 'UserID' - Идентификатор пользователя, от имени которого была запущена задача.

- 'GroupID' - Идентификатор группы, от имени которой задача была поставлена в очередь.
- 'NumNodes' - Число узлов, запрошенных или выделенных на задачу или шаг задачи.
- 'NCPUs' - Общее число процессоров, выделенных задаче.
- 'NumTasks' - Общее число подзаданий в задаче или ее шаге.
- 'CPUs/Task' - Соотношение общего числа процессоров к числу подзадач.
- 'ReqB:S:C:T' - Определяет число различных компонент аппаратных средств, запрошенных для задачи.
- 'Socks/Node' - Число желаемого пользователем отношения количества сокетов к числу узлов вычислителя.
- 'NtasksPerN:B:S:C' - Определяет число подзадач, необходимых к запуску на определенном числе компонент аппаратных средств (узлы, материнские платы, сокет и ядра).
- 'CoreSpec' - Число ядер, которые пользователь зарезервировал на узле для системных нужд.
- 'MinCPUsNode' - Минимальное отношение числа процессоров к количеству узлов.
- 'MinMemoryNode' - Минимальное отношение количества оперативной памяти в МБ к числу узлов.
- 'MinTmpDiskNode' - Минимальное отношение временного дискового пространства в МБ к числу узлов.
- 'JobID' - Идентификационный номер задачи или шага задачи.
- 'Priority' - Приоритет задачи, определенный планировщиком Slurm.
- 'Licenses' - Список лицензий для программного обеспечения, используемого при вычислении задачи. Указывается пользователем при постановке задачи в очередь.

Атрибуты, используемые для машинного обучения по собранной по статистике *scontrol* – данные из базы данных логирования планировщика Slurm после окончания вычисления задач:

- 'TimelimitRaw' – Ограничение по времени для вычисления задачи, заданное пользователем в минутах.
- 'ReqTRES' - Минимальное количество ресурсов, запрошенное задачей в момент начала вычисления.
- 'ElapsedRaw' - Реальное время, за которое вычислялась задача, в секундах.
- 'State' - Статус задачи после окончания вычисления.
- 'CPUTimeRAW' - Использованное процессорное время (реальное время, умноженное на число процессоров), потребленное задачей, в процессоро-секундах.
- 'AveCPU' - Среднее (системное + пользовательское) процессорное время среди всех подзадач внутри задачи.
- 'AveCPUFreq' - Средняя взвешенная частота процессора по всем подзадачам внутри задачи в КГц.
- 'AveDiskRead' - Среднее количество байтов, прочитанных задачей во время вычисления.
- 'AveDiskWrite' - Среднее количество байтов, записанных задачей во время вычисления.
- 'AvePages' - Среднее число отказов страницы (page fault) по всем подзадачам внутри задачи.
- 'AveRSS' - Средний размер страниц памяти, выделенных задаче и в настоящее время находящихся в ОЗУ по всем подзадачам внутри задачи.
- 'AveVMSize' - Средний объем виртуальной памяти по всем подзадачам внутри задачи.
- 'MaxDiskRead' - Максимальное количество байтов, прочитанных задачей во время вычисления.

– 'MaxDiskWrite' - Максимальное количество байтов, записанных задачей во время вычисления.

4.4.3.1.2. Параметр отброса неудачных задач

Была проведена серия экспериментов с различными наборами входных данных и фильтрацией обучающей выборки. В ее ходе было выявлено, что при исключении неуспешно завершенных задач из данных, в некоторых случаях, предсказание осуществляется с повышенной точностью, т.к. такие данные можно считать выбросами (outliers). Пользователи могли ошибаться в логике работы своих вычислений или неверно задавать заявку на постановку задачи в очередь, поэтому при отбросе таких задач уменьшается зашумленность выборки, что может благоприятно повлиять на качество предсказания.

Параметр *drop_failed_jobs* позволяет регулировать фильтрацию неудачных запусков вычислений. Стоит заметить, что если в выборке преобладают задачи, завершенные некорректно по причинам, не связанным с ошибкой пользователя, а, например, вследствие политики работы планировщика, то применение данного параметра может наоборот ухудшить точность предсказания. На рисунке 4.11 представлен график, показывающий соотношение результатов вычислений по некоторым пользователям суперкомпьютера. Синим цветом обозначена доля успешно завершенных задач, красным – завершение задачи с ошибкой в коде программы, желтым – задача отменена пользователем, зеленым – задача снята с вычисления планировщиком задач, оранжевым – другие причины. Как можно заметить, доля задач с ошибкой значительна, что может плохо повлиять на корректность предсказания.

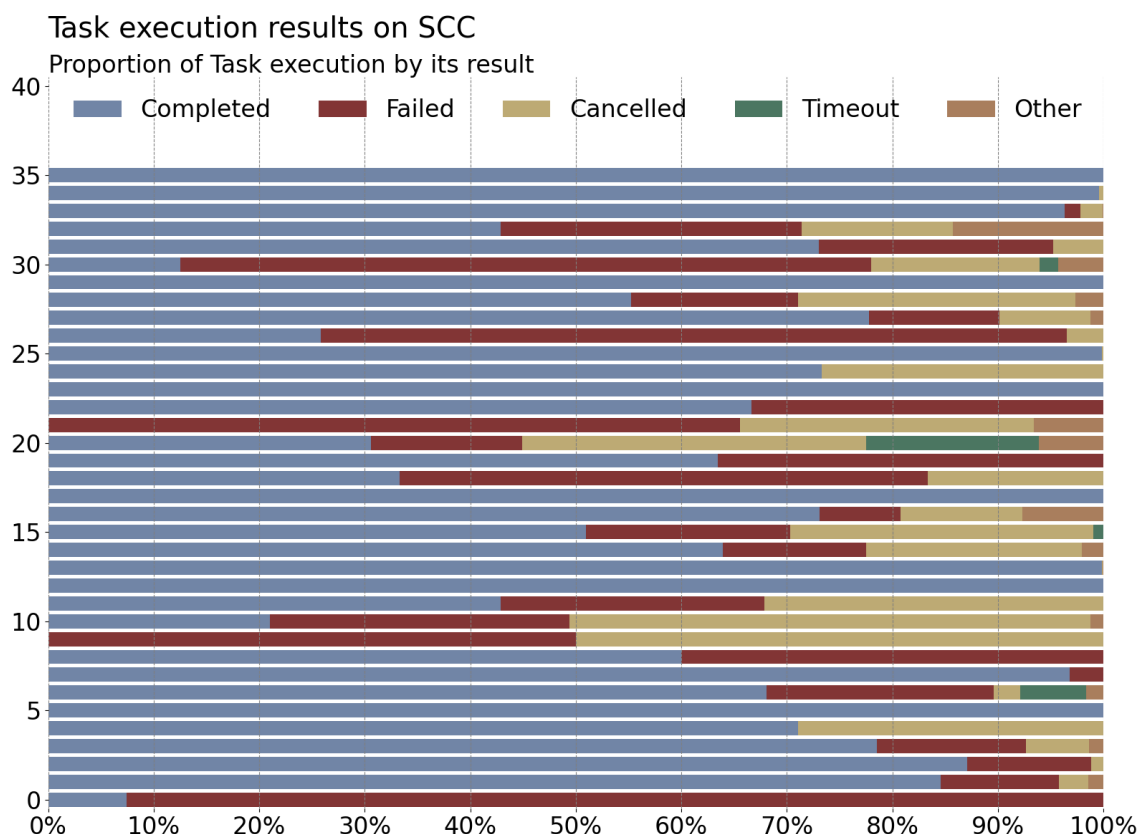


Рисунок 4.11 Распределение результатов вычислений по пользователям. По вертикали отмечены различные пользователи с долями, соответствующими исходам вычислений.

4.4.3.1.3. Число классов

Параметр *class_count* определяет число временных интервалов, на которые будет разбит весь возможный диапазон времени выполнения задач. Границы диапазонов выбираются так, чтобы в каждый класс попало равномерное число задач из обучающей выборки. Данный параметр лучше всего подбирать экспериментально, в зависимости от точности предсказания. При слишком малом числе классов, повысится процент попадания в верный временной диапазон, однако он может быть неприемлемо широким, тем самым точность предсказания будет невысока. С другой стороны, если указать слишком большое количество классов для разбиения, то это уменьшит шанс точного попадания в верный интервал, что также плохо влияет на качество

предсказания, ведь помимо этого модель будет страдать от переобучения (overfitting).

4.4.3.1.4. Учет лицензий

Чтобы качественно оценить задачу, а именно, определить род вычислений, присущие ей алгоритмы, при обработке данных особое внимание уделяется спискам лицензий. Для этого формируется список всех известных серверов лицензий, к которым обращаются задачи перед вычислением. Далее из этого списка формируется таблица меток, определяющих к каким серверам лицензий обращается конкретная задача. Получение данной таблицы называется *one-hot encoding* – процесс, с помощью которого категориальные атрибуты преобразуются в подходящую алгоритмам машинного обучения форму. Полученная таблица присоединяется к уже существующим атрибутам задачи, и таким образом, формируются новые признаки, позволяющие определить основные алгоритмы или ПО, используемое при вычислении.

4.4.3.1.5 Атрибуты обогащения данных

В процессе подготовки данных для машинного обучения, помимо имеющихся изначально атрибутов, включаются новые искусственные атрибуты, которые задают дополнительную функциональную зависимость, между известными данными и целевой переменной предсказания – временем исполнения задачи. Эти дополнительные атрибуты помогают алгоритмам машинного обучения построить дополнительные связи для повышения точности предсказания. Далее приведены некоторые атрибуты обогащения, добавляемые при обработке данных.

Коэффициент средней утилизации ресурсов для пользователя:

- Для каждой из задач в выборке, запущенных на расчёт из-под учётной записи пользователя, рассчитывается отношение фактического времени расчёта задачи к планируемому $ElapsedRaw/TimelimitRaw$.

- Вычисляется среднее значение данного отношения среди всех задач в выборке, запущенных на расчёт из-под учётной записи данного пользователя.

– Рассчитанное значение коэффициента добавляется ко всем задачам, запущенным на расчёт из-под учётной записи данного пользователя, в атрибут под именем *UtilFactor*.

Класс размера задачи:

– Определяется минимальное и максимальное значение фактически использованного задачей процессорного времени среди всей выборки *ElapsedRaw*.

– Разделить полученный интервал на n разных диапазонов, соответствующих классу размера задачи, где n является параметром, который задает пользователь при настройке модели предсказания времени.

– Каждой задаче в выборке присваивается рассчитанный номер диапазона в атрибут под именем *SizeClass*.

Доля неудачных задач пользователя:

– Для каждого из пользователей рассчитывается отношение разности общего числа задач, запущенных под учётной записью пользователя и числа его задач в статусе *COMPLETED* к общему числу пользователя. $K = (N_{all} - N_{completed})/N_{all}$.

– Получившийся коэффициент добавляется ко всем задачам пользователя в выборке в атрибут под именем *FailureRate*.

Среднее время выполнения по лицензиям:

– Производится анализ содержимого атрибута *Licenses*, что позволяет установить, какие лицензии были использованы для выполнения задачи.

– Для каждой из лицензий вычисляется среднее время выполнения задач, для которых использовалась та или иная лицензия.

– Для каждой задачи вычисляется среднее время по всем задействованным лицензиям, полученное значение добавляется в выборке в атрибут под именем *LicenseMeanTime*.

Предварительно обработанные данные могут быть загружены из сохраненных файлов *full_data.csv*, *data.csv* и *time_diapasons.csv*, хранимые в

поддиректории *precompiled* для уменьшения времени исполнения и использования вычислительных ресурсов при проведении экспериментов с нахождением оптимальных параметров обучения. Булевый параметр *load_saved_data* позволяет определить использовать ли пользователю предварительно кешированные данные.

4.4.3.2. Модуль настройки и обучения модели предсказания

4.4.3.2.1. Выбор модели машинного обучения

Пользователю предлагается использовать одну из 8 различных методов машинного обучения:

- RandomForestRegressor
- ExtraTreesRegressor
- GradientBoostingRegressor
- HistGradientBoostingRegressor
- RandomForestClassifier
- ExtraTreesClassifier
- GradientBoostingClassifier
- HistGradientBoostingClassifier

Данные методы относятся к группе ансамблей моделей. Так называют методы, которые построены как процедура параллельного или последовательного построения композиции алгоритмов машинного обучения. Комбинация нескольких алгоритмов обучения, работая вместе, позволяют построить модель более эффективную и точную, чем любая из моделей, построенная с помощью отдельного алгоритма.

Для поставленной задачи выбор данной группы методов машинного обучения обусловлен эффективностью работы с большим набором входных данных и усреднением ошибки выходных данных моделей путем их параллельного или последовательного объединения.

Выбранные методы разделяются на две категории: регрессоры и классификаторы. В процессе реализации программы использовалось несколько типов регрессоров для нахождения наиболее подходящего для

обеспечения высокой точности предсказаний. Для каждого из регрессоров выполнялся перебор возможных значений настроек для поиска набора настроек, обеспечивающего максимальную точность предсказаний. Описание поиска оптимальных параметров моделей обучения приведено в отдельном разделе.

RandomForestRegressor/Classifier

Регрессор на основе модели случайного леса. Ряд классификационных деревьев решений обучается на различных подвыборках данных обучения, и используется усреднение для повышения точности предсказания и избежания переобучения.

Преимущества использования RandomForestRegressor:

- проблема переоснащения решается путем усреднения или объединения результатов различных деревьев решений;
- случайные леса лучше работают с большим количеством атрибутов данных, чем одно дерево решений;
- случайные леса имеют меньшую дисперсию, чем одно дерево решений;
- случайные леса обладают гибкостью и высокой точностью;
- алгоритм случайного леса не требует масштабирования данных;
- алгоритм случайного леса универсален по отношению к различным задачам машинного обучения;
- алгоритм случайного леса нечувствителен к монотонному преобразованию признаков;
- алгоритм случайного леса имеет эффективную реализацию на языке Python.

Недостатки использования RandomForestRegressor:

- сложность и ресурсоёмкость алгоритма случайного леса.

Параметры, на которых достигнута максимальная точность предсказания:

- `criterion="poisson"` - критерий оценки качества разделения данных.

- `n_estimators=1000` - число деревьев.
- `max_depth=None` - максимальная глубина дерева.
- `max_features="sqrt"` - максимальное число, используемых атрибутов при каждом разбиении внутри дерева.
- `max_samples=0.4` – максимальная доля обучающей выборки, используемая при каждом разбиении внутри дерева.
- `min_samples_leaf=1` – минимальное число элементов обучающей выборки для каждого узла дерева.
- `min_sample_split=2` – минимальное число элементов обучающей выборки для каждого разбиения внутри дерева.

ExtraTreesRegressor/Classifier

Ряд рандомизированных деревьев решений обучается на различных подвыборках данных обучения, и используется усреднение для повышения точности предсказания и избежания переобучения.

Преимущества использования ExtraTreesRegressor:

- Уменьшение дисперсии модели за счет несколько большего увеличения смещения, по сравнению со случайным лесом;
- Возможно использование ExtraTreesRegressor в случае переобучения на случайном лесе или градиентном бустинге

Недостатки использования ExtraTreesRegressor:

- Возможна потеря точности предсказания, по сравнению с другими моделями, за счёт внедрения элемента случайности.

Параметры, на которых достигнута максимальная точность предсказания:

- `max_depth=None` - максимальная глубина дерева.
- `max_features=None` - максимальное число, используемых атрибутов при каждом разбиении внутри дерева.
- `min_samples_leaf=3` – минимальное число элементов обучающей выборки для каждого узла дерева.

- `min_sample_split=2` – минимальное число элементов обучающей выборки для каждого разбиения внутри дерева.
- `n_estimators=1000` - число деревьев.

GradientBoostingRegressor/Classifier

В основе лежит построение аддитивной модели оптимизации функции потерь: на каждом этапе дерева обучаются на отрицательном градиенте функции потерь.

Преимущества использования GradientBoostingRegressor:

- возможность выбора произвольной функции потерь позволяет учитывать особенности данных обучения;
- гибкость настройки за счёт возможности использования различных базовых алгоритмов;
- возможность алгоритмической и математической оптимизации для ускорения работы.

Недостатки использования GradientBoostingRegressor:

- трудоёмкость и невысокая скорость работы;
- необходимость построения множества базовых алгоритмов для композиции;
- чувствительность к ошибкам и выбросам;
- построение композиции из сложных и мощных алгоритмов занимает много времени без существенного повышения качества;
- результаты работы бустинга могут быть сложно интерпретируемы, особенно если в композицию входят десятки алгоритмов.

Параметры, на которых достигнута максимальная точность предсказания:

- `criterion="squared_error"` - критерий оценки качества разделения данных
- `init=HistGradientBoostingRegressor/Classifier()` – инициализирующая модель.
- `learning_rate=0.1` – скорость обучения модели по итерациям.

- `max_depth=None` - максимальная глубина дерева.
- `max_features="log2"` - максимальное число, используемых атрибутов при каждом разбиении внутри дерева.
- `min_samples_leaf=3` – минимальное число элементов обучающей выборки для каждого узла дерева.
- `min_sample_split=6` – минимальное число элементов обучающей выборки для каждого разбиения внутри дерева.
- `n_estimators=1000` - число деревьев.
- `subsample=0.6` – доля подвыборки, используемая в каждом отдельном дереве.

HistGradientBoostingRegressor/Classifier

Более быстрый вариант алгоритма *GradientBoostingRegressor/Classifier* для промежуточных выборок, или для тестирования работы моделей.

Преимущества использования HistGradientBoostingRegressor:

- Значительное ускорение построения деревьев решений, по сравнению с GradientBoostingRegressor, за счёт дискретизации значений атрибутов в данных обучения.

Недостатки использования HistGradientBoostingRegressor:

- Сложность выбора оптимального числа диапазонов для гистограмм: слишком низкое число снижает точность предсказания, а слишком высокое приводит к деградации производительности.

Параметры, на которых достигнута максимальная точность предсказания:

- `l2-regularization=1e-05` – параметр для L2 регуляризации.
- `learning_rate=0.3` – скорость обучения модели по итерациям.
- `loss="squared_error"` – функция потерь, используемая для корректировки предсказательных способностей модели.
- `max_bins=100` – максимальное число разбиений выборки для построения гистограмм.
- `max_depth=None` – максимальная глубина каждого дерева.

- `max_iter=1000` – максимальное число итераций обучения.
- `max_leaf_nodes=30` – максимальное число листовых узлов в каждом дереве.
- `min_samples_leaf=5` – минимальное число элементов обучающей выборки для каждого узла дерева.
- `scoring = "r2"` – функция оценки предсказательной способности модели на тестовой выборке.
- `tol=0.001` – допустимая ошибка для сравнения полученной точности по функции потерь.

4.4.3.2.2 Поиск оптимальных параметров моделей машинного обучения

Для каждой из предложенных моделей были найдены оптимальные параметры обучения, называемый в литературе «тюнинг» моделей. Поиск параметров, которые позволяют получить лучшие результаты предсказания на тестовой выборке данных, может быть повторно проведен пользователем, если это понадобится.

Нахождение параметров происходит методом кросс-валидации, при котором данные несколько раз разбиваются в фиксированном соотношении на обучающую и тестовую выборку, вычисляя средний результат предсказания модели по поставленной метрике. Путем выполнения кросс-валидации на каждом из возможных наборов параметров, вычисляется метрика, определяющая качество предсказания. Для регрессоров выбрана метрика «R2», показывая отношение дисперсий вектора значений выхода модели с целевым вектором. Для классификации используется метрика «ассурасу», показывающая процент точного совпадения предсказанного вектора с целевым.

Для каждой из моделей построенная многомерная дискретная сетка параметров, содержит в своих узлах выборочные точки значений параметров, отличающихся между собой на различные порядки. Пример сетки поиска оптимальных параметров для модели *HistGradientBoostingRegressor*:

```
'loss': ['squared_error', 'poisson'],
```

'learning_rate': [0.3, 0.1, 0.05, 0.01],
'max_iter': [100, 500, 1000],
'max_leaf_nodes': [10, 30, 50, None],
'max_depth': [3, 4, 5, None],
'min_samples_leaf': [5, 10, 20, 30],
'l2_regularization': [1e-5, 1e-3, 1e-1],
'max_bins': [100, 255],
'tol': [1e-3, 1e-2, 1e-1]

Для каждой из предложенных моделей были построены сетки поиска, найдены оптимальные параметры обучения и загружены в алгоритм с их автоматическим применением.

4.4.3.2.3 Предсказание времени выполнения задачи

После выбора модели машинного обучения и установки оптимальных параметров обучения, происходит дополнительная предобработка данных с учетом индивидуальных предпочтений пользователя алгоритма.

Модель внимания, веса пользователей

Параметр *use_weights* позволяет решить проблему неравномерности распределения числа задач по пользователям, которая приводит к тому, что наиболее частые пользователи вычислительного кластера вытесняют частотой своих вычислений более редких пользователей. Данный параметр предоставляет возможность регулировать значимость отдельных пользователей путем присвоения им в соответствие весовой функции, которая позволяет уделить большее внимание пользователям, имеющих малое число вычислений относительно других.

На данный момент модель внимания использует весовую функцию, обратно пропорциональную частоте появления задач пользователя в обучающей выборке. Применение параметра приводит к значительному росту точности предсказания задач редких пользователей за счет небольшого снижения точности предсказаний по пользователям с многочисленными задачами. Параметр задается булевым значением, True или False.

Учет пользовательской оценки

Параметр *with_user_prediction* определяет, будет ли использоваться при обучении модели ограничение по времени, которые указывали пользователи вычислительного кластера при постановке задач в очередь. Большая часть пользователей указывает неоправданно высокое значение времени, ограничивающее их вычисление. Пользователи используют такой большой запас времени для дополнительной уверенности в том, что их задача досчитает до конца. По этой причине данный атрибут при обучении может как увеличить точность предсказания, так и наоборот уменьшить в зависимости от средней квалификации и качестве оценок пользователей, выполнявших вычисления на рассматриваемом промежутке времени.

Если пользователи вычислительного кластера указывают ограничение по времени в разы, превышающее действительное, то этот параметр не даст значительного роста точности предсказания. Параметр задается булевым значением, True или False.

Скользящее окно

Необходимо учитывать сезонность вычислений, которые пользователи проводят при пользовании суперкомпьютерного центра. Наиболее частая причина, по которой пользователи проводят свои вычисления, – это работа над проектными заданиями. Эмпирические наблюдения показали, что в рамках одного проекта вычисления крайне схожи между собой, что позволяет, ориентируясь на предыдущий опыт пользователя, предсказать в дальнейшем время вычислений последующих задач. Однако при смене проекта или при переходе на другой этап зачастую меняются и сами потребности в вычислениях, их средства или алгоритмы. По этой причине введен параметр *time_window*, позволяющий указать ширину скользящего окна обучающей выборки. Чтобы модель не обучалась на излишне устаревших данных, вычислениях пользователя, которые он проводил в других проектах, можно ограничить это окно до нескольких месяцев, чтобы уменьшить вероятность зашумленности данных обучения.

В будущих версиях модуля предсказания рассматривается добавление функциональности, задающей ширину скользящего окна для каждого отдельного пользователя. Ширина такого окна должна вычисляться автоматически, оптимизируя точность предсказания на тестовой выборке с использованием метода кросс-валидации. Параметр задается в секундах, целое положительное число.

Таймаут завершения задачи

Чтобы сгладить среднюю ошибку предсказания, алгоритм позволяет регулировать таймаут завершения задачи. Параметр *kill_timeout* задает дополнительное время, которое планировщик выделит задаче, чтобы она успела корректно завершить свою работу. Таким образом, указав фиксированное время, можно существенно понизить процент задач, вынужденно снятых планировщиком. Значение этого параметра лучше всего задавать в паре с параметром *proba_quantile*, который будет описан в следующем подразделе. Подбирая пару этих параметров, необходимо найти оптимум между процентом аварийно-завершенных задач и среднего отклонения предсказанного времени от реального. Чем меньше таймаут завершения, тем меньше сгладится ошибка предсказания, и больше задач окажется снятыми с вычисления планировщиком вычислительного кластера, но при этом уменьшится среднее ожидание задач в очереди. Параметр задается в секундах, целое положительное число. На рисунках 4.12 и 4.13 представлены тепловые карты зависимостей пары параметров таймаута завершения задачи и квантили распределения вероятности от доли аварийно-завершенных задач и средней ошибки соответственно.

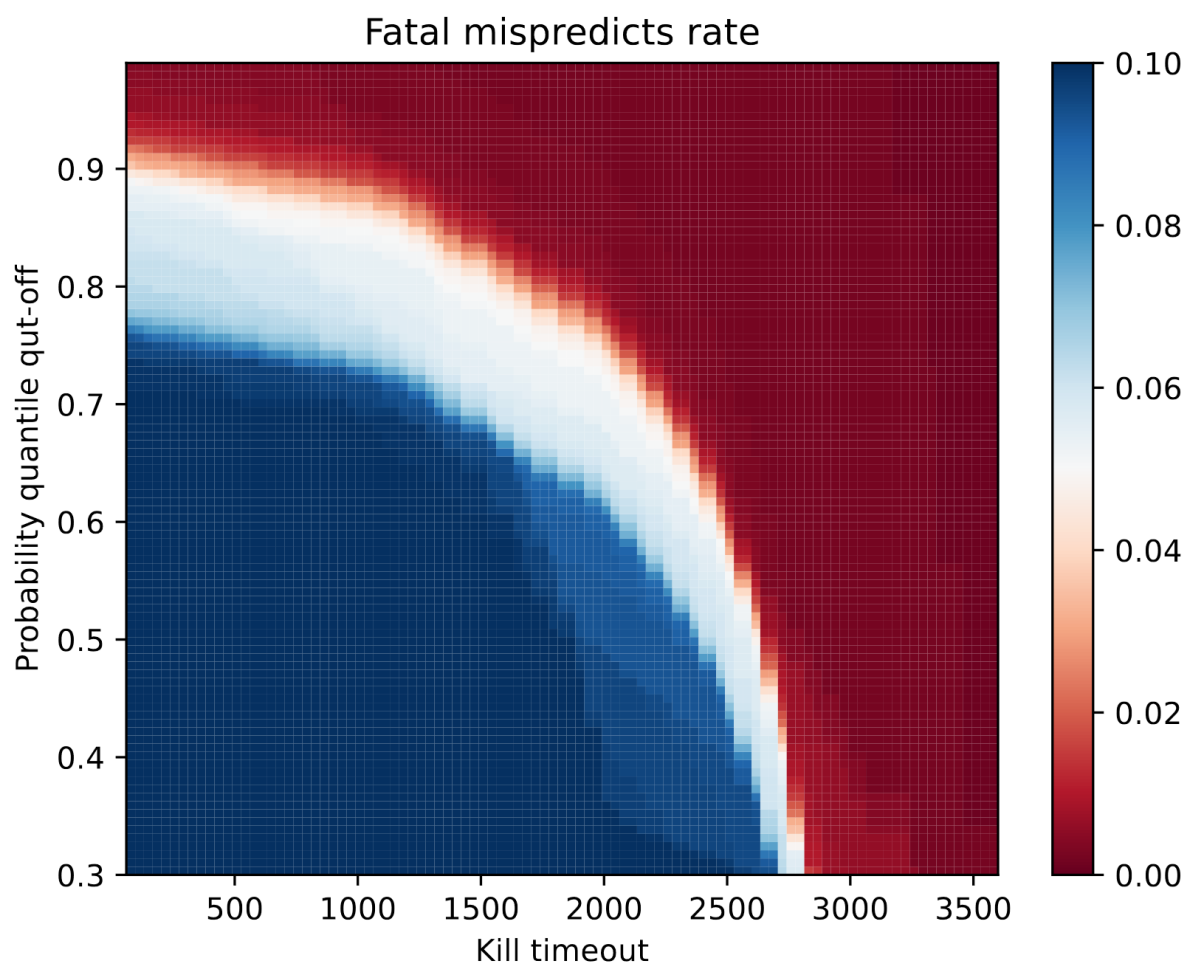


Рисунок 4.12 Тепловая карта (heatmap), показывающая долю аварийно-завершенных задач, в зависимости от пары параметров: квантили распределения вероятности (в долях) по вертикальной оси и таймаута завершения задачи (в секундах) по горизонтальной оси

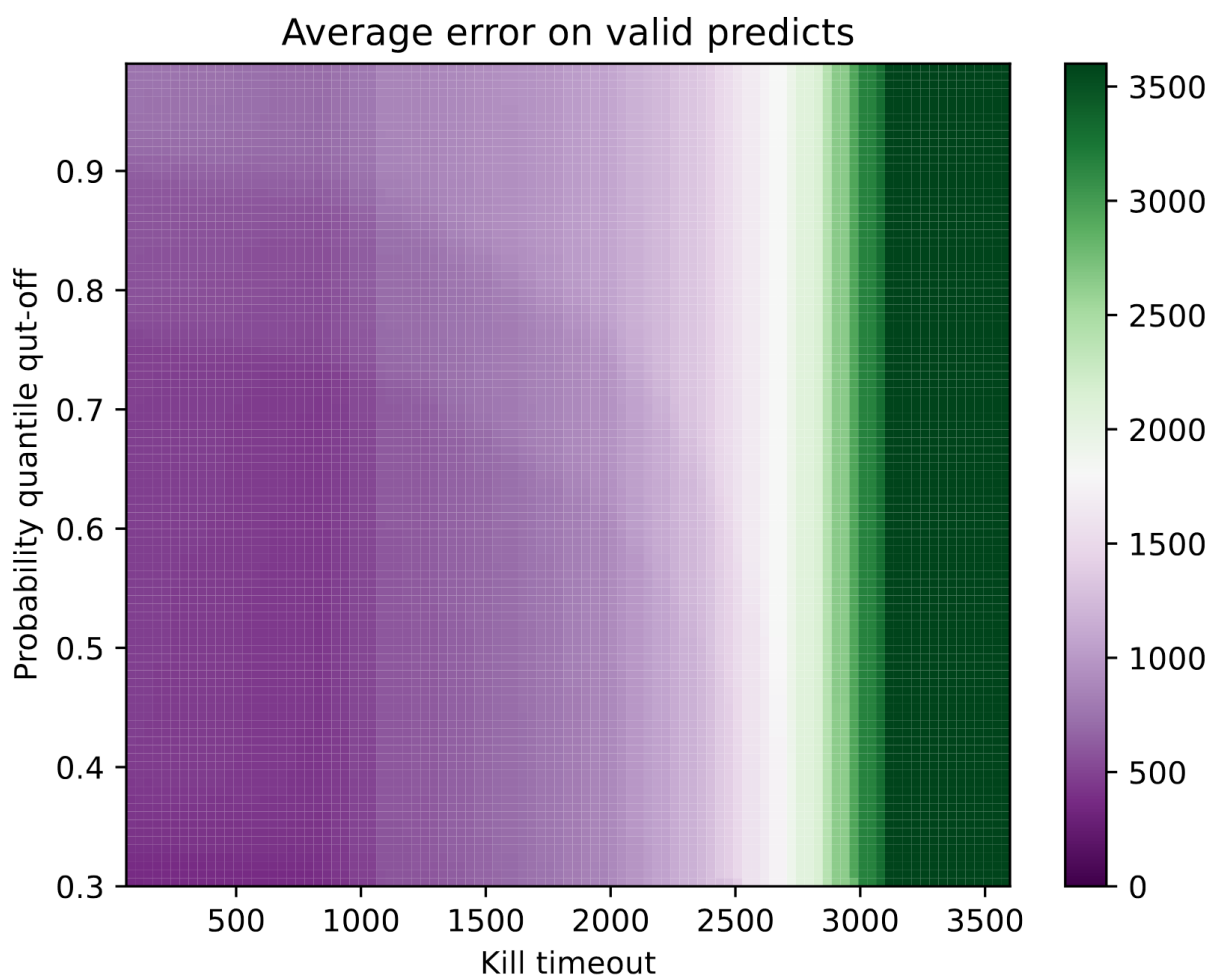


Рисунок 4.13 Тепловая карта (heatmap), показывающая абсолютную среднюю ошибку предсказания (в секундах), в зависимости от пары параметров: квантили распределения вероятности (в долях) по вертикальной оси и таймаута завершения задачи (в секундах) по горизонтальной оси

Квантиль распределения вероятности

Большинство систем, использующих классификационные модели машинного обучения, считают за предсказанный класс, соответствующий наибольшей вероятности. При сборе данных о задачах по логам вычислительного кластера не удастся получить доступ к информации, позволяющей оценить размерность вычислений. Соответственно, различные задачи с похожими значениями статистики вычислений могут по времени выполнения различаться на порядки в зависимости от размерности данных, с которыми работает задача. Чтобы учесть возможный разброс времени исполнения похожих задач при условиях недостатка информации о них,

вместо классического использования математического ожидания, введен параметр квантили распределения вероятности принадлежности задачи к классам по времени. Параметр *proba_quantile* позволяет регулировать уровень нижней квантили, по которой принимается окончательное решение о выборе временного интервала задачи. Чем выше поставить значение квантили, тем меньше будет доля задач, снятых диспетчером. При этом может значительно уменьшиться точность предсказания времени с отклонением в большую сторону. Параметр задается в виде доли, числа с плавающей точкой от 0 до 1.0. График, иллюстрирующий использование данного параметра представлен на рисунке 4.14.

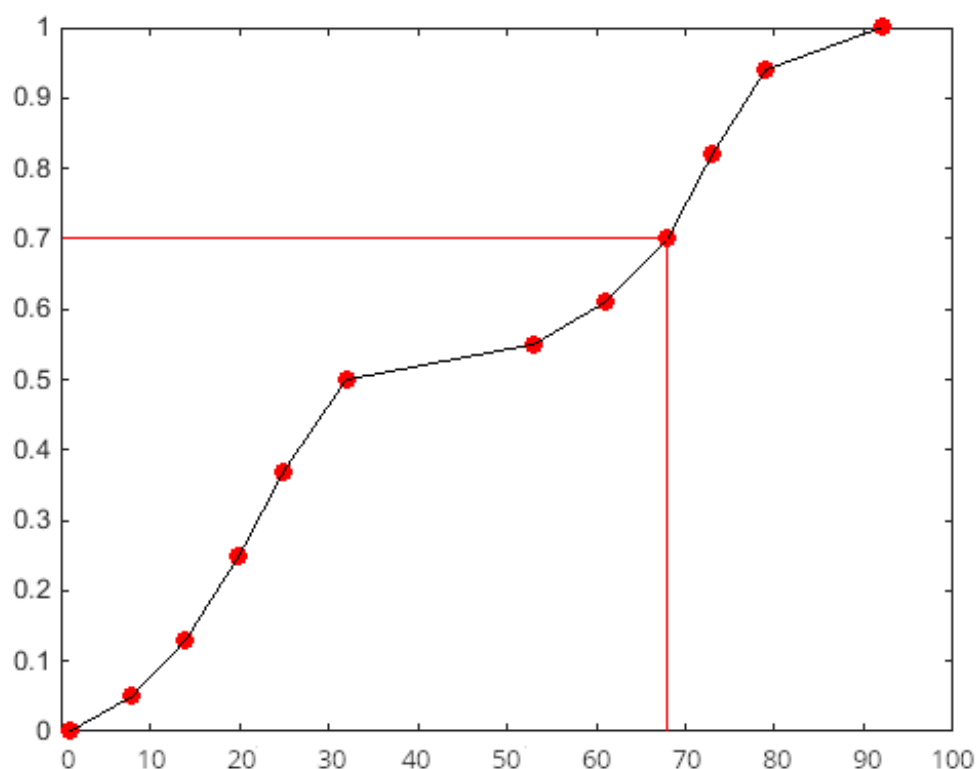


Рисунок 4.14 График функции распределения вероятности принадлежности к классам задач. По горизонтальной оси отмечены классы задач. Красными линиями отмечена нижняя квантиль, равная 0.7, она соответствует 67 классу задачи

4.4.3.3. Модуль использования модели предсказания

После настройки и обучения модели в предыдущем модуле, обученная модель сохраняется в файле *model.pred*, после чего может быть импортирована в модуль использования модели предсказания. Данный модуль предназначен для вызова из консоли с целью мгновенного получения оценки времени выполнения. Для этого необходимо единственным параметром передать строку, содержащую все известные атрибуты на этапе постановки в очередь задачи. Модуль вернет прогнозируемое время исполнения задачи. На рисунке 4.15 представлен график, иллюстрирующий работу предсказателя на 50 последних задачах, запущенных пользователями суперкомпьютера.

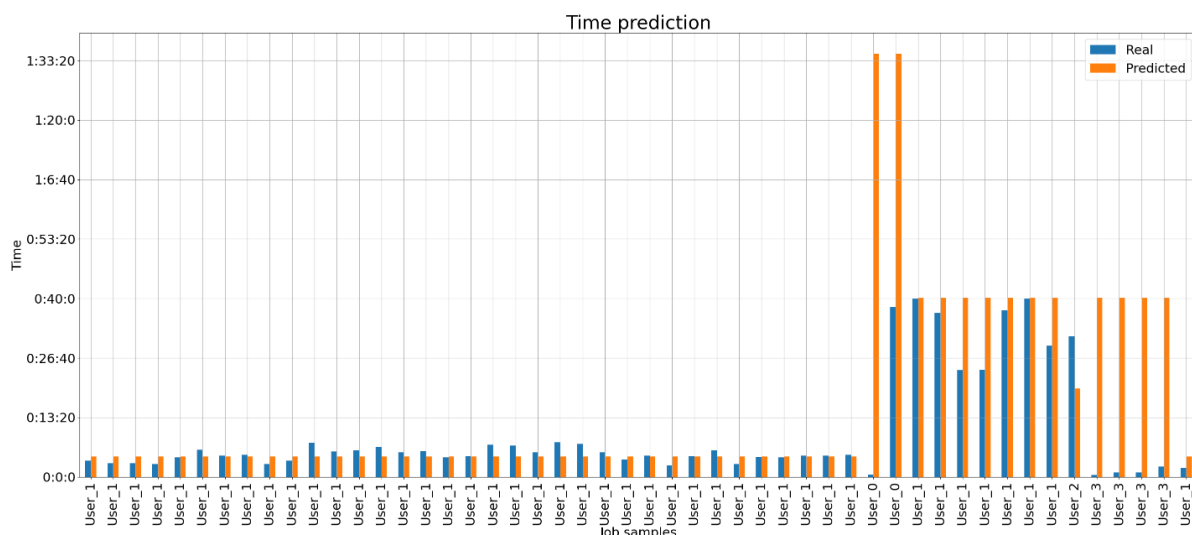


Рисунок 4.15 График работы модуля предсказания. На вертикальной оси представлено время вычисления задачи. По горизонтали отмечены соответственно задачи, запущенные различными пользователями. Синие столбцы соответствуют реальному времени выполнения задачи, оранжевые – предсказанное время модели

Заключение

Разработка статистической модели функционального управления распределением ресурсов гетерогенного кластера позволяет оптимизировать распределение ресурсов между узлами кластера для достижения максимальной эффективности использования ресурсов. Это может включать в

себя адаптивное регулирование нагрузки на различные узлы кластера в зависимости от их текущей нагрузки и доступных ресурсов.

Модель также может использоваться для прогнозирования необходимости дополнительных ресурсов для обеспечения нормальной работы кластера в будущем. Это может помочь в раннем планировании и предотвращении проблем, связанных с нехваткой ресурсов.

Помимо перечисленного, модель может использоваться для определения необходимых мер для снижения нагрузки на отдельные узлы кластера и предотвращения перегрузки. Это может включать в себя меры такие как масштабирование узлов кластера, миграция задач на другие узлы или изменение конфигурации ресурсов на каждом узле.

В частности, модель также может использоваться для определения наиболее эффективных способов распределения ресурсов в зависимости от текущей нагрузки и доступных ресурсов. Это может включать в себя алгоритмы такие как оптимизация распределения ресурсов на основе жадных алгоритмов или методов машинного обучения.

5 Алгоритм мультиагентного диспетчерского управления вычислительными ресурсами гетерогенного суперкомпьютерного кластера с использованием методов искусственного интеллекта

Цель работы мультиагентного диспетчера ГСК заключается в сокращении значения целевой функции при максимизации количества заданий, решенных в рамках своего таймаута с помощью имеющегося множества \mathbf{R} ресурсов ГСК [148]. При этом, как было сказано ранее, необходимо обеспечить самоорганизацию распределенных программных или программно-аппаратных агентов ресурсов для распределения поступающих заданий.

В ходе исследований был рассмотрен ряд вариантов реализации процедуры распределенного диспетчирования и в итоге для реализации диспетчера потока заданий в ГСК, было решено доработать логическую архитектуру системы: ввести в состав системы виртуальных агентов заданий с минимальной нагрузкой.

Таким образом, в состав мультиагентного диспетчера кроме агентов AR_i ($i=1,2,...,N$), представляющих различные ресурсы $R_j \in \mathbf{R}$ ($i=1,2,...,N$), предлагается ввести представителей заданий, фактически выполняющих роль агентов заданий AZ_j ($j=1,2,...,M$) и отвечающих за равномерное распределение нагрузки и сокращение общего времени выполнения того или иного задания $Z_l \subseteq \mathbf{Z}$ ($j=1,2,...,M$). Такие агенты заданий могут функционировать в рамках ДО, на которые пользователь отправил соответствующий экземпляр задания на выполнение (см. рисунок 5.1).

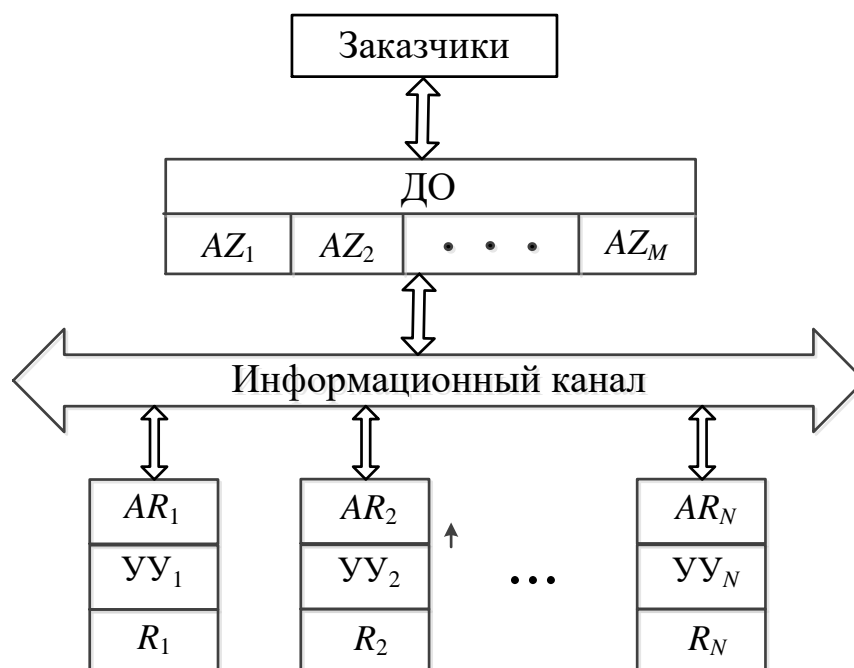


Рисунок 5.1 ГСК с мультиагентным диспетчером, включающим агентов двух типов – агентов заданий AZ и агентами ресурсов AR

Исходя из вышеприведенных соображений, работу алгоритма мультиагентного диспетчерского управления вычислительными ресурсами гетерогенного суперкомпьютерного кластера с использованием методов искусственного интеллекта в укрупненном виде можно представить следующим образом:

1. Агенты AR_j различных ресурсов R_j ($i=1,2,...,N$) последовательно с некоторой периодичностью отправляют на все ДО ГСК данные об актуальных своих параметрах загрузки L_j .
2. ДО ГСК при получении данных о значении параметров загрузки L_j от агента AR_j вычислительного ресурса R_j заносят их таблицу данных и рассчитывают среднее значение загрузки для каждого параметра.
3. В случае, если какой-либо агент AR_j вычислительного ресурса R_j не отправлял информацию о своих параметрах загрузки L_j в течение определенного таймаута, он помечается как отказавший, устаревшие значения

его параметров не используются, а администратору ГСК отправляется соответствующее сообщение.

4. Пользователь формирует дескриптор нового экземпляра задания $Z_l \in \mathbf{Z}$, для этого он;

- берет за основу один из типов заданий P ;
- создает файлы с соответствующими этому типу заданий входными данными;
- указывает дополнительные параметры входных данных, если для выбранного типа задания есть такая возможность;
- указывает время, к которому он хотел бы получить результаты выполнения задания (в ходе формирования задания пользователь должен иметь возможность видеть ретроспективу решения заданий этого типа для того, чтобы иметь возможность выбрать адекватное время для выполнения задания).

5. Пользователь получает на сайте ГСК данные о работающих в рамках системы досках объявлений, их загрузке и скорости передачи данных.

6. Пользователь размещает на ДО дескриптор своего задания и в зависимости от объема входных данных и реализации системы или обеспечивает доступ к входным данным самостоятельно, или загружает их на ДО.

7. В момент поступления задания $Z_l \in \mathbf{Z}$ на одну из ДО ГСК, на данной ДО создается виртуальный агент задания AZ_l и дескриптору присваивается уникальный идентификатор.

8. Агент AZ_l задания Z_l публикует на ДО дескриптор своего задания и разрешает доступ к нему.

9. Агенты AR_j различных ресурсов R_j ($i=1,2,...,N$) последовательно с некоторой периодичностью опрашивают различные ДО ГСК в поисках новых появившихся экземпляров заданий: они запрашивают таблицу всех заданий ДО и выбирают из нее задание Z_l , которое данный агент

еще не пытался выполнить и для которого значение разницы между установленным пользователем таймаутом и средним значение времени решения заданий такого типа минимально (см. рисунок 5.2).

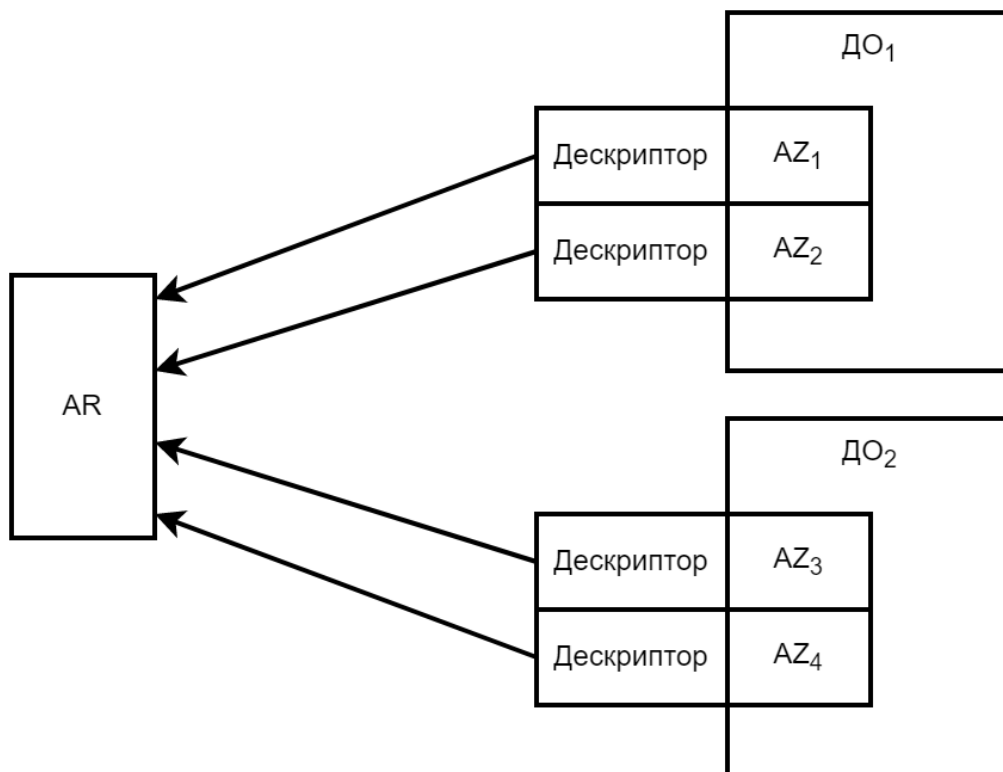


Рисунок 5.2 Процесс сбора агентом данных об актуальных заданиях с агентов заданий на различных ДО

10. Когда агент AR_j выбирает на ДО новое задание Z_l , выставленное на исполнение агентом AZ_l , для проверки, он загружает с ДО дескриптор экземпляра задания Z_l и оценивает возможность своего участия в его выполнении: если он обладает одним из видов вычислительных ресурсов, с применением которых задание соответствующего типа P_l может быть решено, он продолжает работать с дескриптором, иначе он отмечает у себя дескриптор с соответствующим идентификатором как просмотренный и переходит к поиску следующих новых заданий на всех ДО ГСК.

11. Агент AR_j использует методы прогнозирования времени выполнения заданий в ГСК на основе искусственного интеллекта

применительно к типу задания P_l , к которому относится экземпляр задания Z_l , с применением информации о задании, добавленной пользователем в дескриптор. На выходе он получает информацию об ориентировочной загрузке своего ресурса R_j (соответствующие параметры L_j в целевой функции F) при выполнении задания Z_l и об ориентировочном времени решения задания Z_l с применением имеющегося вычислительного ресурса R_j .

12. Агент AR_j оценивает время получения входных данных и исполняемых файлов экземпляра задания Z_l с применением методов оценки времени передачи данных для выполнения задания в ГСК.

13. Агент AR_j отправляет на ДО агенту AZ_l предложение о выполнении задания с информацией об ориентировочных значениях параметров L_j своего вычислительного ресурса R_j , а также суммарное значение ориентировочного времени получения необходимых данных и выполнения экземпляра задания и переходит в режим ожидания ответа от ДО. В случае, если ответ от ДО не приходит в течение определенного времени, агент AR_j отправляет соответствующее сообщение администратору ГСК и переходит к поиску следующих заданий.

14. Агент AZ_l экземпляра задания Z_l получает от агента AR_j предложение о выполнении задания Z_l , включающее соответствующую информацию об ориентировочных значениях параметров L_j вычислительного ресурса R_j , а также ориентировочном времени завершения выполнения задания и сохраняет в таблицу потенциальных исполнителей задания.

15. Агент AZ_l ожидает в течение определённого времени поступление предложений от других агентов ресурсов и заносит все поступившие предложения в таблицу потенциальных исполнителей задания (см. рисунок 5.3).

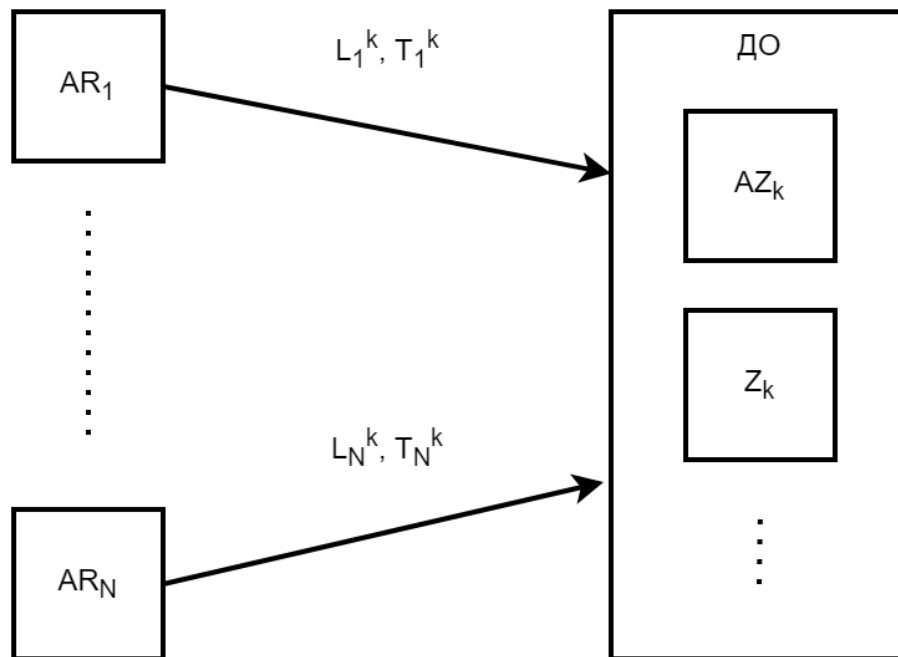


Рисунок 5.3 Поступление предложений от агентов ресурсов AR_j агенту задания AZ_l

16. По истечении времени ожидания агент AZ_l выделяет в таблице потенциальных исполнителей задания те предложения, ориентировочное время в которых удовлетворяет таймауту пользовательского задания с учетом текущего момента времени. Если предложений, ориентировочное время в которых удовлетворяет таймауту, не существуют, то переход к пункту 19.

17. Если предложения, ориентировочное время в которых удовлетворяет таймауту, существуют, то для каждого из них агент AZ_l вычисляет значение целевой функции F с учетом актуальных средних значений параметров загрузки L_j ресурсов в ГСК, имеющихся на его ДО. Затем он выбирает минимальное значение среди полученных и соответствующий указанному значению вычислительный ресурс R_j назначается на выполнение экземпляра задания Z_l , о чем сообщается его агенту AR_j .

18. Остальным агентам, выславшим предложения по исполнению задания Z_l , отправляется информация об отказе, и они продолжают процесс поиска заданий для выполнения.

19. Если предложений, ориентировочное время в которых удовлетворяет таймауту, не существуют, то агент в зависимости от реализации ГСК AZ_l может предложить пользователю отдать задание на выполнение агенту AR_j , разница между ориентировочным временем выполнения и таймаутом для которого минимальна, или воспользоваться каким-либо коэффициентом важности пользователя в ГСК для нахождения оптимального соотношения между значением целевой функции F и разницы во времени относительно таймаута.

20. Как только агент AR_j получает подтверждение о назначении на выполнение экземпляра задания Z_l , он начинает загрузку всех необходимых данных (от пользователя или с ДО) и запускает задание на выполнение.

21. Как только агент AR_j завершает выполнение какого-либо экземпляра задания Z_l , он выполняет следующие действия:

- отправляет пользователю – владельцу задания сообщение о готовности результатов;
- по готовности пользователя отправляет выходные данные задания в его адрес;
- сохраняет и отправляет администратору ГСК файл с ретроспективной информацией о выполнении задания: тип задания P_l , временные графики изменения каждого параметра вычислительного ресурса при выполнении задания, дополнительные сведения о входных данных, предоставленные пользователем и иную информацию при ее наличии.

В общем виде схема распределенного алгоритма, реализующего работу предложенного метода представлен на рисунке 5.4.

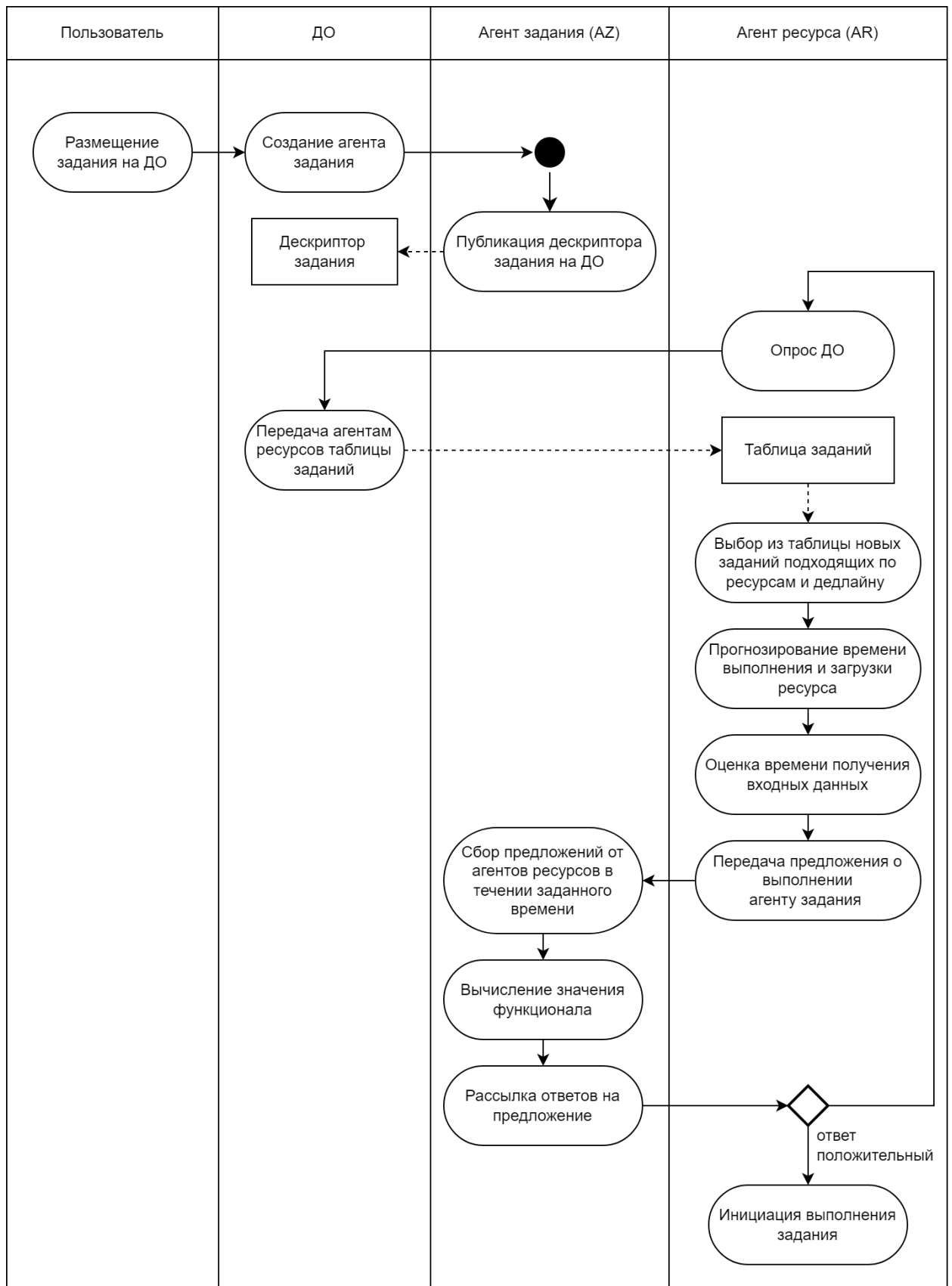


Рисунок 5.4 Схема распределенного алгоритма, реализующего метод функционирования диспетчера потока заданий в ГСК

6 Разработка методов кластеризации, как механизма управления заданиями, исполняемыми с использованием вычислительных ресурсов гетерогенного кластера

6.1 Анализ данных.

Проблемы исходных данных

В ходе анализа данных хотелось бы обозначить следующие проблемы, являющиеся препятствием к построению точных и качественных индуктивных моделей на основании имеющихся данных:

- большой разброс оцениваемой величины (от нескольких секунд до двух недель);
- сильная несбалансированность задач по диапазонам, которым они принадлежат (рисунок 6.1);
- недостаточное количество информации в имеющихся факторах, по которым оценивается целевая величина.

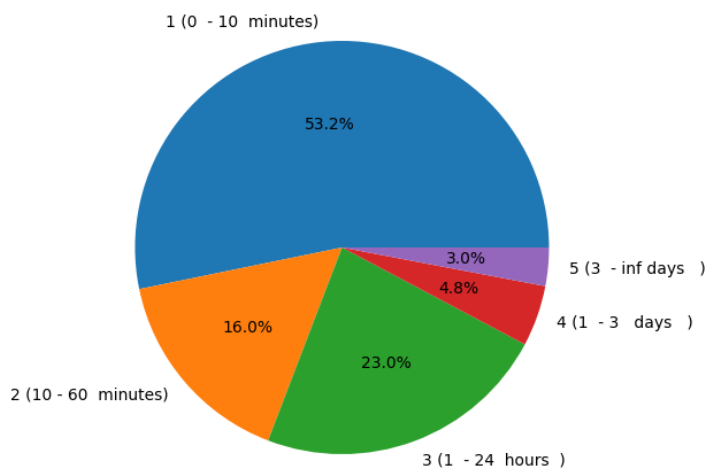


Рисунок 6.1 Распределение задач по временным диапазонам

Постановка задачи

Целевой задачей является оценка времени выполнения задачи Y на основании набора предикторов X . Решение целевой задачи может быть получено через решение известных задач:

– Регрессии. Наиболее типичный подход к решению целевой задачи. В качестве оптимизационного критерия моделей в общем случае используется среднее квадратичное или среднее абсолютное отклонение. Функции отклонения являются информативными и ненасыщенными в сравнении с функциями оптимизации, используемыми при постановке задачи в виде задачи выживаемости либо классификации, т.к. они в лучшей степени отражают критичность ошибки предсказания;

– Выживаемости. При такой постановке оценивается не сама целевая величина, а распределение вероятностей p того, что задача завершится в момент времени t . Оптимизационным критерием является максимизация индекса конкордации. Индекс конкордации - отношение числа пар, в которых соблюдается отношение упорядоченность к общему числу пар. Оптимизации по такому критерию позволяет строить модели, нацеленные на хорошее моделирование тенденции в целом, однако, плохо учитывает критические ошибки конкретных случаев. TODO почему выживаемость лучше чем регрессия + дискуссионные мысли про то, что Y случайная величина и постановка задачи через регрессию не является корректной;

– Классификации. В данном случае целевая задача может быть представлена как классификация временных диапазонов, к которым принадлежит задача. Проблемой такой постановки является то, что в классических функциях оптимизации классификационных моделей (например, кросс-энтропии) не учитываются расстояния между классами. Таким образом, с точки зрения стандартных классификационных моделей, случай, когда мы путаем класс “секунды” с классом “минуты” не отличается от случая когда мы путаем класс “секунды” с классом “дни”, что является довольно критичным при решении целевой задачи.

Геометрический смысл всех постановок исходной задачи приведен на рисунке 6.2

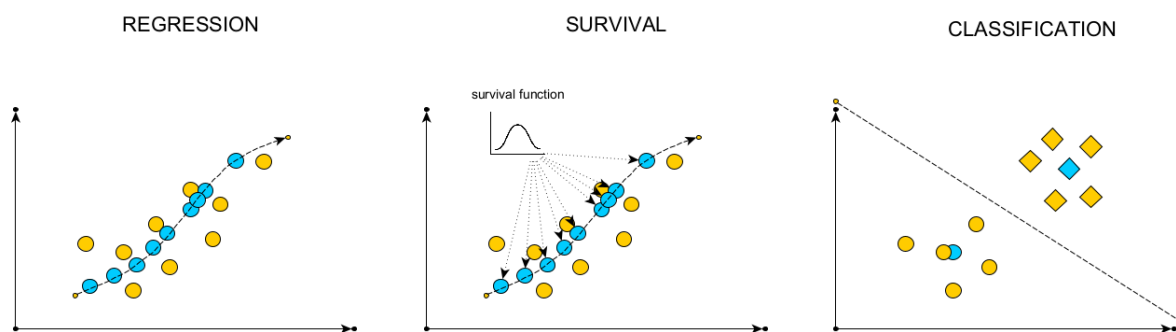


Рисунок 6.2 Геометрический смысл различных постановок исходной задачи

6.2 Выбор методов на основании анализа данных

Случайный лес является комбинацией деревьев решений, основанной на бутстрэпе и методе случайных подпространств.

Основные концепции анализа выживаемости

Набор тренировочных данных в анализе выживаемости состоит из цензурированных и нецензурированных наблюдений. Для первых известно только то, что время до интересующего события (смерти или неудачи), превышает продолжительность наблюдения. Случай же, когда время до наступления события совпадает с продолжительностью наблюдения, соответствует наблюдению без цензуры. Принимая во внимание эти типы наблюдений, можно записать обучающий набор D , который состоит из n триплетов $(\mathbf{x}_i, \delta_i, T_i), i = 1, \dots, n$, таких, что $\mathbf{x}_i^T = (x_1^{(i)}, \dots, x_m^{(i)})$ – вектор признаков i -го примера; T_i – время до наступления интересующего события; $\delta_i \in \{0, 1\}$ – индикатор цензурирования наблюдения. В частности, $\delta_i = 1$, когда наблюдается событие, представляющее интерес, т.е. это цензурирование отсутствует, и $\delta_i = 0$, если наоборот.

Модель выживаемости обучается на множестве D с целью оценить вероятностные характеристики времени T до наступления интересующего события для нового экземпляра \mathbf{x} .

Одним из важных понятий в анализе выживаемости является функция выживания $S(t|\mathbf{x})$, которая представляет собой вероятность выживания экземпляра \mathbf{x} до момента времени t , т.е. $S(t|\mathbf{x}) = \Pr\{T > t|\mathbf{x}\}$. Другим важным

определением является кумулятивная функция риска $H(t|\mathbf{x})$, которая интерпретируется как мера того, что событие наступит в момент времени t при условии выживания объекта до этого момента. SF определяется через CHF следующим образом:

$$S(t|\mathbf{x}) = \exp(-H(t|\mathbf{x}))$$

CHF можно определить еще и как интеграл функции риска $h(t|\mathbf{x})$, которая также используется в анализе выживаемости и определяется как частота событий в момент времени t при условии, что ни одно событие не произошло до этого момента.

Одним из важных показателей, характеризующих модели выживаемости, является индекс конкордации Харрелла. C-index используется для сравнения различных моделей и настройки их параметров. Его также можно рассматривать как ошибку предсказания для оценки обученности модели. C-index измеряет вероятность того, что в случайно выбранной паре примеров тот, который потерпел неудачу первым, имел худший прогнозируемый результат. Он рассчитывается как отношение числа пар, правильно упорядоченных моделью, к общему числу допустимых пар. Пара недопустима, если оба события цензурированы справа или если цензурируется самое раннее время в паре. Если C-index равен 1, то соответствующая модель выживаемости должна быть идеальной, а если равен 0.5, то результат будет не лучше случайного угадывания.

Модель Кокса

Одним из популярных методов для анализа данных о выживаемости, учитывающих особенности обучающих примеров, является полупараметрическая регрессионная модель пропорциональных рисков Кокса, которая вычисляет влияние наблюдаемых ковариат (признаков) на риск возникновения события. Она предполагает, что логарифмический риск интересующего события представляет собой линейную комбинацию ковариат. Это предположение называется условием линейного пропорционального

риска. Модель Кокса является полупараметрической в том смысле, что ее можно разделить на параметрическую часть, которая состоит из вектора параметров регрессии, связанного с ковариатами, и непараметрическую часть, которая может быть оставлена полностью неопределенной.

В соответствии с моделью Кокса функция риска в момент времени t при заданных значениях предиктора \mathbf{x} определяется как

$$h(t|\mathbf{x}, \mathbf{b}) = h_0(t)\exp(\psi(\mathbf{x}, \mathbf{b})),$$

где $h_0(t)$ – базовая функция риска, не зависящая от вектора \mathbf{x} и вектора \mathbf{b} ; $\mathbf{b}^T = (b_1, \dots, b_m)$ – неизвестный вектор коэффициентов или параметров регрессии. Базовая функция риска представляет собой риск, когда ковариаты равны нулю $\mathbf{x}^T = (0, \dots, 0)$. Функция $\psi(\mathbf{x}, \mathbf{b})$ в модели линейна, т.е.

$$\psi(\mathbf{x}, \mathbf{b}) = \mathbf{b}^T \mathbf{x} = \sum_{k=1}^m b_k x_k.$$

Коэффициент риска – это отношение между двумя функциями риска $h(t|\mathbf{x}_1, \mathbf{b})$ и $h(t|\mathbf{x}_2, \mathbf{b})$. Он постоянен для модели Кокса с течением времени.

В рамках модели Кокса функция выживания вычисляется следующим образом:

$$S(t|\mathbf{x}) = \exp\left(-H_0(t)\exp(\psi(\mathbf{x}, \mathbf{b}))\right) = (S_0(t))^{\exp(\psi(\mathbf{x}, \mathbf{b}))},$$

где $H_0(t)$ – базовая кумулятивная функция риска; $S_0(t)$ – базовая функция выживания, которая определяется как SF для нулевого вектора признаков $\mathbf{x}^T = (0, \dots, 0)$. Важно отметить, что функции $H_0(t)$ и $S_0(t)$ не зависят от \mathbf{x} и \mathbf{b} .

Одной из основных проблем использования модели Кокса является предположение о линейной зависимости между ковариатами и логарифмическим риском наступления события, что уменьшает гибкость и эффективность подхода, так как эта связь может быть нелинейной и плохо аппроксимируемой простыми методами. Поэтому было предложено множество моделей машинного обучения, включая глубокие нейронные сети,

метод опорных векторов, случайный лес выживаемости и т.д., которые стали мощными и эффективными инструментами анализа выживаемости.

Объяснение моделей выживаемости

Следует отметить, что большинство моделей, имеющих дело с данными о выживаемости, можно рассматривать как «черные ящики», которые должны быть объяснены. Модель Кокса в этой ситуации играет решающую роль, поскольку она рассматривает линейную комбинацию ковариат примера, коэффициенты которых можно рассматривать как количественное влияние признаков на предсказание. Это свойство позволяет использовать эту модель или ее модификации для аппроксимации «черных ящиков» в анализе выживаемости с целью их интерпретации.

Несмотря на важность моделей выживаемости, особенно в медицине, необходимо признать, что для интерпретации предсказаний существуют только расширения LIME. Эти методы, названные SurvLIME, используют в качестве основы для получения объяснений как раз модель Кокса. Однако, они основаны на линейной аппроксимации «черных ящиков» и позволяют получить только локальные интерпретации.

Поэтому интересно расширить модель Кокса с помощью GAM вместо линейной зависимости, т.е. заменяя $\psi(\mathbf{x}, \mathbf{b})$ на функцию

$$\psi(\mathbf{x}, \mathbf{g}) = g_1(x_1) + \dots + g_m(x_m).$$

Такая комбинация уже рассматривалась несколькими авторами. Если же при этом использовать NAM, то путем модификации этого метода для обучения нейронной сети с целью аппроксимации расширенной модели Кокса можно будет получить объяснения предсказаний модели выживаемости типа «черный ящик» в виде набора функций влияния признаков входных данных.

Случайные и глубокие леса выживаемости

Некоторые непараметрические модели выживаемости основаны на использовании случайных лесов выживаемости (СЛВ), которые можно рассматривать как расширение случайных лесов с учетом цензурированности

обучающих данных. Эти модели дают превосходящие результаты, когда обучающих данных мало или когда в обучающей выборке имеется много цензурированных примеров. Для расширения и улучшения случайных лесов выживаемости был предложен глубокий лес выживаемости в качестве расширения так называемого глубокого леса на случай цензурированных данных для анализа выживаемости. Глубокий лес является моделью на основе композиции случайного леса, которая включает в себя набор случайных лесов, организованных в виде уровней каскада лесов, аналогично слоям в нейронных сетях. Однако нейроны заменяются различными случайными лесами, которые играют ту же роль, что и нейроны. Случайные леса в глубоком лесу выживаемости заменяются на случайные леса выживаемости. Однако в отличие от исходного глубокого леса, глубокий лес выживаемости использует специальную схему стекинга, которая реализует связи между уровнями глубокого леса.

Случайный лес выживаемости аналогичен регрессионному случайному лесу, но каждое дерево решений в случайном лесу выживаемости использует специальные правила расщепления, которые устанавливают процедуру деления подмножества точек обучающих данных на два подмножества. Существует несколько правил расщепления. В отличие от случайного леса, выход которого - распределение вероятностей классов, случайный лес выживаемости предсказывает кумулятивную функцию риска, обозначенную $H(t|\mathbf{x})$, или функцию выживаемости (функцию надежности), обозначенную $S(t|\mathbf{x})$, которая рассчитывается путем усреднения оценок функций, полученных всеми деревьями случайного леса выживаемости.

Существует несколько показателей для сравнения различных моделей выживаемости. Наиболее популярным показателем, характеризующим эффективность прогнозирования модели, является С-индекс или индекс конкордации Харела. Он оценивает, насколько правильно модель ранжирует времена жизни. Его также определяют как меру согласия между прогнозируемой и наблюдаемой функциями выживаемости. Для оценки С-

индекса рассмотрим часть множества D , состоящего из допустимых пар $\{(\mathbf{x}_i, \delta_i, T_i), (\mathbf{x}_j, \delta_j, T_j)\}$ для $i \leq j$. Пара является недопустимой, если оба события цензурируются справа или, если наименьшее время в паре является цензурированным. Тогда С-индекс - это отношение количества пар, правильно упорядоченных моделью, к общему количеству допустимых пар. Если С-индекс равен 1, то соответствующая модель выживаемости считается идеальной. Если это 0.5, то модель не лучше, чем случайное угадывание. Пусть t_1, \dots, t_n - заранее определенные моменты времени. Если выход алгоритма - прогнозируемая функция выживаемости то С-индекс рассчитывается как:

$$C = \frac{1}{M} \sum_{i: \delta_i=1} \sum_{j: t_i < t_j} \mathbf{1}[\hat{S}(t_i | \mathbf{x}_i) > \hat{S}(t_j | \mathbf{x}_j)].$$

Здесь M - число всех допустимых пар; $\mathbf{1}[a]$ - индикаторная функция, принимающая значение 1, если условие a выполняется, и 0, иначе.

6.3 Полученные результаты моделирования

Построение моделей

Исходные данные были разделены на обучающую и тестовую подвыборки. В обучающую выборку были включены задачи с 9 сентября по 17 октября. В тестовую выборку были включены задачи с 17 октября по 17 ноября. Размеры обучающей и тестовой выборки составили 26 и 8 тысяч задач соответственно.

Для построения моделей, наиболее устойчивых к общим тенденциям, имеющимся в данных, используется алгоритм k-fold кросс-валидации с числом разбиений равным 5. Число разбиений выбрано эвристически. Лучшая модель выбирается на основании наименьший средней ошибки по 5 разбиениям обучающей выборки. Таким образом построение модели на зависит от тестовых данных и тестовая выборка может быть также рассмотрена как оценочная, а по результатам модели на тестовой выборке можно судить не только о ее способности к интерполяции, но также и к

экстраполяции целевой функции зависимости. Схема построения и выборка моделей приведена на рисунке 6.3.

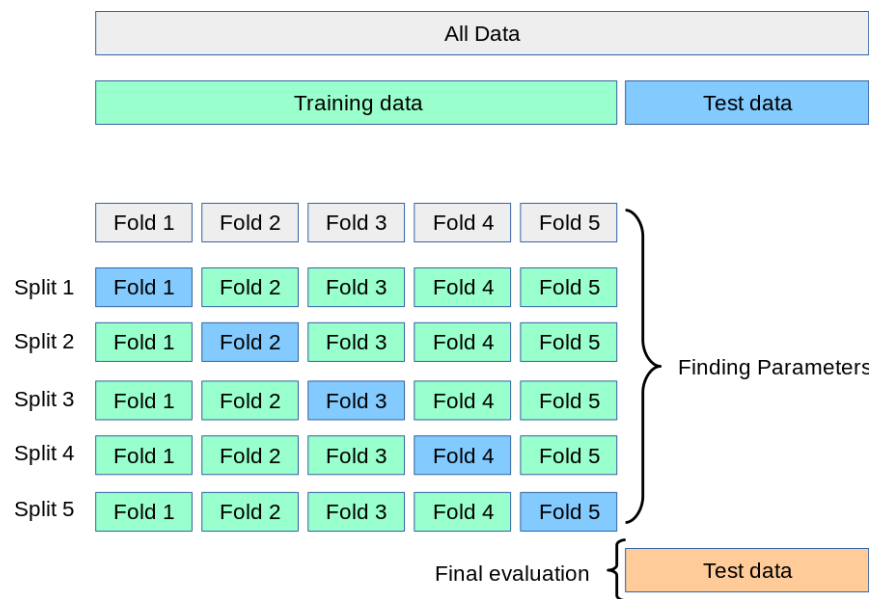


Рисунок 6.3 Схема построения и выбора моделей

Критерии оценки

В качестве критериев оценки всех моделей используются:

- r-коэффициент корреляции по Пирсону

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

- r²-коэффициент детерминации

$$r^2 = \frac{\sum_{i=1}^n (y_i - x_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

– матрица ошибок. При построении матрицы ошибок регрессионные оценки времени по всем моделям представляются в виде диапазонов, в который попала оценка. Для лучшей интерпретируемости в работе используются как матрицы ошибок, нормализованные по общему числу задач, так и по числу задач, попадающих в диапазон.

– Polynomial confusion accuracy – специальная метрика, отражающая предметную специфику. Рассчитывается на основании матрицы ошибок. Наиболее точной моделью считается модель с наиболее интенсивной

диагональю. С увеличением расстояния до диагонали вклад каждого элемента матрицы уменьшается. Вклад элементов под диагональю (оценка ниже факта) считается ниже вклада элементов над диагональю (оценка выше факта).

$$Polynomial\ confusion\ accuracy = \sum_{i=0}^{cols} \sum_{j=0}^{rows} weighted_acc_{i,j},$$

где $weighted_{acc_{i,j}} = acc_{i,j} * 0.5^{|i-j|}$ - на элементах, где оценка выше факта ($j - i > 0$); $weighted_{acc_{i,j}} = acc_{i,j} * 0.1^{|i-j|}$ - на элементах, где оценка ниже факта ($j - i < 0$); $weighted_{acc_{i,j}} = acc_{i,j}$ - вклад диагональных элементов не взвешивается.

Оценки пользователей

На рисунке 6.4 приведена матрица ошибок, построенная на основании пользовательских оценок из тестовой выборки, нормализованная по общему числу задач. На матрице видно, что 17 % всех задач, имеющих фактическую длительность [0, 10] минут указывают в заявке примерное время выполнения [1, 24] часа, а еще 13 % всех задач, имеющих фактическую длительность [0, 10] минут указывают в заявке примерное время выполнения [1-3] дней. Также 18 % задач, имеющие фактическую длительность [1, 24] часа указывают в заявке примерное время выполнения [3, 15] дней.

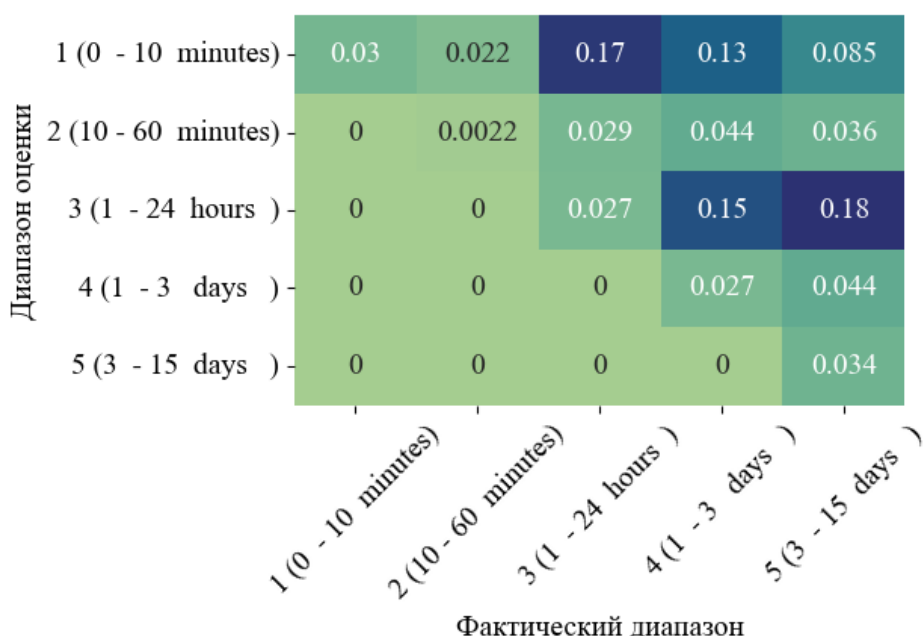


Рисунок 6.4 Матрица ошибок на основании пользовательских оценок из тестовой выборки, нормализованная по общему числу задач

На рисунке 6.5 представлена матрица ошибок на основании пользовательских оценок, нормализованная по числу задач, попадающих в диапазон. Матрица демонстрирует, что высокую точность определения диапазона ($> 80\%$) демонстрируют лишь задачи, фактическая длительность задач которых попадает в диапазон $[3, 15]$ дней. В то же время для задач, имеющих малое время выполнения, точность попадания оценки в правильный диапазон является довольно низкой - лишь для 6.8% задач, имеющих время выполнения $[0, 10]$, 2% задач, имеющих время выполнения $[10, 60]$ минут и 7.8% задач, имеющих время выполнения $[1, 24]$ диапазон пользовательской оценки совпадает с фактическим диапазоном времени выполнения.

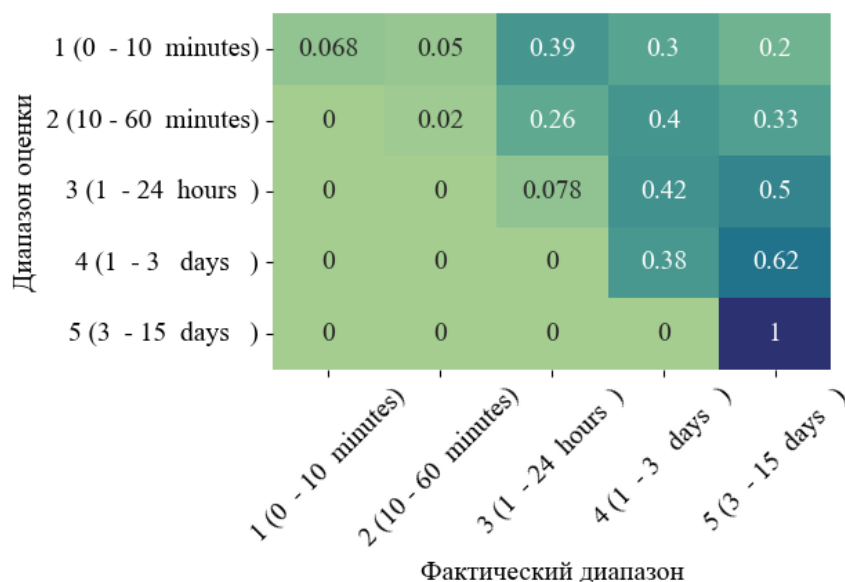


Рисунок 6.5 Матрица ошибок на основании пользовательских оценок, нормализованная по числу задач, попадающих в диапазон.

Таким образом, плюсом пользовательских оценок несомненно является то, что оценка пользователя никогда не оказывается ниже фактического времени расчета задачи. Однако в то же время в большей части заявок указываемое время оказывается сильно больше требуемого для решения задачи, что может мешать планировщику максимально эффективно использовать вычислительные ресурсы СКЦ.

С использованием методов машинного обучения хотелось бы получить модель, матрица ошибок которой будет иметь более интенсивную диагональ и допускать критических ошибок при оценке времени выполнения пользовательской задачи. Под критической ошибкой подразумевается случай, когда время требуемое для решения пользовательской задачи было сильно занижено.

Случайный лес

На рисунке 6.6 приведен график зависимости фактического времени выполнения задач от оцененного моделью случайного регрессионного леса. Точность оценки по критерию r составила 0.68, по критерию r^2 - 0.40.

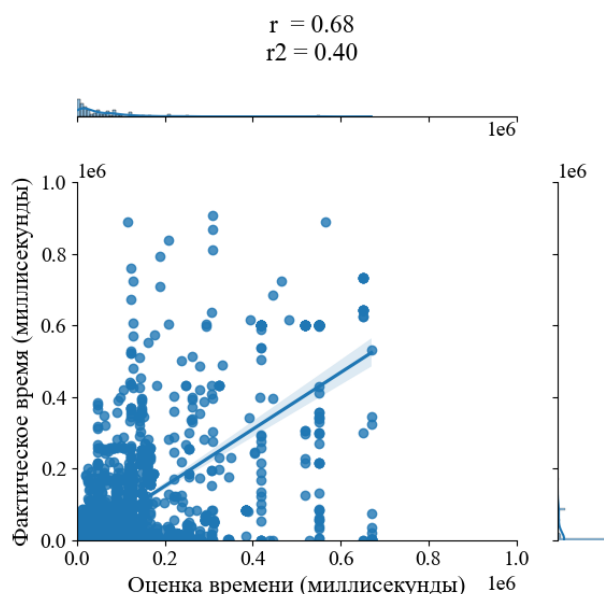


Рисунок 6.6 График зависимости фактического времени от оцененного моделью случайного регрессионного леса

На рисунке 6.7 приведена матрица ошибок модели случайного леса, нормализованная по общему числу задач в тестовой выборке. Матрица демонстрирует высокую точность соответствия для задач, имеющих длительность $[1, 24]$ часа. Однако, если проанализировать предсказания модели, можно заметить, что наибольшее число предсказаний модели ($\sim 0.78\%$) попадают в диапазон $[1, 24]$ часа. Таким образом модель демонстрирует

низкую вариативность предсказаний и по большей части сходится к предсказанию среднего значения.

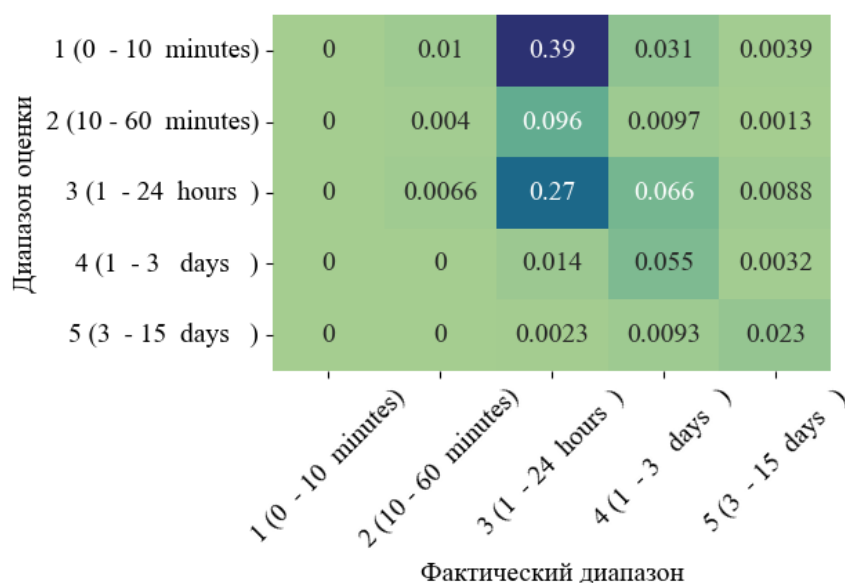


Рисунок 6.7 Матрица ошибок модели случайного регрессионного леса, нормализованная по общему числу задач в тестовой выборке

Таблица 6.1 Сетка параметров модели случайного леса

Параметр	Вектор выбираемых значений
n_estimators	26, 52 , 104
bootstrap	True , False
max_depth	10, 20, 30
max_features	0.5 , 1.00
min_samples_leaf	1, 2, 4, 8

Градиентный бустинг с асимметричной функцией ошибки

На рисунке 6.8 приведен график зависимости фактического времени выполнения задач от оцененного моделью случайного регрессионного леса. Точность оценки по критерию r составила 0.61, по критерию r^2 - 0.11.

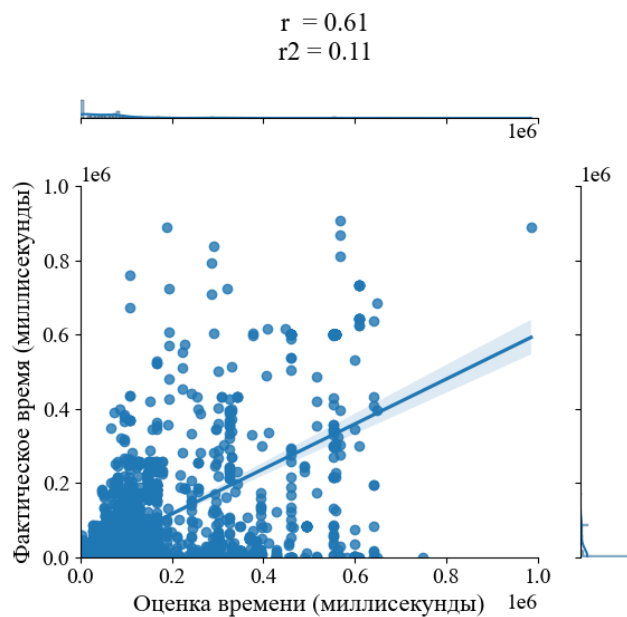


Рисунок 6.8 График зависимости фактического времени от оцененного моделью градиентного бустинга

На рисунке 6.9 приведена матрица ошибок модели градиентного бустинга, нормализованная по общему числу задач в тестовой выборке. Матрица демонстрирует, что сравнительно с моделью случайного леса модель градиентного бустинга демонстрирует меньшую предрасположенность к предсказанию длительности задачи в диапазоне $[1, 24]$ часа. Значительной части задач, имеющих длительность $[0, 10]$ минут соответствуют предсказания в диапазоне $[10, 60]$ минут, что может поспособствовать оптимизации использования ресурсов СКЦ.

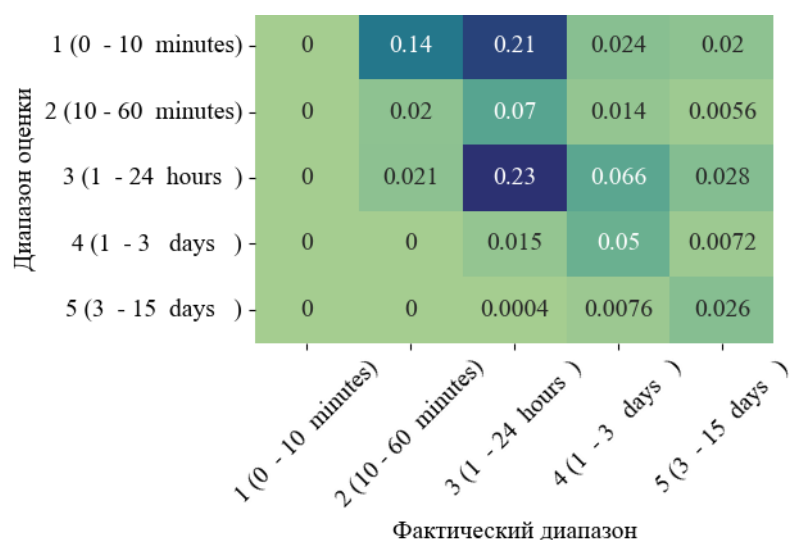


Рисунок 6.9 Матрица ошибок модели градиентного бустинга, нормализованная по общему числу задач в тестовой выборке

Таблица 6.2 Сетка параметров модели градиентного бустинга

Параметр	Вектор выбираемых значений
n_estimators	26, 52, 104
bootstrap	True, False
max_depth	10, 20, 30
max_features	0.5, 1.00
min_child_samples	1, 2, 4, 8

Случайный лес выживаемости

На рисунке 6.10 приведен график зависимости фактического времени выполнения задач от оцененного моделью случайного леса выживаемости. Точность оценки по критерию r составила 0.2, по критерию r^2 - 0.00.

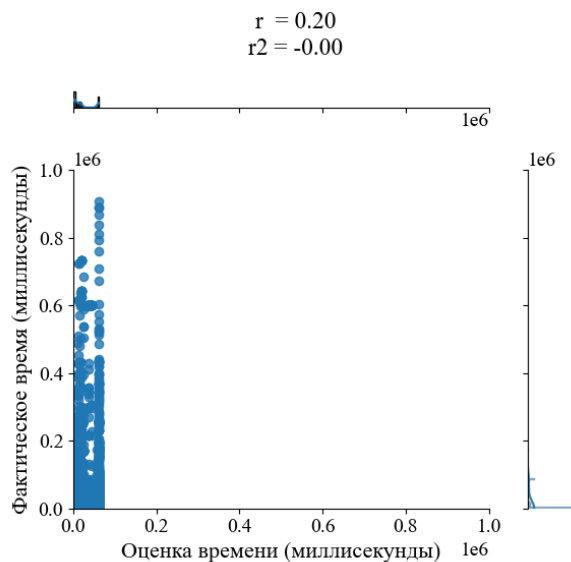


Рисунок 6.10 График зависимости фактического времени от оцененного моделью градиентного бустинга

На рисунке 6.11 приведена матрица ошибок модели градиентного бустинга, нормализованная по общему числу задач в тестовой выборке. Матрица демонстрирует плохую вариативность предсказаний модели. Модель леса выживаемости не подходит для прямого использования на исходных данных из-за их высокого разброса (от 1 секунды до 15 дней), а индекс конкордации Харела, используемый в качестве критерия оптимизации модели, имеет большую чувствительность к общему числу ошибок, а не к величинам ошибок в конкретных предсказаниях, из-за чего модель игнорирует ошибки предсказания на задачах с большим временем исполнения, поскольку их число в выборке сравнительно мало (порядка 3%).

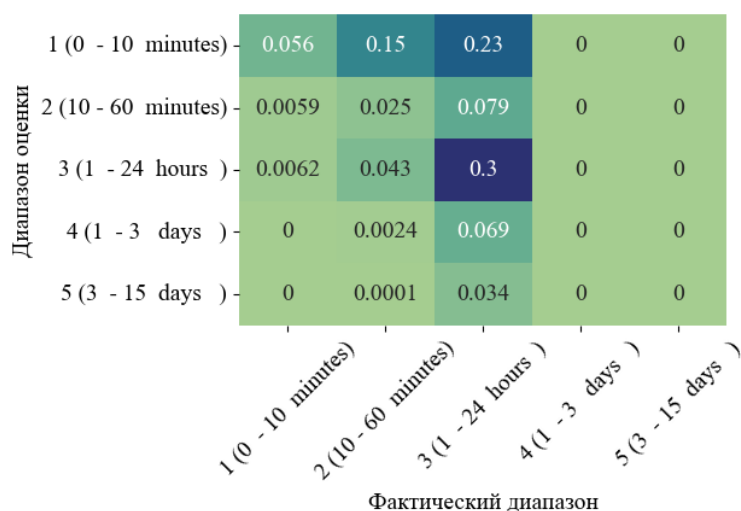


Рисунок 6.11 Матрица ошибок модели леса выживаемости, нормализованная по общему числу задач в тестовой выборке

Таблица 6.3 Сетка параметров модели случайного леса выживаемости

Параметр	Вектор выбираемых значений
n_estimators	26, 52, 104
bootstrap	True, False
max_samples	500, 1000, 2000
max_depth	10 , 20, 30
max_features	0.5, 1.00
min_child_samples	1, 2, 4 , 8

Кластеризация + градиентный бустинг с асимметричной функцией ошибки + случайный лес выживаемости

На рисунке 6.12 приведен график зависимости фактического времени выполнения задач от оцененного моделью "кластеризация + градиентный бустинг с асимметричной функцией ошибки + случайный лес выживаемости". Точность оценки по критерию r составила 0.42, по критерию r_2 - 0.11.

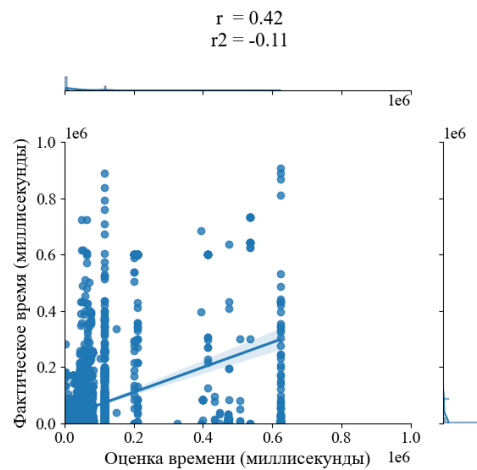


Рисунок 6.12 График зависимости фактического времени от оцененного моделью «кластеризация + градиентный бустинг с асимметричной функцией ошибки + случайный лес выживаемости»

На рисунке 6.13 приведена матрица ошибок модели “кластеризация + градиентный бустинг с асимметричной функцией ошибки + случайный лес выживаемости”, нормализованная по общему числу задач в тестовой выборке. Матрица демонстрирует наиболее интенсивную диагональ чем в других моделях. Особенно стоит отметить способность модели как к правильной оценке времени легковесных задач $[0, 10]$ минут, так и задач средней длительности $[1, 24]$ часа.

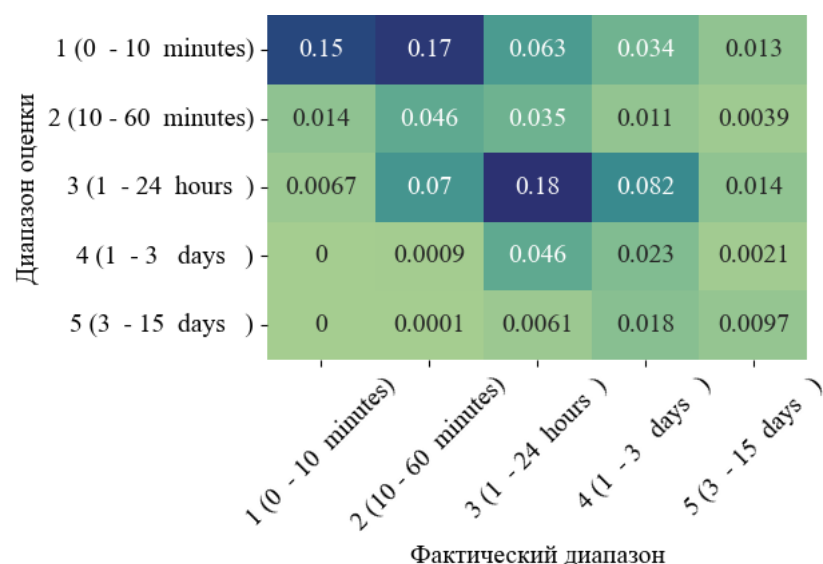


Рисунок 6.13 матрица ошибок модели ”кластеризация + градиентный бустинг с асимметричной функцией ошибки + случайный лес выживаемости”, нормализованная по общему числу задач в тестовой выборке

Таблица 6.4 сетка параметров модели «кластеризация + градиентный бустинг + лес выживаемости»

Параметр	Описание	Вектор выбираемых значений
clusters_n	Число кластеров	1, 2, 3, 4
lgbm_n_estimators	Число деревьев в модели lgbm	26, 52, 104
lgbm_bootstrap	Обучение деревьев на различных подвыборках исходной выборки в модели lgbm	True, False
lgbm_max_depth	Максимальная глубина дерева в модели lgbm	10, 20, 30
lgbm_max_features	Максимальное число факторов в дереве в модели lgbm	0.5, 1.00
lgbm_min_child_samples	Минимальное число примеров в листе дерева модели lgbm	1, 2, 4, 8
rsf_n_estimators	Число деревьев в модели rsf	26, 52, 104

Продолжение таблицы 6.4

rsf_bootstrap	Обучение деревьев на различных подвыборках исходной выборки в модели rsf	True, False
rsf_max_samples	Максимальное число примеров в дереве в модели rsf	500, 1000, 2000
rsf_max_depth	Максимальная глубина дерева в модели rsf	10 , 20, 30
rsf_max_features	Максимальное число факторов в дереве в модели rsf	0.5, 1.00
rsf_min_child_samples	Минимальное число примеров в листе дерева модели rsf	1, 2, 4 , 8

Общие выводы по результатам

Общие результаты по всем моделям приведены в таблице 6.5. С точки зрения привычных метрик проблемы регрессии r и r^2 лучший результат демонстрирует модели RF и LGBM, однако такие метрики не отражают несбалансированности обучающей выборки. С точки зрения решаемой задачи, наиболее важной метрикой считается Polynomial confusion accuracy, поскольку эта метрика в большей степени учитывает специфику предметной области, разнородность и несбалансированность обучающей выборки. Наилучшие результаты по метрике демонстрируют методы RSF и гибридный метод «Clustering + LGBM + RSF». Результаты демонстрируют неоднозначность использования общих регрессионных критериев оценки моделей машинного обучения в целевой задаче оценки времен исполнения задач на СК. Актуальной видится проблема дальнейшего развития критериев оценки качества моделей машинного обучения при решении целевой задачи.

Таблица 6.5. Общие результаты по всем моделям

Модель	r	r2	Polynomial confusion accuracy
User	0.31	-18.29	0.36
RF	0.68	0.4	0.55
LGBM	0.61	0.11	0.54
RSF	0.2	0	0.57
Clustering + LGBM + RSF	0.42	-0.11	0.60

7 Анализ протоколов взаимодействия пользователей, размещаемых задач и ресурсов гетерогенного кластера при решении прикладных задач в предметных областях

7.1 Постановка исследовательской задачи

7.1.1 Описание данных

В соответствии с п. 2.2.3 настоящего отчета, при размещении задачи пользователь в основном запрашивает в СКЦ в качестве ресурсов время, необходимое для задачи, и количество вычислительных узлов. Могут быть также запрошены число процессоров, количество памяти и другие ресурсы.

Для анализа были собраны данные за период с середины июня по середину ноября 2022 года. Выбор периода обусловлен такими фактами, как летние отпуска, в которые активность размещения задач в СКЦ предположительно снижается, начало учебного года, когда активность пользователей должна возрасти, а также отчётный период в последнем квартале года, когда активность так же может быть высокой. На рисунке 7.1 показана загрузка СКЦ в рассматриваемый период времени. В июне (второй половине) и июле было размещено намного меньше задач, чем в остальных месяцах. Больше всего задач пришлось на август, а в сентябре и октябре наблюдалась средняя загруженность СКЦ. Не смотря на количество размещённых задач, самым загруженным месяцем по выделенному времени и процессорному времени был сентябрь, а размер задач июля немногим уступал размеру задач августа (в то время как по количеству задач эти месяцы отличаются в 4 раза). По количеству выделенных узлов и процессоров лидировал август. Сентябрь и октябрь были средними по загруженности, и меньше всего загружены июль и июнь. Поскольку была рассмотрена только первая половина ноября, и данные по второй половине не были включены в анализ, по имеющимся графикам можно предположить, что ноябрь по загруженности близок к сентябрю и октябрю.

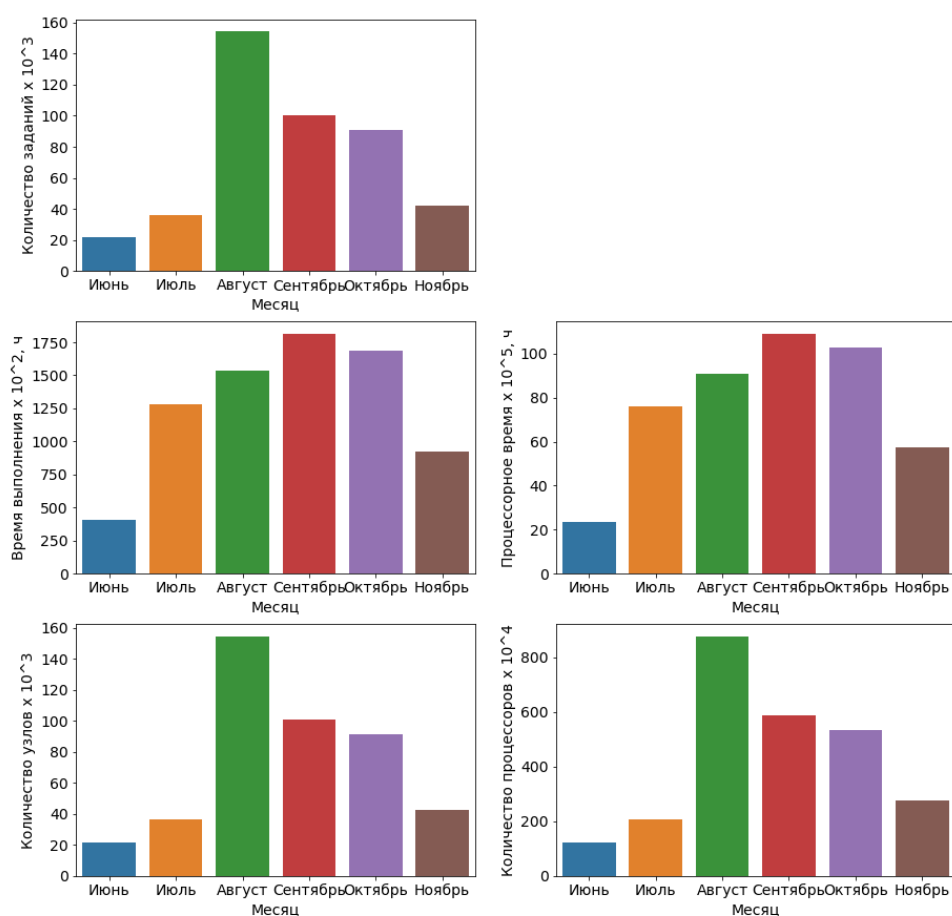


Рисунок 7.1 Гистограммы загрузки СКЦ

Рисунок 7.2 иллюстрирует загрузку очередей, в которые размещаются задания в СКЦ. Наименования очередей tornado, tornado-k40 и cascade соответствуют описанию СКЦ, приведённому в п. 3.2.2 настоящего отчета. Подавляющее большинство задач (92%) попадают в очередь tornado.

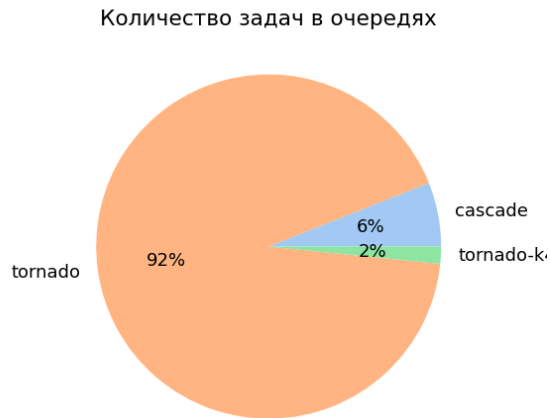


Рисунок 7.2 Распределение задач по очередям различных кластеров

7.1.2 Описание источников данных

Для анализа использовались данные планировщика задач, формируемые перед постановкой задачи в очередь. Часть данного набора составляют параметры требуемых ресурсов – количество вычислительных узлов, процессорных ядер, времени и т. п. Так же там содержатся атрибуты пользователя – имя учетной записи и группы, соответствующие идентификаторы; данные о запускаемой задаче – путь к стартовому скрипту, рабочий каталог, файлы для перенаправления потоков стандартного вывода и ошибок. Используя эти данные, планировщик принимает решение о месте и времени запуска задачи.

Второй набор данных взят из системы статистики планировщика (accounting), в которой сохраняется часть входных параметров задачи, но, главное, есть список задействованных вычислительных узлов, время выполнения задачи (и подзадач при наличии), потребленное процессорное время, объём использованной памяти. Объединяя и фильтруя эти наборы данных можно разносторонне изучить выполняемые задачи, чтобы разработать способ уточнения входных параметров задачи для повышения эффективности работы планировщика.

Помимо сбора базовых данных о процессах задач планировщик имеет механизм учета потребляемых ресурсов таких, как электроэнергия, файловые системы и сетевой трафик. Данные механизмы недоступны в стандартной конфигурации, т. к. требуют обеспечения поддержки соответствующих аппаратных и программных подсистем.

В работе сложных вычислительных (и не только) систем применяются различные механизмы наблюдения за компонентами и контроля их состояния – системы мониторинга и диспетчеризации. На данный момент существует множество решений систем мониторинга с разным уровнем возможностей. Наиболее популярными являются приложения с открытым исходным кодом nagios (nagios.org) и zabbix (zabbix.com) и их производные, например, icinga (icinga.com). Так же активно используется связка TIG – Telegraf-InfluxDB-Grafana (www.influxdata.com, grafana.com) для решения комплексных задач по мониторингу оборудования и различных сервисов или Prometheus+Grafana (prometheus.io). Можно выделить системы, ориентированные на сбор и визуализацию данных без существенных возможностей их анализа и реакции на события – Ganglia, collectd, cacti (ganglia.info, collectd.org, cacti.net).

При использовании в высокопроизводительных системах необходимо учитывать, что будут собираться большие объемы однотипных данных в силу специфики кластерной архитектуры таких систем. Далеко не все метрики можно получать с минимальными затратами и, соответственно, влиянием на производительность оборудования. Потому предпочтение отдается легковесным системам сбора данных (Ganglia, collectd). Однако, при аккуратной настройке возможно использование и других систем мониторинга.

Основной сложностью использования данных, собираемыми системами мониторинга, является их пересечение с данными планировщика задач. В общем случае, когда единицей планирования является процессорное ядро, данные мониторинга практически непригодны для решения рассматриваемой задачи. Но в случае выделения целых узлов есть возможность выделения данных, соответствующих интервалу выполнения задачи.

Еще одним источником данных могут выступать системы диспетчеризации, применяемые для управления инженерным оборудованием вычислительного центра таким, как системами электропитания и охлаждения. Однако данные системы не обладают желаемой детализацией сбора данных, что существенно ограничивает их полезность для решения рассматриваемой задачи.

7.1.3 Описание исследовательской задачи

Для того, чтобы получить более полную картину поведения пользователей при размещении задач в СКЦ, анализ был выполнен по данным, сгруппированным непосредственно по пользователям, а также по научным группам пользователей и предметным областям. За обозначенный период времени свои задачи разместили 145 пользователей, 63 группы и 10 предметных областей.

В процессе работы с данными было обнаружено, что 88% задач разместили пользователи научной группы с наименованием «geovation» (далее geovation без кавычек), относящейся к предметной области геофизики. Известно, что пользователи geovation размещают свои задачи посредством вспомогательного программного обеспечения, то есть автоматизированным образом. Поскольку доля задач, размещённых этими пользователями очень велика и оказывает большое влияние на анализируемые показатели, набор данных был разделён на две части – задачи пользователей geovation и задачи остальных пользователей. Каждая часть исследовалась отдельно.

В анализе поведения пользователей были использованы элементы описательной статистики, вычислены и построены графически следующие показатели:

- суммарное и среднее потребление ресурсов – размещённые задачи, время в СКЦ, количество вычислительных узлов и процессоров. Суммарные величины показывают, сколько всего ресурсов было выделено пользователям за анализируемый период. Средние величины с СКО и

медианные величины характеризуют распределения ресурсов по пользователям, группам и предметным областям;

- количество и доли задач, недооценённых и переоценённых пользователями, а также задач, время для которых пользователи оценили правильно с точностью 95% (ошиблись на $\pm 5\%$). Недооценённые задачи – это те задачи, время выполнения которых превысило время, запрошенное пользователями. Таким образом, задача потребовала больше времени, чем изначально запросил пользователь. Переоценёнными задачами являются те, которые выполнялись меньшее время, чем запросили пользователи;

- количество и доли задач с установками по умолчанию (предположительно двое суток, один узел), а также количество и доли недооценённых и переоценённых задач среди задач с установками по умолчанию. Если пользователь, настраивая переменные окружения при постановке задачи в СКЦ, не устанавливает время выполнения задачи и количество узлов, то планировщик автоматически подставляет значения двое суток и один узел. В предположении, что все задачи с такими значениями являются задачами с установками по умолчанию, когда пользователь игнорирует выбор значений для своей задачи, и был выполнен анализ.

На графиках добавлены значения среднего, медианы и квантилей для общей оценки распределений.

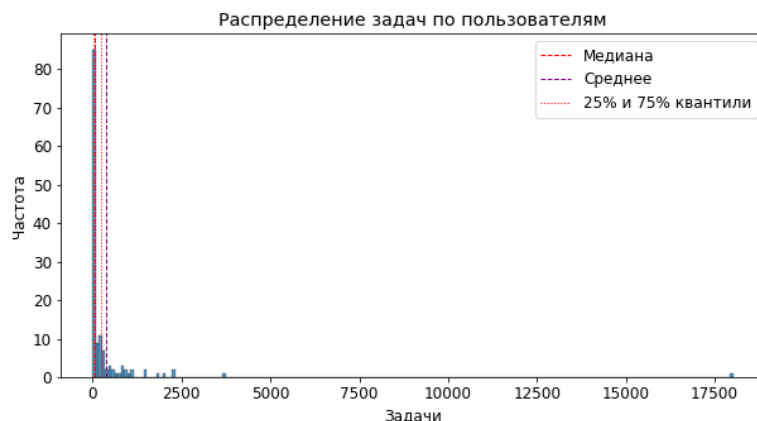
7.2 Анализ ресурсов, выделяемых пользователям

7.2.1 Анализ пользователей СКЦ

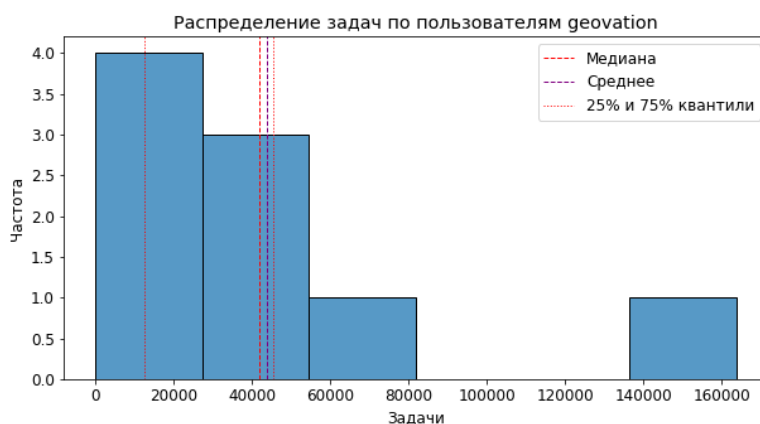
В этом параграфе приведены выделенные суммарные и средние ресурсы (размещённые задачи, время в СКЦ, количество вычислительных узлов и процессоров), распределённые по пользователям.

Ввиду того, что пользователи группы geovation были отделены от остальных, соотношение пользователей в двух частях набора данных стало равным 9 (geovation) и 136 соответственно.

На рисунке 7.3 показаны распределения задач по пользователям, не относящимся и относящимся к geovation соответственно. Большинство пользователей размещали в СКЦ менее 100-300 задач. Отдельные пользователи разместили от 500 до 18000 задач, поэтому распределение асимметрично. Около 7 пользователей geovation разместили менее 60 тыс. задач и двое – от 60 до 160 тыс. задач.



а)



б)

Рисунок 7.3 а) Распределение задач по пользователям, не относящимся к geovation; б) Распределение задач по пользователям geovation

Рисунки 7.4 и 7.5 иллюстрируют распределение суммарного, медианного, среднего времени выполнения задач и СКО по пользователям, не относящимся и относящимся к geovation соответственно. На первом графике из четырёх шкала времени показана логарифмической ввиду больших

значений суммарного времени выполнения задач отдельных пользователей. Графики указывают на то, что задачи более 90 пользователей выполнялись за очень короткое время. Однако по причине вклада нескольких пользователей с много большими значениями времени, распределение в целом асимметрично (медианные и средние значения не совпадают), как и распределение задач.

Распределение времени выполнения задач по пользователям

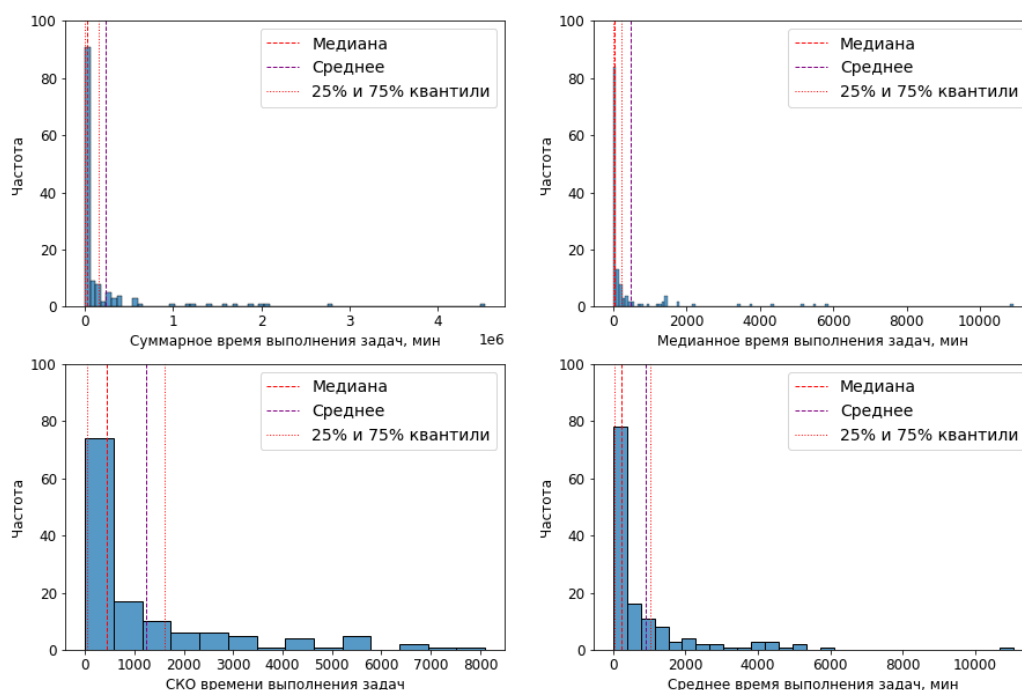


Рисунок 7.4 Распределение суммарного, медианного, среднего времени выполнения задач и СКО по пользователям, не относящимся к geovation

Распределение медианного времени выполнения задач пользователей geovation говорит о том, что задачи, размещённые более 8 пользователями, выполнялись за крайне короткие промежутки времени (менее 50 минут, в то время как график среднего времени показывает 200-500 минут), а задачи одного пользователя превысили 2500 минут. На этом примере можно убедиться в том, что поведение пользователей внутри группы может сильно отличаться.

Распределение времени выполнения задач по пользователям geovation

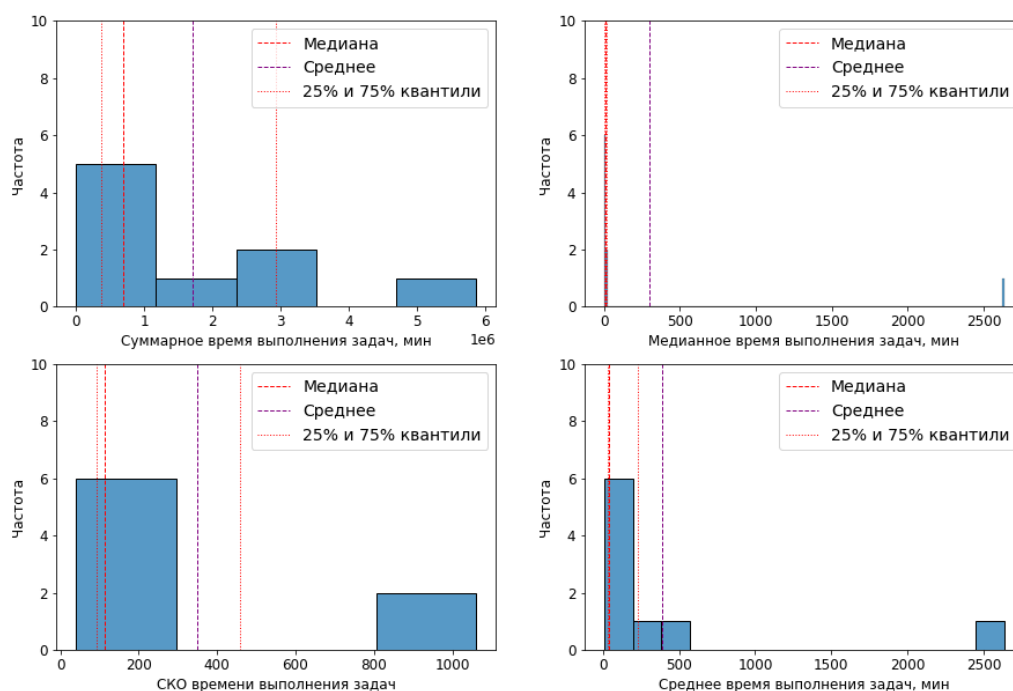


Рисунок 7.5 Распределение суммарного, медианного, среднего времени выполнения задач и СКО по пользователям geovation

Более 125 пользователей, не относящихся к geovation, в среднем использовали всего один вычислительный узел (рис. 7.6) и 56 процессоров (рис. 7.7) для выполнения задач. Все пользователи geovation использовали один узел (рис. 7.8) и в среднем 56 или 96 процессоров (рис. 7.9).

Распределение выделенных узлов по пользователям

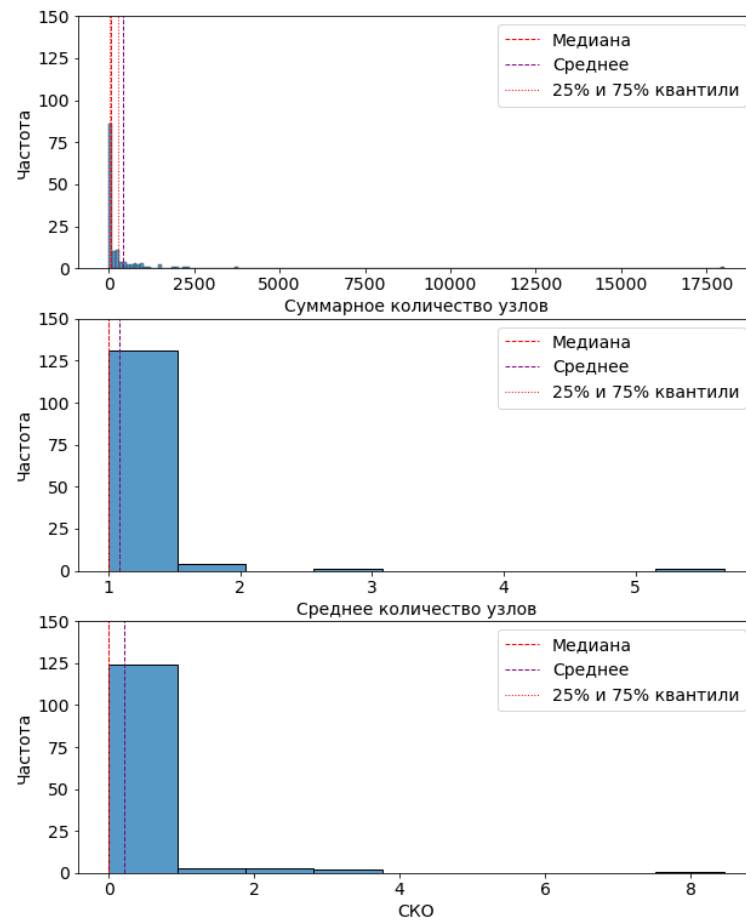


Рисунок 7.6 Распределение суммарного, среднего количества выделенных узлов и СКО по пользователям, не относящимся к geovation

Распределение выделенных процессоров по пользователям

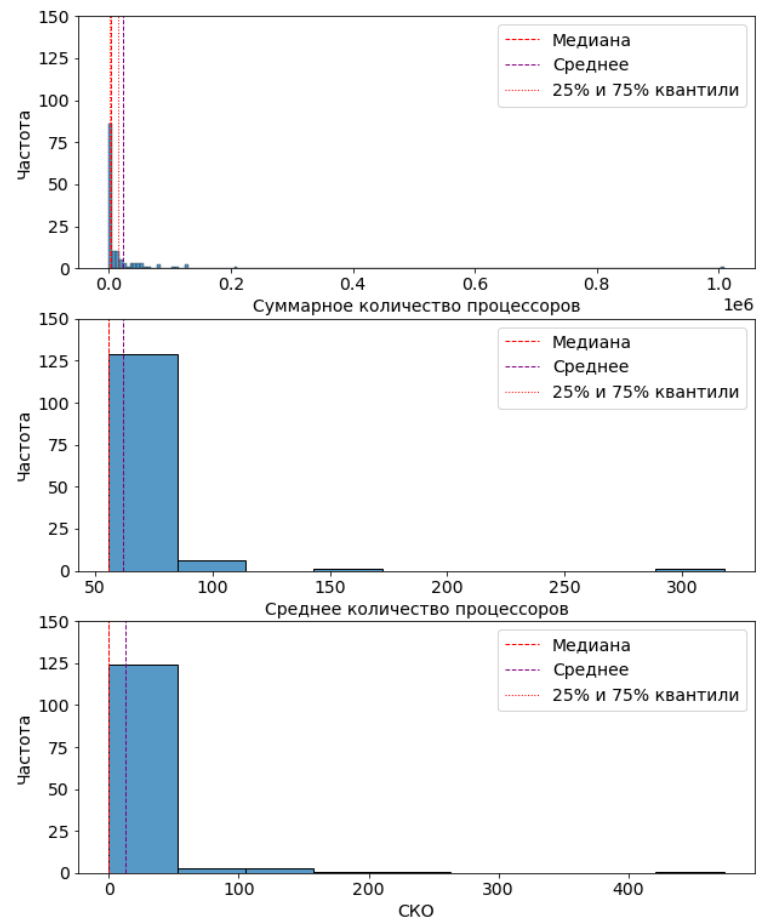


Рисунок 7.7 Распределение суммарного, среднего количества выделенных процессоров и СКО по пользователям, не относящимся к geovation

Распределение выделенных узлов по пользователям geovation

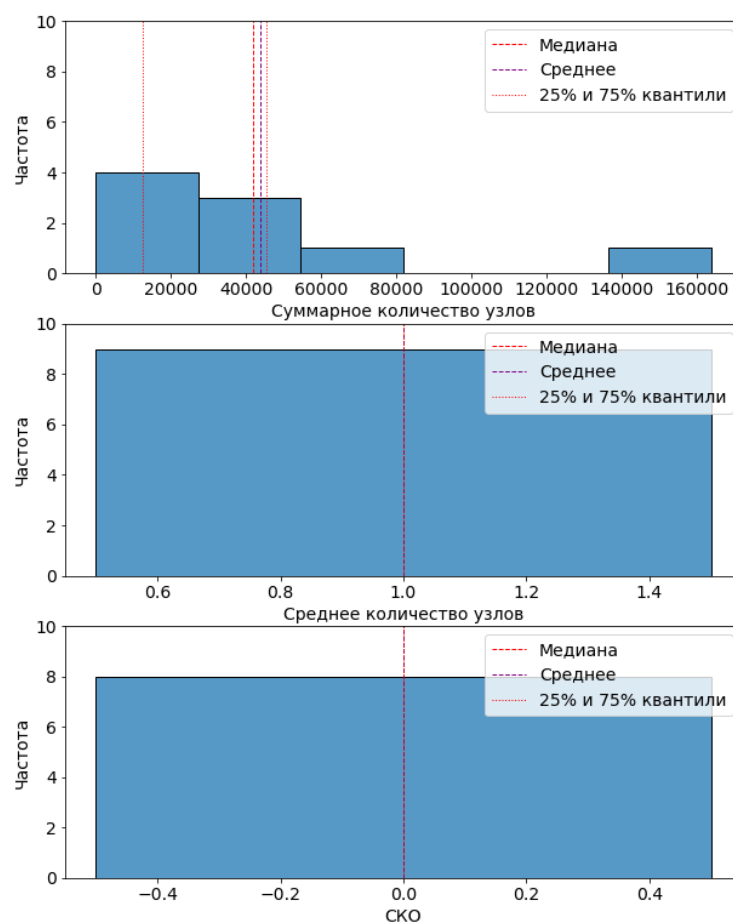


Рисунок 7.8 Распределение суммарного, среднего количества выделенных узлов и СКО по пользователям geovation

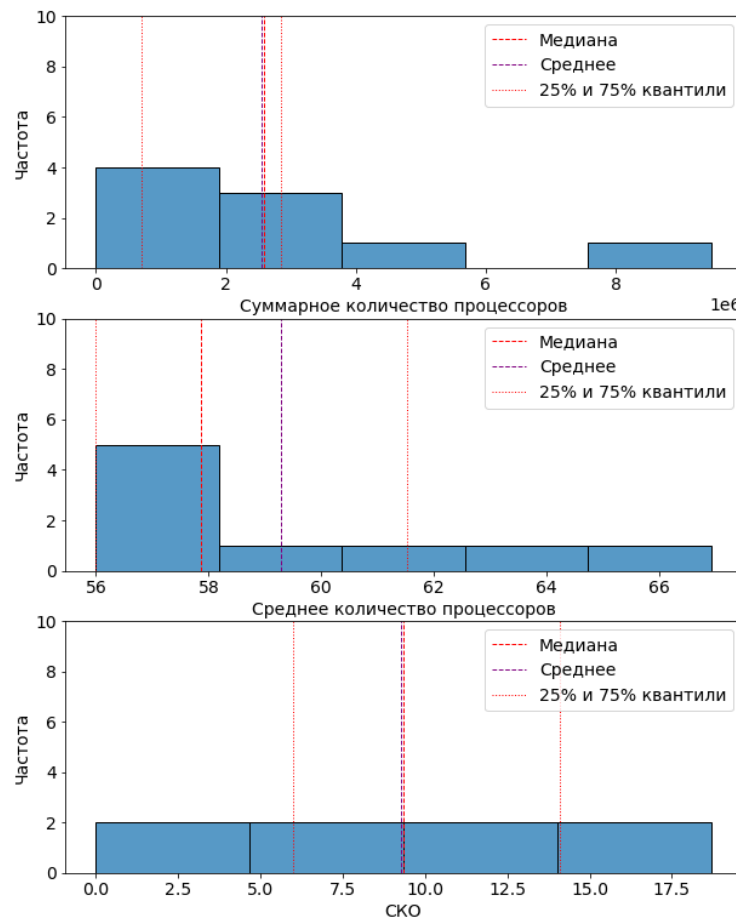


Рисунок 7.9 Распределение суммарного, среднего количества выделенных процессоров и СКО по пользователям geovation

7.2.2 Научные группы пользователей

Несмотря на то, что распределения суммарных и средних ресурсов, распределённых по научным группам пользователей, схожи с распределениями по пользователям, подобный анализ позволяет посмотреть на набор данных под другим углом и способен выявить дополнительные закономерности. Всего обнаружено 63 группы, включая geovation.

Большинство групп размещали в СКЦ менее 100-300 задач (рис. 7.10), что эквивалентно поведению пользователей в п. 7.2.1 настоящего отчета. Отдельные группы разместили от 1500 до 18000 задач, поэтому распределение асимметрично.



Рисунок 7.10 Распределение задач по группам пользователей

Задачи более 30 групп пользователей выполнялись за очень короткое время (рис. 7.11). Меньшая доля групп превысила 200 минут в медианных и 500 минут в средних значениях.

Распределение времени выполнения задач по группам пользователей

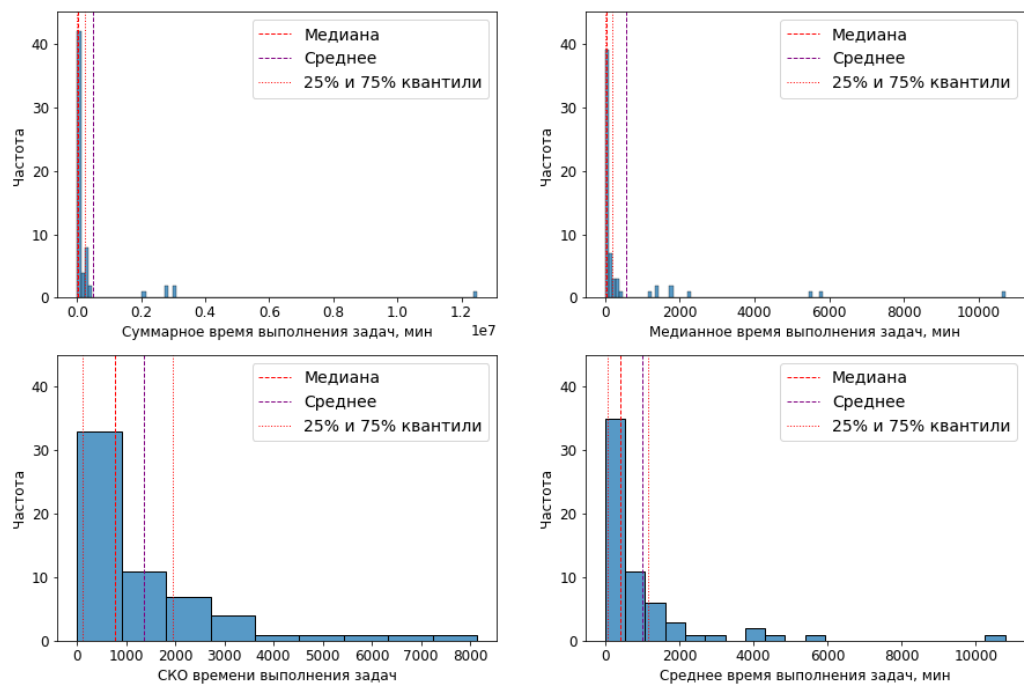


Рисунок 7.11 Распределение суммарного, медианного, среднего времени выполнения задач и СКО по группам пользователей

Более 50 групп в среднем использовали один вычислительный узел (рис. 7.12) и 56 процессоров (7.13) для выполнения задач.

Распределение выделенных узлов по группам пользователей

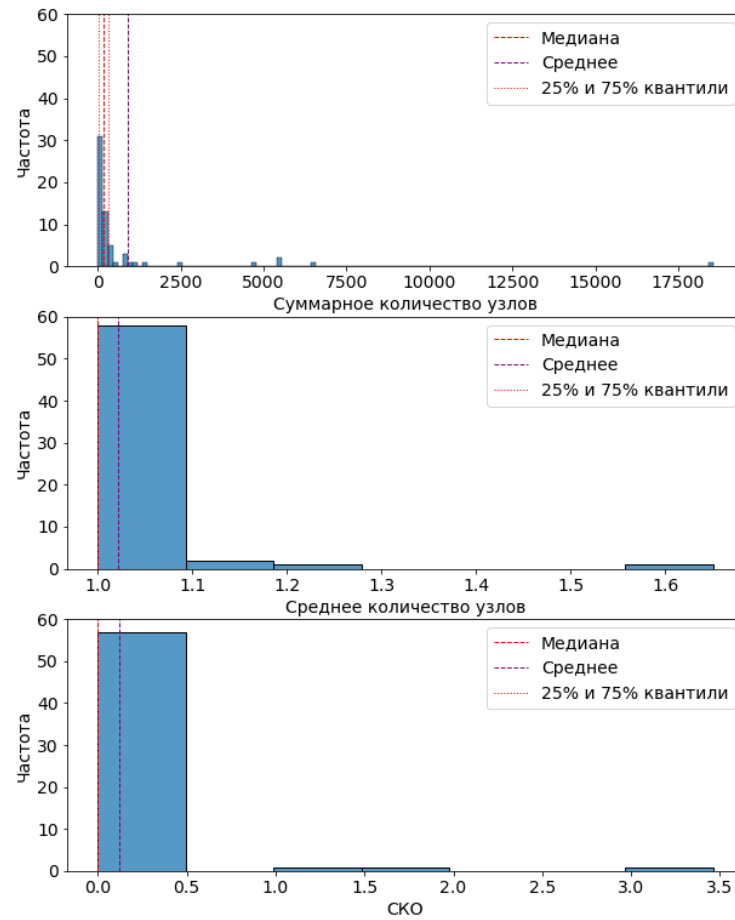


Рисунок 7.12 Распределение суммарного, среднего количества выделенных узлов и СКО по группам пользователей.

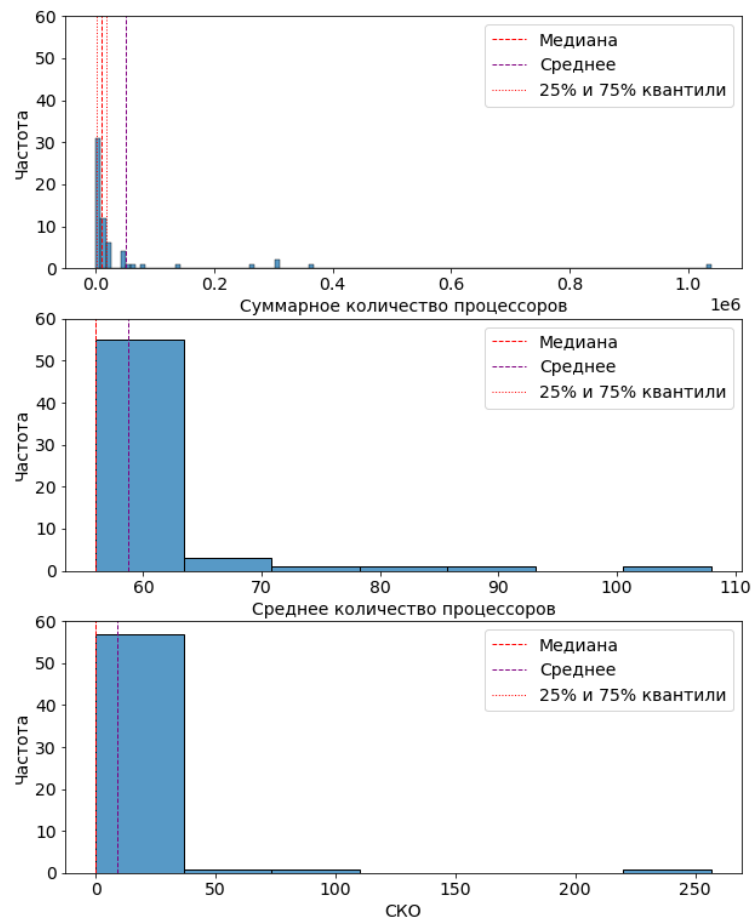


Рисунок 7.13 Распределение суммарного, среднего количества выделенных процессоров и СКО по группам пользователей

Рисунок 7.14 иллюстрирует распределение выделенных ресурсов группы geovation. Первые три графика показывают распределение времени выполнения задач в разных масштабах.

Группа пользователей geovation, чья доля задач составила 88% (более 350 тыс. задач) из всего набора данных, размещала задачи в основном короткой длительности менее 250 минут (более 250 тыс. задач). Лишь малая доля задач (менее 100) превысила 1000 минут. Все задачи выполнялись на одном вычислительном узле и в основном на 56 процессорах, небольшое количество выполнялось на 96 процессорах.

Распределения выделенных ресурсов группы geovation

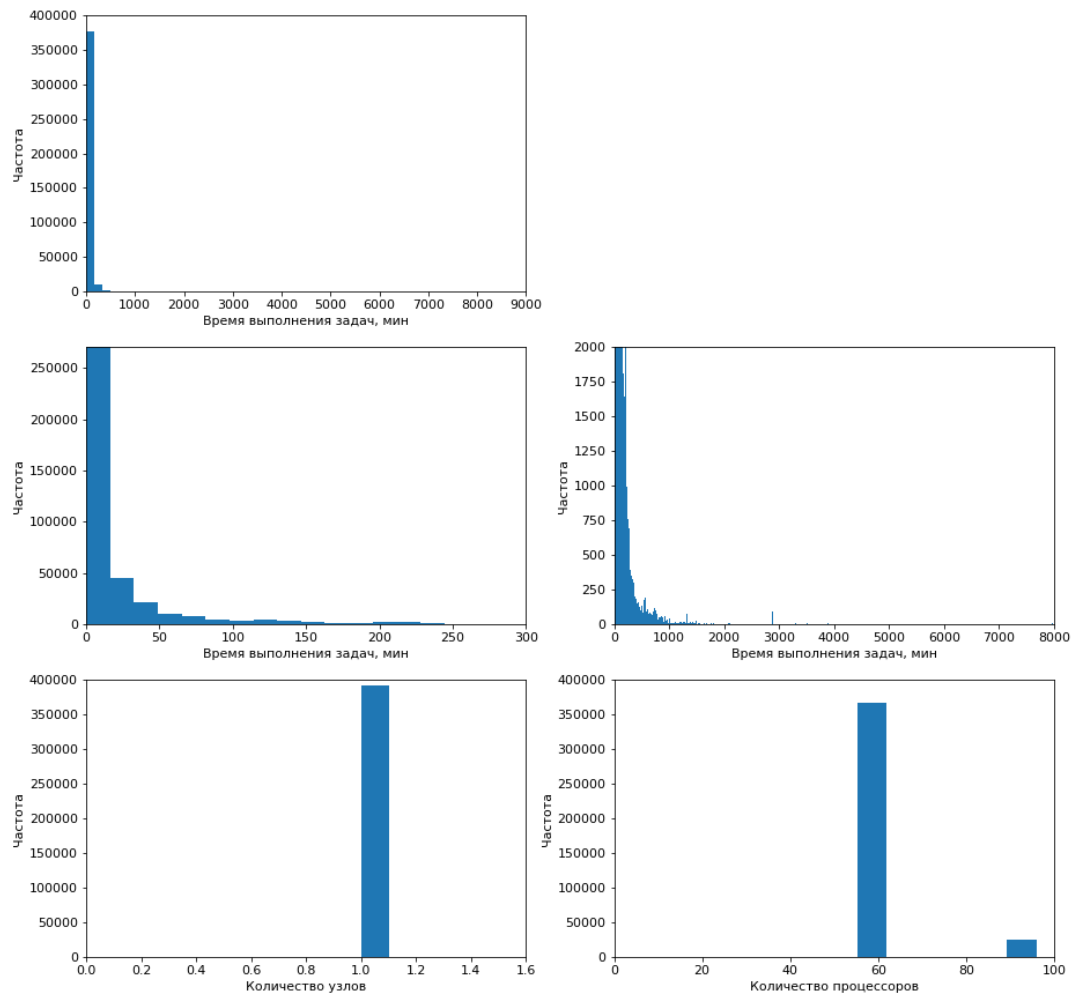


Рисунок 7.14 Распределение задач и долей задач с установками по умолчанию по пользователям группы geovation

Распределения ресурсов по группам показывают такие же результаты, как и по пользователям. В случае с пользователями данные были объединены по 136 и 9 пользователям, групп же получилось 63. В зависимости от цели исследования и постановки исследовательской задачи удобнее анализировать поведение пользователей или научных групп пользователей. Целесообразно также рассматривать поведение пользователей внутри групп, насколько похожи параметры задач, размещаемых пользователями одной группы.

7.2.3 Анализ задач по предметным областям

В отличие от пользователей и научных групп, было условно выделено всего десять предметных областей – ИТ (информационные технологии), астрофизика, биоинформатика, биофизика, энергетика, геофизика, механическая инженерия, механика, физика, радио физика. Группа пользователей geovation исключена из набора данных и была проанализирована в пп. 7.2.1 и 7.2.2 настоящего отчета.

Шесть из десяти предметных областей разместили менее 5 тыс. задач, две области разместили более 10 тыс., но менее 18 тыс. (рис. 7.15, 7.16).

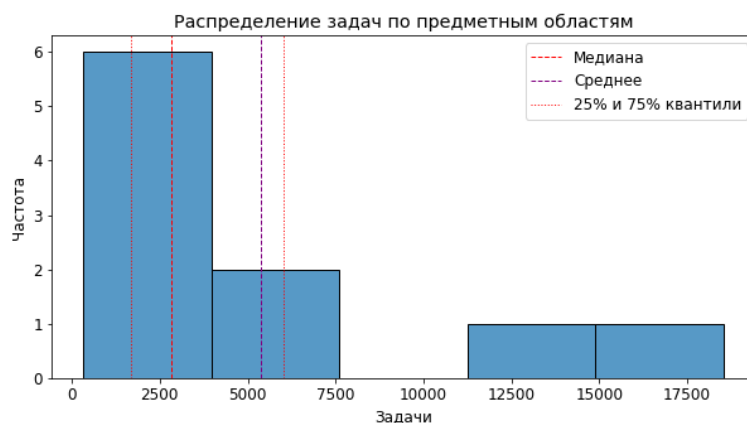


Рисунок 7.15 Распределение задач по предметным областям

Распределения медианного и среднего времени выполнения задач по предметным областям отличаются. Медианное время большинства областей не превышает 150 минут, тогда как среднее время распределяется между областями до более, чем 1750 минут.

Распределение времени выполнения задач по предметным областям

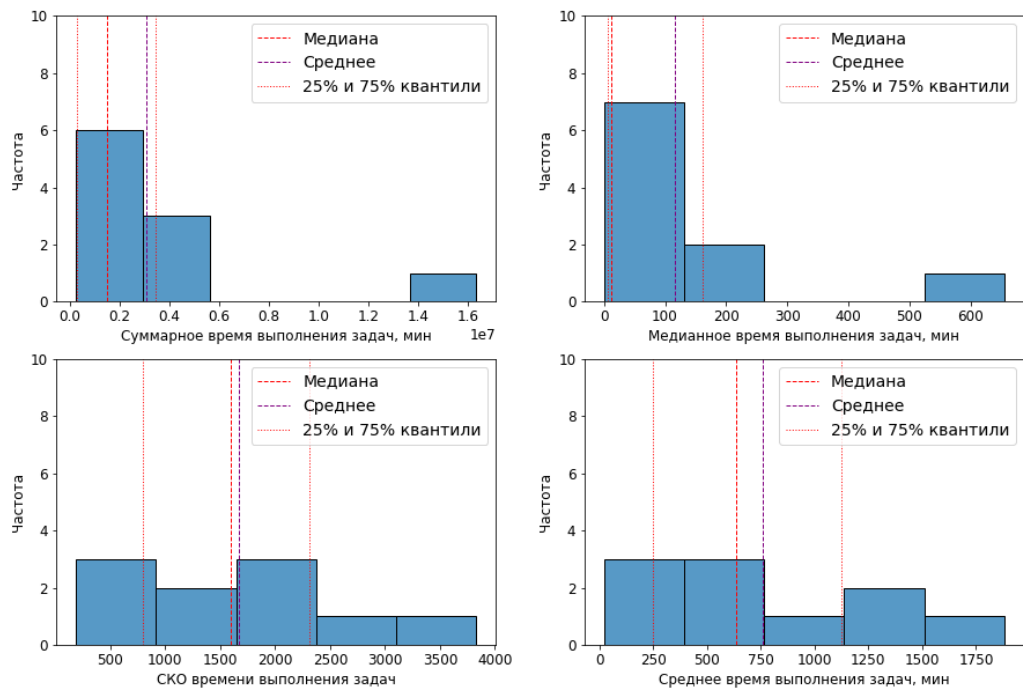


Рисунок 7.16 Распределение суммарного, медианного, среднего времени выполнения задач и СКО по предметным областям

Семь предметных областей в среднем использовали один вычислительный узел для выполнения задач (рис. 7.17). Пять областей в среднем использовали 56 процессоров (рис. 7.18). Значения остальных превышают 56 ввиду того, что по всей видимости, эти области использовали, как 56, так и 96 процессоров, а также другие значения одинаково часто.

Распределение выделенных узлов по предметным областям

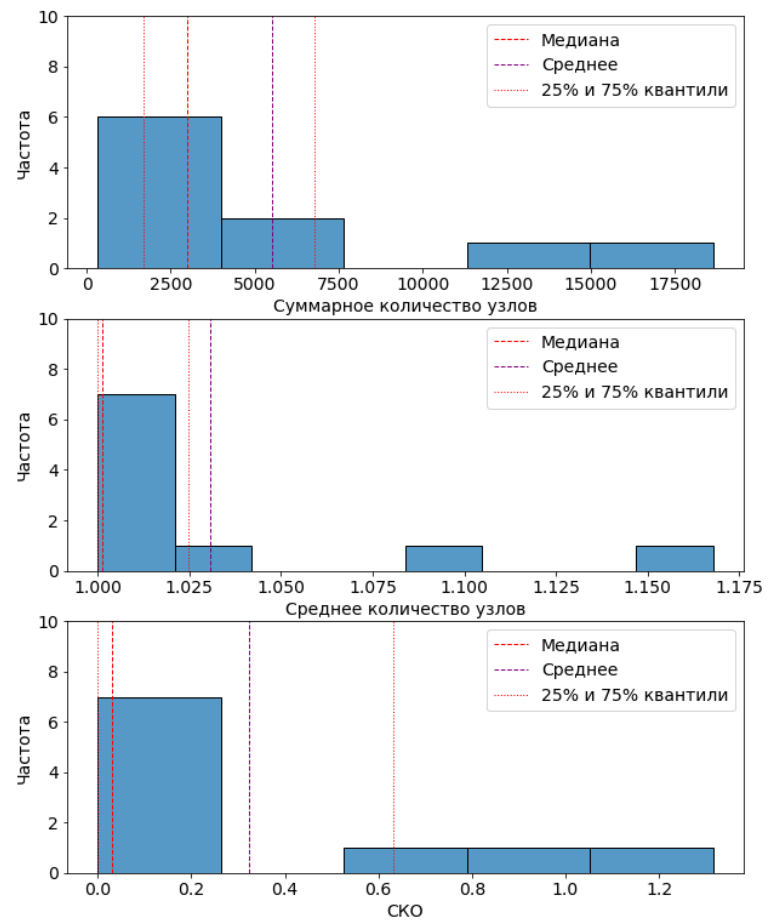


Рисунок 7.17 Распределение суммарного, среднего количества выделенных узлов и СКО по предметным областям

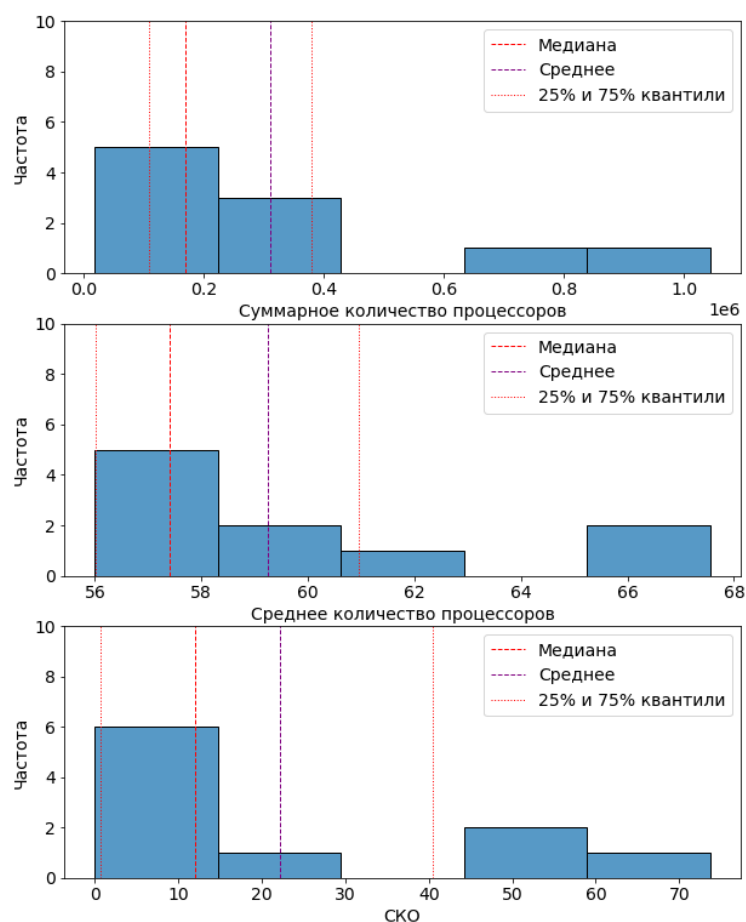


Рисунок 7.18 Распределение суммарного, среднего количества выделенных процессоров и СКО по предметным областям

Показатели по предметным областям отличаются от показателей по пользователям и группам. Различия обнаружены в распределениях времени выполнения задач и процессоров. Соотношения между областями не совпадают с соотношениями между пользователями и группами.

7.3 Оценка ресурсов, запрашиваемых пользователями

7.3.1 Анализ запрашиваемых ресурсов в разрезе пользователей

В этом параграфе приведены распределения задач и долей задач, недооценённых и переоценённых пользователями, задач, время для которых

пользователи оценили правильно с точностью 95% или ошиблись на $\pm 5\%$, а также распределения задач, долей задач с установками по умолчанию, недооценённых и переоценённых задач и долей задач среди них.

Доля недооценённых задач для более 90 пользователей близка к 0, в то время как доля переоценённых задач превышает 0,85 (рис. 7.19). Таким образом, пользователи в основном переоценивали время, необходимое для выполнения задач, и в редких случаях запрашиваемое время совпадало с выделенным на задачу с погрешностью $\pm 5\%$ (последняя пара графиков). Доля недооценённых задач пользователей geovation ничтожна, как и доля задач, время для которых было установлено с 95%-точностью, они переоценивали время почти всех своих задач (рис. 7.20).

Распределение задач и долей задач, оценённых с 95%-точностью, недооценённых и переоценённых пользователями

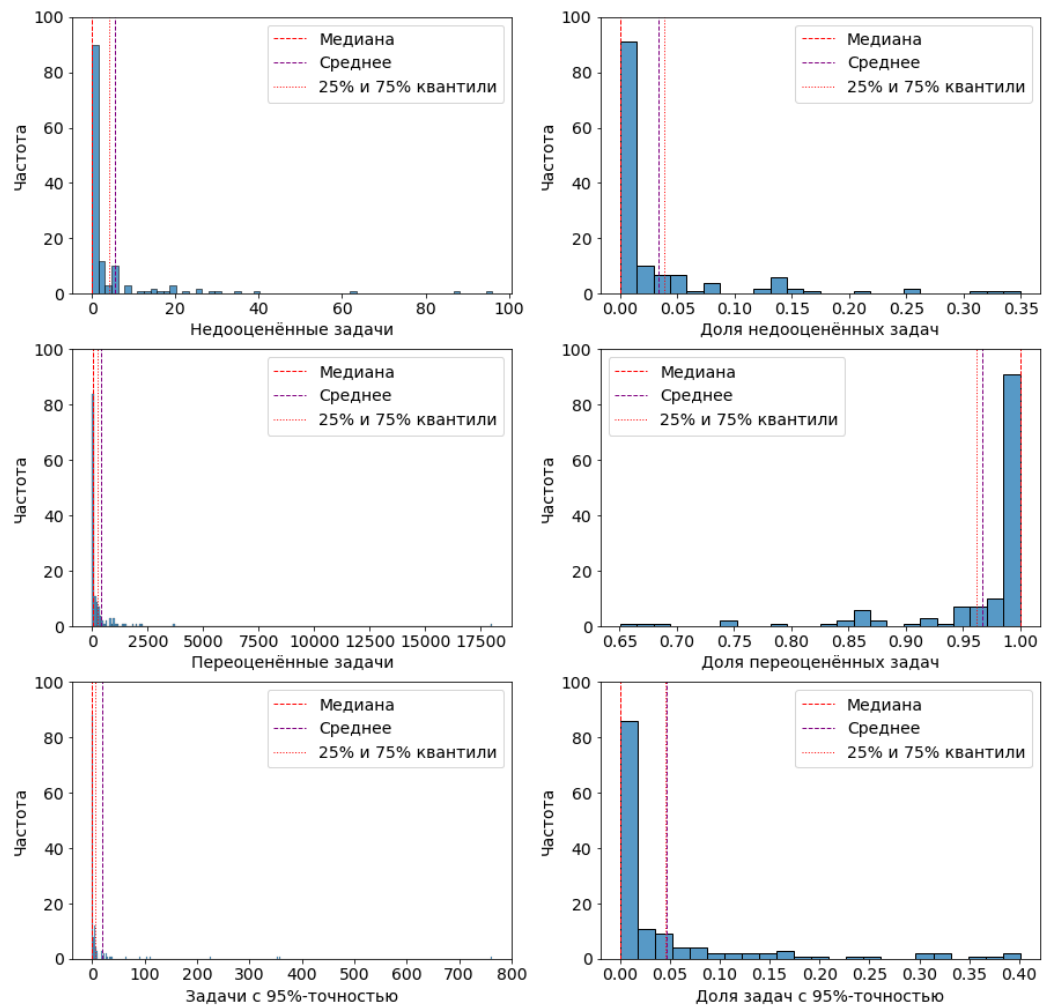


Рисунок 7.19 Распределение задач и долей задач, оценённых с точностью 95%, недооценённых и переоценённых пользователями, не относящимися к geovation

Распределение задач и долей задач, оценённых с 95%-точностью, недооценённых и переоценённых пользователями geovation

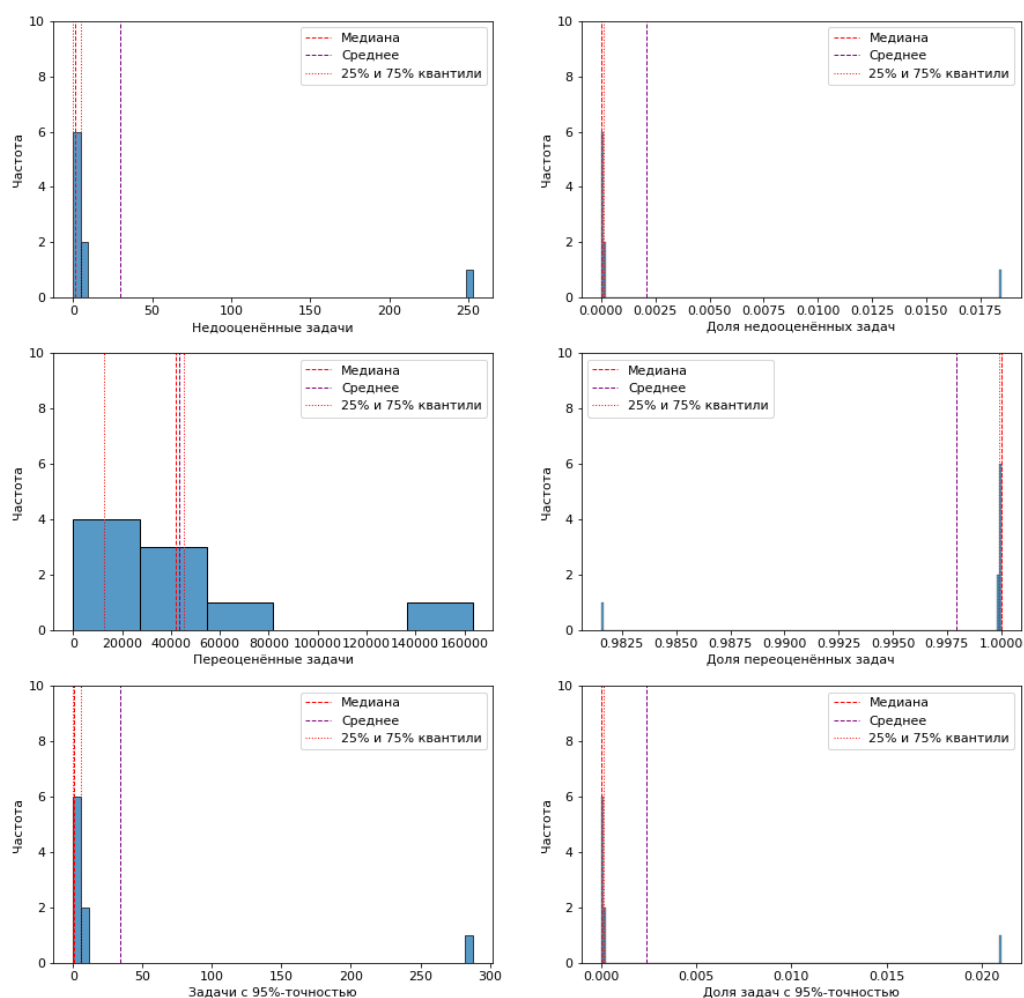


Рисунок 7.20 Распределение задач и долей задач, оценённых с точностью 95%, недооценённых и переоценённых пользователями geovation

Более 120 пользователей не размещали задачи с установками по умолчанию (рис. 7.21), несколько пользователей размещали различные доли таких задач (от 0,2 до 1). Те, кто размещал подобные задачи, в основном переоценивали время, необходимое для выполнения задачи. Среди пользователей geovation доля задач с установками по умолчанию оказалась крайне высока – от 0,7 до 1, и почти все они были переоценены (рис. 7.22).

Распределение задач и долей задач с установками по умолчанию

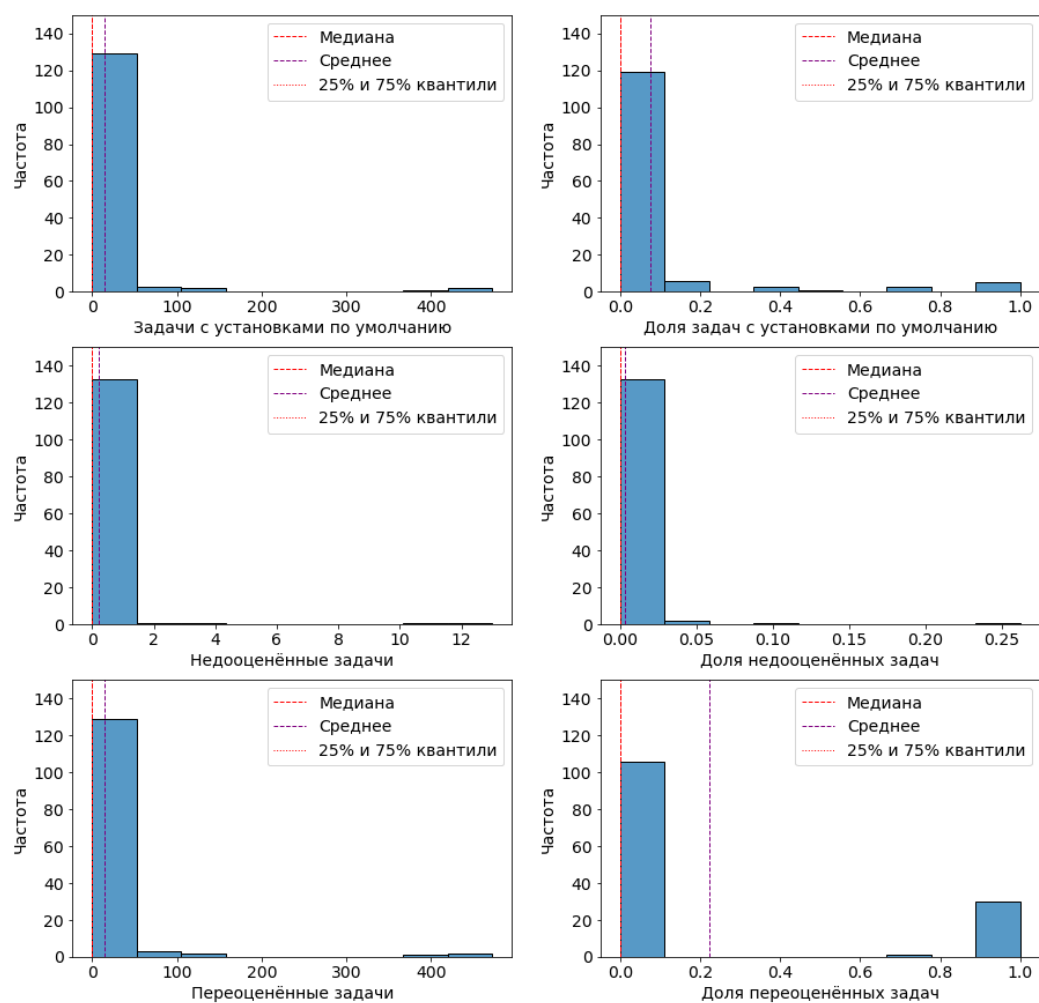


Рисунок 7.21 Распределение задач и долей задач с установками по умолчанию по пользователям

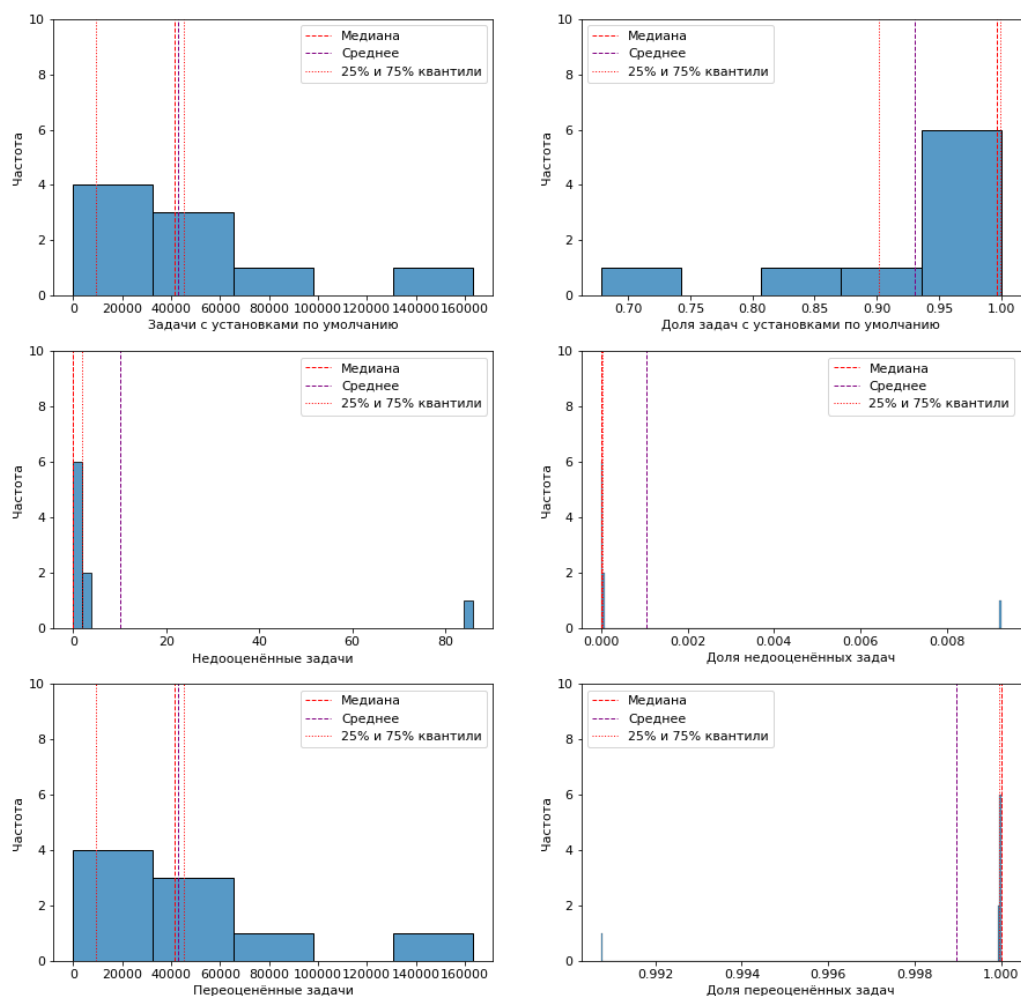


Рисунок 7.22 Распределение задач и долей задач с установками по умолчанию по пользователям geovation

Обобщая результаты исследования, приведённые в этом параграфе, можно сделать вывод о том, что пользователи geovation размещали преимущественно задачи с установками по умолчанию и тем самым переоценивали время, необходимое для выполнения своих задач, тогда как среди остальных пользователей мало кто размещал подобные задачи, но в целом и остальные пользователи переоценивали время для размещённых задач.

7.3.2 Анализ запрашиваемых ресурсов в разрезе групп пользователей

Те же оценочные показатели ресурсов, что и в п. 7.3.1 настоящего отчета для пользователей, рассматривались для групп пользователей.

Доля недооценённых задач для подавляющего большинства групп (немногим меньше 60) составляет менее 0,15, в то время как доля переоценённых задач превышает 0,85, а для более, чем 35 групп равна 1 (рис. 7.23). Группы, как и пользователи, в основном переоценивали время, необходимое для выполнения задач, и в редких случаях запрашиваемое время совпадало с выделенным на задачу с погрешностью $\pm 5\%$.

Распределение задач и долей задач, оценённых с 95%-точностью, недооценённых и переоценённых группами

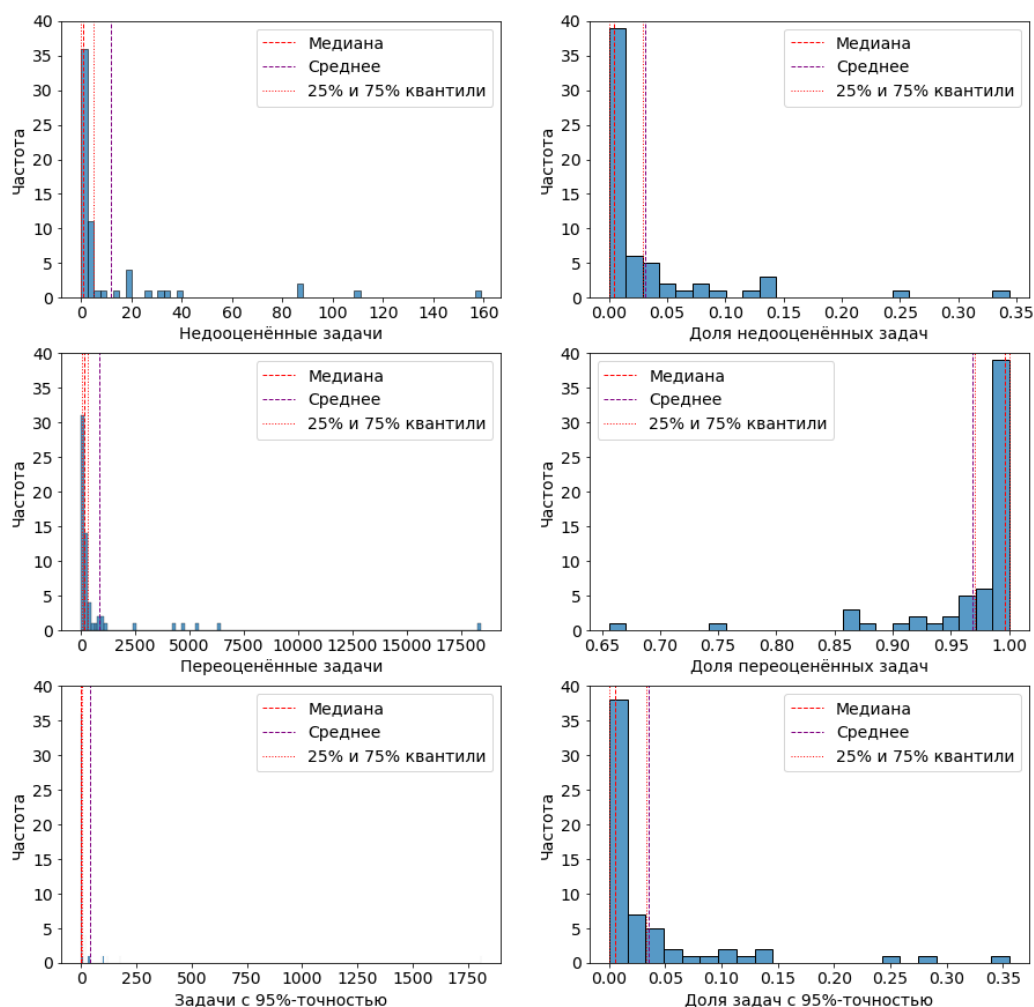


Рисунок 7.23 Распределение задач и долей задач, оценённых с точностью 95%, недооценённых и переоценённых группами пользователей

Рисунок 7.24 иллюстрирует распределение задач и долей задач с установками по умолчанию по группам пользователей. Более 45 групп не имели таких задач, остальные размещали малые доли задач с установками по умолчанию, за редким исключением. Те, кто всё-таки размещал подобные задачи, в основном переоценивали время, необходимое для выполнения задачи.

Распределение задач и долей задач с установками по умолчанию

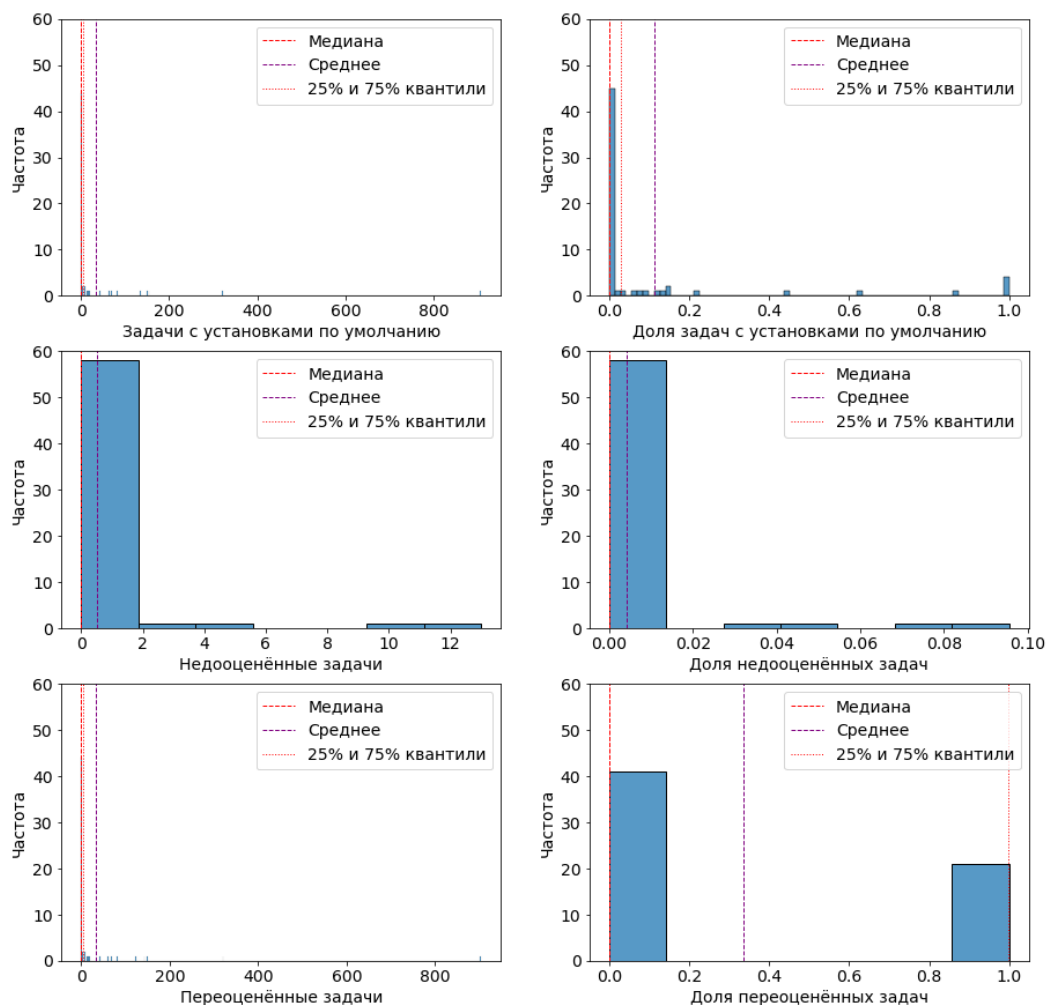


Рисунок 7.24 Распределение задач и долей задач с установками по умолчанию по группам пользователей

Группа geovation переоценивала время, необходимое для выполнения задач в 99,9% случаев и разместила 98% задач с установками по умолчанию, 99,98% из которых оказались переоценёнными (табл. 7.1).

Таблица 7.1 Доли задач, недооценённых и переоценённых пользователями geovation, доли задач с установками по умолчанию

Показатель	Значение
Доля недооценённых задач	0.000686
Доля переоценённых задач	0.999314
Доля задач, оценённых с точностью 95%	0.000783
Доля задач с установками по умолчанию	0.982884
Доля недооценённых задач среди задач с установками по умолчанию	0.000239
Доля переоценённых задач среди задач с установками по умолчанию	0.999761

7.3.3 Анализ запрашиваемых ресурсов в разрезе предметных областей

Так же, как и в п. 7.2.3 настоящего отчета, при рассмотрении набора данных, объединённого по предметным областям, группа пользователей geovation исключена и была проанализирована в пп.7.3.1 и 7.3.2 настоящего отчета.

Доля задач, недооценённых предметными областями составила менее 0,07, а доля переоценённых – более 0,94 и доля точно оценённых – менее 0,08 для 8 из 10 областей (рис. 7.25). По сравнению с распределениями для пользователей и групп распределение для предметных областей выглядит ещё более радикально в сторону переоценки времени выполнения задач.

Распределение задач и долей задач, оценённых с 95%-точностью, недооценённых и переоценённых по предметным областям

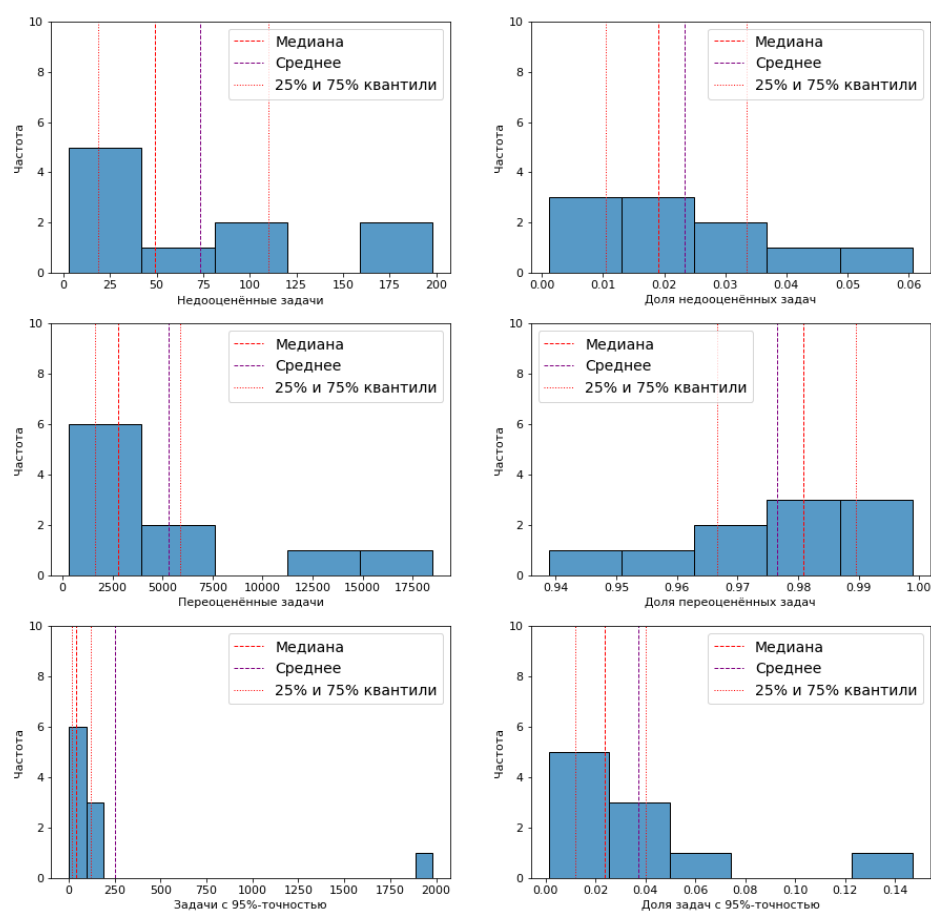


Рисунок 7.25 Распределение задач и долей задач, оценённых с точностью 95%, недооценённых и переоценённых предметными областями

Распределение долей задач с установками по умолчанию располагается в диапазоне от 0 до 0,16 для предметных областей, среди них крайне мало недооценённых задач, а доля переоценённых составила менее 0,2 в трёх областях и от 0,8 до 1 в остальных областях (рис. 7.26).

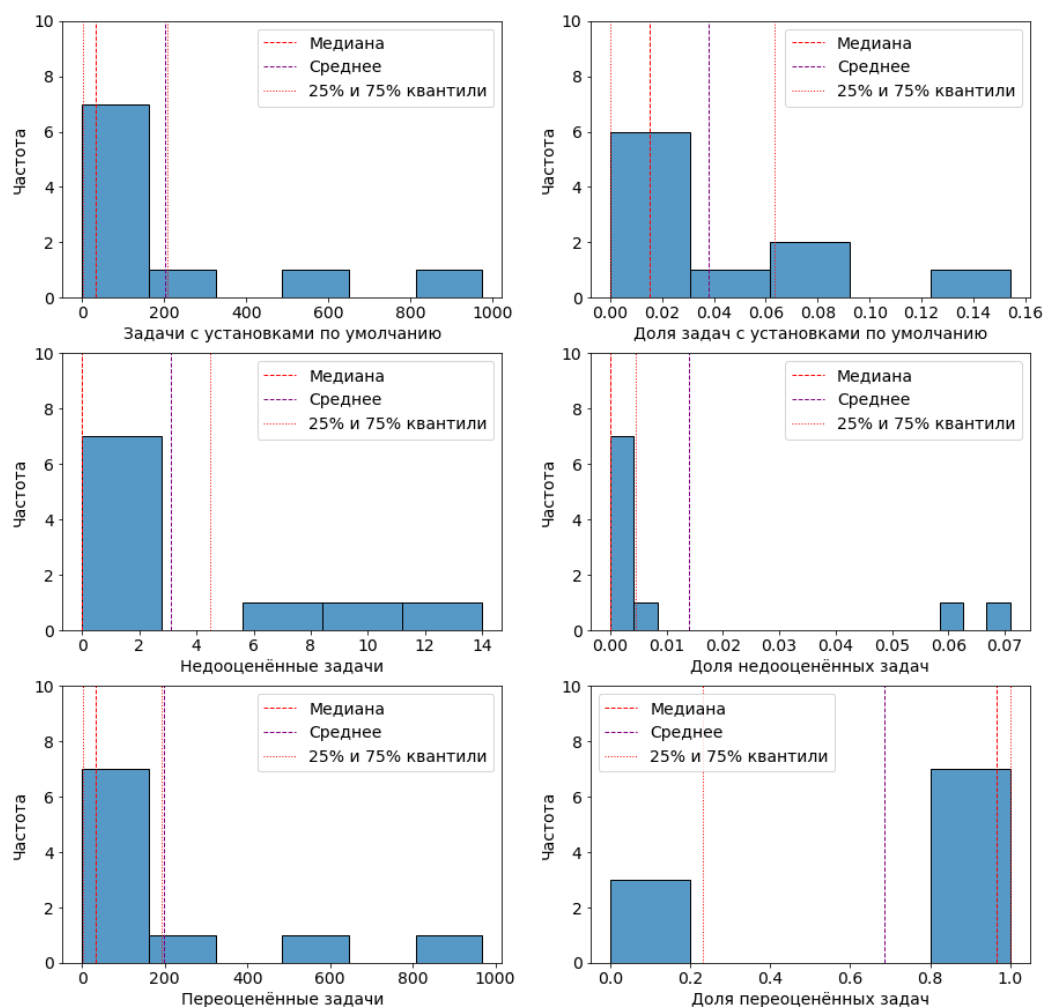


Рисунок 7.26 Распределение долей задач с установкой по умолчанию

Как и в п. 7.2 настоящего отчета при анализе показателей выделенных ресурсов, распределения показателей оценки времени по предметным областям отличаются от распределений по пользователям и научным группам, поскольку распределения на предметные области являются более крупными объединениями данных.

7.4 Анализ предполагаемых зависимостей между различными факторами и целевой величиной

В этом параграфе приведены результаты анализа предполагаемых корреляций между атрибутами задач (факторами), имеющимися в наборе данных, и временем выполнения задачи, принимаемым здесь в качестве целевой величины.

Рисунок 7.27 иллюстрирует вектор корреляций между целевой величиной и 12 наиболее значимыми с точки зрения корреляции факторами, а на рисунке 7.28 показана матрица корреляций между самими факторами. На рисунках видно, что наиболее статистически значимым фактором с точки зрения оценки времени выполнения задачи является `TimelimitRaw` – время, запрошенное пользователем – с коэффициентом корреляции 0.369. Факторы `State` (состояние задачи), `User` (пользователь) и `AssocID` (идентификатор ассоциации, к которой принадлежит пользователь) имеют коэффициенты корреляции с целевой величиной, равные -0.255, 0.233 и -0.232 соответственно, при низкой корреляции друг с другом, что говорит о наличии независимой статистической информации. Высокую корреляционную зависимость 0.53 имеют факторы `User` и `ReqTRES` (запрашиваемые ресурсы), что говорит о взаимосвязи между пользователем и числом запрашиваемых ресурсов. Один из факторов загрузки `WCKeyID` (идентификатор ключа) имеет корреляцию -0.111 с целевой величиной, однако также высокую корреляцию 0.763 с фактором `AssocID`, что свидетельствует о малом количестве статистической информации в этом факторе. Остальные факторы `Account_saact` (аккаунт, с которого была запущена задача), `NCPUs_scontrol` (количество выделенных процессоров), `GroupID_saact` (идентификатор группы пользователя), `ExitCode_saact` (код выхода задачи), `WCKey_saact` (ключ) имеют низкую корреляцию менее 0.1 с целевой величиной и не выглядят статистически значимыми.

TimelimitRaw	0.36900
State	-0.25500
User	0.23300
AssocID	-0.23200
ReqTRES	0.12500
TRES	-0.11100
WCKeyID	-0.11100
Account_sacct	-0.06200
NCPUs_scontrol	0.04200
GroupID_sacct	-0.01500
ExitCode_sacct	0.00200
WCKey_sacct	0.00100

Рисунок 7.27 Вектор корреляций между целевой величиной и 12 наиболее значимыми с точки зрения корреляции факторами

	Account_sacct	AssocID	ExitCode_sacct	GroupID_sacct	NCPUs_scontrol	ReqTRES	State	TRES	TimelimitRaw	User	WCKeyID	WCKey_sacct
Account_sa...	1.00000	-0.29400	-0.15100	-0.05100	-0.19900	-0.30700	0.18700	0.19400	-0.18000	-0.49000	-0.32000	-0.17600
AssocID	-0.29400	1.00000	0.07200	0.06800	0.05000	0.10400	0.06100	0.04300	-0.33900	0.03000	0.76300	0.56300
ExitCode_s...	-0.15100	0.07200	1.00000	-0.00200	0.23000	0.24400	-0.29900	0.04900	0.23500	0.06000	0.06000	0.08500
GroupID_sa...	-0.05100	0.06800	-0.00200	1.00000	0.00200	0.00500	-0.01000	0.01800	-0.00600	-0.01000	0.04300	0.03900
NCPUs_sco...	-0.19900	0.05000	0.23000	0.00200	1.00000	0.20400	-0.06400	-0.24000	0.13800	0.23000	-0.02900	0.09800
ReqTRES	-0.30700	0.10400	0.24400	0.00500	0.20400	1.00000	-0.02500	-0.56200	0.19200	0.53100	0.29600	0.23000
State	0.18700	0.06100	0.04400	-0.01000	-0.06400	-0.02500	1.00000	0.01900	-0.27800	-0.14100	0.02000	-0.07700
TRES	0.19400	0.04300	-0.29900	0.01800	-0.24000	-0.56200	0.01900	1.00000	-0.10800	-0.32800	-0.02200	0.06400
TimelimitRa...	-0.18000	-0.33900	0.04900	-0.00600	0.13800	0.19200	-0.27800	-0.10800	1.00000	0.27300	-0.05200	0.14400
User	-0.49000	0.03000	0.23500	-0.01000	0.23000	0.53100	-0.14100	-0.32800	0.27300	1.00000	0.18200	0.25200
WCKeyID	-0.32000	0.76300	0.06000	0.04300	-0.02900	0.29600	0.02000	-0.02200	-0.05200	0.18200	1.00000	0.73900
WCKey_sac...	-0.17600	0.56300	0.08500	0.03900	0.09800	0.23000	-0.07700	0.06400	0.14400	0.25200	0.73900	1.00000

Рисунок 7.28 Матрица корреляций между наиболее значимыми факторами

Заключение

В результате анализа поведения пользователей и взаимодействия ресурсов СКЦ на наборе данных за период с середины июня по середину ноября 2022 года был отмечен ряд важных для исследования выводов:

1) 88% задач было размещено одной научной группой пользователей автоматизированным образом с помощью вспомогательного программного обеспечения. Все задачи являлись задачами с установками по умолчанию – двое суток и один вычислительный узел, тогда как большинство задач выполнялось менее 250 минут. Таким образом, пользователи группы переоценивали время задач, вынуждая планировщик перераспределять ресурсы после окончания коротких задач, когда было запрошено много

большее время, и ресурсы освобождались. Задачи одного из пользователей превысили 2500 минут и вычислялись на одном узле. Здесь можно было бы предложить пользователю больше узлов, тогда его задачи выполнялись бы быстрее. В целом поведение пользователей внутри группы отличается. Нельзя сказать, что пользователи одной группы размещают задания одинаковой длительности,

2) большая часть остальных пользователей разместила в СКЦ менее 100-300 задач, а отдельные пользователи – от 500 до 18000 задач. Задачи выполнялись в основном за очень короткое время – менее 150 минут по медианному времени для большинства предметных областей, за исключением долгих задач отдельных пользователей, что привело к асимметричному распределению времени. Для выполнения задач большинства пользователей в среднем был задействован всего один вычислительный узел и в основном 56 процессоров, в некоторых случаях – 96. Здесь, как и в случае geovation, можно было бы предложить пользователям больше вычислительных узлов для продолжительных задач, таким образом обеспечив как равномерную загрузку СКЦ и тем самым равномерные энергопотребление и износ оборудования, так и ускоренное выполнение задачи, что важно для пользователя. Пользователи переоценивали время, необходимое для выполнения задач (доля переоценённых задач превышает 0,85) и размещали мало задач с установками по умолчанию, пытаясь всё же угадать время, но также переоценивая его,

3) распределения показателей по пользователям, научным группам и предметным областям несколько отличаются, поскольку распределение на предметные области является более крупным объединением данных по отношению к группам (10 областей и 63 группы), а группы – к пользователям (145 пользователей). Например, доля переоценённых задач среди пользователей и групп составляет более 0,85, а среди предметных областей – более 0,94. В зависимости от цели исследования и постановки исследовательской задачи удобнее анализировать поведение пользователей, научных групп пользователей или предметных областей. Разукрупнение

объединений данных помогает анализировать поведение отдельных пользователей или групп на фоне предметных областей в целом и выявлять возможные скрытые свойства,

4) для предсказания времени выполнения нельзя использовать средние значения по пользователю или группе, поскольку распределения асимметричны, и средние значения не совпадают с медианными. Использование средних значений неизбежно приведёт к ошибкам.

Анализ возможных зависимостей между атрибутами задачи, имеющимися в наборе данных, и временем выполнения задачи показал, что из всех рассмотренных атрибутов только время, запрошенное пользователем на задачу, значимо влияет на итоговое время выполнения задачи.

Продолжение работы по изучению поведения пользователей и влияния различных атрибутов из набора данных на время выполнения задачи, подбору и построению моделей, на которые можно было бы опираться в решении глобальной задачи оптимизации загрузки СКЦ, заключается в исследовании поведения пользователей внутри научных групп, насколько похожи параметры задач, размещаемых пользователями одной группы, а также ключевым образом в получении дополнительных данных о размещаемых задачах пользователей. Например, какие функции каких библиотек вызывает пользователь для выполнения своей задачи, по этой информации можно определить наиболее и наименее ресурсоёмкие функции библиотек и вычислить необходимый объём ресурсов с помощью алгоритмов машинного обучения до постановки задачи в очередь. Другой интересной особенностью задач является точность вычислений, закладываемая пользователем, данные о которой так же пока недоступны. Очевидно, что задачи с требованием высокой точности расчёта потребуют больше ресурсов, чем задачи с требованием низкой точности расчёта.

Реализация возможностей получения дополнительных данных, описанных в п. 7.1.2, поможет достичь обозначенных целей дальнейших этапов исследования. Имеющиеся в планировщике механизмы учёта

потребляемых ресурсов – электроэнергии, файловых систем и сетевого трафика потребуют настройки аппаратных и программных подсистем. Для координации систем мониторинга и диспетчеризации таких, как Ganglia и других с планировщиком Slurm понадобится разработка протоколов соответствия данных и соответствующей настройки систем, а также последующее их внедрение.

8 Разработка аппаратно-программного макетного образца мультиагентного диспетчера ресурсов гетерогенного вычислительного кластера

8.1. Разработка лабораторного стенда вычислительного кластера

АППАРАТНАЯ ЧАСТЬ

Для выполнения экспериментов, состоящих в выполнении пользовательской вычислительной нагрузки в полностью контролируемом окружении, протоколировании параметров запуска и выполнения задач, настройке и тестировании параметров окружения без влияния на реальные производственные процессы с использованием действующего СКЦ, была спроектирована функциональная схема лабораторного стенда, состоящая из:

- управляющего узла УУ-1;
- вычислительных узлов ВУ-1..ВУ-3, ВУ-5;
- беспроводного маршрутизатора РТ-1;
- проводных и беспроводных каналов связи.

Функциональная схема приведена на рисунке 8.1.

Данная схема была также успешно собрана в полноценный вычислительный кластер. На него также установлено программное обеспечение, которое настроено с максимальным приближением к режиму работы СКЦ.

Особенностью управляющего узла УУ-1 является его реализация в виде виртуальной машины в среде виртуализации VirtualBox, что позволяет гибко работать с различными программными конфигурациями, сохраняя их состояние в виде так называемых снапшотов – мгновенных снимков файловой подсистемы – с возможностью в любой момент восстановить ранее сохранённое состояние за минимальное время (порядка нескольких секунд). Также возможно подготовить несколько различных программных конфигураций, сохранить их в виде отдельных файлов виртуальных жёстких дисков и оперативно переключаться между ними для выполнения различных

экспериментальных задач. Эмулируется следующая конфигурация аппаратного обеспечения:

- два ядра центрального процессора Intel Core i7-10700 (наследуется от физической машины, на которой выполняется виртуальная среда);
- оперативная память DDR4 объёмом 2 ГБ;
- жёсткий диск объёмом 50 ГБ.

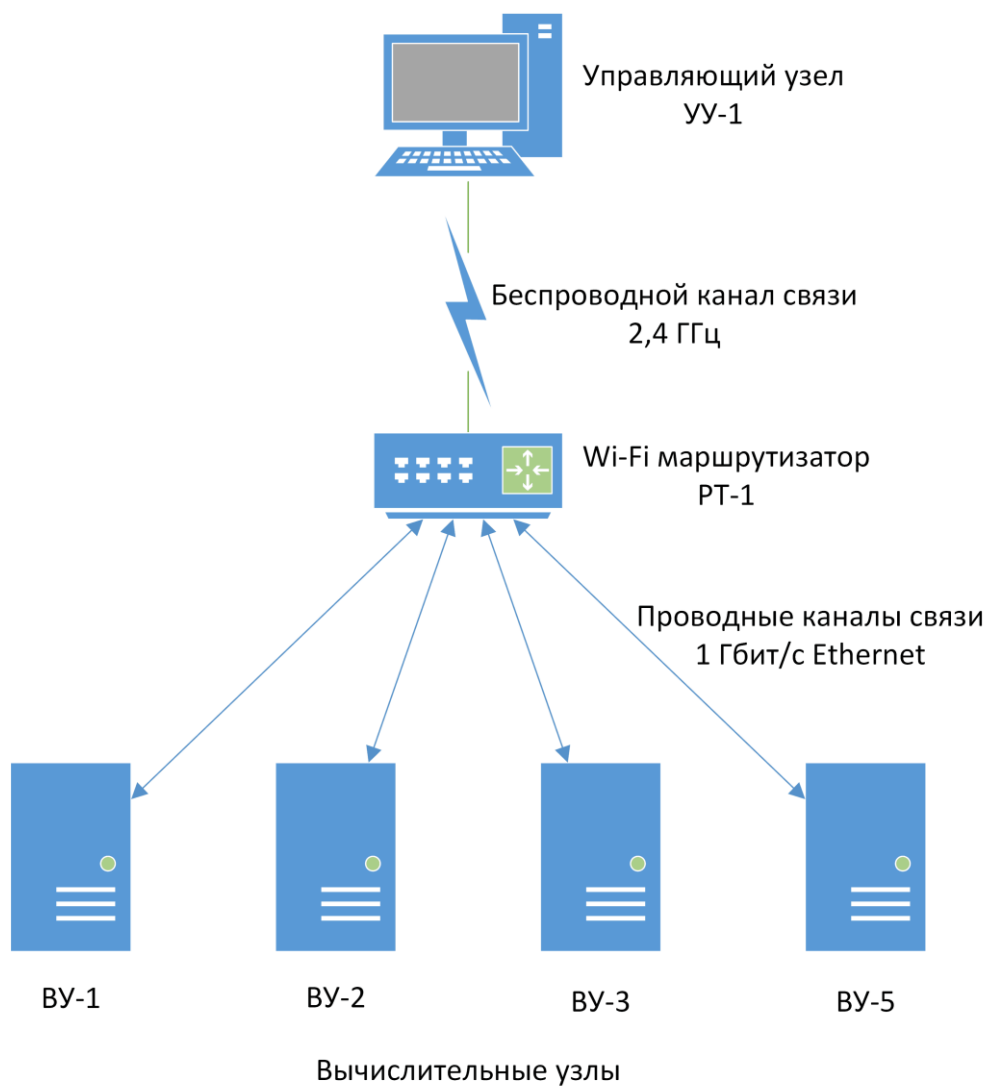


Рисунок 8.1 Функциональная схема лабораторного стенда

Необходимо заметить, что виртуальная среда в данном случае не создаёт дополнительных ограничений для выполнения экспериментов, поскольку пользовательские задания выполняются только на вычислительных узлах, где для них непосредственно доступны аппаратные вычислительные ресурсы.

Вычислительные узлы ВУ-1..ВУ-3 предназначены для выполнения пользовательских заданий, задействующих ресурсы процессоров общего назначения, и представлены следующей конфигурацией аппаратного обеспечения:

- 4-ядерный центральный процессор Intel Core i5-4440;
- оперативная память DDR3 объёмом 8 ГБ;
- жёсткий диск объёмом 320 ГБ.

Вычислительный узел ВУ-5 предназначен для выполнения пользовательских заданий, задействующих ресурсы не только процессора общего назначения, но и графического ускорителя посредством модели CUDA. Конфигурация аппаратного обеспечения данного вычислительного узла следующая:

- 4-ядерный (8 «виртуальных» ядер) центральный процессор Intel Core i7-2600;
- графический ускоритель Nvidia Quadro K6000;
- оперативная память DDR3 объёмом 16 ГБ;
- жёсткий диск объёмом 320 ГБ;

Таким образом, используя различные конфигурации аппаратного обеспечения вычислительных узлов – с ускорителем вычислений и без него – моделируется гетерогенная вычислительная среда, отражающая аппаратные особенности высокопроизводительного комплекса.

Для организации коммуникационной сети стенда используется беспроводной маршрутизатор РТ-1, оснащённый неблокирующим 4-х портовым коммутатором 1 Гбит/с Ethernet, а также двухдиапазонной беспроводной точкой доступа стандарта 802.11n. Вычислительные узлы ВУ1..ВУ-3, ВУ5 подключены к маршрутизатору посредством проводных интерфейсов 1 Гбит/с Ethernet, а управляющий узел УУ-1 – посредством беспроводного интерфейса 2,4 ГГц 802.11n. Это позволяет, с одной стороны, обеспечить высокоскоростное взаимодействие вычислительных узлов в процессе выполнения вычисления, а с другой – позволяет при необходимости

перемещать виртуальную машину управляющего узла без привязки к фиксированной сетевой инфраструктуре.

ПРОГРАММНАЯ ЧАСТЬ

При разработке мультиагентного диспетчера ресурсов гетерогенного вычислительного кластера обязательным этапом является разработка алгоритмов оптимизации для распределения вычислительных задач между узлами кластера. Сюда относятся алгоритмы планирования, оптимизации и управления задачами в зависимости от запрошенных ими вычислительных ресурсов. Также должна быть разработана система управления качеством обслуживания, которая будет обеспечивать необходимое качество работы приложений, запущенных на кластере.

Кроме того, при разработке подобного диспетчера также необходимо провести исследование и оценку различных подходов и методов управления ресурсами для гетерогенного вычислительного кластера. На этом этапе выполняется исследование и оценка различных парадигм управления ресурсами, таких как децентрализованное, централизованное и смешанное управление ресурсами. А также здесь проводится исследование различных методов адаптации и регулирования ресурсов, таких как машинное обучение, оптимизация и прогнозирование.

В итоге, разработка аппаратно-программного реконфигурируемого макетного образца мультиагентного диспетчера ресурсов гетерогенного вычислительного кластера является комплексным процессом, который включает в себя исследование, разработку и оценку различных подходов и методов для эффективного управления ресурсами гетерогенного вычислительного кластера. Этот процесс включает в себя разработку алгоритмов для распределения ресурсов, мониторинга использования ресурсов, адаптивного регулирования использования ресурсов, системы управления качеством обслуживания и многое другое. Для реализации такого решения могут использоваться различные технологии, такие как мультиагентные системы, машинное обучение, макетирование, и так далее. В

итоге, результатом разработки должен быть макет, который может эффективно управлять ресурсами гетерогенного вычислительного кластера и адаптироваться к изменениям в его составе и нагрузке.

ПОДХОДЫ

Различные подходы и методы для эффективного управления ресурсами гетерогенного вычислительного кластера могут быть оценены по различным критериям, таким как эффективность, надежность, гибкость и адаптивность.

Децентрализованный подход: в этом подходе ресурсы распределяются между узлами кластера без централизованного контроля. Этот подход может быть эффективным в случаях с большим количеством узлов и малым количеством зависимостей между ними, но может быть менее надежным и менее гибким в случаях с большим количеством зависимостей между узлами.

Централизованный подход: в этом подходе ресурсы управляются централизованным контроллером. Этот подход может быть более надежным и гибким, но может быть менее эффективным в случаях с большим количеством узлов или сложными зависимостями между ними, так как централизованный контроллер может быть нагружен и иметь ограниченную масштабируемость.

Машинное обучение и оптимизационные подходы: в этом подходе используются методы машинного обучения и оптимизации для прогнозирования и адаптивного регулирования ресурсов в кластере. Этот подход может быть очень эффективным, поскольку он использует динамическую информацию о состоянии кластера и запросах для регулирования ресурсов. Однако, этот подход требует наличия достаточного количества данных и мощных вычислительных ресурсов для обучения и запуска моделей.

Нет одного оптимального подхода для управления ресурсами гетерогенного вычислительного кластера, каждый из них имеет свои преимущества и недостатки. Выбор подхода зависит от специфики

конкретной ситуации и задачи, и может включать использование сочетания различных методов и подходов.

СБОРКА КЛАСТЕРА

Сборка вычислительного кластера с использованием Slurm требует некоторого опыта и знаний в области системного администрирования и сетевой инфраструктуры. В общих чертах, процесс сборки кластера с использованием Slurm может быть разделен на следующие шаги:

- Подготовка инфраструктуры: необходимо подготовить инфраструктуру, такую как сетевое оборудование, хосты и диски, для развертывания кластера.
- Развертывание узлов: необходимо развернуть и настроить узлы кластера.
- Настройка сети: необходимо настроить сетевые настройки для обеспечения связи между узлами кластера.
- Установка и настройка ПО: необходимо установить и настроить необходимое ПО, такое как Slurm, для управления кластером.
- Тестирование и отладка: необходимо протестировать и отладить кластер, чтобы убедиться в его работоспособности и производительности.

Настройка мониторинга: необходимо настроить системы мониторинга для отслеживания состояния кластера и обнаружения проблем.

В дополнение к настройке конфигурационных файлов для каждого узла, необходимо также настроить некоторые другие аспекты узлов для обеспечения их корректной работы в кластере.

Одним из таких аспектов является установка и настройка дистрибутива операционной системы, который будет использоваться в кластере. Помимо непосредственно операционной системы необходимо также выполнить установку нужных драйверов и пакетов, чтобы обеспечить корректную работу оборудования и программного обеспечения. Также необходимо настроить автоматическое обновление и обслуживание для обеспечения безопасности и стабильности кластера.

Для подготовки операционной системы необходимо установить дистрибутив CentOS 7.9 на управляющий и вычислительные узлы. Для этого рекомендуется выполнить минимальную установку и указать сетевой репозиторий для установки дополнительного программного обеспечения: https://mirror.yandex.ru/centos/7/os/x86_64/.

Затем необходимо установить драйверы для графических ускорителей, соответствующие их модели. Рекомендуется выполнять установку драйвера, а также программного обеспечения CUDA ToolKit, следуя инструкциям из документации: <https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>. Необходимо выполнить описанные по ссылке пункты 1-6, соответствующие дистрибутивам RHEL 7 / CentOS 7, за исключением пункта 4, где вместо установки драйвера из пакетного менеджера необходимо скачать подходящий драйвер самостоятельно и установить его, указав путь к исходным файлам ядра Linux. Для ускорителей NVIDIA драйвер можно скачать по ссылке <https://www.nvidia.com/download/index.aspx>. Сделать это можно следующим образом:

1) скачиваем установщик драйвера утилитой `wget`

```
# wget https://us.download.nvidia.com/XFree86/Linux-x86_64/515.57/NVIDIA-Linux-x86_64-515.57.run
```

2) запускаем установщик, указав в значении параметра `--kernel-source-path` путь к каталогу с исходными файлами ядра Linux

```
# ./NVIDIA-Linux-x86_64-515.57.run --kernel-source-path /usr/src/kernels/3.10.0-1160.66.1.el7.x86_64
```

3) убедиться в правильности установки драйверов можно командой `nvidia-smi`, которая покажет информацию об ускорителях в системе.

Другой важной частью настройки узлов является настройка сетевой инфраструктуры. Это может включать в себя настройку сетевых интерфейсов, конфигурирование VLAN, настройку маршрутизации и настройку

безопасности сети. Это необходимо для обеспечения эффективного общения между узлами кластера и доступа к ресурсам.

Одним из этапов настройки сетевой части является обеспечение возможности обращения к узлам кластера по именам, что облегчает дальнейшую работу. Для этого в файле `/etc/hosts` каждого узла необходимо указать соответствие между IP адресами и именами для каждого узла в кластере. Например, таким образом:

```
172.16.0.100 gateway gateway.slurm # шлюз для выхода в интернет
172.16.0.101 controller01 controller01.slurm # УУ-1
172.16.0.200 node01 node01.slurm # ВУ-1
172.16.0.201 node02 node02.slurm # ВУ-2
172.16.0.202 node03 node03.slurm # ВУ-3
172.16.0.203 node04 node04.slurm # ВУ-4, не используется
172.16.0.204 node05 node05.slurm # ВУ-5
```

Другим важным элементом обеспечения взаимодействия управляющей службы Slurm с подчинёнными вычислительными узлами является служба аутентификации Munge. На этом этапе рекомендуется подготовить учётные записи пользователей и группы, от имени которых будут выполняться службы Munge и Slurm. Для этого нужно на всех узлах:

1) сохранить в новую переменную окружения идентификатор учётной записи пользователя и группы (любой, не используемый ни на одном узле)

```
# export MUNGEUSER=991
# export SLURMUSER=992
```

2) создать группы

```
# groupadd -g $MUNGEUSER munge
# groupadd -g $SLURMUSER slurm
```

3) создать учётные записи пользователей, указав путь к домашнему каталогу в значении параметра `-d` и путь к исполняемой оболочке в значении параметра `-s`

```
# useradd -m -c "MUNGE Uid 'N' Gid Emporium" -d /var/lib/munge -u  
$MUNGEUSER -g munge -s /sbin/nologin munge
```

```
# useradd -m -c "SLURM workload manager" -d /var/lib/slurm -u  
$SLURMUSER -g slurm -s /bin/bash slurm
```

Для установки Munge необходимо:

1) добавить сетевой репозиторий EPEL

```
# yum install epel-release -y
```

2) установить пакеты munge, munge-libs и munge-devel

```
# yum install munge munge-libs munge-devel -y
```

После установки Munge необходимо сгенерировать новый ключ, который должен быть одинаковым на всех узлах, которые входят в состав одного кластера Slurm. Рекомендуется выполнять это действие на управляющем узле. Для этого потребуется:

1) установить пакет rng-tools

```
# yum install rng-tools -y
```

2) обновить пул энтропии

```
# rngd -r /dev/urandom
```

3) создать новый ключ с помощью утилиты create-munge-key

```
# /usr/sbin/create-munge-key -r
```

или вручную, не забыв изменить владельца, группу и права доступа к файлу ключа

```
# dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
```

```
# chown munge:munge /etc/munge/munge.key
```

```
# chmod 400 /etc/munge/munge.key
```

Для того, чтобы передать сгенерированный ключ всем остальным вычислительным узлам, можно воспользоваться утилитой безопасного копирования scp, выполняя операцию копирования на каждый узел:

```
# scp /etc/munge/munge.key root@node01:/etc/munge
```

...

```
# scp /etc/munge/munge.key root@node05:/etc/munge
```

После копирования ключа необходимо подключиться к каждому узлу и установить корректные разрешения на каталоги с файлами конфигурации и журналов Munge, а также запустить службу Munge. Это можно сделать следующим образом:

1) установить munge владельцем и группой для каталогов Munge

```
# chown -R munge:munge /etc/munge/ /var/log/munge/
```

2) установить права на доступ к каталогам Munge только пользователю и группе munge

```
# chmod 0700 /etc/munge/ /var/log/munge/
```

3) включить автоматический запуск службы при старте операционной системы

```
# systemctl enable munge
```

4) запустить службу немедленно

```
# systemctl start munge
```

Проверить правильность настройки Munge можно, например, с управляющего узла следующими командами:

```
# munge -n
```

```
# munge -n | munge
```

```
# munge -n | ssh node01 unmunge
```

```
# remunge
```

Если после выполнения этих команд сообщений об ошибках не последовало – служба Munge функционирует корректно.

Для выполнения установки диспетчера Slurm необходимо:

1) установить пакеты, от которых зависит сам Slurm

```
# yum install openssl openssl-devel pam-devel numactl numactl-devel hwloc  
hwloc-devel lua lua-devel readline-devel rrdtool-devel ncurses-devel man2html  
libibmad libibumad -y
```

2) скачать исходные файлы Slurm, далее для примера используется версия 19.05.3-2

```
# wget http://134.100.28.207/files/src/slurm/slurm-19.05.3-2.tar.bz2
```

3) установить систему сборки rpm-пакетов и пакет разработчика Mysql

```
# yum install rpm-build mysql-devel -y
```

4) собрать из исходных файлов rpm-пакет Slurm с поддержкой Mysql

```
# rpmbuild -ta slurm-19.05.3-2.tar.bz2 --with mysql
```

5) скопировать полученный каталог rpmbuild со всем содержимым на все вычислительные узлы

```
# scp -r /root/rpmbuild/ root@node01:/root/
```

...

```
# scp -r /root/rpmbuild/ root@node05:/root/
```

6) на каждом узле перейти в скопированный каталог rpmbuild/RPMS/x86_64 и выполнить установку пакетов

```
# cd /root/rpmbuild/RPMS/x86_64
```

```
# yum --nogpgcheck localinstall *
```

В зависимости от конфигурации сетевого окружений может потребоваться настроить брандмауэр на всех узлах. В нашем случае к узлам ВУ1..ВУ5 нет доступа из внешней сети непосредственно, поэтому на этих вычислительных узлах брандмауэр отключен следующим образом:

```
# systemctl stop firewalld
```

```
# systemctl disable firewalld
```

На управляющем узле, к которому в нашем случае имеется непосредственный доступ из сети лаборатории, отключать брандмауэр неблагоразумно, поэтому в нём разрешены подключения на порты 6817..6819 для корректной работы Slurm.

```
# firewall-cmd --permanent --zone=public --add-port=6817/udp
```

```
# firewall-cmd --permanent --zone=public --add-port=6817/tcp
```

```
# firewall-cmd --permanent --zone=public --add-port=6818/udp
```

```
# firewall-cmd --permanent --zone=public --add-port=6818/tcp
```

```
# firewall-cmd --permanent --zone=public --add-port=6819/udp
# firewall-cmd --permanent --zone=public --add-port=6819/tcp
# firewall-cmd --reload
```

Для детального протоколирования работы контроллера Slurm на управляющий узел необходимо также установить СУБД Mysql (в новых версиях название изменено на MariaDB):

```
# yum install mariadb-server mariadb-devel -y
```

А также предоставить службе аккаунтинга Slurm доступ к созданной базе данных slurm_acct_db:

```
MariaDB[(none)]> GRANT ALL ON slurm_acct_db.* TO
'slurm'@'localhost' IDENTIFIED BY '1234' with grant option;
```

```
MariaDB[(none)]> SHOW VARIABLES LIKE 'have_innodb';
```

```
MariaDB[(none)]> FLUSH PRIVILEGES;
```

```
MariaDB[(none)]> CREATE DATABASE slurm_acct_db;
```

```
MariaDB[(none)]> quit;
```

После чего можно проверить привилегии:

```
$ mysql -p -u slurm
```

```
MariaDB[(none)]> show grants;
```

```
MariaDB[(none)]> quit;
```

Далее потребуется донастройка конфигурации Mysql, для чего необходимо создать файл /etc/my.cnf.d/innodb.cnf:

```
# touch /etc/my.cnf.d/innodb.cnf
```

И поместить в него следующее содержимое:

```
[mysqld]
```

```
innodb_buffer_pool_size=1024M
```

```
innodb_log_file_size=64M
```

```
innodb_lock_wait_timeout=900
```

После чего необходимо перезапустить службу сервера БД:

```
# systemctl stop mariadb
```

```
# mv /var/lib/mysql/ib_logfile? /tmp/
```

```
# systemctl start mariadb
```

Проверить настройки СУБД можно следующим образом:

```
MariaDB[(none)]> SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
```

ОСОБЕННОСТИ НАСТРОЙКИ

При создании вычислительного кластера с использованием Slurm, существует несколько особенностей настройки, которые следует учитывать:

- Конфигурация узлов: необходимо настроить каждый узел в кластере с указанием его ресурсов (процессоры, память, диски) и сетевых настроек.
- Конфигурация контроллера: необходимо настроить контроллер Slurm (службу slurmd), которая будет управлять ресурсами кластера и диспетчером задач. В этой части настройки можно указать параметры, связанные с кэшированием, распределением задач и планированием.
- Настройка политик доступа: необходимо настроить политики доступа к ресурсам кластера, например, ограничения на использование ресурсов для определенных групп пользователей или приложений.
- Настройка мониторинга и отчетности: необходимо настроить системы мониторинга и отчетности для отслеживания состояния кластера и анализа использования ресурсов.

Первой стадией является настройка узлов. Для этого необходимо создать конфигурационный файл для каждого узла в кластере, в котором будут указаны его ресурсы (количество процессоров, объем памяти, диски), а также сетевые настройки.

Следующей стадией является настройка контроллера Slurm, который будет управлять ресурсами кластера и диспетчером задач. В этой части

настройки можно указать параметры, связанные с кэшированием, распределением задач и планированием.

Далее необходимо настроить политики доступа к ресурсам кластера, например, ограничения на использование ресурсов для определенных групп пользователей или приложений. Это помогает обеспечить равномерное распределение ресурсов и избежать недопустимого использования.

Наконец, необходимо настроить системы мониторинга и отчетности для отслеживания состояния кластера и анализа использования ресурсов. Это поможет администратору кластера получить полное представление о том, как ресурсы используются, и идентифицировать проблемные области, которые требуют улучшения.

НАСТРОЙКА УЗЛОВ

Настройка узлов является ключевым этапом при создании вычислительного кластера с использованием Slurm. Этот этап включает в себя создание конфигурационного файла для каждого узла в кластере, в котором будут указаны его ресурсы и сетевые настройки.

В конфигурационном файле для каждого узла следует указать следующую информацию:

- Количество процессоров: это позволяет Slurm определить количество доступных ресурсов для запуска задач.
- Объем памяти: это позволяет Slurm определить объем доступной памяти для запуска задач.
- Диски: если в кластере используются дисковые ресурсы, то их также следует указать в конфигурационном файле, чтобы Slurm мог определить доступность и настроить их использование.
- Сетевые настройки: также необходимо указать информацию о сетевых интерфейсах, которые используются для общения между узлами кластера и доступа к ресурсам.

После создания конфигурационных файлов для всех узлов, их необходимо разместить в соответствующей директории, чтобы Slurm мог найти их и использовать для управления ресурсами кластера. Также необходимо настроить доступ к конфигурационным файлам только для администратора кластера, чтобы избежать несанкционированного изменения настроек.

Рекомендуется подготовить конфигурационный файл Slurm на управляющем узле кластера, а затем скопировать его на остальные узлы. Для этого необходимо:

1) создать файл конфигурации `slurm.conf`

```
# touch /etc/slurm/slurm.conf
```

2) заполнить конфигурацию в текстовом редакторе

```
# vim /etc/slurm/slurm.conf
```

Пример содержимого файла конфигурации приведен ниже; при необходимости можно воспользоваться документацией по данному файлу, которая расположена по адресу <https://slurm.schedmd.com/slurm.conf.html>

```
ClusterName="LabStand"
```

```
ControlMachine="controller01"
```

```
AuthType=auth/munge
```

```
CacheGroups=0
```

```
CryptoType=crypto/munge
```

```
PrivateData=accounts,jobs,users
```

```
MpiDefault=none
```

```
ProctrackType=proctrack/cgroup
```

```
PropagateResourceLimits=NONE
```

```
SchedulerParameters=bf_window=4320,defer,default_queue_depth=2000
```

```
ReturnToService=1
```

```
RebootProgram=/usr/sbin/reboot
```

```
SlurmctldPidFile=/var/run/slurmctld.pid
```

```
SlurmctldPort=6817
```


SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/var/spool/slurm/d
SlurmUser=root
StateSaveLocation=/var/spool/slurm/ctld
SwitchType=switch/none
TaskPlugin=task/cgroup
TIMERS
CompleteWait=60
EpilogMsgTime=4000
KillWait=120
MessageTimeout=30
ResvOverRun=5
MinJobAge=300
SlurmctldTimeout=3660
SlurmdTimeout=300
WaitTime=150
MaxArraySize=100000
TCPTimeout=4
TOPOLOGY
TopologyPlugin=topology/none
SCHEDULING
SchedulerType=sched/backfill
SchedulerPort=7321
SelectType=select/linear
JobRequeue=0
InactiveLimit=900
MaxTasksPerNode=56
JOB PRIORITY
PriorityType=priority/multifactor

```

PriorityDecayHalfLife=0
PriorityUsageResetPeriod=none
PriorityWeightAge=2000
PriorityWeightAssoc=0
PriorityWeightFairshare=2000
PriorityWeightJobSize=1000
PriorityWeightPartition=1
PriorityWeightQOS=4000
# LOGGING AND ACCOUNTING
AccountingStorageEnforce=limits,qos,wkeys
AccountingStorageHost=controller01
AccountingStoragePort=6819
AccountingStorageType=accounting_storage/slurmdbd
AcctGatherFilesystemType=acct_gather_filesystem/lustre
AccountingStorageTRES=fs/lustre
JobCompType=jobcomp/none
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/cgroup
SlurmctldDebug=debug5
SlurmdDebug=debug5
# POWER SAVE SUPPORT FOR IDLE NODES (optional)
CpuFreqDef=Performance
# Licenses
Licenses=ansyspar:1156
#      Changes      prompt      after      salloc      command.      See
/etc/profile.d/slurm_bash_prompt.sh
SallocDefaultCommand=/bin/bash
GresTypes=gpu
# COMPUTE NODES

```

```
NodeName=node[01-04] CPUs=4 Boards=1 SocketsPerBoard=1  
CoresPerSocket=4 ThreadsPerCore=1 RealMemory=7694
```

```
NodeName=node05 CPUs=4 Boards=1 SocketsPerBoard=1  
CoresPerSocket=4 ThreadsPerCore=1 RealMemory=15567  
Gres=gpu:quadro_k6000-12gb:1
```

```
PartitionName=intel Nodes=node[01-03] Default=YES MaxTime=20160  
DefaultTime=720 State=UP
```

```
PartitionName=nvidia Nodes=node[05] MaxTime=20160 DefaultTime=720  
State=UP
```

Здесь в поле `NodeName` необходимо указать доменные имена узлов, которые можно указать через запятую (`node01, ... , node04`), или массивом (`node[01-04]`). Узел `node05` содержит в себе ускоритель, об этом говорит атрибут `Gres` (`Generic Resources`) вида `x:y:z`, где `x` – это название типа (группы) ускорителей, здесь выбрано название `gpu`, `y` – название конкретной модели, `z` – их число в данном узле. Также отдельно должна быть строка конфигурации `GresTypes`, которая содержит в себе объявления типов ускорителей. Остальные данные о вычислительных ресурсах (`CPUs`, `Boards`, `RealMemory`) этих узлов можно получить, введя команду `slurmd -C`. Результат выполнения этой команды на одном из узлов тестового кластера:

```
NodeName=node05 CPUs=4 Boards=1 SocketsPerBoard=1  
CoresPerSocket=4 ThreadsPerCore=1 RealMemory=15567  
UpTime=6-21:52:39
```

Эти данные соответствуют узлу `node05` и были записаны в файл конфигурации. Далее идет объявление `PartitionName`. Это логические разделы кластера, которые, как правило, объединяют в себе гомогенные узлы. В данном случае, раздел `intel` содержит в себе 3 узла `node01-node03` без ускорителей, а раздел `nvidia` содержит один узел `node05` с ускорителем.

Также помимо `slurm.conf` на узлах с ускорителями нужен файл `gres.conf`. Создать и отредактировать его можно таким образом:

```
# touch /etc/slurm/gres.conf
# vim /etc/slurm/gres.conf
```

Пример содержимого такого файла приведен ниже.

```
#####
# Slurm's Generic Resource (GRES) configuration file
# Define GPU devices with MPS support, with AutoDetect sanity checking
#####
#AutoDetect=nvml
Name=gpu Type=quadro_k6000-12gb File=/dev/nvidia0 Cores=0-3
```

Чтобы автоматически определить ускорители в системе, можно раскомментировать строку `#AutoDetect=nvml` и перезапустить службу `slurmd` командой `systemctl restart slurmd`, после чего информация об обнаруженных в системе ускорителях будет помещена в файл журнала. Просмотреть его можно командой `journalctl -xe`. Чтобы автоопределение работало, нужно установить пакет `nvml`, если он не установлен:

```
# yum install cuda-nvml-devel-11-7.x86_64
```

Более тонко настроить ускорители вычислений можно, обратившись к документации по этому файлу, доступной по адресу <https://slurm.schedmd.com/gres.conf.html>.

После того, как конфигурационный файл `slurm.conf` будет подготовлен на управляющем узле, его необходимо скопировать на остальные узлы:

```
# scp /etc/slurm/slurm.conf root@node01:/etc/slurm/
...
# scp /etc/slurm/slurm.conf root@node05:/etc/slurm/
```

На управляющем узле необходимо создать файлы и рабочие каталоги служб Slurm и задать правильные разрешения:

```
# mkdir /var/spool/slurm
```

```
# chown slurm: /var/spool/slurm/
# chmod 755 /var/spool/slurm/
# mkdir /var/log/slurm
# touch /var/log/slurm/slurmctld.log
# chown slurm: /var/log/slurm/slurmctld.log
# touch /var/log/slurm/slurm_jobacct.log /var/log/slurm/slurm_jobcomp.log
#          chown          slurm:          /var/log/slurm/slurm_jobacct.log
/var/log/slurm/slurm_jobcomp.log
```

На всех вычислительных узлах также необходимо создать файлы и рабочие каталоги Slurm и установить разрешения. Здесь меньшее число файлов и каталогов, поскольку на вычислительных узлах не выполняется служба контроллера Slurm:

```
# mkdir /var/spool/slurm
# chown slurm: /var/spool/slurm
# chmod 755 /var/spool/slurm
# touch /var/log/slurm/slurmd.log
# chown slurm: /var/log/slurm/slurmd.log
```

По завершении настройки можно запустить службу Slurm на вычислительных узлах следующим образом:

```
# systemctl enable slurmd.service
# systemctl start slurmd.service
```

Проверить состояние службы можно следующей командой:

```
# systemctl status slurmd.service
```

Для завершения настройки службы аккаунтинга Slurm на управляющем узле необходимо создать файл конфигурации slurmdbd.conf и настроить права доступа:

```
# touch /etc/slurm/slurmdbd.conf
```

```
# vim /etc/slurm/slurmdbd.conf
# chown slurm: /etc/slurm/slurmdbd.conf
# chmod 600 /etc/slurm/slurmdbd.conf
# touch /var/log/slurmdbd.log
# chown slurm: /var/log/slurmdbd.log
```

Содержимое файла slurmdbd.conf в нашем примере приведено ниже:

```
DbdAddr=localhost
DbdHost=localhost
DbdPort=6819
StorageUser=slurm
StoragePass=1234
StorageLoc=slurm_acct_db
StorageType=accounting_storage/mysql
AuthType=auth/munge
DebugLevel=debug5
LogFile=/var/log/slurm/slurmdbd.log
PidFile=/var/run/slurmdbd.pid
```

После завершения настройки службы аккаунтинга Slurm на управляющем узле можно включить автоматический запуск службы и сразу запустить её:

```
# systemctl enable slurmdbd
# systemctl start slurmdbd
# systemctl status slurmdbd
```

Если при запуске службы не возникло ошибок – теперь можно запустить службу контроллера Slurm на управляющем узле:

```
# systemctl enable slurmctld
# systemctl start slurmctld
# systemctl status slurmctld
```

НАСТРОЙКА ПОЛИТИКИ ДОСТУПА К РЕСУРСАМ КЛАСТЕРА

Настройка политики доступа к ресурсам кластера является важным этапом при создании и управлении вычислительным кластером с использованием Slurm. Эта политика определяет, какие пользователи и группы имеют доступ к ресурсам кластера и как этот доступ контролируется.

Одним из ключевых аспектов настройки политики доступа является создание и настройка аккаунтов и групп пользователей. Это позволяет определить, какие пользователи имеют доступ к кластеру и какие ресурсы доступны для использования. В частности, можно настроить различные группы пользователей с различными уровнями доступа к ресурсам, включая ограничения на количество ядер и оперативной памяти.

Другой важный аспект настройки политики доступа – это настройка партиций и очередей. Партиции – это наборы ресурсов, которые могут быть назначены конкретным группам пользователей или задачам. Очереди – это механизм, который управляет доступом к ресурсам в партициях и определяет какой задаче будет назначен ресурс. Это позволяет более тонко контролировать доступ к ресурсам и обеспечить эффективность использования ресурсов.

Также можно настроить дополнительные механизмы контроля доступа, такие как лимиты на количество запущенных задач или ограничения на время и способы распределения ресурсов. Например, можно настроить систему резервирования ресурсов для ключевых задач или приоритетных пользователей. Также можно реализовать механизмы балансировки нагрузки и перераспределения ресурсов в зависимости от изменения нагрузки на кластер.

Важно понимать, что настройка политики доступа к ресурсам кластера является динамическим процессом и требует постоянного мониторинга и адаптации в зависимости от изменения нагрузки и потребностей пользователей.

НАСТРОЙКА СИСТЕМ МОНИТОРИНГА

Настройка систем мониторинга состояния кластера является важным этапом при создании и управлении вычислительным кластером с использованием Slurm. Это позволяет отслеживать ресурсы кластера, задачи и состояние каждого узла. Также это важно для оптимизации использования ресурсов и обнаружения и устранения проблем.

Существует множество различных систем мониторинга, которые могут быть использованы для мониторинга состояния кластера. Одной из наиболее распространенных является система мониторинга Ganglia. Эта система предоставляет возможность сбора метрик и генерации графиков для отслеживания ресурсов кластера. Другой популярной системой является Prometheus, которая также позволяет собирать метрики и генерировать графики, но имеет более широкую функциональность и более современный интерфейс. Также можно использовать системы типа Nagios или Zabbix для мониторинга состояния сетевых интерфейсов, доступности сервисов и других элементов инфраструктуры кластера.

Настройка систем мониторинга также может включать интеграцию с инструментами аналитики данных, такими как Grafana или Kibana, для визуализации метрик и обнаружения закономерностей в данных. Также может быть настроена система оповещений, чтобы уведомлять администраторов о критических проблемах или предсказывать потенциальные проблемы.

В общем, настройка систем мониторинга состояния кластера является важным этапом при создании и управлении вычислительным кластером. Это позволяет отслеживать ресурсы кластера, задачи и состояние каждого узла, что в свою очередь позволяет оптимизировать использование ресурсов, обнаруживать и устранять проблемы и предотвращать будущие проблемы.

В частности, системы мониторинга могут предоставлять информацию о загрузке процессоров, использовании памяти, дисковом пространстве, сетевых интерфейсах и доступности служб. Это может включать в себя

информацию о нагрузке на каждый узел, списке запущенных задач и использовании ресурсов каждой задачей.

Настройка систем мониторинга также может включать в себя интеграцию с инструментами оповещения, такими как Nagios или Zabbix, чтобы отправлять уведомления администраторам о критических проблемах или предсказывать потенциальные проблемы. Это может включать в себя настройку порогов для определенных метрик и настройку правил для генерации оповещений в случае их превышения.

8.2. Разработка протокола сбора метаданных о вычислительных процессах

В дальнейшей работе необходимо проводить сбор и структуризацию метаданных, несущих информацию об атрибутах заданий, отправленных на выполнение с помощью планировщика задач «SLURM» для дальнейшего оптимального планирования.

Для подготовки исходных данных о ходе вычислительного процесса выделяется три основных шага:

- 1) сбор параметров запуска задания, которые указал пользователь явно при выставлении задачи в очередь на исполнение;
- 2) сбор данных об атрибутах, при которых задание было запущено; они могут отличаться от параметров, которые указал пользователь, так как планировщики задач могут их переопределять для более эффективного решения задания;
- 3) сбор информации после выполнения задачи, содержащая результат вычисления, метаданные и протокол хода исполнения программы.

В ходе работы получена структурированная форма данных о вычислительном процессе, в частности параметров запуска задания, которые пользователь указал явно при постановке задачи в очередь на исполнение диспетчеру SLURM в СКЦ. Рассмотренная форма представления данных

может быть использована при проектировании автоматизированных планировщиков заданий с использованием методов машинного обучения.

Ниже представлена структура JSON файла, состоящего из всех возможных 89 атрибутов для тонкой настройки команды sbatch SLURM версии 21.08. Атрибуты, в соответствии с настраиваемой сущностью, были структурированы и распределены в 5 различных групп: «Информация о пользователе», «Учёт заданий», «Управление ресурсами», «Управление заданием», «Взаимодействие задания».

```
{
  "userInfo": { // группа «Информация о пользователе»
    "account ": < account >,
    "comment ": <string >,
    "gid ": <group >,
    "uid ": <user >
  },
  "jobAccounting": { // группа «Учёт заданий»
    "acctg-freq ": < <datatype>=<interval> >,
    "mcs-label ": <mcs >,
    "profile ": <all|none|[energy[,|task[,|lustre[,|network]]]] >,
    "qos ": <qos >,
    "wckey ": <wckey >
  },
  "resourcesManagement": { // группа «Управление ресурсами»
    "extra-node-info ": < sockets[:cores[:threads]] >,
    "constraint ": <list >,
    "contiguous ": < bool >,
    "cores-per-socket ": <cores >,
    "cpu-freq ": < p1[-p2[:p3]] >,
    "cpus-per-task ": <ncpus >,
    "exclusive ": < bool[=user|mcs] >,
    "nodefile ": <node file >,
    "gres ": <list >,
    "gres-flags ": <enforce-binding >,
    "hint ": <type >,
    "clusters ": <string >,
    "mem ": <MB >,
    "mem-per-cpu ": < MB >,
    "mem_bind ": < [{quiet,verbose},]type >,
  }
}
```

```

"mincpus ": < n >,
"nodes ": < minnodes[-maxnodes] >,
"ntasks ": < number >,
"ntasks-per-core ": < ntasks >,
"ntasks-per-socket ": < ntasks >,
"ntasks-per-node ": < ntasks >,
"overcommit ": < bool >,
"partition ": < partition_names >,
"power ": < flags >,
"propagate ": < [=rlimitfR] >,
"reboot ": < bool >,
"reservation ": < name >,
"oversubscribe ": < bool >,
"core-spec ": < num >,
"sockets-per-node ": < sockets >,
"switches ": < <count>[@<max-time>] >,
"tasks-per-node ": < n >,
" thread-spec ": < num >,
"threads-per-core ": < threads >,
"tmp ": < MB >,
"odelist ": < node name list >,
"exclude ": < node name list >
},
"jobManagement": { // группа «Управление заданием»
    "begin ": < time >,
    "deadline ": < OPT >,
    "hold ": < bool >,
    "ignore-pbs ": < bool >,
    "job-name ": < jobname >,
    "jobid ": < jobid >,
    "immediate ": < bool >,
    "no-kill ": < bool >,
    "kill-on-invalid-dep ": < yes|no >,
    "licenses ": < license >,
    "distribution ": < arbitrary|<block|cyclic|plane >>,
    "nice ": < [=adjustment] >,
    "no-requeue ": < bool >,
    "priority ": < value >,
    "requeue ": < bool >,
    "time ": < time >,
    "test-only ": < bool >,
    "time-min ": < time >,
    "wait ": < bool >,

```

```

    "wait-all-nodes ": <value >,
    "wrap ": <command string >
  },
  "jobInteraction": { // группа «Взаимодействие задания»
    "bb ": <spec >,
    "checkpoint-dir ": <directory >,
    "workdir ": <directory >,
    "dependency ": <dependency_list >,
    "export ": <environment variables | ALL | NONE >,
    "export-file ": <filename | fd >,
    "get-user-env ": <bool[=timeout][mode] >,
    "error ": <filename pattern >,
    "input ": <filename pattern >,
    "mail-type ": <type >,
    "mail-user ": <user >,
    "network ": <type >,
    "output ": <filename pattern >,
    "open-mode ": <append|truncate >,
    "parsable ": <bool >,
    "quiet ": <bool >,
    "signal ": < [B:]<sig_num>[@<sig_time>] >,
    "verbose ": <bool >
  }
}

```

Таким образом, была получена структурированная форма данных о вычислительном процессе, в частности параметров запуска задания, которые пользователь указал явно при постановке задачи в очередь на исполнение диспетчеру SLURM в СКЦ. Рассмотренная форма представления данных также может быть использована в дальнейшем при проектировании интеллектуального диспетчера задач с использованием методов машинного обучения.

8.3. Разработка базы данных хранения метаданных о вычислительных процессах

Также была спроектирована структура базы данных, в которой хранятся данные о задачах, запущенных на СКЦ. Она состоит из трех таблиц «scontrol»,

«sacct» и «sbatch», соответствующие источникам собранных данных. Каждая таблица содержит целочисленное поле ID задачи, а также два поля типа VARCHAR(500), которая позволяет динамически выделять память под объем загружаемый в нее атрибут, не превышающий 500 символов. Эти два поля позволяют задать пару ключ-значение для атрибута и соответствующего ему значения.

Была также разработана программа для извлечения, лексической обработки и отправки в БД данных о задачах. Программа отлажена и протестирована на лабораторном стенде запуском тестовых задач. В итоге, в таблице «scontrol» для каждой задачи собирается по 60 атрибутов, в «sacct» - 107 атрибутов, а в таблице «sbatch» хранится ровно столько записей, сколько параметров указал пользователь явно в скрипте запуска. Пользователь, в среднем, указывает не более 8 параметров в скрипте запуска. Итого, для каждого ID задачи база данных содержит около 180 параметров, при этом некоторые поля могут быть пустыми, если SLURM не смог получить информацию по данному атрибуту. На рисунке 8.2 приведен пример запроса к одной из таблиц базы данных, на нем видна часть атрибутов для задачи с ID 60, и часть от задачи с ID 61.

60	Priority	4294901722
60	Partition	debug
60	QOS	normal
60	QOSRAW	1
60	Reason	None
60	ReqCPUFreq	Unknown
60	ReqCPUFreqMin	Unknown
60	ReqCPUFreqMax	Unknown
60	ReqCPUFreqGov	Unknown
60	ReqCPUS	1
60	ReqMem	0n
60	ReqNodes	1
60	ReqTRES	billing=1,cpu=1,node=1
60	Reservation	
60	ReservationId	
60	Reserved	00:00:00
60	ResvCPU	00:00:00
60	ResvCPURAW	0
60	Start	2022-06-10T12:38:29
60	State	COMPLETED
60	Submit	2022-06-10T12:38:29
60	Suspended	00:00:00
60	SystemCPU	00:00.003
60	SystemComment	
60	Timelimit	365-00:00:00
60	TimelimitRaw	525600
60	TotalCPU	00:00.006
60	TRESUsageInAve	
60	TRESUsageInMax	
60	TRESUsageInMaxNode	
60	TRESUsageInMaxTask	
60	TRESUsageInMin	
60	TRESUsageInMinNode	
60	TRESUsageInMinTask	
60	TRESUsageInTot	
60	TRESUsageOutAve	
60	TRESUsageOutMax	
60	TRESUsageOutMaxNode	
60	TRESUsageOutMaxTask	
60	TRESUsageOutMin	
60	TRESUsageOutMinNode	
60	TRESUsageOutMinTask	
60	TRESUsageOutTot	
60	UID	0
60	User	root
60	UserCPU	00:00.002
60	WCKey	
60	WCKeyID	0
60	WorkDir	/root
61	Account	root
61	AdminComment	
61	AllocCPUS	1
61	AllocNodes	1
61	AllocTRES	billing=1,cpu=1,node=1
61	AssocID	2
61	AveCPU	
61	AveCPUFreq	
61	AveDiskRead	

Рисунок 8.2 Пример запроса к таблице «scontrol» с обращением ко всем полям

Далее представлен краткий алгоритм работы программы, схема представлена на рисунке 8.3:

- 1) Создание пользователем скрипта запуска задачи
- 2) Размещение скрипта в рабочую директорию пользователя
- 3) Вызов на постановку скрипта в очередь (sbatch)

- 4) Срабатывание события «job allocated» (выделены ресурсы для задачи)
- 5) Вызов скрипта анализа, получение ID процесса
- 6) Поиск скрипта запуска по ID процесса
- 7) Анализ скрипта запуска и извлечение из него параметров
- 8) Сохранение параметров во внутреннем формате программы
- 9) Загрузка данных в БД хранения параметров
- 10) Получение ответа от БД

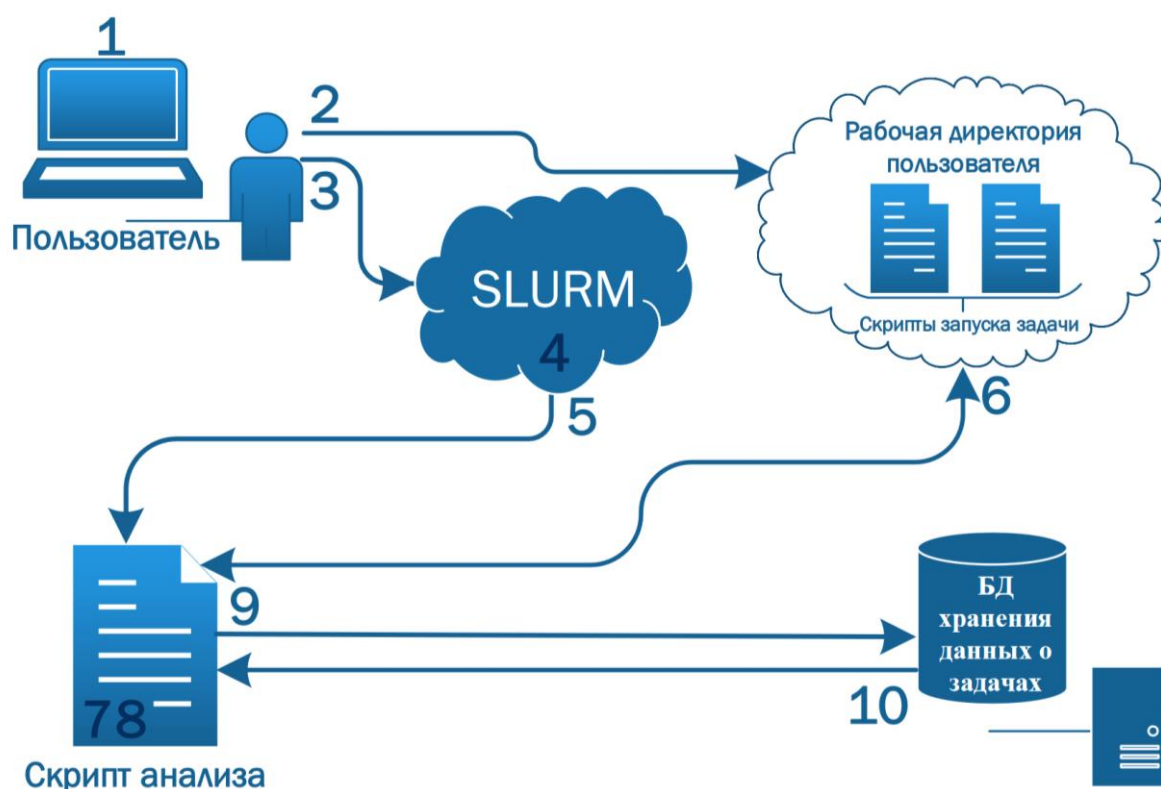


Рисунок 8.3 Схема извлечения, обработки и сохранения данных о задачах

Программа сбора состоит из около 300 строк программного кода без учета подключенных к ней библиотек. Вызов программы должен осуществляться по событию выделения ресурсов для задачи, а также по событию завершения задачи. Таким образом, на выходе программы будут две соответствующие базы данных. Время работы для каждого вызова составляет около 4-5 секунд из-за ожидания обращений к удаленному серверу БД, поэтому

программа должна запускаться в фоновом режиме, чтобы не вызывать задержку для планировщика. Программа корректно обрабатывает ошибочные состояния, бесперебойно продолжая свою работу.

ВЫВОДЫ

Полученный мультиагентный вычислительный кластер позволяет решать сложные задачи многопоточной обработки больших объемов данных, например, машинное обучение, вычисления в области инженерной и естественных наук, моделирование и вычислительное моделирование.

Кластер позволяет собирать следующую статистику:

- Использование ресурсов каждым узлом кластера, включая ЦПУ, память, диск и сеть.
- Использование ресурсов пользователями и задачами, включая процессорное время, использованную память и дисковое пространство.
- Статус и доступность узлов кластера, включая их местоположение, количество ядер и другие характеристики.
- Статистика очередей задач и задержки в очередях.
- Статистика отказов и ошибок в работе кластера.

Эта статистика может использоваться в дальнейшем для мониторинга и оптимизации работы кластера, а также для диагностики и устранения проблем. Она может быть также использована для оптимизации распределения ресурсов между задачами и пользователями, чтобы обеспечить максимальную производительность и эффективность. Также статистика может использоваться для прогнозирования и планирования расширения кластера в будущем.

Мультиагентный вычислительный кластер может использоваться для разработки и тестирования интеллектуального диспетчера ресурсов. Использование мультиагентной системы позволяет создавать реалистичные ситуации и тестировать диспетчер в реальных условиях.

Для разработки интеллектуального диспетчера, можно использовать методы машинного обучения и искусственного интеллекта, чтобы создать модели, которые могут принимать решения о распределении ресурсов в зависимости от текущей нагрузки и состояния кластера. В ходе тестирования, можно сравнивать производительность интеллектуального диспетчера с другими методами управления ресурсами и оценивать его эффективность.

9 Разработка «высокоуровневой» архитектуры аппаратных компонент мультиагентного диспетчера, использующего каналы информационного обмена между агентами

9.1 Проблемы повышения производительности вычислителя с фиксированной архитектурой

Средства и подходы, традиционно используемые для повышения производительности вычислений хорошо известны. Это и многопоточное, многоядерное и/или многопроцессорное пространственное распараллеливание, реализуемое на традиционных процессорах, имеющих N физических ядер и позволяющих реализовать до $N \times 2$ потоков. Это и графические карты, работающие в режиме вычислителя, в англоязычной литературе называемом General Purpose Graphic Processing Unit (GPGPU), и имеющие систолическую SIMD (Single Instruction Multiple Data) – одна инструкция много данных – архитектуру. Это и FPGA (Field Programmable Gate Arrays) – СБИС (Сверхбольшие Интегральные Схемы), часто называемые в ПЛИС, с внутренней архитектурой, аппаратно реконфигурируемой под выполняемый алгоритм.

В вычислителях с фиксированной архитектурой (многоядерные/многопоточковые процессоры и GPGPU) возможности повышения производительности ограничены особенностями конкретного вычислителя: числом вычислительных блоков; существующими связями между вычислительными блоками и их быстродействием; объемом локальной/распределенной памяти, шириной каналов доступа к памяти; быстродействием и производительностью памяти. Эти ограничения являются фиксированными для конкретного вычислителя с фиксированной архитектурой (Мультипроцессорной Вычислительной Системы – МВС) и определяют зависимость между производительностью вычислителя и типом решаемой на нем задачи (см. рисунок 9.1).

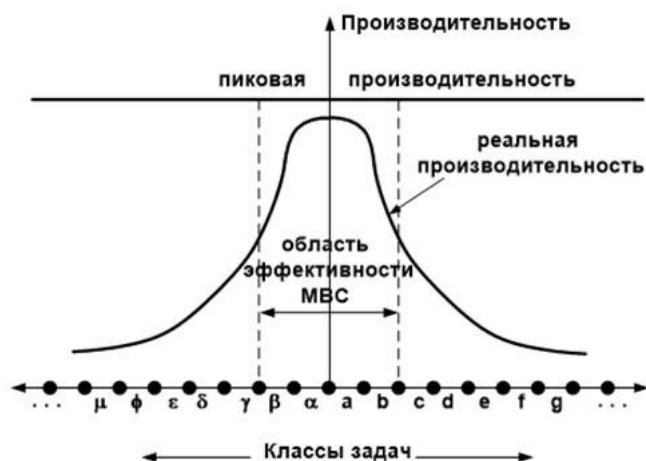


Рисунок 9.1 Зависимость производительности от класса задач

9.2 Влияние процессов мультиагентного планирования SLURM на реальную производительность гетерогенных вычислительных кластеров

В любой суперкомпьютерной вычислительной системе запуск задачи пользователя на исполнение не гарантирует ее успешное завершение, нужны метрики «умных» решений и «тонкая» настройка заданий пользователей с использованием механизма целевой реконфигурации аппаратной архитектуры вычислителя, когда каждому параметру задания пользователя ставится в соответствие дескриптор метрики производительности (см. рисунок 9.2), который позволяет определять какие:

- параметры в описание задания, в частности оценка времени исполнения, следует принять, чтобы повысить вероятность успешного завершения или «выживания» задания в процессе его исполнения в данной аппаратно-программном окружении;
- стратегии исполнения заданий, а именно выбор моментов времени запуска задания на исполнение, могут обеспечить равномерную загрузку всех имеющихся аппаратных ресурсов суперкомпьютерного кластера.

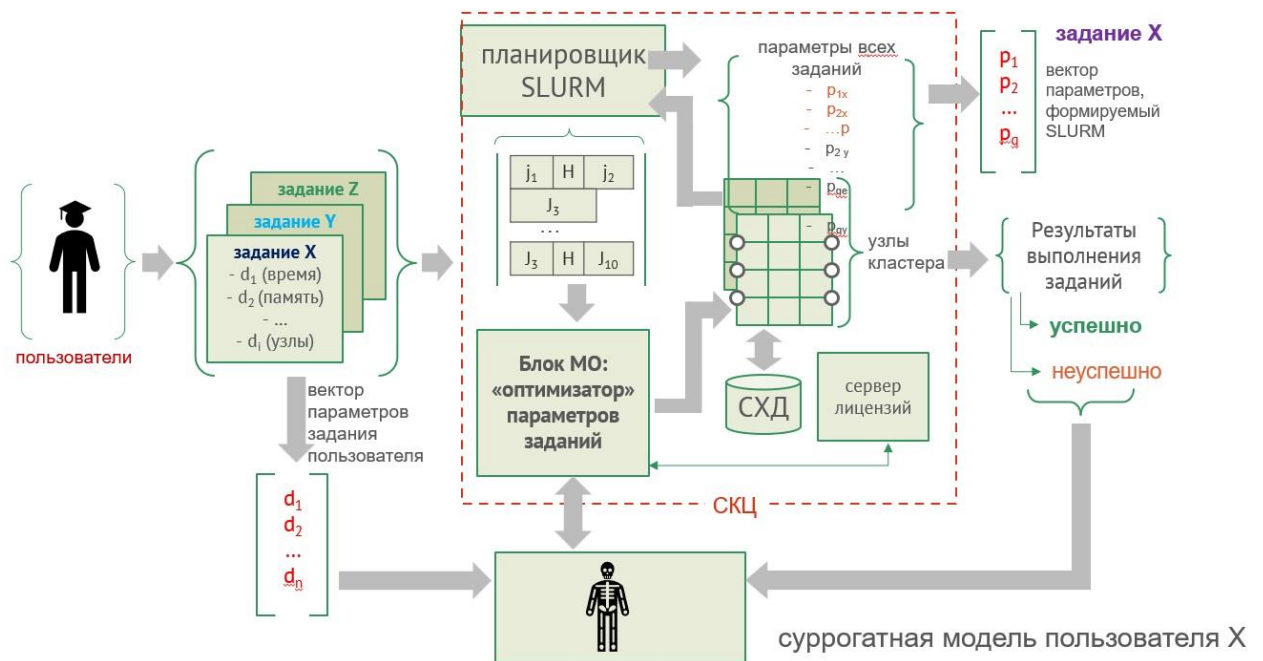


Рисунок 9.2 Стратегия исполнения заданий

Так как каждый узел гетерогенного кластера информирует блок принятия решения интеллектуального диспетчера (SLURM) о текущем состоянии ресурсов, то анализируя данные об результатах выполнения различных классов заданий можно сделать вывод о существенном влиянии особенностей архитектуры кластера на эффективность исполнения различного класса заданий (см. таблицу 9.1).

Приведенные выше данные показывают, что использование каналов информационного обмена между агентами отдельных узлов гетерогенного кластера можно использовать для повышения его реальной производительности за счет:

- точного прогнозирования времени исполнения заданий, которое используется интеллектуальным диспетчером SLURM, однако требует дополнительного встраивания в контур исполнения заданий «умного» блока оценки корректности данных, представленных пользователем, и обобщения опыта выполненных вычислений с учетом значений выбранных параметров заданий (построение суррогатной модели пользователя);

- целевого перераспределения потока заданий на специализированные и аппаратно реконфигурируемые аппаратные блоки, используя предварительно созданные аппаратные реализации алгоритмов решаемых задач, обладающие высокой реальной производительностью выполнения алгоритмов решаемых задач;
- оперативной настройки за счет формирования целевого запроса к базе данных с файлами аппаратных конфигураций ПЛИС, вычислительных блоков кластера, архитектура которых отражает особенности аппаратных ускорителей используемых вычислительных алгоритмов.

Таблица 9.1 эффективность выполнения задач в разных областях знаний

Область знаний	Число запусков			Эффективность		
	успешно	снято	всего	Число успешных	время	ресурсы
Астрофизика						
Биоинформатика						
Биофизика						
Энергетика						
Геофизика						
ИТ						
Инжиниринг						
Механика						
Физика						
Радиофизика						

9.3 Использование технологии аппаратной реконфигурации для повышения реальной производительности гетерогенного кластера

Вычислители с фиксированной архитектурой обеспечивают рост производительности в основном за счет увеличения числа вычислительных

ядер (процессоров) и соответствующего увеличения потребляемой мощности (смотри рисунок 9.3): повышение производительности (R_{max}) в 2.8 раз достигнуто за счет повышения потребления энергии в 2.8 раза и увеличения количества вычислительных ядер в 3 раза. При такой тенденции для достижения уровня производительности в 1 Экзо Флопс (по тестам Linpack) потребуется более 15М ядер, а потребление энергии составит около 60МВт.

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
		3.0x	2.8x	2.5x	2.8x
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096

Рисунок 9.3 Источники роста производительности суперкомпьютеров
ТОП 500

Гетерогенные распределённые аппаратно реконфигурируемые вычислительные системы во многом лишены указанных выше недостатков фиксированных архитектур. Предлагаемая и реализуемая в СКЦ Политехнический концепция построения вычислительной системы включает три уровня: уровень Объяснения; уровень Агрегации; уровень доступа и предобработки больших данных (смотри рисунок 9.4). Архитектура вычислителей на всех уровнях иерархии структурно эквивалентна и является гетерогенной в широком, современном, понимании этого термина: т.е. включает аппаратно реконфигурируемые под решаемую задачу вычислительные узлы. Это могут быть, в зависимости от иерархии, аппаратно реконфигурируемые кластеры, состоящие из реконфигурируемых блоков; реконфигурируемые блоки; реконфигурируемые модули и логические части отдельных микросхем, на базе которых реализуются системы на кристалле.

Кластеры/блоки/узлы/модули с аппаратно реконфигурируемой внутренней архитектурой, подстраиваемой под задачу, традиционно реализуемые на ПЛИС, во многом лишены указанных выше недостатков

фиксированных архитектур и позволяют получать экстремум функции приведенной на рисунке 9.1 для решаемой в данный момент задачи.

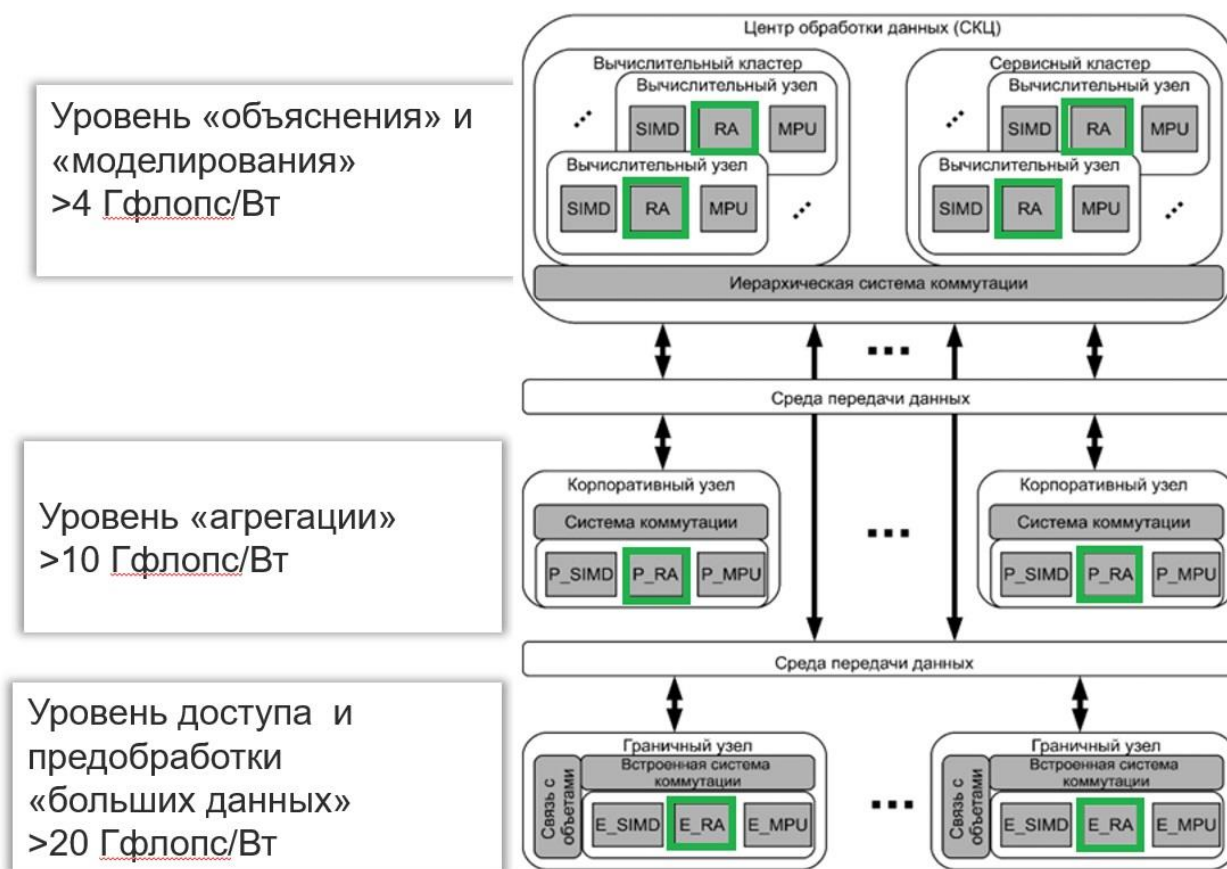


Рисунок 9.4 Концепция построения вычислительной системы в СКЦ
Политехнический

Структурно, системы с аппаратной реконфигурацией архитектуры под решаемую задачу (более точно – под реализуемый алгоритм решаемой задачи) – Аппаратно Реконфигурируемые Вычислительные Системы (АРВС), подобно системам параллельных вычислений, разделяют на классы, определяемые мерами: объем работы, выполняемый системой; издержки связи между элементами системы. Подобную классификацию называют детализацией или зернистостью или гранулярностью системы. В данной классификации выделяют: крупнозернистые, мелкозернистые АРВС и промежуточные – среднезернистые АРВС.

Для крупнозернистых АРВС характерна относительная независимость вычислений от других вычислительных устройств в системе, с редким обменом данными (как обрабатываемыми данными, так и конфигурационными данными, обеспечивающими аппаратную реконфигурацию вычислителя под реализуемый алгоритм) как между элементами системы, так и с центральным вычислителем, осуществляющим планирование и диспетчеризацию задач, т.е. с вычислителем, реализующим интеллектуальный мультиагентный диспетчер. Другими словами, крупнозернистые АРВС ориентированы на решение крупных вычислительных задач с изолированным набором обрабатываемых (исходных и результирующих) данных, т.е. для задач с так называемой пространственно-временной локализацией данных. Крупнозернистые АРВС обладают значимым аппаратным ресурсом и «высокой стоимостью», в контексте производительности вычислений, каналов связей с центральным вычислителем.

Для мелкозернистых АРВС характерна высокая скорость обмена данными (как обрабатываемыми данными, так и данными, обеспечивающими аппаратную реконфигурацию вычислителя под реализуемый алгоритм) с центральным вычислителем, осуществляющим планирование и диспетчеризацию задач, т.е. с вычислителем, реализующим интеллектуальный мультиагентный диспетчер. Другими словами, мелкозернистые АРВС ориентированы на решение функций (процедур, классов), входящих в состав решаемой задачи, с динамически меняющимися данными. Обычно мелкозернистые АРВС реализованы на отдельных ПЛИС, зачастую обладающими существенными аппаратными ресурсами для аппаратной реализации алгоритмов реализуемых функций, и имеют локальные модули хранения данных большого объема и высокой пропускной способностью каналов обмена. Модули хранения данных в мелкозернистых АРВС могут быть предназначены как для хранения оперативных данных, объем таких модулей измеряется десятками Гигабайт, так и для хранения системных

данных, объем таких модулей измеряется десятками Терабайт. Мелкозернистые АРВС обладают аппаратным ресурсом достаточным для аппаратной реализации алгоритмов функций решаемой задачи и «низкой стоимостью», в контексте производительности вычислений, каналов связей с центральным вычислителем.

Среднезернистые АРВС – промежуточное решение, обычно реализуемое подобно мелкозернистой АРВС, но имеющее не одну, а несколько ПЛИС, с высокой степенью связанности между собой, и существенный объем модулей для хранения оперативных данных для каждой ПЛИС.

Для реализации первого шага в создании гетерогенной распределенной аппаратно реконфигурируемой суперкомпьютерной вычислительной системы и построения на ее базе «высокоуровневой» архитектуры аппаратных компонент мультиагентного диспетчера, использующего каналы информационного обмена между агентами, в рамках текущего этапа проекта выполнена закупка, установка, настройка и ввод в эксплуатацию аппаратно-программного реконфигурируемого вычислительного блока на базе «Терциус-3», выпущенного компанией ООО "НИЦ СЭ И НК" (РФ, г. Таганрог), предназначенного для отладки разрабатываемого программного обеспечения интеллектуального диспетчера.

Блок предназначен для управления высокопроизводительным реконфигурируемым вычислительным кластером (ВРВК) СКЦ, реализованным также на базе блоков «Терциус-3». В состав кластера входит 10 Реконфигурируемых Вычислительных Блоков «Терциус-3» (РВБ «Терциус-3»), объединенных друг с другом медными и оптоволоконными каналами Ethernet 1000 через расположенные в стойке коммутаторы (смотри рисунок 9.5).

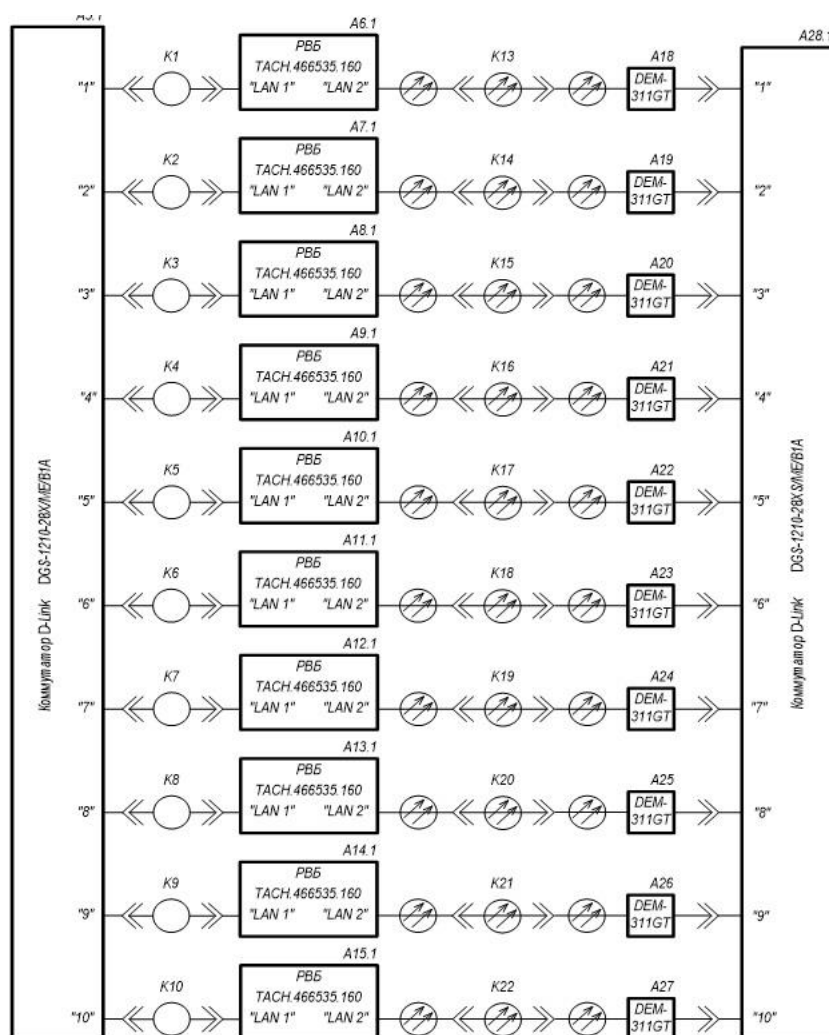


Рисунок 9.5 Структура ВРВК «Терциус-3»

Система коммутации состоит из двух независимых коммутаторов:

- D-Link DGS-1210-28X/ME/B1A для подключения к проводным каналам Ethernet;
 - D-Link DGS-1210-28XS/ME/B1A для подключения к оптическим каналам Ethernet,
- позволяющих объединять все РВБ в единую сеть по внутренним 10-ти проводным и 10 оптическим каналам Ethernet и подключать эту сеть к внешним сетям по 8 оптическим каналам Ethernet (по 4 канала для каждого из коммутаторов).

РВБ «Терциус-3» (смотри рисунок 9.6) предназначен для ускорения вычислительно трудоёмких функций при автоматизированной обработке данных. Заявленная производителем (ООО "НИЦ СЭ И НК")

производительность, пиковая – 5.6 ТФлопс, при максимальной потребляемой мощности 1000Вт.

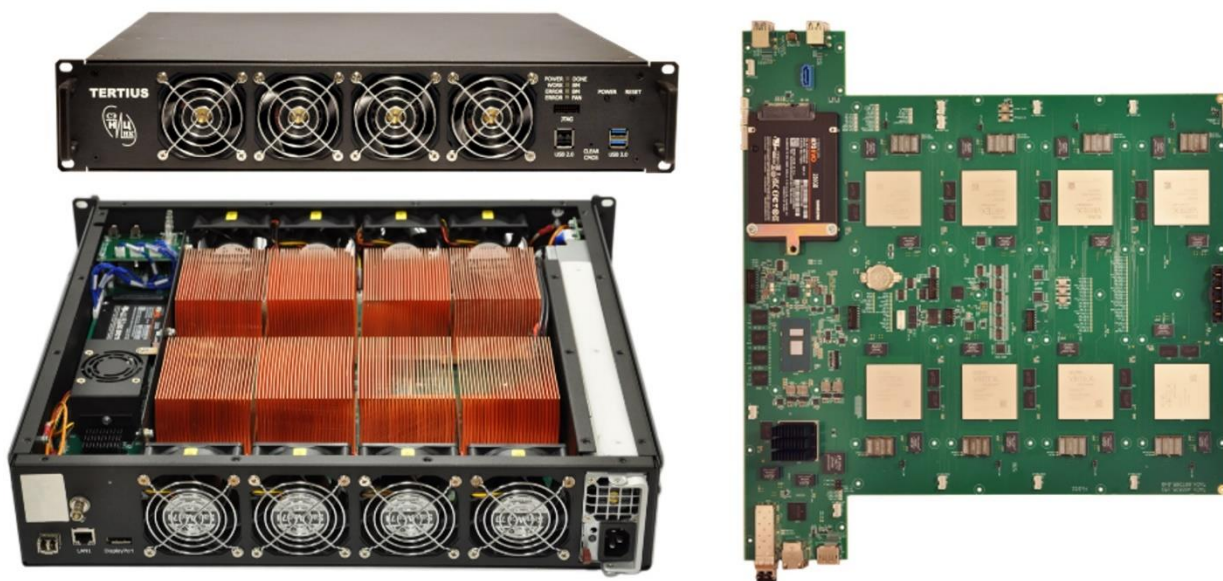


Рисунок 9.6 РВБ "Терциус 3" - общий вид и вид платы без корпуса

Каждый РВБ «Терциус-3» имеет в своём составе (смотри рисунок 9.7):

- плату вычислительного модуля TACH.466535.159, которая содержит:
 - восемь программируемых логических интегральных схем (ПЛИС) XCVU095-1FFVB1760C фирмы Xilinx;
 - управляющий модуль (УМ), включающий в себя:
 - процессор Core i56300U фирмы Intel,
 - четыре микросхемы оперативной памяти MT52L1G32D4PG107WT (объёмом 32 Гбит каждая);
 - модуль загрузки и управления на базе ПЛИС XC7K325T-2FFG900I;
 - порты интерфейсов USB2, USB3,
 - DisplayPort – порт для подключения монитора;
 - Ethernet порты – медный и оптический;
- плату индикации TACH.468365.002 содержащую:
 - элементы управления и индикации;

- порт интерфейса JTAG;
- блок питания SUPERMICRO PWS-1K21P1R,
- систему воздушного охлаждения (не показанную на рисунке 9.7),

которая включает в себя:

- вентилятор TACH.632550.058 - для охлаждения процессора Core i56300U;
- вентиляторы TACH.632550.057 для охлаждения элементов вычислительного поля - ПЛИС XCVU095-1FFVB1760С фирмы Xilinx.

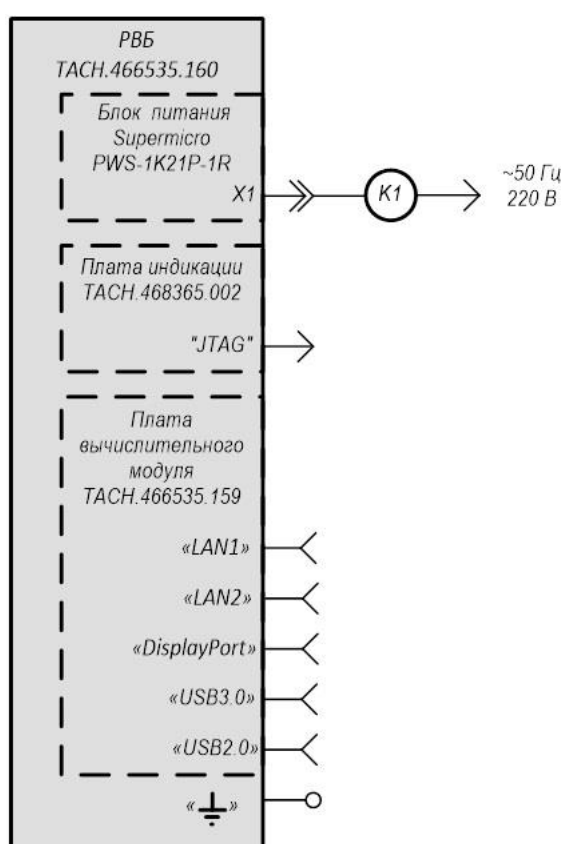


Рисунок 9.7 Структура подключения РВБ "Терциус 3"

Структура вычислительного поля одного РВБ «Терциус-3» содержит:

- 8 микросхем ПЛИС XCVU095-1FFVB1760С фирмы Xilinx, являющихся вычислительными (ПЛИС ВП);
- одну ПЛИС XC7K325Т, которая выполняет функции контроллера платы и устройства сопряжения с ПК (ПЛИС МЗУ).

Структурная схема связей вычислительных ПЛИС, включает как связи, имеющиеся между ПЛИС, так и связи с модулями памяти DDRL3 RAM, выполненными на микросхемах MT41K512M16HA, емкостью 1Гбайт каждый (смотри рисунок 9.8).

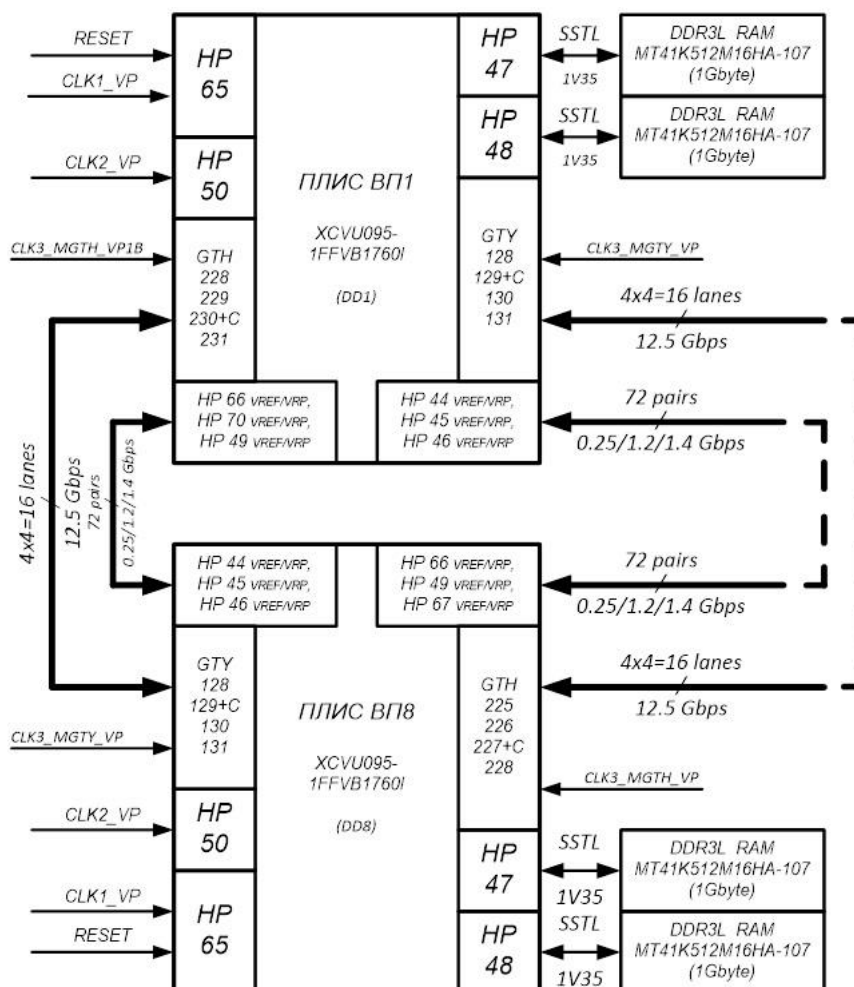


Рисунок 9.8 Структурная схема связей вычислительных ПЛИС

ПЛИС ВП подключены в цепочку, согласно кольцевой топологии: DD1=>DD2=>DD3=>DD4=>DD5=>DD6=>DD7=>DD8=>DD1. Для связи ПЛИС ВП между собой, допускается использовать стандарты LVDS, LVCMOS18 и MGT.

Для информационного обмена между ПЛИС DD1 и ПЛИС МЗУ следует использовать отдельный интерфейс, который предполагает работу в двух режимах: FAST и SLOW. В режиме SLOW одной командой производится

чтение/запись одного данного. В режиме FAST с помощью одной команды можно выполнить потоковое чтение/запись множества данных (до 256 Мбайт). Интерфейс является синхронным, все его сигналы со стороны ПЛИС МЗУ формируются на частоте 250 МГц. Сигналы со стороны ПЛИС DD1 должны быть также сформированы на частоте 250 МГц.

РВБ «Терциус-3» относятся к классу крупнозернистых АРВС по следующим параметрам:

- Логическая емкость, совокупность аппаратных ресурсов для реализации вычислительного алгоритма, высока. Что определяется наличием в РВБ «Терциус-3» восьми микросхем XCVU095 сверхвысокой логической емкости (смотри рисунок 9.8).

- Стоимость, в контексте производительности вычислительной системы, обмена обрабатываемыми данными с центральным вычислителем, реализующим интеллектуальный мультиагентный диспетчер, и с другими вычислительными устройствами (включая РВБ «Терциус-3», собранные в Кластер) высока. Что определяется:

- наличием только низкоскоростных каналов Ethernet 1000 (смотри рисунок 9.5) связи с внешними устройствами,
- включением в процедуру обмена обрабатываемыми данными низкоскоростного процессора Core i56300U УМ;
- особенностями архитектуры РВБ «Терциус-3» позволяющей осуществлять обмен данными с внешними вычислительными устройствами только через первую и восьмую ПЛИС в цепочке (смотри рисунок 9.8).

- Стоимость, в контексте производительности, обмена конфигурационными данными с центральным вычислителем, реализующим интеллектуальный мультиагентный диспетчер. Что определяется:

- наличием только низкоскоростных каналов Ethernet 1000 (смотри рисунок 9.5) связи с внешними устройствами,

- включением в процедуру обмена конфигурационными данными, т.е. в процедуру конфигурирования ПЛИС в РВБ «Терциус-3», промежуточного (относительно центрального вычислителя с мультиагентным диспетчером) звена - низкоскоростного процессора Core i56300U модуля УМ и модуля загрузки и на базе ПЛИС XC7K325T;
- особенностями архитектуры РВБ «Терциус-3» позволяющей осуществлять обмен данными с внешними вычислительными устройствами только через первую и восьмую ПЛИС в цепочке (смотри рисунок 9.8).

Одним из недостатков использования APBC, относящимся к любому классу зернистости, является сложность традиционных процедур разработки аппаратных реализаций алгоритмов на ПЛИС, основанных как на схемном описании, так и на использовании языков описания аппаратуры (Hardware Description Language -HDL), например, языков VHDL, Verilog/SystemVerilog HDL.

Процесс создания вычислителя, аппаратно оптимизированного под алгоритм решаемой функции, а тем более задачи, трудоемок и требует существенных временных затрат как на этапе его разработки, так и на этапе его отладки. Это зачастую не позволяет провести исследование и сравнительных анализ разных вариантов аппаратных реализаций алгоритма решаемой функции/задачи. Что приводит к аппаратным решениям по производительности близким к решениям на основе вычислителей с фиксированными архитектурами. К настоящему времени существует ряд работ, связанных с созданием аппаратных реализаций вычислительно сложных алгоритмов и демонстрирующих указанные выше особенности и недостатки.

Современным подходом при создании аппаратной реализации алгоритма решаемой функции/задачи является использование возможностей средств высокоуровневого синтеза – синтеза аппаратных решений из

описаний, созданных на высокоуровневых языках программирования, обычно на языках Си и C++. Подобные средства предоставляют как ведущие производители FPGA, такие как компания Xilinx и Intel PSG, так и компании, занимающиеся созданием средств разработки электронных устройств, например, компания Mentor Graphics.

В состав Аппаратно-Программного Комплекса РВБ «Терциус-3» (АПК РВБ «Терциус-3») входит Программное Обеспечение (ПО), предназначенное для высокоуровневого синтеза аппаратных реализаций решаемых задач:

- системное ПО процессора Core i56300U, входящего в состав УМ;
- интегрированная среда разработки (IDE);
- транслятор и синтезатор языка COLAMO;
- библиотеки схемотехнических ядер интерфейсов обмена данными;
- библиотеки схемотехнических ядер обработки данных.

Системное ПО обеспечивает диагностику неисправностей РВБ «Терциус-3» и мониторинг его эксплуатационных параметров (напряжения, тока, мощности, состояния вентиляторов).

Интегрированная среда разработки предназначена для создания на языке высокого уровня COLAMO описаний аппаратных средств, разрабатываемых для решения прикладных задач на базе РВБ «Терциус-3» и обеспечивает:

- редактирование и трансляцию исходных программ на языке высокого уровня COLAMO с последовательным запуском транслятора языка программирования высокого уровня COLAMO и синтезатора конфигурационных файлов для синтеза загрузочных конфигурационных файлов многокристального схемотехнического решения вычислительного поля РВБ и управляющей программы для УМ РВБ «Терциус-3»;
- отображение информационных, предупреждающих и сообщений об ошибках, возникающих при трансляции исходной программы на языке высокого уровня COLAMO РВБ.

Подобные средства позволяют не только создать некоторое, базовое, не оптимизированное аппаратное решение, но и провести исследование и

сравнительный анализ различных вариантов аппаратных решений, отличающихся: степенью параллелизма, количеством ступеней конвейера, интерфейсом к памяти данных, структурной организацией памяти данных и другими параметрами, связанными с эффективностью (критерии: производительность, аппаратные затраты) создаваемых аппаратных решений.

Эффективность, производительность и аппаратные затраты, конечного результата, аппаратной реализации алгоритма решаемой задачи, существенно зависят от выбранного алгоритма решения задачи и подобранных параметров процедуры высокоуровневого синтеза. Процедура получения оптимального (критерии: производительность, аппаратные затраты) конечного результата (аппаратной реализации алгоритма решаемой задачи) не формализована, эвристическая, и требует проведения исследований с использованием имитационного моделирования и сравнительного анализа.

Поэтому, в общем случае, подобные «оптимальные решения» создаются заранее, т.е. не в процессе решения вычислительной задачи, или как часто говорят – offline для конкретного РВБ. Такие «оптимальные решения» хранятся в центральном вычислителе и по мере необходимости, при принятии соответствующего решения интеллектуальным мультиагентным диспетчером, динамически загружаются в аппаратно реконфигурируемый вычислительный узел (например, в РВБ «Терциус-3»).

Подобная процедура управления на аппаратном уровне архитектурой вычислительной системы, реализуемая путем выбора «оптимальной» из «возможных» (заранее собранных) аппаратных реализаций для алгоритма решения прикладной задачи, реализуемая интеллектуальным мультиагентным диспетчером, может рассматриваться как простейшая реализация интеллектуальным диспетчером машины Гёделя, т.е. реализация «Искателя Доказательств» Гёделя со слабым искусственным интеллектом (ИИ).

9.4 Направления развития гетерогенной распределенной аппаратно реконфигурируемой суперкомпьютерной вычислительной системы СКЦ Политехнический и интеллектуального диспетчера

Дальнейшими направлениями развития Гетерогенной Распределенной Аппаратно Реконфигурируемой Суперкомпьютерной Вычислительной Системы СКЦ Политехнический и реализации следующего поколения «высокоуровневой» архитектуры аппаратных компонент мультиагентного диспетчера и алгоритмов, реализуемых мультиагентным диспетчером, являются:

– закупка, установка, настройка и ввод в эксплуатацию кластера мелкозернистых APBC (смотри рисунок 9.9);

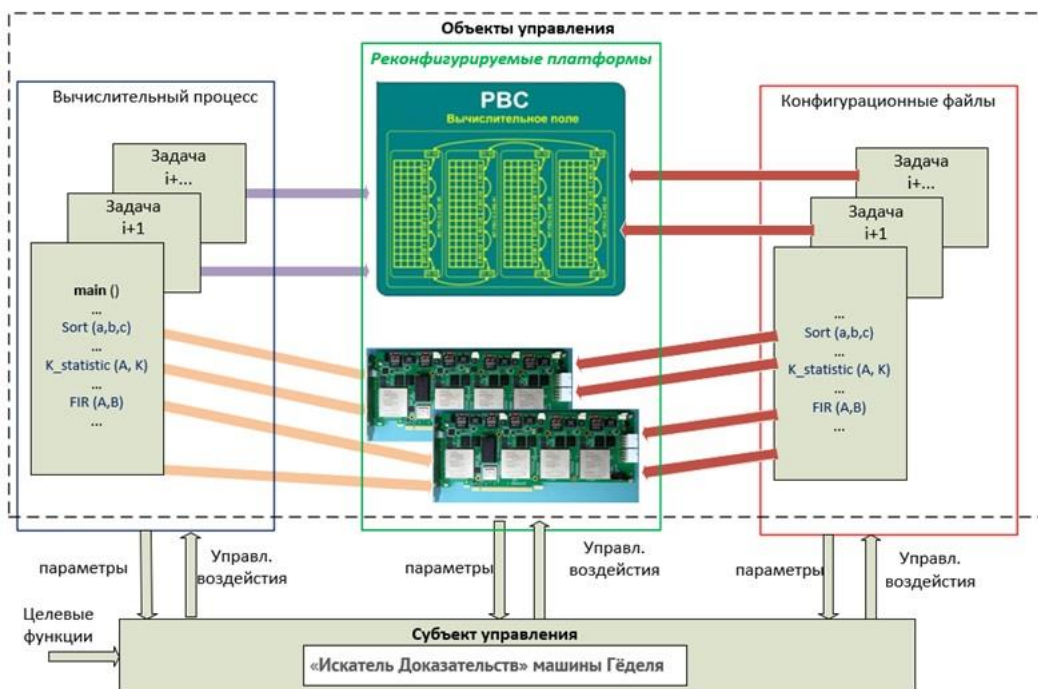


Рисунок 9.9 Планируемый подход к интеграции и управлению мелкозернистой APBC в СКЦ Политехнический

– реализация интеллектуальным мультиагентным диспетчером «Искателя Доказательств» Гёделя с сильным искусственным интеллектом (ИИ), обеспечивающим управление процессом вычислений на основе

реализации процедур «обучения» вычислительной системы «решать» различные задачи, а именно:

- на аппаратном уровне – аппаратно реконфигурировать ресурсы используемых РВБ в соответствии с алгоритмом решаемой задачи;
 - на программном уровне - управлять «траекторией» алгоритма решения задачи/комплекса связанных задач в пространстве аппаратных возможностей вычислительной системы;
 - на системном (интеллектуальном) уровне - реализовывать автоматизированное создание «оптимальных решений» для аппаратных реализаций алгоритмов решаемых задач.
- построение распределенного интеллектуального мультиагентного диспетчера, учитывающего распределенных характер объектов управления (вычислительных ресурсов), разнородную архитектуру и разные целевые функции, характерные для уровней Объяснения, Агрегации, Доступа и предобработки больших данных (смотри рисунок 9.10).

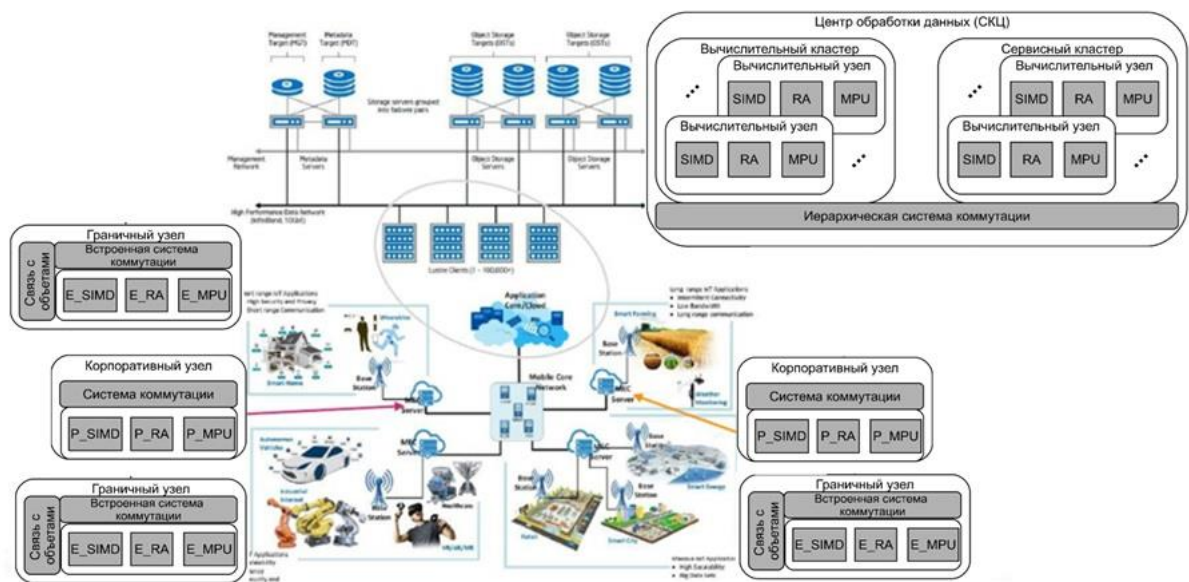


Рисунок 9.10 Концепция распределенного интеллектуального мультиагентного диспетчера

10 Разработка протоколов агрегации информационно-вычислительных и коммуникационных ресурсов агентов, участвующих в диспетчерском управлении ресурсами гетерогенного суперкомпьютерного кластера

10.1 Концепция организации данных информационно-вычислительных ресурсов агентов в интересах системы машинного обучения средств диспетчеризации

Протоколы агрегации информационно-вычислительных и коммуникационных ресурсов агентов, участвующих в диспетчерском управлении ресурсами гетерогенного суперкомпьютерного кластера, определяются как наборы данных и правила их преобразования для получения объективного состояния и выбора оптимальной стратегии управления вычислительным суперкомпьютерным кластером. Для этого применяются как локальные решения, оптимизирующие процедуры диспетчеризации планировщика SLARM [149], так и комплексные подходы на примере агрегатной модели Everest [150] или OSTI.

В рассматриваемой концепции протоколы являются информационным обеспечением системы управления суперкомпьютерного кластера, а передаваемая ими информация являться как исходными данными для непосредственного принятия решений, так и опосредовано, в качестве источника данных для пополнения выборок систем машинного обучения.

Рассмотрев, типовой процесс подготовки, диспетчеризации, выполнения и анализа результатов процесса исполнения задачи, на гетерогенном суперкомпьютере, изложенный в разделе 2.2.3 настоящего отчета, при реализации мультиагентного подхода, необходимо выделить набор специфических агентов, обеспечивающих реализацию задач каждого этапа.

Специфика агентов состоит в:

- направленности взаимодействия – с аппаратурой или пользователями;
- скорости информационного взаимодействия: от микросекунд до недель;
- объемами генерируемых данных;
- способами взаимодействия с другими агентами: централизованный или децентрализованный;
- способом реализации: программное программно-аппаратное или аппаратное решение.

С функциональной точки зрения агенты могут быть классифицированы на агентов сбора данных и агентов, принимающих решения. Действия каждого агента основаны на модели, некоторые из которых являются интеллектуальными, в частности, обучающимися, примером такого агента является система моделирования системы управления заданиями [151].

В этом случае можно сформировать иерархическую многоагентную среду управления (см. рисунок 10.1), спецификой которой является ее применение для оптимизации работы гетерогенного суперкомпьютерного кластера, примером которой может являться обучающая система на основе параметрической настройки алгоритмов работы агентов [152,153,154]. Каждый уровень иерархии представляет множество экземпляров агентов, обеспечивающих реализацию конкретной функции.

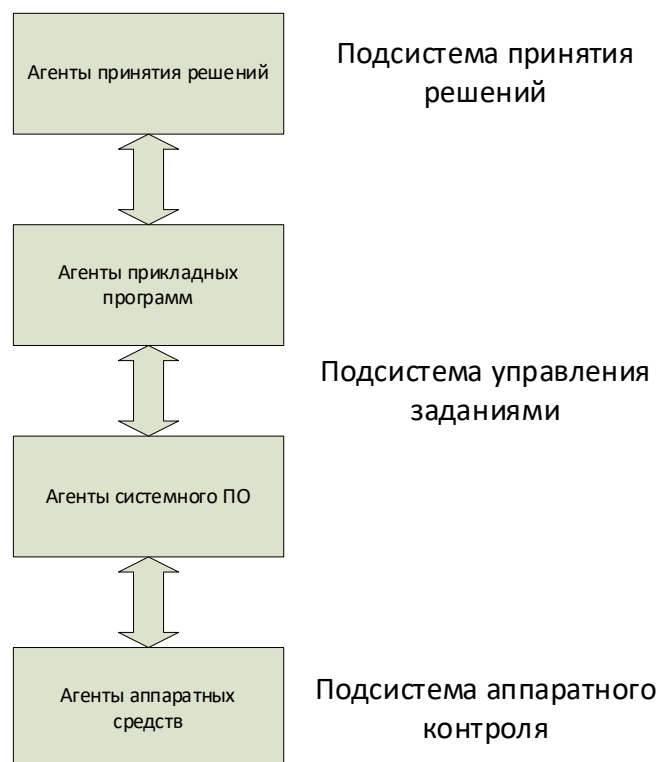


Рисунок 10.1 Иерархия агентов гетерогенной суперкомпьютерной системы

Агенты принятия решений обеспечивают высокоуровневый анализ и определение стратегии управления задачами в суперкомпьютерной системе. В задаче суперкомпьютерного управления эти агенты обеспечивают анализ и формирование обучающих выборок, применение вывода для получения прогноза, объяснения правил вывода результатов. Агенты прикладных программ анализируют состояния прикладных процессов и их заявок и поставляют исходные данные для агентов принятия решений. Агенты системного ПО контролируют состояния функционирования процессов для определения времен фактического старта и завершения задач пользователя. Агенты системного ПО осуществляют контроль за загрузкой и выгрузкой заданий пользователей на аппаратные средства и предоставляют такие функции, как загрузки виртуальных машин, подготовка и запуск, мониторинг и останов вычислительных процессов пользователей. Агенты аппаратных средств осуществляют мониторинг аппаратных средств, средств межпроцессорной коммуникации и общей памяти для динамического

определения доступных аппаратных ресурсов с целью обеспечения эффективного планирования.

Число и специализация агентов мультиагентной распределенной системы определяются исходя из особенностей гетерогенной суперкомпьютерной платформы, и, в целом агенты уровня принятия решений функционируют каждый в единственном экземпляре, агенты прикладных программ – по числу запускаемых пользователем прикладных процессов, а агенты системного ПО и аппаратных средств группируются исходя из наборов компонентов аппаратуры.

10.2 Протоколы организации взаимодействия агентов в интересах системы машинного обучения средств диспетчеризации

Протоколы обмена данными между агентами обеспечивают прием и передачу данных, что обеспечивает взаимодействие между уровнями иерархии. По мере повышения уровня иерархии обеспечивается обобщение данных и их преобразование для пригодного для выполнения анализа вида. Принимаемые агентами верхнего уровня решения детализируются и превращаются в отдельные команды для управления прикладными задачами, системным ПО и оборудованием.

Типовой процесс подготовки задачи пользователя в суперкомпьютерной системе приведен на рисунке 10.2.

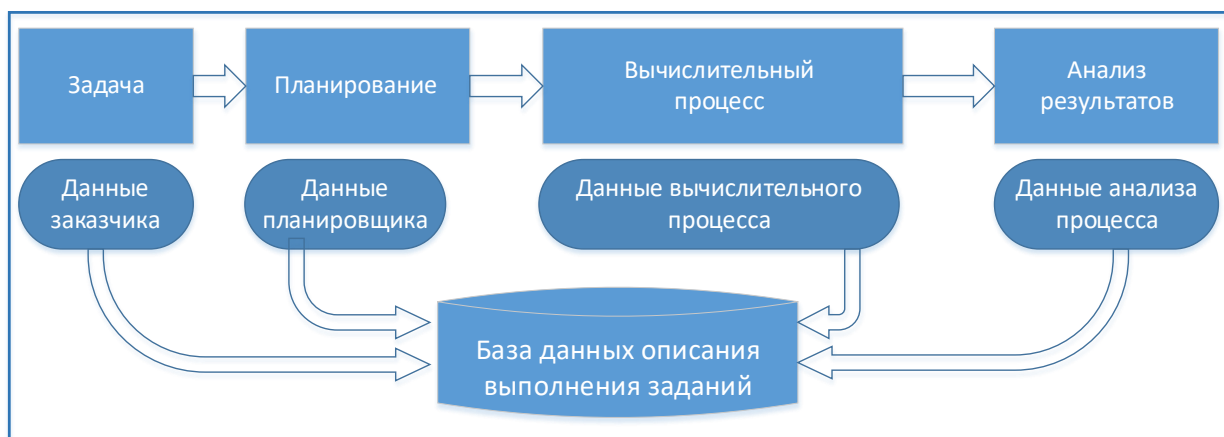


Рисунок 10.2 Типовой процесс выполнения задачи пользователя и генерируемые ими данные

Учитывая специфику процесса, можно выделить набор агентов для каждого из этапов для каждого из уровней иерархии. Перечень основных агентов, их функции и кратность присутствия в системе приведены в таблице 10.1.

Таблица 10.1 Перечень агентов системы диспетчеризации гетерогенной суперкомпьютерной платформы

№	Агент	Функции	Кратность в системе	Уровень иерархии
1	2	3	4	5
Описания задачи				
1	Анализа задачи	Выделение объективных характеристик программы: язык программирования, размер, объем запрашиваемых ресурсов, используемые библиотеки, запрашиваемое время на выполнение, желаемый момент запуска	По числу задач активных задач пользователя в очереди	Прикладных программ

Продолжение таблицы 10.1

1	2	3	4	5
2	Анализа профайла пользователя	Выделение объективных характеристик профайла пользователя, запускающего задачу: периодичность выполнения всех задач и конкретной задачи, успешность прошлых запусков, соответствие ранее выделенного требуемого и запрашиваемого времен для других задач	По числу пользователей с активными задачами	Прикладных программ
Планирования				
3	Анализа доступных ресурсов	Сбор и мониторинг доступных программных и аппаратных ресурсов: доступных узлов, процессоров и ядер, локальной и глобальной оперативной памяти, пропускной способности шин данных, лицензий на прикладное программное обеспечение	один на гетерогенный узел	Системных программ
4	Анализа процессов	мониторинг текущего состояния	Один на гетерогенный узел	Системных программ

Продолжение таблицы 10.1

1	2	3	4	5
5	Анализа очереди процессов	мониторинг запросов пользователя на постановку задач в очередь.	Один в системе	Системных программ
Вычислительного процесса				
6	Анализа задач пользователя	мониторинг вычислительных процессов выполняемых задач	по числу вычислительных процессов	Системных программ
7	Анализа аппаратных средств	мониторинг технического состояния процессов пользователя	по числу модулей в гетерогенной системе	Аппаратных средств
Анализа результатов				
8	Анализа результатов	Сбора данных о результате вычислительного процесса	по числу вычислительных процессов	Системных программ
9	Настройки задания пользователя	Формирование оптимальных параметров запуска пользовательских заданий.	один в системе	принятия решений

Схема взаимодействия агентов в иерархии представлена на рисунке 10.3.

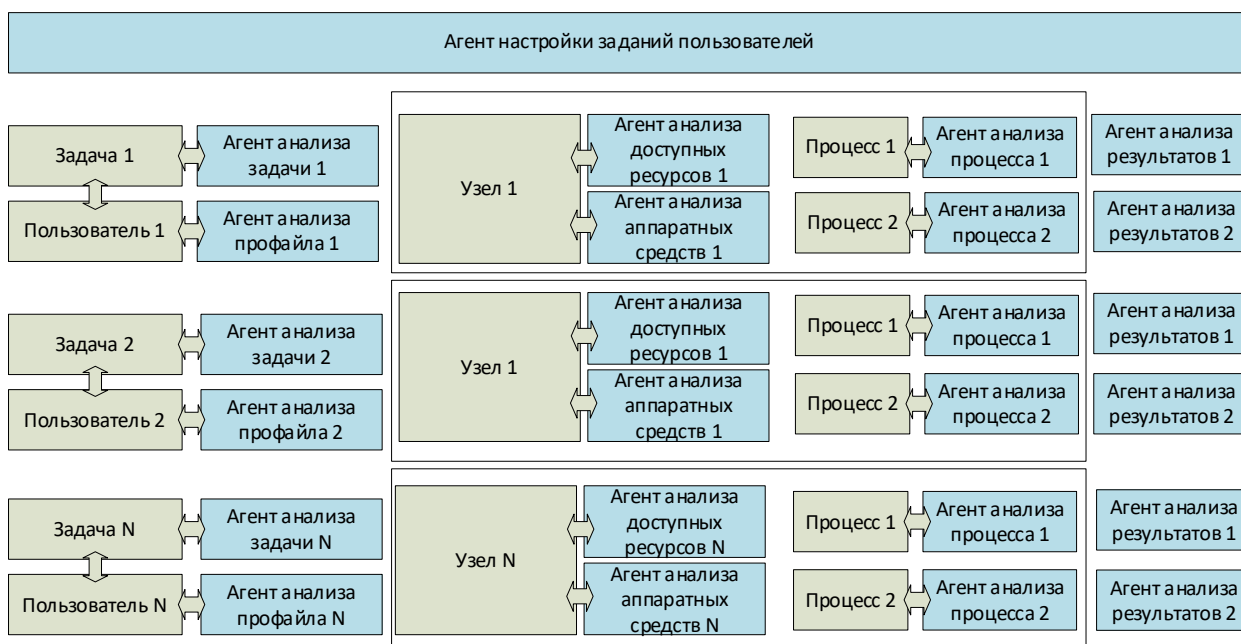


Рисунок 10.3 Схема взаимодействия агентов в иерархии

Протоколы, обеспечивающие взаимодействие между агентами, осуществляют передачу данных для согласованного и непрерывного потока управления вычислительным процессом в автоматическом режиме. Источником данных в агентах для формирования потоков данных на каждом уровне иерархии могут являться:

- аппаратные события, содержащие данные о состоянии аппаратуры, признак занятости канала передачи данных, исправности узла;
- программные события, содержащие сообщения о запуске, останове, аварийном или нормальном завершении работы программ;
- события, генерируемые пользователями, к которым относятся запрос пользователя на постановку задачи в очередь или удаление задачи из очереди, изменения приоритета выполнения программы, добавления или уменьшения объема аппаратного ресурса для программы;
- результаты работы алгоритмов интеллектуального предсказания результатов выполнения программы.

Реализуемые протоколы передачи данных представляют децентрализованную распределённую систему обмена сообщениями между агентами. Агенты запускаются и завершают работу автоматически на каждом

этапе жизненного цикла функционирования задачи. Схема потоков данных межагентного взаимодействия приведена на рисунке 10.4.

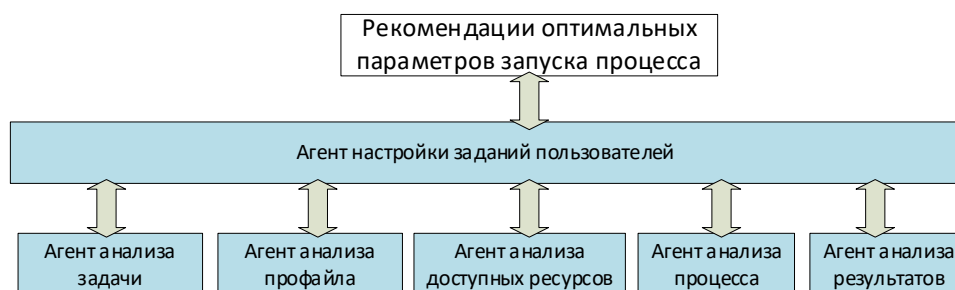


Рисунок 10.4 Схема взаимодействия агентов в иерархии

Для информационного обмена между агентами выделяются и используются ресурсы серверных узлов и локальной вычислительной сети со стандартными протоколами информационного обмена. Информационное взаимодействие осуществляется по стандартизированным протоколам таким, как SOAP, REST, XML-RPC в форматах JSON и/или XML.

Технологически, при разработке агентной модели может быть использован шаблон «представление – модель – контроллер» (Model-View-Controller, MVC).

Основными компонентами данного метода являются три компонента: модель, представление и контроллер.

Модель – объектно-ориентированное представление прикладных данных; модель реализует бизнес-логику прикладного уровня и является независимой от возможных способов визуализации данных.

Представление – интерфейс, обеспечивающий взаимодействие агентов и управляющих элементов.

Контроллер – основной элемент шаблона MVC, обеспечивающий прием пользовательского запроса, выполнение необходимых операций с объектами модели и генерацию представления, содержащего результаты выполнения запроса.

Схема архитектуры «модель-представление-контроллер» приведена на рисунке 10.5.

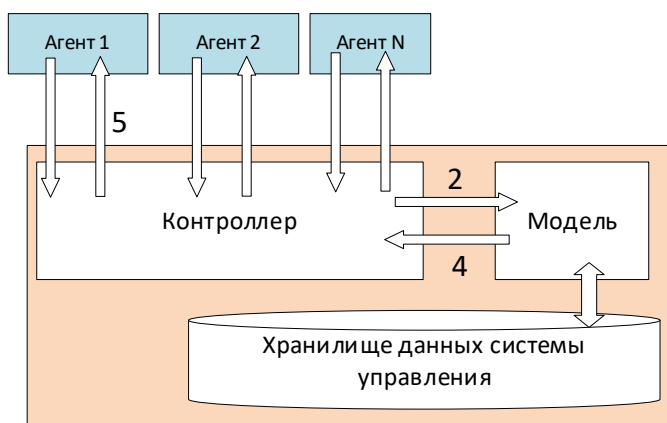


Рисунок 10.5 Схема архитектуры «модель-представление-контроллер»

Протокол взаимодействия компонентов состоит из 5 последовательно выполняемых шагов.

Агент через перечень заранее разработанных представлений строит запрос к приложению и отправляет его на сервер.

Контроллер принимает запрос и передает на выполнение объектам модели.

Объекты модели взаимодействуют с базой данных в процессе выполнения запроса.

Результат выполнения запроса передается контроллеру.

Контроллер строит представление, содержащее результаты выполнения запроса, и возвращает его Агенту пользователя.

Представленная модель обеспечивает динамическое асинхронное взаимодействие произвольного числа агентов разного уровня иерархии.

10.3. Управление потоком задач непрерывной подготовки данных для совершенствования технологий машинного обучения

Решение задачи управления данными о диспетчеризации систем требует разработки и реализации специализированного бизнес-процесса организации разработки и функционирования задачи интеллектуальной диспетчеризации.

Рассматриваемый процесс «как есть», основан на режиме автоматического подбора параметров запуска прикладных программ на гетерогенной суперкомпьютерной платформе, и основан на выборе средних характеристик, не учитывающих специфику аппаратного обеспечения, программных реализаций и вычислительной платформы. Типичное описание процесса подготовки данных приведено на рисунке 10.6.

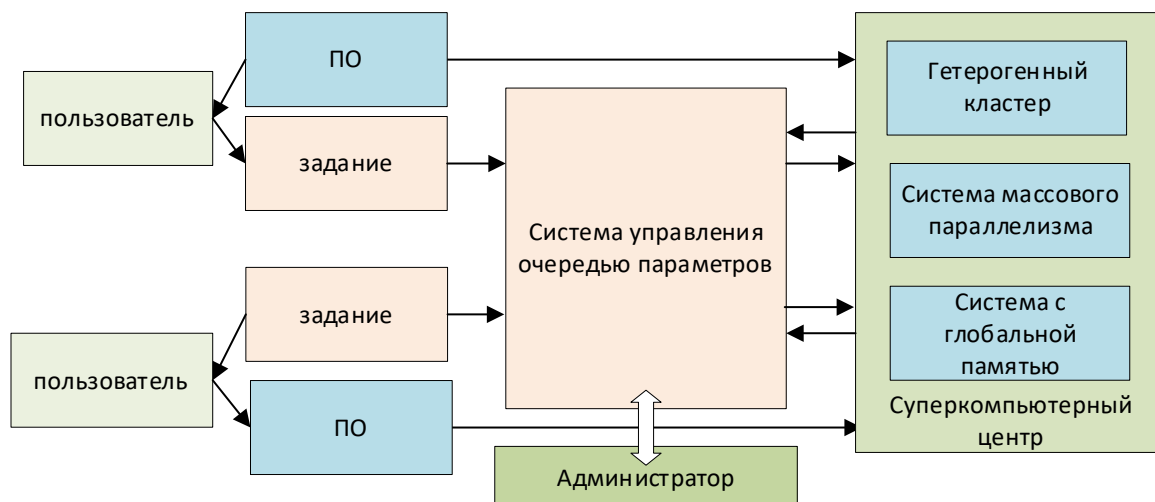


Рисунок 10.6 Типовой процесс реализации планирования выполнения задачи в суперкомпьютерной системе «как есть»

Типовой процесс подготовки процедур планирования выполнения заданий состоит в:

- формировании пользователем пакета программного обеспечения;
- представлении пользователем задания на загрузку пакета программного обеспечения;
- постановку задания в очередь системой управления очередью;
- вычисления момента загрузки процесса в одну из подсистем суперкомпьютерного центра;
- загрузки программного обеспечения;
- запуска вычислительного процесса;
- мониторинга процесса и записи логов;
- получения результата вычислений и передачу их пользователю.

В ходе процесса сохраняется масса дополнительных данных, требующих ручного анализа. В указанный процесс может вмешаться администратор, который в ручном режиме может обеспечить изменение параметров запуска той или иной задачи.

Ручное изменение параметров администратором не обеспечивает оптимальное управление системой в целом, а только ускоряет или замедляет выполнение конкретной задачи.

Для оптимизации функционирования системы управления заданиями в целом необходим комплексный подход к формированию параметров заданий на основе данных как о запускаемой задаче, так и о контексте системы в целом. Предложенный подход состоит в разработке дополнительной системы управления, которая может не только корректировать отдельные параметры запускаемых программ, но и корректировать общие состояния системы в целом. Пример реализации такой схемы приведен на рисунке 10.7.

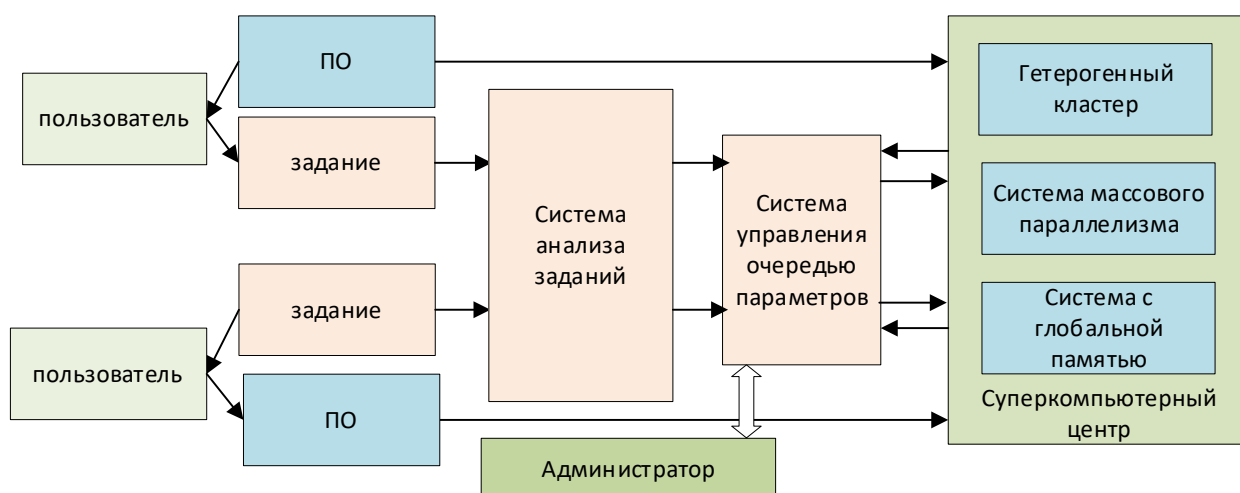


Рисунок 10.7 Планирования выполнения задачи с использованием переназначения параметров заданий в суперкомпьютерной системе «как будет»

Обновленный процесс подготовки процедур планирования выполнения заданий состоит в:

- формировании пользователем пакета программного обеспечения;

- представлении пользователем задания на загрузку пакета программного обеспечения;
- анализ параметров заданий в соответствии с текущим состоянием вычислительной системы;
- передачу задания планировщику заданий;
- постановку задания в очередь системой управления очередью;
- вычисления момента загрузки процесса в одну из подсистем суперкомпьютерного центра;
- загрузки программного обеспечения;
- запуска вычислительного процесса;
- мониторинга процесса и записи логов;
- получения результата вычислений и передачу их пользователю.

Функцией системы анализа заданий является определение оптимальных параметров запуска новой задачи, и таких изменений параметров, уже ожидающих в очереди, но еще не выполняющихся задач так, чтобы общее время выполнения задания было оптимально. Такая система может функционировать в автоматическом режиме, выбирая и переписывая значения параметров заданий так, чтобы общее время выполнения задания было оптимальным.

В целом, такая задача может быть решена средствами машинного обучения, для чего требуется формирование дополнительного контура ручного управления, который обеспечивает формирование репрезентативных выборок данных, что обеспечивается этапом разработки системы анализа заданий.

Такая подсистема может быть организована статически, когда данные исследованы, а процедуры реализованы на этапе разработки системы или динамически, когда в процессе функционирования обеспечивается изменение наборов данных и формирование обучающей компоненты непосредственно в динамическом режиме.

Первый способ организации подсистемы обеспечивает устойчивость функционирования системы в стационарных режимах, а второй обеспечивает гибкую настройку при значительных изменениях в задачах, решаемых на суперкомпьютерном кластере.

Заключение

В рамках первого этапа проекта осуществлялась разработка методов повышения реальной производительности гетерогенных суперкомпьютерных платформ на основе применения технологий искусственного интеллекта, мультиагентного управления и машинного обучения.

Проведён анализ существующих методов диспетчеризации ресурсов суперкомпьютерных вычислителей, показаны проблемы оптимизации работы суперкомпьютерной системы с помощью планировщика Slurm, приведено описание структуры и функционирования СКЦ «Политехнический», планировщика задач Slurm и жизненного цикла задачи пользователя, а также выполнен обзор аналогичных исследований.

Показано, что потенциал Slurm позволяет достигать высокой производительности вычислений, повышать пропускную способность системы, сокращать время обработки процесса, снижать время ожидания для задач пользователей и время ответа на запросы пользователей. Использование Slurm для управления гетерогенными вычислительными кластерами предоставляет мощные возможности для оптимизации ресурсов, но также требует формирования подходов к настройке и обслуживанию системы. При этом выявлена проблема несоответствия между ресурсами, запрошенными пользователем для выполнения задачи, и ресурсами, выделенными планировщиком, когда время, запрашиваемое пользователем для задачи, не совпадает с временем, которое фактически было потрачено на выполнение задачи, что приводит к неэффективному использованию ресурсов и неравномерному энергопотреблению и задержкам в выполнении других задач.

Показано, что внедрение мультиагентного подхода и методов машинного обучения в систему управления ресурсами Slurm позволит значительно улучшить эффективность использования суперкомпьютера и предоставить более высококачественный сервис для пользователей, решив проблему несоответствия между запрошенными и выделенными ресурсами. При этом по результатам анализа источников показано, что среди аналогичных исследований единый

эффективный подход к решению задачи планирования заданий пользователя на текущий момент отсутствует.

Представлено описание мультиагентного подхода к диспетчированию ресурсов СК. Приведена архитектура мультиагентного диспетчера потока заданий, функционирующего с использованием досок объявлений. Предложен метод мультиагентного диспетчирования для обеспечения многоуровневого адаптивного распределения ресурсов гетерогенного кластера. Показано, что в идеальном случае агенты должны четко и однозначно понимать всю специфику задания и стараться решить задания оптимальным образом с точки зрения суперкомпьютерной системы в целом.

Предложен метод информационного взаимодействия в мультиагентных диспетчерах на базе отдельно функционирующих серверов, выполняющих роль досок объявлений – открытой платформы информационного обмена между агентами и постановщиком заданий.

В работе проведена разработка статистической модели функционального управления распределением ресурсов гетерогенного кластера, которая позволяет оптимизировать распределение ресурсов между узлами кластера для достижения максимальной эффективности использования ресурсов. Модель может использоваться для прогнозирования необходимости дополнительных ресурсов для обеспечения нормальной работы кластера в будущем. Это может помочь в раннем планировании и предотвращении проблем, связанных с нехваткой ресурсов. Помимо перечисленного, модель может использоваться для определения необходимых мер для снижения нагрузки на отдельные узлы кластера и предотвращения перегрузки. Это может включать в себя меры такие как масштабирование узлов кластера, миграция задач на другие узлы или изменение конфигурации ресурсов на каждом узле. В частности, модель также может использоваться для определения наиболее эффективных способов распределения ресурсов в зависимости от текущей нагрузки и доступных ресурсов.

В работе разработан и приведен алгоритм мультиагентного диспетчерского управления вычислительными ресурсами гетерогенного суперкомпьютерного кластера с использованием методов искусственного интеллекта, позволяющий эффективно управлять ресурсами гетерогенного суперкомпьютера.

Проведен анализ данных журналов пользователей. Показано, что с точки зрения решаемой задачи, наиболее важной метрикой считается Polynomial confusion accuracy, поскольку эта метрика в большей степени учитывает специфику предметной области, разнородность и несбалансированность обучающей выборки. Наилучшие результаты по данной метрике демонстрируют методы RSF и гибридный метод «Clustering + LGBM + RSF». Результаты демонстрируют неоднозначность использования общих регрессионных критериев оценки моделей машинного обучения в целевой задаче оценки времен исполнения задач на СК. Актуальной видится проблема дальнейшего развития критериев оценки качества моделей машинного обучения при решении целевой задачи.

В результате анализа журналов поведения пользователей и взаимодействия ресурсов СКЦ на наборе данных за период с середины июня по середину ноября 2022 года был отмечен ряд выводов о структуре заданий на СКЦ «Политехнический», показано, что процент успешно выполненных задач существенно зависит от квалификации пользователей и предметной области исследований.

Создан гетерогенный стенд для моделирования процессов диспетчеризации при помощи slurm. Собранный на стенде статистическая информация может использоваться в дальнейшем для мониторинга и оптимизации работы кластера, а также для диагностики и устранения проблем. Она может быть также использована для оптимизации распределения ресурсов между задачами и пользователями, чтобы обеспечить максимальную производительность и эффективность. Также статистика может использоваться для прогнозирования и планирования расширения кластера в будущем. Созданный прототип мультиагентного вычислительного кластера может использоваться для разработки и тестирования интеллектуального диспетчера ресурсов.

Использование мультиагентной системы позволяет создавать реалистичные ситуации и тестировать диспетчер в реальных условиях.

В рамках текущего этапа проекта выполнена закупка, установка, настройка и ввод в эксплуатацию аппаратно-программного реконфигурируемого вычислительного блока на базе «Терциус-3», выпущенного компанией ООО "НИЦ СЭ И НК" (РФ, г. Таганрог), предназначенного для отладки разрабатываемого программного обеспечения интеллектуального диспетчера. Разработана концепция построения вычислительной системы СКЦ Политехнический, которая включает три уровня: уровень Объяснения; уровень Агрегации; уровень доступа и предобработки больших данных. Архитектура вычислителей на всех уровнях иерархии структурно эквивалентна и является гетерогенной в широком, современном, понимании этого термина: т.е. включает аппаратно реконфигурируемые под решаемую задачу вычислительные узлы (например, аппаратно реконфигурируемые кластеры, состоящие из реконфигурируемых блоков).

В работе также предложены методы информационно взаимодействия агентов, участвующих в процессе диспетчеризации. Разработана иерархическая модель информационного взаимодействия агентов гетерогенной суперкомпьютерной системы. Показана модель планирования выполнения задачи с использованием переназначения параметров заданий в суперкомпьютерной системе.

Задачи первого этапа работы выполнены в полном объеме в срок, соответствующий заданию на выполнение работ по первому этапу темы государственного задания № FSEG-2022-0001.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 J. D. Ullman, "Np-complete scheduling problems," Journal of Computer and System sciences, vol. 10, no. 3, pp. 384–393, 1975.
- 2 Timothy G. Mattson, An Overview of the Intel TFLOPS Supercomputer // MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) URL: http://www.ai.mit.edu/projects/aries/course/notes/ascii_red.pdf (Дата обращения: 25.12.2022).
- 3 Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. / International J. Supercomputer Applications, 15(3), 2001.
- 4 D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," in Workshop on Job Scheduling Strategies for Parallel Processing, 1997, pp. 1–34.
- 5 A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," in IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 6, pp. 529–543, June 2001, doi: 10.1109/71.932708.
- 6 D. Talby and D.G. Feitelson, "Supporting Priorities and Improving Utilization of the IBM SP Scheduler Using Slack-Based Backfilling," Proc. 13th Int'l Parallel Processing Symp., pp. 513–517, Apr. 1999.
- 7 W. Tang, Z. Lan, N. Desai and D. Buettner, "Fault-aware, utility-based job scheduling on Blue, Gene/P systems," 2009 IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–10, doi: 10.1109/CLUSTER.2009.5289206.
- 8 Fatos Xhafa, Ajith Abraham, "Computational models and heuristic methods for Grid scheduling problems", Future Generation Computer Systems, Volume 26, Issue 4, 2010, Pages 608–621, <https://doi.org/10.1016/j.future.2009.11.005>.
- 9 Fatih Say, Cüneyt F. Bazlamaçcı, A reconfigurable computing platform for real time embedded applications, Microprocessors and Microsystems, Volume

36, Issue 1, 2012, Pages 13-32, <https://doi.org/10.1016/j.micpro.2011.08.013>.

10 Akihiro Misawa, Susumu Date, Keichi Takahashi, Takashi Yoshikawa, Masahiko Takahashi, Masaki Kan, Yasuhiro Watashiba, Yoshiyuki Kido, Chonho Lee, and Shinji Shimojo. 2017. Highly Reconfigurable Computing Platform for High Performance Computing Infrastructure as a Service: Hi-IaaS. In Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017). SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 163–174. <https://doi.org/10.5220/0006302501630174>

11 Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour, “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Advances in Neural Information Processing Systems 12, pp. 1057-1063, MIT Press, 2000.

12 Anurag Agarwal, Selcuk Colak, Varghese S. Jacob, Hasan Pirkul, “Heuristics and augmented neural networks for task scheduling with non-identical machines”, European Journal of Operational Research, Volume 175, Issue 1, 2006, Pages 296-317, <https://doi.org/10.1016/j.ejor.2005.03.045>.

13 Seyed Mehdi Mohtavipour, Hadi Shahriar Shakhoseini, A quad-form clustered mapping approach for large-scale applications of reconfigurable computing systems, Computers & Electrical Engineering, Volume 97, 2022, 107637, <https://doi.org/10.1016/j.compeleceng.2021.107637>.

14 Derya Eren Akyol, G. Mirac Bayhan, “A review on evolution of production scheduling with neural networks”, Computers & Industrial Engineering, Volume 53, Issue 1, 2007, Pages 95-122, <https://doi.org/10.1016/j.cie.2007.04.006>.

15 Feitelson, D.G. (2001). Metrics for Parallel Job Scheduling and Their Convergence. In: Feitelson, D.G., Rudolph, L. (eds) Job Scheduling Strategies for Parallel Processing. JSSPP 2001. Lecture Notes in Computer Science, vol 2221. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45540-X_11

16 Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. 2020. RLScheduler: an automated HPC batch job scheduler using reinforcement learning. In Proceedings of the International Conference for High Performance

Computing, Networking, Storage and Analysis (SC '20). IEEE Press, Article 31, 1–15.

17 Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16). Association for Computing Machinery, New York, NY, USA, 50–56. <https://doi.org/10.1145/3005745.3005750>

18 Danilo Carastan-Santos and Raphael Yokoingawa de Camargo, “Obtaining Dynamic Scheduling Policies with Simulation and Machine Learning”, SC’17 -2 International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing), Nov 2017, Denver, United States. hal-01618940

19 Dror G Feitelson, Dan Tsafir, and David Krakov. 2014. Experience with using the parallel workloads archive. J. Parallel and Distrib. Comput. 74, 10 (2014), 2967–2982.

20 Ayala I., Amor M., Fuentes L. The Sol agent platform: Enabling group communication and interoperability of self-configuring agents in the Internet of Things //Journal of Ambient Intelligence and Smart Environments. – 2015. – T. 7. – №. 2. – C. 243-269. DOI: 10.3233/AIS-150304

21 Campbell, A., Wu, A.S. Multi-agent role allocation: issues, approaches, and multiple perspectives. Auton Agent Multi-Agent Syst 22, 317–355 (2011). <https://doi.org/10.1007/s10458-010-9127-4>

22 El Fallah-Seghrouchni, A., Piette, F., Caval, C. and Taillibert, P. (2018) ‘A multi-agent platform for the deployment of ambient systems’, Int. J. Agent-Oriented Software Engineering, Vol. 6, Nos. 3/4, pp.369–401, doi.org/10.1504/IJAOSE.2018.096435

23 L. Xue, C. Sun, D. Wunsch, Y. Zhou and F. Yu, "An adaptive strategy via reinforcement learning for the prisoner’s dilemma game," in IEEE/CAA Journal of Automatica Sinica, vol. 5, no. 1, pp. 301-310, Jan. 2018, doi: 10.1109/JAS.2017.7510466.

- 24 H. Wang, T. Huang, X. Liao, H. Abu-Rub and G. Chen, "Reinforcement Learning for Constrained Energy Trading Games With Incomplete Information," in *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3404-3416, Oct. 2017, doi: 10.1109/TCYB.2016.2539300.
- 25 Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., & Yu, Y. (2018). MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1). <https://doi.org/10.1609/aaai.v32i1.11371>
- 26 Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6382–6393.
- 27 M. Pipattanasomporn, M. Kuzlu, W. Khamphanchai, A. Saha, K. Rathinavel and S. Rahman, "BEMOSS: An agent platform to facilitate grid-interactive building operation with IoT devices," 2015 IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA), Bangkok, Thailand, 2015, pp. 1-6, doi: 10.1109/ISGT-Asia.2015.7387018.
- 28 Anubhav Choudhary, Indrajeet Gupta, Vishakha Singh, Prasanta K. Jana, A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing, *Future Generation Computer Systems*, Volume 83, 2018, Pages 14-26, <https://doi.org/10.1016/j.future.2018.01.005>.
- 29 Wang, Y., Jiang, J., Xia, Y., Wu, Q., Luo, X., & Zhu, Q. (2018). A Multi-stage Dynamic Game-Theoretic Approach for Multi-Workflow Scheduling on Heterogeneous Virtual Machines from Multiple Infrastructure-as-a-Service Clouds. *IEEE International Conference on Services Computing*.
- 30 Eiman Iranpour, Saeed Sharifian, A distributed load balancing and admission control algorithm based on Fuzzy type-2 and Game theory for large-scale SaaS cloud architectures, *Future Generation Computer Systems*, Volume 86, 2018, Pages 81-98, <https://doi.org/10.1016/j.future.2018.03.045>.

- 31 R. Duan, R. Prodan and X. Li, "Multi-Objective Game Theoretic Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds," in *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 29-42, Jan.-March 2014, doi: 10.1109/TCC.2014.2303077.
- 32 Lei Wu & Yuandou Wang, 2018. "Scheduling Multi-Workflows Over Heterogeneous Virtual Machines With a Multi-Stage Dynamic Game-Theoretic Approach," *International Journal of Web Services Research (IJWSR)*, IGI Global, vol. 15(4), pages 82-96.
- 33 C. A.C. Coello, G. T. Pulido, and M. S. Lechuga. 2004. Handling multiple objectives with particle swarm optimization. *Trans. Evol. Comp* 8, 3 (June 2004), 256–279. <https://doi.org/10.1109/TEVC.2004.826067>
- 34 Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., & Xie, H. (2019). Multi-Objective Workflow Scheduling With Deep-Q-Network-Based Multi-Agent Reinforcement Learning. *IEEE Access*, 7, 39974-39982.
- 35 Mandeep Kaur, Sanjay Kadam, A novel multi-objective bacteria foraging optimization algorithm (MOBFOA) for multi-objective scheduling, *Applied Soft Computing*, Volume 66, 2018, Pages 183-195, <https://doi.org/10.1016/j.asoc.2018.02.011>.
- 36 Longxin Zhang, Kenli Li, Changyun Li, Keqin Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, *Information Sciences*, Volume 379, 2017, Pages 241-256, <https://doi.org/10.1016/j.ins.2016.08.003>.
- 37 Durillo, J.J., Prodan, R. Multi-objective workflow scheduling in Amazon EC2. *Cluster Comput* 17, 169–189 (2014). <https://doi.org/10.1007/s10586-013-0325-0>
- 38 Amandeep Verma, Sakshi Kaushal, A hybrid multi-objective Particle Swarm Optimization for scientific workflow scheduling, *Parallel Computing*, Volume 62, 2017, Pages 1-19, <https://doi.org/10.1016/j.parco.2017.01.002>.
- 39 Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, and Shiyan Hu. 2019. Minimizing cost and makespan for workflow scheduling in

cloud using fuzzy dominance sort based HEFT. *Future Gener. Comput. Syst.* 93, C (Apr 2019), 278–289. <https://doi.org/10.1016/j.future.2018.10.046>

40 Arabnejad, H., Barbosa, J.G. A Budget Constrained Scheduling Algorithm for Workflow Applications. *J Grid Computing* 12, 665–679 (2014). <https://doi.org/10.1007/s10723-014-9294-7>

41 Tao, M., Dong, S. & Zhang, L. A multi-strategy collaborative prediction model for the runtime of online tasks in computing cluster/grid. *Cluster Comput* 14, 199–210 (2011). <https://doi.org/10.1007/s10586-010-0145-4>

42 K. Li, X. Tang, B. Veeravalli and K. Li, "Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems," in *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 191-204, Jan. 2015, doi: 10.1109/TC.2013.205.

43 Zhou, N., Qi, D., Wang, X., Zheng, Z., and Lin, W. (2016), A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table, *Concurrency Computat.: Pract. Exper.*, doi: 10.1002/cpe.3944

44 Guoqi Xie, Renfa Li, Keqin Li, Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems, *Journal of Parallel and Distributed Computing*, Volume 83, 2015, Pages 1-12, <https://doi.org/10.1016/j.jpdc.2015.04.005>.

45 D. Bozdag, F. Ozguner and U. V. Catalyurek, "Compaction of Schedules and a Two-Stage Approach for Duplication-Based DAG Scheduling," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857-871, June 2009, doi: 10.1109/TPDS.2008.260.

46 Tong, Z., Chen, H., Deng, X. et al. A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization. *Soft Comput* 23, 11035–11054 (2019). <https://doi.org/10.1007/s00500-018-3657-0>

47 Liang A, Pang Y. A Novel, Energy-Aware Task Duplication-Based Scheduling Algorithm of Parallel Tasks on Clusters. *Mathematical and Computational Applications*. 2017; 22(1):2. <https://doi.org/10.3390/mca22010002>

- 48 Q. Wang, H. Zhang, C. Qu, Y. Shen, X. Liu, J. Li. RLSchert: An HPC Job Scheduler Using Deep Reinforcement Learning and Remaining Time Prediction. MDPI, Applied Sciences, октябрь 2021 10.3390/app11209448
- 49 K. Lamar, A. Goponenko, C. Peterson, B. A. Allan, J. M. Brandt, D. Dechev. Backfilling HPC Jobs with a Multimodal-Aware Predictor. IEEE, 2021 IEEE International Conference on Cluster Computing (CLUSTER), октябрь 2021 10.1109/Cluster48925.2021.00093
- 50 Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, Z. Lan. Trade-off between Prediction Accuracy and Underestimation Rate in Job Runtime Estimates. IEEE, 2017 IEEE International Conference on Cluster Computing (CLUSTER), сентябрь 2017 10.1109/CLUSTER.2017.11
- 51 Q. Wang, Y. Shen, J. Li. User-level Workload Analysis for Supercomputers. 2021 The 4th International Conference on Software Engineering and Information Management, январь 2021 <https://doi.org/10.1145/3451471.3451483>
- 52 T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, D. Tiwari. Job Characteristics on Large-Scale Systems: Long-Term Analysis, Quantification, and Implications. IEEE, SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, февраль 2021 10.1109/SC41405.2020.00088
- 53 S. Banerjee, J. Hecker. A Multi-Agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing. Springer, First Complex Systems Digital Campus World E-Conference 2015, сентябрь 2015 10.1007/978-3-319-45901-1_4 <https://arxiv.org/abs/1509.06420>
- 54 J. P. Pabico. A Framework for a Multiagent-based Scheduling of Parallel Jobs. arXiv, июнь 2015 <https://arxiv.org/abs/1506.07964>
- 55 Y. Fan, Z. Lan. DRAS-CQSim: A reinforcement learning based framework for HPC cluster scheduling. Elsevier, Software Impacts, май 2021 <https://doi.org/10.1016/j.simpa.2021.100077>
- 56 Y. Fan, B. Li, D. Favorite, N. Singh, T. Childers, P. Rich, W. Allcock,

M. E. Papka, Z. Lan. DRAS: Deep Reinforcement Learning for Cluster Scheduling in High Performance Computing. IEEE Transactions on Parallel and Distributed Systems (декабрь 2022) 10.1109/TPDS.2022.3205325

57 A. Iulian, O. Florin, P. I. Raicuc. New scheduling approach using reinforcement learning for heterogeneous distributed systems. Elsevier, Journal of Parallel and Distributed Computing, июль 2018 <https://doi.org/10.1016/j.jpdc.2017.05.001>

58 Y. Hu, C. de Laat, Z. Zhao. Learning Workflow Scheduling on Multi-Resource Clusters. IEEE, 2019 IEEE International Conference on Networking, Architecture and Storage (NAS), сентябрь 2019 10.1109/NAS.2019.8834720

59 N. Grinsztajn, O. Beaumont, E. Jeannot, P. Preux. READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling. IEEE, 2021 IEEE International Conference on Cluster Computing (CLUSTER), сентябрь 2021 10.1109/Cluster48925.2021.00031

60 M. Cheong, H. Lee, I. Yeom, H. Woo. SCARL: Attentive Reinforcement Learning-Based Scheduling in a Multi-Resource Heterogeneous Cluster. IEEE, IEEE Access, октябрь 2019 10.1109/ACCESS.2019.2948150

61 Jiang Y., Jiang J. Contextual resource negotiation-based task allocation and load balancing in complex software systems //IEEE Transactions on Parallel and Distributed Systems. – 2008. – Т. 20. – №. 5. – С. 641-653.

62 Каляев А.И., Каляев И.А. Метод мультиагентного диспетчирования ресурсов в облачных вычислительных средах // Известия Российской академии наук. Теория и системы управления. 2016. № 2. С. 51.

63 Городецкий В. И., Самойлов В. В., Троцкий Д. В. Базовая онтология коллективного поведения автономных агентов и ее расширения //Известия Российской академии наук. Теория и системы управления. – 2015. – №. 5. – С. 102-102.

64 Каляев И.А., Капустян С.Г. Метод мультиагентного управления «умным» интернет-производством // Робототехника и техническая кибернетика. 2018. № 1 (18). С. 34-48.

- 65 Wooldridge M. An introduction to multiagent systems. – John Wiley & sons, 2009. – 488 p.
- 66 Herrera M., Pérez-Hernández M., Kumar Parlikad A., Izquierdo J. Multi-agent systems and complex networks: Review and applications in systems engineering // Processes. – 2020. – v.8. – №. 3. – 312 p.
- 67 Anatoly Kalyaev; Korovin I., Adaptive Multiagent Organization of the Distributed Computations // AASRI Procedia 6 - 2014: 49–58p.
- 68 Carmel D., Markovitch S. Opponent modeling in multi-agent systems // International Joint Conference on Artificial Intelligence. – Springer, Berlin, Heidelberg, 1995. – С. 40-52.
- 69 Коровин Я.С., Капустян С.Г., Хисамутдинов М.В., Каляев А.И., Иванов Д.Я. Алгоритм функционирования агента облачного сервиса системы поддержки принятия решений // Суперкомпьютерные технологии (СКТ-2018). Материалы 5-й Всероссийской научно-технической конференции: в 2-х томах. 2018. С. 87-92.
- 70 Suganuma T., Oide T., Kitagami S., Sugawara K., Shiratori N. Multiagent-based flexible edge computing architecture for IoT //IEEE Network. – 2018. – Т. 32. – №. 1. – С. 16-23.
- 71 Каляев А.И. Теоретические основы создания самоорганизующихся диспетчеров распределенных систем на базе мультиагентного социоинспирированного подхода // Известия ЮФУ. Технические науки 2021 №4, С 6-21.
- 72 Shoham Y., Leyton-Brown K. Multiagent systems: Algorithmic, game-theoretic, and logical foundations. – Cambridge University Press, 2008.
- 73 Тарасов В.Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. М.: Эдиториал УРСС, 2002. - 353 с.
- 74 Городецкий В.И. Самоорганизация и многоагентные системы. I. Модели многоагентной самоорганизации // Известия Российской академии наук. Теория и системы управления. – 2012. – №. 2. – С. 92-92.

- 75 Ross Ashby W. Principles of the self-organizing dynamic system //The Journal of General Psychology. – 1947. – Т. 37. – №. 2. – С. 125-128.
- 76 Kauffman S.A., Stuart A. The origins of order: Self-organization and selection in evolution. – Oxford University Press, USA, 1993.
- 77 Карпов В.Э. Модели социального поведения в групповой робототехнике // Управление большими системами: сборник трудов. – 2016. – №. 59. – С. 65-232.
- 78 Новиков Д.А. Математические модели формирования и функционирования команд. – Москва: ФИЗМАТЛИТ. – 2008. – 185 С.
- 79 Georgeff M. Communication and interaction in multi-agent planning // Readings in distributed artificial intelligence. – Morgan Kaufmann, 1988. – С. 200-204.
- 80 Кулинич А.А. Модель кооперации агентов (роботов) // Труды Четырнадцатой национальной конференции по искусственному интеллекту с международным участием КИИ-2014. – 2014. – С. 24–27.
- 81 Легович Ю. С., Максимов Д. Ю. Выбор исполнителя в группе интеллектуальных агентов // Управление большими системами: сборник трудов. – 2015. – №. 56. – С.78-94.
- 82 Котенко И. В., Уланов А. В. Многоагентное моделирование защиты информационных ресурсов в сети Интернет //Известия Российской академии наук. Теория и системы управления. – 2007. – №. 5. – С. 74-88.
- 83 Conte R., Edmonds B., Moss S., Sawyer R.K. Sociology and social theory in agent based social simulation: A symposium // Comput. Math. Organ. Theory. Springer, 2001. – Vol. 7, № 3. – P. 183–205.
- 84 Кулинич А. А. Модель командного поведения агентов в качественной семиотической среде. Ч. 2. Модели и алгоритмы формирования и функционирования команд агентов // Искусственный интеллект и принятие решений. – 2018. – №. 1. – С. 29-40.
- 85 Semwal T., Jha S. S., Nair S. B. Tartarus: A multi-agent platform for bridging the gap between cyber and physical systems //Proceedings of the 2016

International Conference on Autonomous Agents & Multiagent Systems. – 2016. – С. 1493-1495.

86 Stone P., Kaminka G.A., Kraus S., Rosenschein J.S. Ad hoc autonomous agent teams: Collaboration without pre-coordination //Twenty-Fourth AAAI Conference on Artificial Intelligence. – 2010. – P. 1504-1509.

87 Palanca J., Terrasa A., Carrascosa C., Julián V. SimFleet: a new transport fleet simulator based on MAS //International Conference on Practical Applications of Agents and Multi-Agent Systems. – Springer, Cham, 2019. – С. 257-264.

88 P. Saint-Andre, Extensible Messaging and Presence Protocol (XMPP): Core, document RFC 6120, Internet Requests for Comments, RFC Editor, Mar. 2011. // URL: <http://www.rfc-editor.org/rfc/rfc6120.txt> (дата обращения: 26.12.2022).

89 Кулинич А. А. Модель командного поведения агентов в качественной семиотической среде. Часть 1. Качественная среда функционирования. Основные определения и постановка задачи // Искусственный интеллект и принятие решений. – 2017. – №. 3. – С. 38-48.

90 Li X., Bilbao S., Martín-Wanton T., Bastos J., Rodriguez J. SWARMs ontology: A common information model for the cooperation of underwater robots // Sensors. – 2017. – Vol. 17. – №. 3. – p. 569.

91 Капустян С., Каляев И., Гайдук А. Модели и алгоритмы коллективного управления в группах роботов. – Litres, 2018. – 289 С.

92 Palanca J., Terrasa A., Carrascosa C., Julián V. Improving the programming skills of students in multiagent systems master courses //Computer Applications in Engineering Education. – 2019. – Т. 27. – №. 4. – С. 836-845.

93 Mell J., Gratch J. Grumpy & Pinocchio: answering human-agent negotiation questions through realistic agent design //Proceedings of the 16th conference on autonomous agents and multiagent systems. – 2017. – С. 401-409.

94 Sanchis A., Juliá, V., Corchado J.M., Billhardt H., Carrascosa C. Using natural interfaces for human-agent immersion //International Conference on Practical Applications of Agents and Multi-Agent Systems. – Springer, Cham, 2014.

– С. 358-367.

95 Chalupsky H., Gil Y., Knoblock C.A., Lerman K., Oh J., Pynadath D.V., Tambe M. Electric Elves: Applying Agent Technology to Support Human Organizations //IAAI. – 2001. – Т. 1. – С. 51-58.

96 Малинецкий Г.Г. Теория самоорганизации. На пороге IV парадигмы // Компьютерные исследования и моделирование. – 2013. – Т. 5. – №. 3. – С. 315-366.

97 Кондратьев В. В., Жевнерчук Д. В. Применение методов теории самоорганизации в задачах управления профилированием и конфигурированием вычислительных систем // Доклады академии наук. – Федеральное государственное унитарное предприятие Академический научно-издательский, производственно-полиграфический и книгораспространительский центр Наука, 2014. – Т. 459. – №. 4. – С. 409-409.

98 Serugendo G.D.M., Gleizes M.P., Karageorgos A. Self-organization in multi-agent systems //The Knowledge engineering review. – 2005. – Т. 20. – №. 2. – С. 165-189.

99 Gaston M.E., DesJardins M. Agent-organized networks for dynamic team formation //Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. – 2005. – С. 230-237.

100 Just J.E., Cornwell M.R., Huhns M.N. Agents for establishing ad hoc cross-organizational teams // Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004. (IAT 2004). – IEEE, 2004. – С. 526-530.

101 O'Brien P. D., Nicol R. C. FIPA—towards a standard for software agents // BT Technology Journal. – 1998. – Т. 16. – №. 3. – С. 51-59.

102 Jelasity M., Babaoglu O. T-Man: Gossip-based overlay topology management //International Workshop on Engineering Self-Organising Applications. – Springer, Berlin, Heidelberg, 2005. – С. 1-15.

103 Охтилев М.Ю., Мустафин Н.Г., Миллер В.Е., Соколов Б.В. Концепция проактивного управления сложными объектами: теоретические и

технологические основы // Известия высших учебных заведений. Приборостроение. – 2014. – Т. 57. – №. 11. – С.7-14.

104 Calegari R., Ciatto G., Mascardi V., Omicini, A. Logic-based technologies for multi-agent systems: A systematic literature review. *Autonomous Agents and Multi-Agent Systems*, 35(1), –2021. – P. – 1-67.

105 Ulicny B., Thalmann D. Crowd simulation for interactive virtual environments and VR training systems // *Computer animation and simulation 2001*. – Springer, Vienna, 2001. – C. 163-170.

106 Almeida J. E., Rosseti R. J. F., Coelho A. L. Crowd simulation modeling applied to emergency and evacuation simulations using multi-agent systems // *arXiv preprint arXiv:1303.4692*. – 2013.

107 Grosz B. J., Kraus S. Collaborative plans for complex group action // *Artificial Intelligence*. – 1996. – Т. 86. – №. 2. – С. 269-357.

108 Jiang Y., Zhou Y., Wang W. Task allocation for undependable multiagent systems in social networks // *IEEE Transactions on Parallel and Distributed Systems*. – 2012. – Т. 24. – №. 8. – С. 1671-1681.

109 Cardoso R.C., Ferrando A. A Review of Agent-Based Programming for Multi-Agent Systems. *Computers* 2021, V.10 (16). – 2021.

110 Mahela O.P., Khosravy M., Gupta N., Khan B., Alhelou H. H., Mahla R., Siano P. Comprehensive overview of multi-agent systems for controlling smart grids // *CSEE Journal of Power and Energy Systems*. – 2020.

111 Simmonds J., Gómez J. A., Ledezma A. The role of agent-based modeling and multi-agent systems in flood-based hydrological problems: a brief review // *Journal of Water and Climate Change*. – 2020. – Т. 11. – №. 4. – С. 1580-1602.

112 Ma Z., Schultz M.J., Christensen K., Værbak M., Demazeau Y., Jørgensen B. N. The application of ontologies in multi-agent systems in the energy sector: a scoping review // *Energies*. – 2019. – Т. 12. – №. 16. – С. 3200.

113 Harley J.M., Bouchet F., Hussain M.S., Azevedo R., Calvo R. A multi-componential analysis of emotions during complex learning with an intelligent

multi-agent system //Computers in Human Behavior. – 2015. – T. 48. – C. 615-625.

114 Parunak H.V.D. " Go to the ant": Engineering principles from natural multi-agent systems //Annals of Operations Research. – 1997. – T. 75. – C. 69-101.

115 González-Briones A., De La Prieta F., Mohamad M.S., Omatu S., Corchado J.M. Multi-agent systems applications in energy optimization problems: A state-of-the-art review //Energies. – 2018. – T. 11. – №. 8. – C. 1928.

116 Calvaresi D., Dubovitskaya A., Calbimonte J.P., Taveter K., Schumacher M. Multi-agent systems and blockchain: Results from a systematic literature review // International conference on practical applications of agents and multi-agent systems. – Springer, Cham, 2018. – C. 110-126.

117 Liang C., Shanmugam B., Azam S., Karim A., Islam A., Zamani M., Idris N.B. Intrusion detection system for the internet of things based on blockchain and multi-agent systems //Electronics. – 2020. – T. 9. – №. 7. – C. 1120.

118 Gregori M. E., Cámara J. P., Bada G. A. A jabber-based multi-agent system platform //Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. – 2006. – C. 1282-1284.

119 Берман А.Ф., Николайчук О.А., Павлов А.И. Самоорганизующийся алгоритм формирования решений для обеспечения требуемого технического состояния сложных опасных объектов //Системный анализ и информационные технологии (САИТ-2017). – 2017. – С. 377-384.

120 Bellifemine F., Poggi A., Rimassa G. JADE–A FIPA-compliant agent framework //Proceedings of PAAM. – 1999. – T. 99. – №. 97-108. – C. 33.

121 Bergenti F., Caire G., Monica S., Poggi A. The first twenty years of agent-based software development with JADE //Autonomous Agents and Multi-Agent Systems. – 2020. – T. 34. – №. 2. – C. 1-19.

122 Poslad S., Buckle P., Hadingham R. The FIPA-OS agent platform: Open source for open standards //proceedings of the 5th international conference and exhibition on the practical application of intelligent agents and multi-agents. – 2000. – T. 355. – №. 20.

123 Caire G., Gotta D., Banzi M. Wade: a software platform to develop

mission critical applications exploiting agents and workflows //Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track. – 2008. – С. 29-36.

124 Bigus J.P., Schlosnagle D.A., Pilgrim J.R., Mills III W.N., Diao Y. ABLE: A toolkit for building multiagent autonomic systems //IBM Systems Journal. – 2002. – Т. 41. – №. 3. – С. 350-371.

125 Gutknecht O., Ferber J. The M ad K it agent platform architecture //Workshop on infrastructure for scalable multi-agent systems at the international conference on autonomous agents. – Springer, Berlin, Heidelberg, 2000. – С. 48-55.

126 Galland S., Gaud N., Rodriguez S., Hilaire V. Janus: Another yet general-purpose multiagent platform //Seventh AOSE Technical Forum, Paris. – 2010.

127 Baldoni M., Baroglio C., Capuzzimati F., Micalizio R. Commitment-based agent interaction in JaCaMo+ // Fundamenta Informaticae. – 2018. – Т. 159. – №. 1-2. – P. 1-33.

128 Bordini R. H., Hübner J. F. BDI agent programming in AgentSpeak using Jason //International workshop on computational logic in multi-agent systems. – Springer, Berlin, Heidelberg, 2005. – С. 143-164.

129 Ricci A., Viroli M., Omicini A. CArtAgO: A framework for prototyping artifact-based environments in MAS //International Workshop on Environments for Multi-Agent Systems. – Springer, Berlin, Heidelberg, 2006. – С. 67-86.

130 Куприянов М. С., Кочетков А. В. Мультиагентная модель самоорганизующейся распределенной системы // Известия СПбГЭТУ "ЛЭТИ". – 2016. – №. 2. – С. 12.

131 Гайдук А.Р. Алгоритмическое обеспечение самоорганизующихся регуляторов с экстраполяцией // Известия РАН. Теория и системы управления. – 2002. – №. 3. – С. 56-63.

132 Васильев С.Н. От классических задач регулирования к интеллектуальному управлению. I // Интеллектуальные системы. – 1999. – Т. 4. – №. 1-2. – С. 19-72.

133 Васильев С. Н., Опарин Г. А., Феокистов А. Г. Интеллектуальный подход к автоматизации моделирования сложных управляемых систем //Труды Международной конференции RDAMM-2001.–Новосибирск. – 2001. – Т. 6. – С. 159-168.

134 Махутов Н.А., Берман А.Ф., Николайчук О.А. Некоторые принципы самоорганизации для управления риском техногенных катастроф //Проблемы анализа риска. – 2015. – Т. 12. – №. 4. – С. 6-17.

135 Aydogan R., Sanchez V., Julian V., Broekens J., Jonker C. Guest editorial: computational approaches for conflict resolution in decision making: new advances and developments //Cybernetics and Systems. – 2014. – Т. 45. – №. 3. – С. 217-221.

136 Лопота А.В., Юревич Е.И. Самоорганизация в кибернетике и робототехнике // Робототехника и техническая кибернетика. – 2014. – №. 4. – С. 4-5.

137 Bernon C., Camps V., Gleizes M.P., Picard G. Tools for self-organizing applications engineering //International Workshop on Engineering Self-Organising Applications. – Springer, Berlin, Heidelberg, 2003. – С. 283-298.

138 Колесников А. В., Кириков И. А., Листопад С. В. Гибридные интеллектуальные системы с самоорганизацией: координация, согласованность, спор //М.: ИПИ РАН. – 2014.

139 Литвинцева Л. В., Ульянов С. В. Интеллектуальные системы управления. I. Квантовые вычисления и алгоритм самоорганизации //Известия Российской академии наук. Теория и системы управления. – 2009. – №. 6. – С. 102-141.

140 Mamei M., Menezes R., Tolksdorf R., Zambonelli, F. Case studies for self-organization in computer science //Journal of Systems Architecture. – 2006. – Т. 52. – №. 8-9. – С. 443-460.

141 Omicini A., Gardelli L. Self-Organisation & MAS. An Introduction [Электронный ресурс] // URL: <https://core.ac.uk/download/pdf/11193005.pdf> (дата обращения: 26.12.2022).

142 Жевнерчук Д. В. Моделирование процессов самоорганизации распределенных пространственно-временных ресурсов //Вестник Нижегородского университета им. Н.И. Лобачевского. – 2014. – №. 2-1. – С.218-222.

143 Маслобоев А.В. Самоорганизация проблемно ориентированных мультиагентных виртуальных пространств на основе градиентных вычислительных полей //Под редакцией д. филос. н. ЕА Никитиной Рецензенты: д. ф. м. н., проф. ВГ Редько д. филос. н., проф. Т. Н Семенова. – 2013. – С. 31-36.

144 Граничин О.Н. Круглый стол «Самоорганизация и искусственный интеллект в группе автономных роботов: методология, теория, практика» // Стохастическая оптимизация в информатике. – 2020. – Т. 16. – №. 1. – С. 5-12.

145 Кузнецова В. Л., Раков М. А. Самоорганизация в технических системах. – Наукова думка, 1987. – 200 С.

146 Каляев И.А., Гайдук А.Р., Капустян С.Г. Самоорганизация в мультиагентных системах // Известия Южного федерального университета. Технические науки. – 2010. – Т. 104. – №. 3. – С.14-20.

147 Anatoly Kalyaev; Korovin I., New Method to Use Idle Personal Computers for Solving Coherent Tasks // 2014 AASRI Conference on Circuit and Signal Processing (CSP 2014) Volume 9, 2014, P. 131–137

148 Коровин Я.С., Капустян С.Г., Каляев А.И., Хисамутдинов М.В., Иванов Д.Я. Сообщества агентов облачного сервиса системы поддержки принятия решений в облаке корпоративных вычислительных ресурсов нефтедобывающей компании // Суперкомпьютерные технологии (СКТ-2018). Материалы 5-й Всероссийской научно-технической конференции: в 2-х томах. 2018. С. 93-97.

149 Леоненков С. Н., Жуматий С. А. Оптимизация алгоритма Backfill и системы планирования заданий для использования на суперкомпьютере "Ломоносов" //Вычислительные технологии в естественных науках. Методы суперкомпьютерного моделирования. – 2015. – С. 140-149.

150 Сухорослов, О. В. Комбинированное использование высокопроизводительных ресурсов и грид-инфраструктур в рамках облачной платформы Everest / О. В. Сухорослов // Суперкомпьютерные дни в России : Труды международной конференции, Москва, 28–29 сентября 2015 года / Суперкомпьютерный консорциум университетов России, Федеральное агентство научных организаций России. – Москва: Московский государственный университет им. М.В. Ломоносова (Издательский Дом (Типография), 2015. – С. 706-711.

151 Кочин В. П., Жерело А. В. Облачный сервис PaaS для управления программными проектами пользователей суперкомпьютера СКИФ-БГУ. – 2016.

152 Феоктистов Александр Геннадьевич, Костромин Роман Олегович Извлечение знаний агентами в системе управления распределенными вычислениями // Информационные и математические технологии в науке и управлении. 2017. №3 (7).

153 Бычков И. В., Опарин Г. А., Феоктистов А. Г., Богданова В. Г., Пашинин А. А. Мультиагентные методы и инструментальные средства управления в сервис-ориентированной распределенной вычислительной среде // Труды ИСП РАН. 2014. №5.

154 Воеводин В. В., Жуматий С. А., Никитенко Д. А. Octoshell: система для администрирования больших суперкомпьютерных комплексов //Суперкомпьютерные дни в России. – 2015. – С. 69-83.