ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте на тему:

Разработка подсистемы динамического изменения приоритета вычислительных задач в очереди суперкомпьютера для расширения системы HPC TaskMaster

Выполнил студент:

группы БПМИИ225, 3 курса Рыбаков Георгий Сергеевич

Принял руководитель проекта:

Костенецкий Павел Сергеевич
начальник отдела суперкомпьютерного моделирования
к.ф.-м.н., доцент

Москва 2025

Содержание

1 Аннотация

С ростом числа задач в различных областях науки, требующих высокопроизводительных вычислений, правильное распределение приоритетов задач и квот пользователей стало одним из важнейших факторов влияющих на эффективную работу суперкомпьютерных центров [20]. Ручное изменение приоритетов задач, а также квот пользователей, позволяет подстраивать поток задач под внешние условия, например под сезонную нагрузку или снижение количества задач в выходные дни, но часто это сопряжено с задержками, связанными с человеческим фактором. В данной статье описана разработка подсистемы динамического изменения приоритетов задач и пользовательских квот, позволяющая более эффективно распоряжаться ресурсами суперкомпьютера.

Ключевые слова

Суперкомпьютер, эффективность, планировщик задач

2 Введение

2.1 Обзор предметной области

В настоящее время, особенно в области машинного обучения, становится все больше задач, требующих значительных вычислительных мощностей. Для решения таких задач в НИУ ВШЭ был построен суперкомпьютер «cHARISMa», имеющий пиковую производительность 2.2 Петафлопс FP64 и, на данный момент, обслуживающий более 1500 пользователей [18].

Чтобы эффективно распределять ресурсы и запускать задачи на суперкомпьютере «cHARISMa» используется планировщик задач SLURM [8]. Это один из самых распространенных планировщиков, использующийся крупнейшими суперкомпьютерными комплексами, входящими в мировой список Top500 [1]. Его популярность обуславливается открытостью исходного кода и используемыми эффективными алгоритмами планирования задач. Более подробно алгоритмы будут рассмотрены в разделе 3.1.

Кроме того, на суперкомпьютерном кластере НИУ ВШЭ используется собственная система мониторинга эффективности задач - HPC TaskMaster [9]. Эта система позволяет в реальном времени отслеживать потребляемые ресурсы, а также выявлять неэффективные задачи с помощью индикаторов и тегов.

Описанное выше программное обеспечение позволяет добиться комфортной работы для всех зарегистрированных пользователей суперкомпьютера. Например, пока TaskMaster позволяет выявлять неэффективные задачи, SLURM позволяет сохранять стабильность самой системы. Один из простых случаев, когда SLURM не дает блокировать работу суперкомпьютера: если один пользователь поставит в очередь множество задач, только первые будут запущены сразу, остальные же уйдут в очередь и получат меньший приоритет.

Загруженность суперкомпьютера и, вследствие этого, возможное максимальное количество выполняемых одновременно задач, определяется еще внешними факторами, например, такими как выходные или праздничные дни. Основываясь на подобных факторах, можно повышать или понижать ограничения задач, находящихся в очереди, для повышения итоговой производительности в количестве выполненных задач. На данный момент это происходит в ручном режиме, но это сопряжено с задержками, связанными с человеческим фактором.

2.2 Постановка цели и задач

Цель проекта: разработать подсистему динамического изменения приоритета вычислительных задач в очереди суперкомпьютера. Она должна работать как сторонний сервис, работающий в интеграции с HPC TaskMaster и включаться только на выходных. Для оптимизации работы, студентом должен быть предложен метод по оптимизации потока задач на базе статистики и эвристики.

Задачи, поставленные перед студентом:

- Провести анализ данных загруженности суперкомпьютера.
- Предложить архитектуру сервиса.
- Разработать систему.
- Сделать полное покрытие тестами.
- Упаковать систему в Docker-контейнер.

2.3 Структура работы

Сначала необходимо выполнить обзор литературы: узнать, какие алгоритмы использует планировщик задач «cHARISMa», как они работают, и какие методы использует система мониторинга эффективности. Далее необходимо воссоздать локально суперкомпьютер

НИУ ВШЭ через установку TaskMaster и настройку SLURM с теми же параметрами, как и на суперкомпьютере НИУ ВШЭ. После этого основная цель работы будет состоять в разработке подсистемы для управления планировщиком задач.

На момент написания промежуточного отчета была подготовлена и отправлена статья о разработке подсистемы на конференцию "Параллельные вычислительные технологии 2025".

3 Обзор литературы

3.1 Планировщик задач SLURM

3.1.1 Планирование потока задач

В разделе 2.1 уже упоминалось, что в «cHARISMa» используется планировщик задач SLURM. Он включает в себя несколько базовых концептов [14]:

- Задача - вычислительная задача, выполнение которой нужно пользователю.
- Узел - вычислительный узел в кластере, который может быть виртуальным или физическим. На каждом узле работает демон slurmd, отвечающий за управление и передачу информации одним определенным узлом.
- Управляющий узел - узел, на котором пользователи управляют выполнением задач. На этом узле расположен центральный демон slurmctld, который управляет всеми узлами кластера.
- Разделы - логические наборы узлов, определяемые ограничениями по максимальному времени выполнения задач, приоритетами и доступностью ресурсов для пользователя, максимальными размерами задач и т.д.
- Приоритет - устанавливаемый алгоритмами SLURM параметр задачи, на основе которого определяется, какая задача будет выполнена первой.

На базовом уровне планировщик практически не выполняет никакой работы по приоритизации. Без дополнительных конфигураций, SLURM использует принцип FIFO (First-in-First-out), что не позволяет грамотно распределять задачи. Максимальный приоритет получает первая задача, а выполнение задач выполняется строго по порядку их поступления.

Для того, чтобы дополнить базовые алгоритмы, в том числе и на суперкомпьютере «cHARISMa» используются встроенные плагины.

Чтобы не допускать нечестного выполнения задач, в SLURM существует модуль мультифакторной приоритизации. При выполнении он учитывает сразу девять факторов, ниже приведены некоторые из них:

- Возраст задачи - сколько задача находится в очереди;
- Fair-Share - признак "честного"распределения, зависящий от того, сколько ресурсов требуется для выполнения задачи и сколько ресурсов уже было занято пользователем;
- QoS - заранее заданные лимиты для каждого пользователя или групп пользователей.

Общая формула имеет следующий вид:

$Job_{priority} = site_{factor} + WeightAge * age_{factor} + WeightFairshare * fairshare_{factor} + WeightJobSize * jobsize_{factor} + ... - nice_{factor}$

В конечном итоге это позволяет правильнее распределять приоритеты задач, основываясь на разных метриках, тем самым давая большему количеству пользователей выполнять расчеты одновременно, и не позволяя парализовать работу суперкомпьютера в исключительных случаях [22].

В дополнение к Multifactor существует стандартный плагин Backfill. Он позволяет значительно ускорить производительность кластера за счет выполнения маленьких низкоприоритетных задач в промежутке между крупными, если их выполнение не увеличит общее время работы. Его примерная работа показана на Рисунке 3.1.

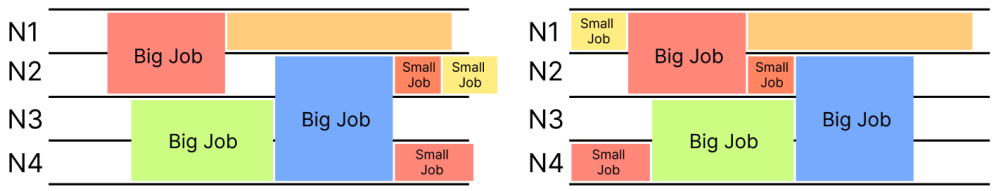


Рис. 3.1: Результат работы алгоритма backfill.

Работа плагина состоит из нескольких этапов. Сначала Backfill рассматривает каждое запущенное задание и рассчитывает их примерное время запуска, исходя из приоритета и дополнительных факторов. Если модуль понимает, что определенная задача может быть запущена на освободившихся ресурсах немедленно, при этом не влияя на уже рассчитанное время запуска задания выше его по приоритету, то SLURM начинает ее выполнение. Если же это невозможно, то плагин зарезервирует ресурсы узлов, чтобы задача была выполнена в планируемое время.

Каждое задание участвует в распределении столько раз, сколько разделов для него было запрошено. Главный фактор немедленного выполнения - максимальное время выполне-

ния задачи. Поэтому без определенных заранее аргументов планирование будет затруднено или даже бесполезно. В конечном итоге модуль позволяет сократить время простоя кластера и увеличить количество выполняемых задач за определенный промежуток времени.

Выше были описаны модули планирования. Но для моего проекта нужно рассмотреть еще два не менее важных аспекта работы SLURM - пользовательские лимиты и механизм вытеснения.

Механизм вытеснения — это часть группового планирования (Gang Scheduling). Групповое планирование — это алгоритм, позволяющий сразу нескольким задачам исполняться в одном разделе и получать доступ к определенным ресурсам поочередно, что может сильно сократить время выполнения. Механизм вытеснения в свою очередь необходим, чтобы задачи с более высоким приоритетом могли "вытеснить"те, что ниже их и исполниться в первую очередь. В зависимости от конфигурации, выполняемое до этого задание может быть приостановлено в фоне и возобновлено в будущем, отправлено заново в очередь, совсем отменено, либо перенаправлено в другой раздел. Кроме того, действие с вытесняемой задачей может зависеть от ее приоритета. Все выполняется в соответствии с параметром $PreemptMode$.

3.1.2 Ограничения вычислительных ресурсов

SLURM позволяет гибко ограничивать пользователей в доступных ресурсах, а также задачи в потребляемой мощности. Так, пользователь может лично проконтролировать лимиты своей задачи, с помощью определенных параметров, например: TimeLimit (ограничение по времени выполнения), MaxNodes (максимальное число задействованных вычислительных узлов) или MaxCPUs (максимальное число ядер, которое может использовать задача). Этих параметров намного больше, и их грамотное использование, по большей части, отслеживает система HPC TaskMaster. Более того, для защиты от неправомерного доступа на суперкомпьютере НИУ ВШЭ используется модуль cgroup, который не позволяет физически получить доступ к "чужим"ресурсам.

Вместе с личным ограничением задач, администраторы суперкомпьютерного комплекса устанавливают квоты на использование тех или иных ресурсов для определенных пользователей, исходя из их поставленных задач.

Для пользователей могут быть установлены лимиты на каждый аспект его работы, включая FairShare и QoS факторы, влияющие на приоритеты задач и определенные ограничения. Например, QoS позволяет не только ограничивать пользователей лично, но и объединять их в группы и изменять их параметры одновременно.

3.2 HPC TaskMaster

Второй важной системой на суперкомпьютере «cHARISMa» является HPC TaskMaster. Эта система разработана отделом Суперкомпьютерного моделирования НИУ ВШЭ и предоставляет мониторинг эффективности задач, выполняющихся на суперкомпьютере. Главной ее задачей является сбор и анализ данных и последующее выявление неэффективных или неисполняющихся задач. Сами данные и метрики задач TaskMaster получает из SLURM и Telegraf — серверный агент для сбора данных о состоянии узлов кластера, таких как утилизация CPU и GPU, энергопотребление и использование сети [17].

Система мониторинга определяет эффективность задач на базе трех сущностей: индикаторы, теги и выводы [23]. Индикаторы определяют загруженность компонентов, используемых в каждой задаче, и принимают значения от 0 - максимальная загруженность, до 1 - минимальная загруженность. Теги определяют свойства задачи, например, тип выполняемой задачи: jupyter notebook или salloc, и длительность задачи: более 7 дней. Выводы - это результаты, предоставляемые пользователям по итогам работы задачи. Они формируются на основе предыдущих двух сущностей и определяют, была ли эффективна задача или нет. Система может распознавать множество неэффективных задач, например, непараллельные задачи, ошибочно запущенные в параллельном режиме.

4 Анализ данных

Перед реализацией системы была поставлена задача анализа данных и поиска подхода, по которому система будет запускать задачи. Для анализа использовались данные загруженности суперкомпьютера за последние 4 года в формате:

- **cpu_load** – загрузка процессора в процентах (число).
- **gpu_load** – загрузка видеокарты в процентах (число).
- **datetime** – временная метка в формате строки.

Пример JSON-данных:

Листинг 1: Пример JSON-формата данных

```
{
    "cpu_load": 41.9,
    "gpu_load": 70.2,
    "datetime": "2025/02/21 00:15"
}
```

Все эти данные формируют один большой временной ряд, изображенный на графиках 4.1 и 4.2, с датами от января 2021 года по март 2025 года с промежутками обновления данных в 15 минут.

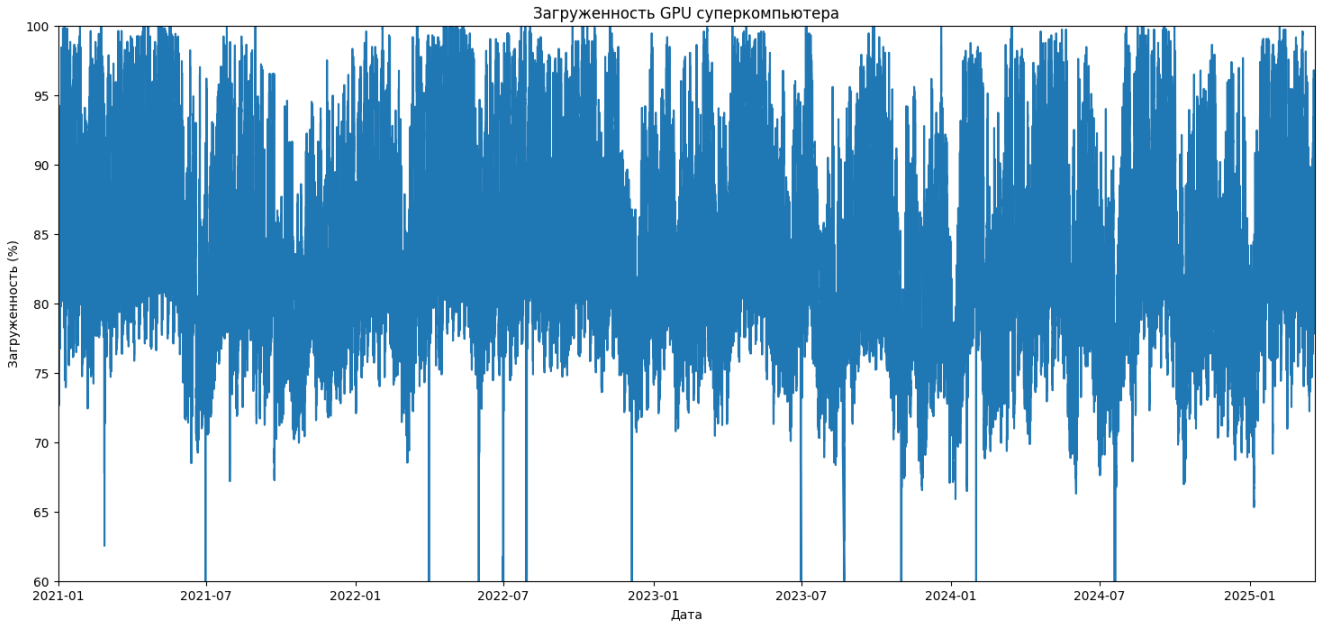


Рис. 4.1: Временной ряд загруженности GPU суперкомпьютера с 2021 по 2025

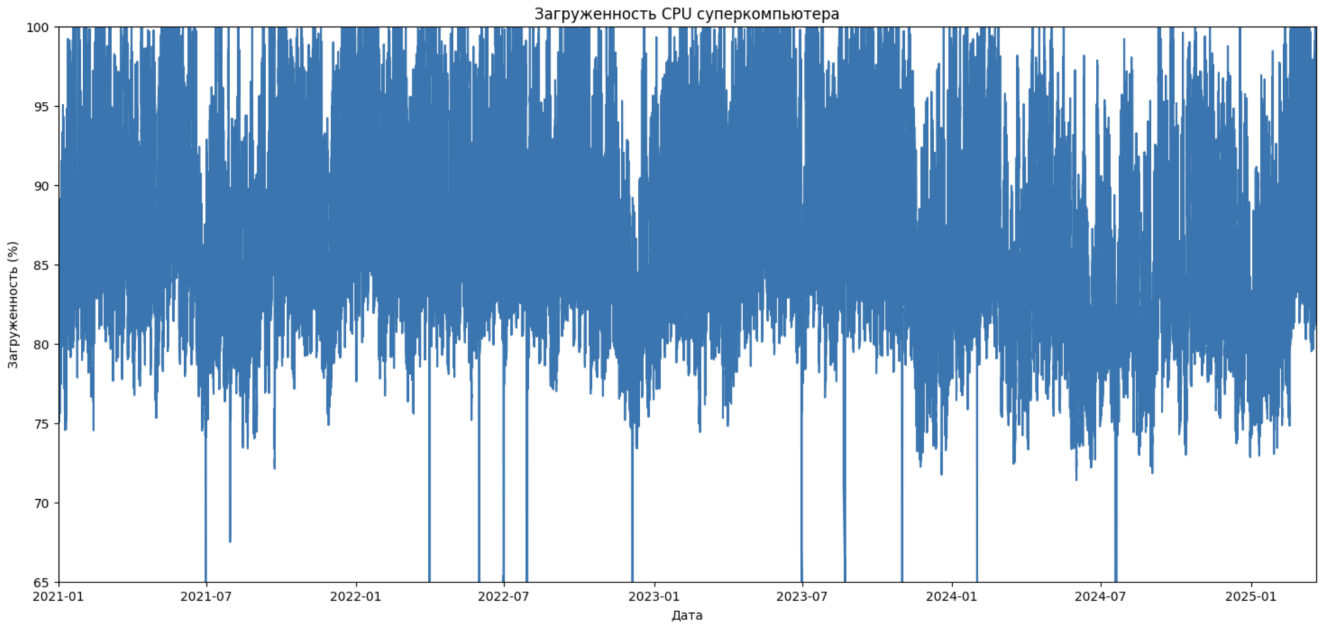


Рис. 4.2: Временной ряд загруженности CPU суперкомпьютера с 2021 по 2025

4.1 Предобработка данных

При первичном анализе временной ряд был очищен от выбросов и аномалий, связанных с проведением профилактических работ, техническим обслуживанием суперкомпьютера и процессом развития систем мониторинга HPC TaskMaster. На графике 4.3 таблица частот показывает, что таких значений достаточно много, и они могли влиять на прогнозирование временных рядов.

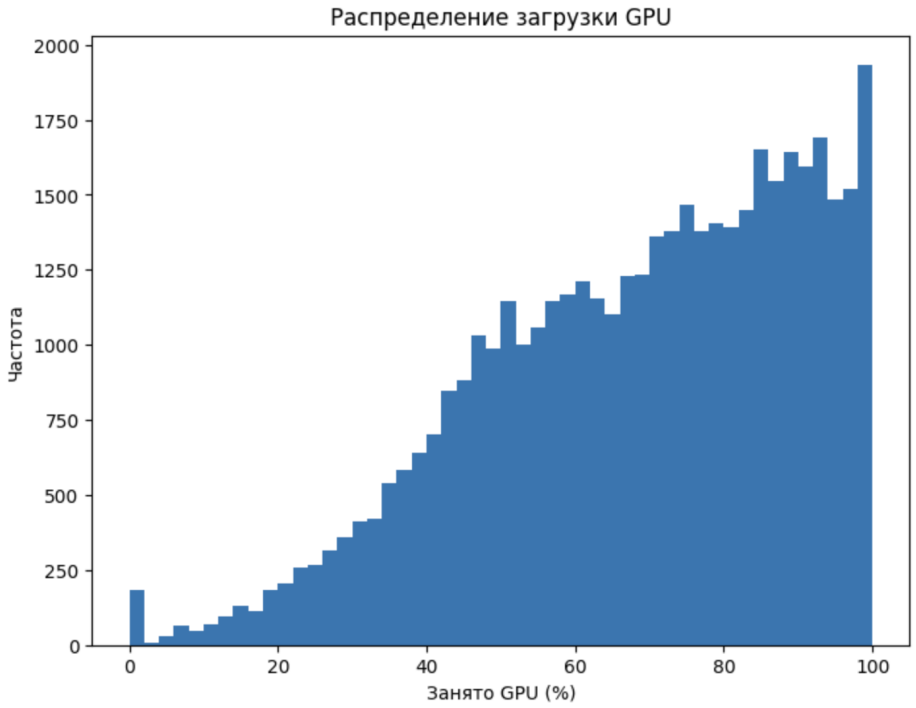
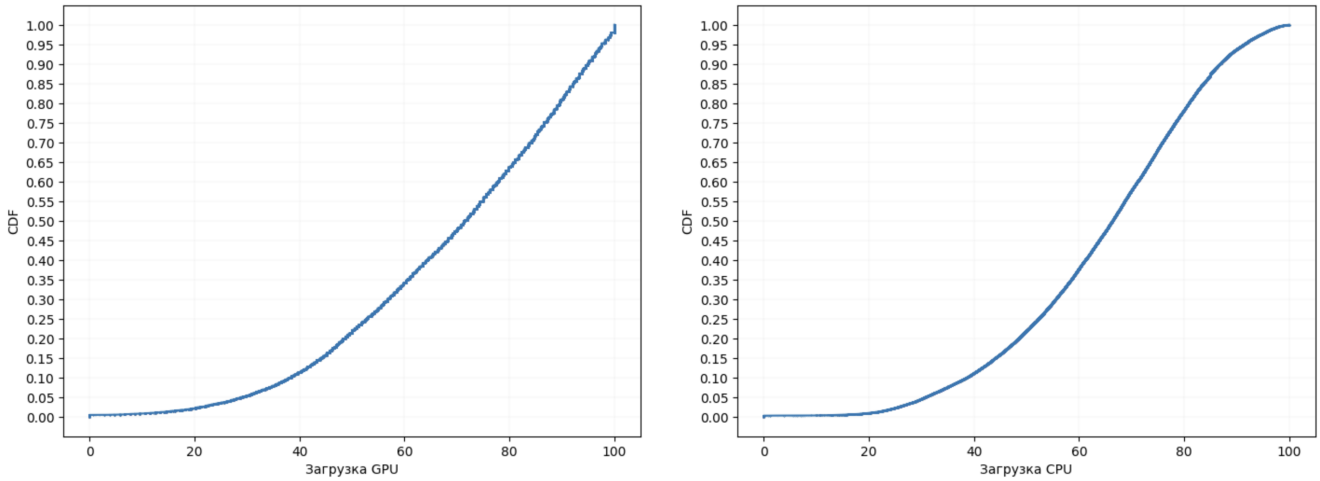


Рис. 4.3: Таблица частот загруженности GPU суперкомпьютера

Около нулевые моменты загруженности суперкомпьютера неразрывно связаны с промежутками, когда кластер перезапускался после профилактических работ. Это значит, что все околонулевые значения, ниже определенного перцентиля, можно считать аномалиями, которые можно удалить.

Для поиска необходимого перцентиля были построены таблицы функции распределения 4.4.



(a) Распределение GPU (b) Распределение CPU

Рис. 4.4: График функции распределения загруженности CPU/GPU суперкомпьютера

По графику кумулятивной функции видно, что брать обычные квартили, например, первый - нерационально. В таком случае мы отрежем существенные данные. В таком случае, за аномалию старта можно взять загрузку ниже 10. Брать выше - опасно. Как видно из данных, в некоторых моментах настоящая загрузка была 15%. В данном случае перцентили следующие:

- для CPU: 12.82,
- для GPU: 18.18.

Поскольку пропуски редко повторялись в определенной сезонности и разбросаны по всему ряду, замена их на что-то не повлияет на обобщение данных. Хоть некоторые пакеты предсказания временных рядов способны работать с пропущенными значениями, их замена все равно является хорошей практикой. В данном случае были заменены удаленные данные с помощью линейной интерполяции [3].

Слишком большие значения не рассматривались как аномалии. На это есть несколько причин. Первая - слишком сложно отличить аномальную высокую загруженность от обычной, ввиду того, что данные за 2021-2023 год слишком завышены из-за того, что неэффек-

тивные задачи в этих годах еще не отключались автоматически. Вторая причина - это не принесет никакой практической пользы. Во многих моментах перепрогнозирование может оказаться даже более полезным, потому что лучше думать, что ресурсов будет меньше, чем занять то, что должно было использоваться.

4.2 Анализ данных

Поскольку поставленной задачей является "запуск задач в выходные то анализировать нужно преимущественно именно этот тип дней. Это сможет подтвердить некоторые гипотезы и поможет в формировании параметров для будущей модели.

Для начала имелось две гипотезы:

1. Загруженность в выходные ниже, чем в будние.
2. Загруженность в выходные зависит от будних и, в частности, от пятницы.

4.2.1 Поиск признаков

Первая гипотеза подтвердила бы смысл существования разрабатываемой системы. Вторая нужна для поиска новых признаков, необходимых для обучения модели.

Для этого были построены и анализированы два scatter [21] графика 4.5, отображающие средние значения за день в пятницу и субботу, а также в будние и выходные. Уже исходя из них можно сделать вывод о наличии корреляции в обоих случаях.

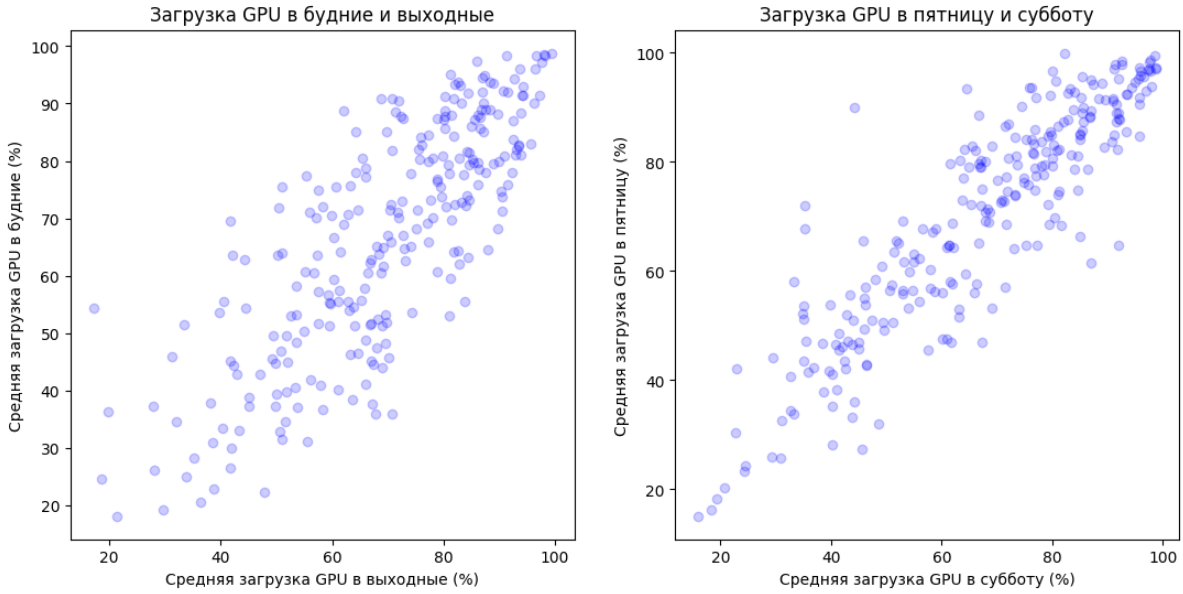


Рис. 4.5: Средняя загруженность: "будние/выходные"и "пятница/суббота"

После этого были высчитаны обычное среднее и медиана, а их процентная разница представлена в таблице 4.1:

Таблица 4.1: Процентная разница загрузки CPU и GPU

Показатель	**Пятница vs суббота**	**Будни vs выходные**
CPU загрузка		
Среднее	6.44%	6.87%
Медиана	9.37%	5.84%
GPU загрузка		
Среднее	7.14%	5.39%
Медиана	12.28%	8.46%

Данная таблица и графики уже подтвердили первую гипотезу. В среднем, загрузка в выходные ниже на 5-10 процентов уже с учетом того, что администраторы суперкомпьютерного комплекса изменяют приоритеты вручную. Из этого можно сделать вывод, что разработка системы оправдана.

Теперь измерим коэффициент корреляции Пирсона для подтверждения второй гипотезы. Результаты представлены в таблице 4.2:

Таблица 4.2: Корреляционный анализ загрузки CPU и GPU

Гипотеза	**Коэффициент корреляции Пирсона**	**p-value**
Пятница/суббота CPU	0.8438	10^{-5}
Пятница/суббота GPU	0.8528	10^{-8}
Будни/выходные CPU	0.6927	10^{-5}
Будни/выходные GPU	0.7389	10^{-8}

Исходя из этих данных можно косвенно, но подтвердить, вторую гипотезу о наличии явной зависимости между будними днями и выходными. Более того, можно сделать вывод, что зависимость от пятницы намного выше, чем от всей недели. Это, скорее всего, обусловлено тем, что пятница - переходный день, и многие оставляют свои задачи до следующей недели.

4.2.2 Поиск горизонта прогнозирования

Данные результаты помогут в будущем при построении модели предсказания временных рядов.

Для дальнейших рассуждений необходимо определить, нужно ли предсказывать два дня или только один, то есть хватит только субботы. Определить это можно достаточно тривиально - посмотреть на разницу средних за субботу и воскресенье. Если в целом загруженность в воскресенье будет меньше или равна субботе, то можно не рассматривать воскресенье для прогнозирования. Для этого был построен усредненный график разницы между субботой и воскресеньем в процентах, изображенный на графике 4.6.

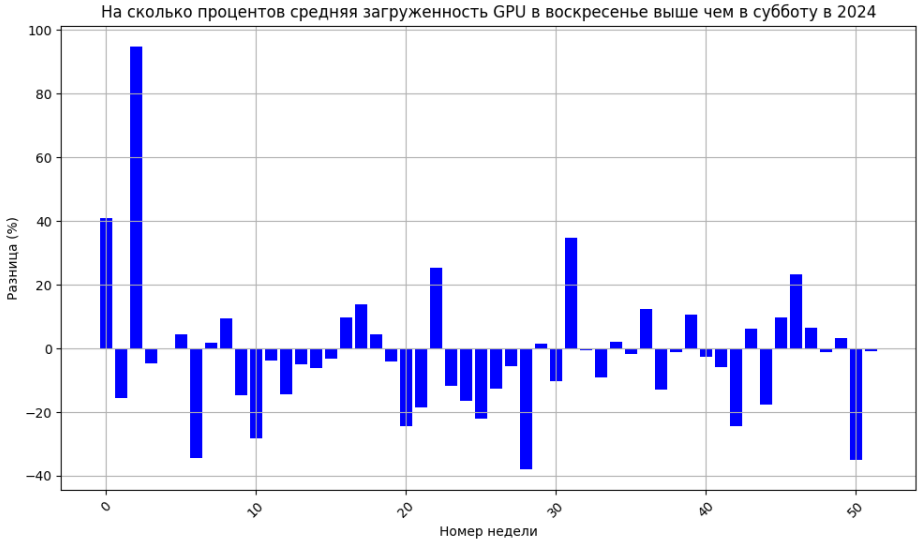


Рис. 4.6: График процентной разницы загруженности GPU между субботой и воскресеньем

Недель, когда загруженность в воскресенье выше, чем в субботу, больше 20, что равно почти половине года. Следовательно, можно сделать вывод, что прогнозирование воскресенья такая же важная задача, как и прогнозирование субботы.

4.2.3 Поиск сезонностей

Разница загруженности между субботой и пятницей это лишь часть сезонностей, которые отчетливо видны на общем графиках загруженности 4.1 и 4.2. Поэтому анализ был продолжен, чтобы выделить новые характеристики ряда.

Для поиска сезонностей ряд сначала был детрендирован, то есть для каждого типа сезонности из ряда вычтен соответствующий тренд. Потом все данные были сгруппированны по средним значениям, после чего были получены следующие графики сезонностей:

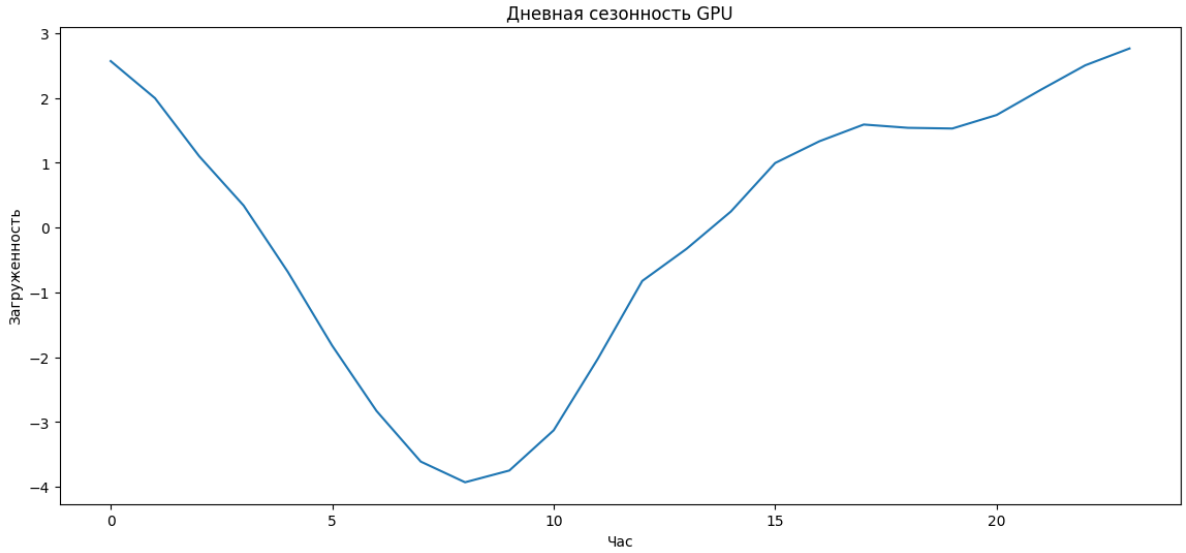


Рис. 4.7: График дневной сезонности загрузки суперкомпьютера

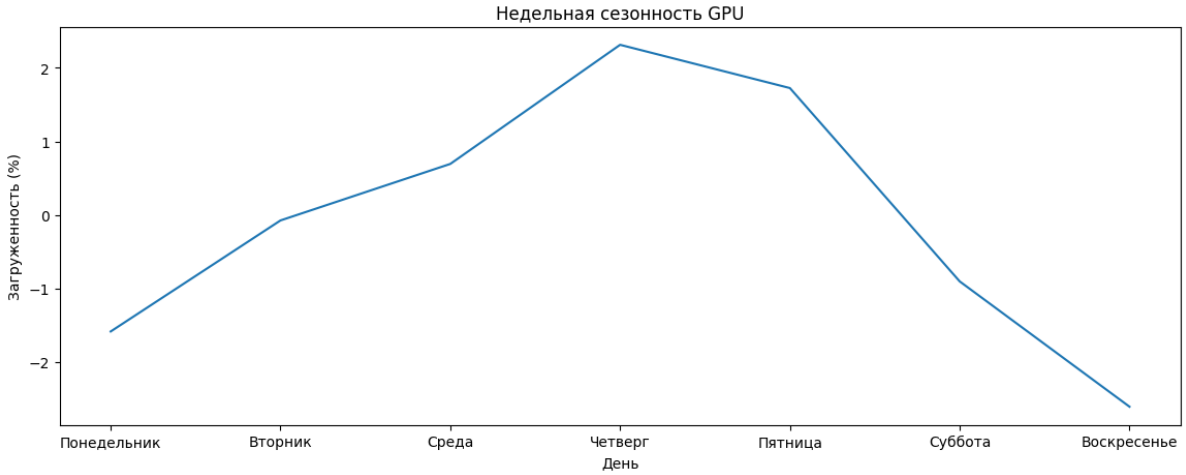


Рис. 4.8: График недельной сезонности загрузки суперкомпьютера

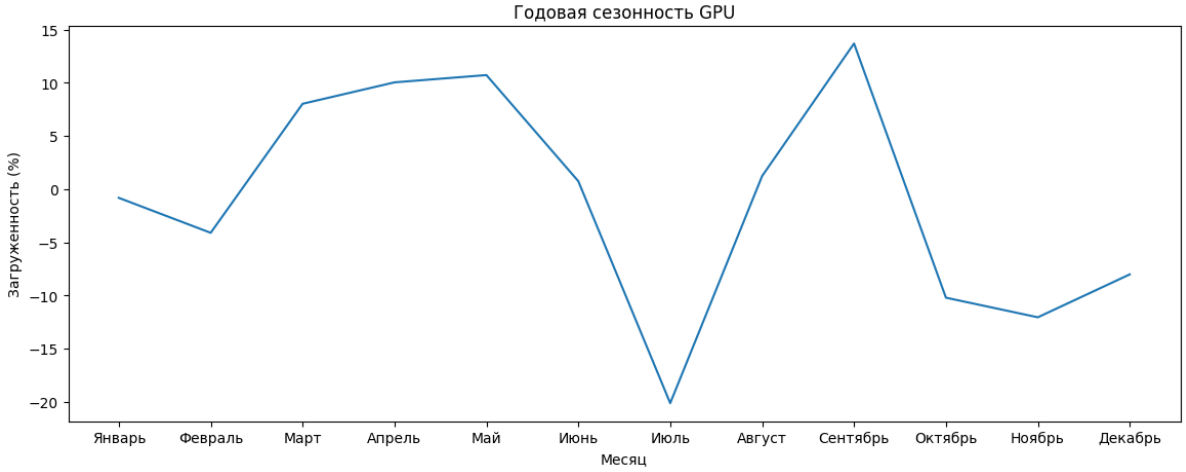


Рис. 4.9: График годовой сезонности загрузки суперкомпьютера

Благодаря графикам можно сделать следующие выводы. Так, на графике 4.7 видно,

что ночью загруженность становится ниже, так как хоть очередь не пустая, но новых задач не добавляется. То же самое видно и на графике 4.8 - пользователи оставляют работу над проектами на будние.

На графике 4.9 месячной сезонности видны определенные шаблоны в поведении. Перед крупными дедлайнами, например, перед курсовыми, ВКР и сдачей работ в мае, загруженность повышается, а зимой, особенно в период праздников, резко падает.

4.2.4 Итоги анализа

Суммируя все вышесказанное, в этой главе были определены три важных свойства, которые помогут на следующих этапах разработки системы:

1. Средняя загруженность в выходные ниже, чем в будние дни.
2. Выходные зависят от предшествующей им недели.
3. Необходимо предсказание на оба выходных дня: субботу и воскресенье.
4. Существует сразу несколько типов сезонностей: дневная, недельная и годовая.

На этом основной анализ окончен, дальнейший анализ будет проводиться уже при создании моделей предсказания.

5 Проектирование

После анализа данных, с пониманием того, в каких условиях и с какими дополнительными параметрами система будет работать, можно приступить к выбору инструментов и проектированию архитектуры.

5.1 Запуск задач

Изначально планировалось с помощью подсистемы динамически изменять квоты пользователей, чтобы дать им больше свободных ресурсов в выходные. Но данная идея не была одобрена руководителем отдела из-за того, что изменение квот может быть опасным или, наоборот, неэффективным. Например, повысив квоты, мы можем запустить длительную вычислительную задачу, которая будет выполняться не только выходные, но и всю следующую неделю, при этом заняв большое количество ресурсов, что является плохим решением. Неэффективность же появляется, когда в очередь попадает задача с условными восьмью требуемыми GPU, когда квоты повышены максимум до 7, и, хотя кластер способен запустить

такую задачу, она все равно останется в очереди. Поэтому было принято решение работать не с пользователями, а с отдельными задачами, ожидающими запуска.

Следующим этапом стало рассмотрение изменения приоритетов. Так, для задачи, у которой указан короткий срок выполнения и у кластера свободны ресурсы для ее выполнения, задаче мог бы быть повышен приоритет. Но изменение приоритета также не является панацеей, так как запуском задач будет заниматься планировщик задач SLURM суперкомпьютера, который, в том числе, ориентирует на квоты пользователей и множество других критериев. Разрабатываемая система в этом случае не будет располагать достаточным пониманием того, когда именно задача с повышенным приоритетом будет запущена.

Финальным предложенным подходом стал запуск задач в обход алгоритма планировщика. Если внутри подсистемы реализовать свой аналог модуля, определяющего загруженность и доступные ресурсы, то можно определять без SLURM, какие задачи могут быть запущены. Так же было принято решение о том, что разрабатываемая подсистема должна отдавать сигнал SLURM через API HPC TaskMaster для запуска задач в обход квот и приоритетов.

5.2 Определение загруженности

Перед написанием подсистемы предполагалось, что определение будущей загруженности будет происходить на основе математических и статистических эвристик, но в ходе поиска возможных решений этой задачи стало понятно, что данный подход не является достаточно подходящим. Дело в том, что загруженность постоянно меняется. Изменяются сезонности, дни недели, дедлайны, и для всех недель писать одну математическую модель невозможно. Исходя из этого, искались более гибкие способы анализировать данные, что, в конечном итоге, привело к выбору методов машинного обучения.

5.3 Предсказание временных рядов

Главный компонент всей системы - планировщик, который определяет доступные задачи и запускает их. Но чтобы он работал, система должна понимать, как именно можно загрузить систему, чтобы сильно не изменить логику работы SLURM.

Для этого в архитектуре был описан еще один модуль - модуль предсказания временных рядов. Он был предложен, потому что анализ всей очереди планировщика будет бесполезным. Более того, по ходу запуска задач могут добавляться другие дополнительные задачи, запущенные уже в выходные.

Из анализа данных, проведенного в предыдущей главе, можно выделить несколько ключевых моментов:

- В данных присутствует несколько сезонностей.
- Тренд слабовыражен.

Также, немаловажным фактором является короткий горизонт прогнозирования. Два дня, необходимых для предсказаний, слишком малый временной интервал для некоторых моделей, что будет показано ниже. Кроме того, немаловажным критерием выбора модели была ее интерпретируемость, чтобы можно было примерно понять, почему модель приняла то или иное решение.

5.3.1 Описание библиотек

SARIMAX

Один из простейших вариантов для предсказаний временных рядов является ARIMA. Это авторегрессионная модель, которая для предсказаний использует предыдущие значения ряда с определенными, подобранными коэффициентами.

SARIMAX является продолжением идеи ARIMA, добавив сезонную компоненту, а также дополнительные регрессоры, например, экономические.

TBATS

TBATS можно считать продолжением ARIMA и улучшенным аналогом SARIMA. В TBATS уже возможен учет нескольких сезонностей, есть определение тренда и преобразование Бокса-Кокса.

В общем виде модель выглядит так:

$$y_t^{(\lambda)} = \ell_{t-1} + \phi\, b_{t-1} + \sum_{i=1}^{K} s_{t-m_i}^{(i)} + d_t,$$

где:

- $y_t^{(\lambda)}$ — значение ряда после Box–Cox-преобразования,
- ℓ_t — уровень (level),
- b_t — тренд (trend), демпфируемый через коэффициент $\phi \in [0, 1]$,
- $s_{t-m_i}^{(i)}$ — сезонная компонента i-го периода длины m_i, представляемая набором тригонометрических функций,

- d_t — ARMA-ошибка (остаток), моделируемая как ARMA(p, q).

Prophet

Самой популярной библиотекой для предсказаний является Prophet. Это open-source проект, который для предсказаний использует довольно простую в интерпретации формулу:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t,$$

где:

- $g(t)$ — тренд,
- $s(t)$ — сезонность,
- $h(t)$ — эффект праздников,
- $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ — гауссовский шум.

Prophet не использует авторегрессию, а все прогнозы считает лишь по этим трем предподсчитанным компонентам, где сезонность - это ряд Фурье с определенной точностью.

Из этого вытекает его проблема - он плохо подстраивается под новые данные. Если в последние дни были аномальные данные, то prophet не подстроится под них, а выдаст прогноз, основанный на уже посчитанных на других элементах.

NeuralProphet

Решает проблему отсутствия авторегрессии другая open-source библиотека, которая в основу берет тот же подход, что и Prophet - в виде разделения модели на три главных компоненты, добавляя новый этот элемент. Кроме этого, автогрегрессия построена не на скользящем среднем, как в TBATS или SARIMAX, а на основе нейросети [16].

Итоговая формула модели NeuralProphet выглядит так:

$$\hat{y}_t = g(t) + s(t) + h(t) + \hat{r}_t.$$

Авторегрессия в этой библиотеке построена на базе AR-Net. После того, как был высчитан тренд, сезонность и праздники, получается остаток r:

$$r_t = y_t - (g(t) + s(t) + h(t)).$$

В обычной модели он воспринимается как шум, но AR-net использует его для авторегрессии, предсказывая его будущее значение по векторам его прошлых значений. Это позволяет добиться наибольшей гибкости в модели [15].

5.4 Сравнение моделей предсказания

5.4.1 Условия сравнения

Для обучения использовался датасет загруженности суперкомпьютера за последние три года (2022-2024).

Для модели TBATS была определена дневная, недельная и годовая сезонность, также было определено использование метода Бокса-Кокса для нормализации данных.

Для модели SARIMAX была определена годовая сезонность и определен лаг в три дня.

Интерфейс Prophet и NeuralProphet крайне похож. Для них были определены праздники в регионе «RU», количество лагов в 288 измерений (3 дня), а также разрешены недельная, годовая и дневная сезонности.

В качестве ошибки использовалось MSSE (Mean Segmented Squared Error) [12]:

$$\mathrm{MSSE} = \frac{1}{K}\sum_{k=1}^{K}\left(\left(\bar{y}_k - \bar{\hat{y}}_k\right)^2\right), \quad \text{где} \quad \bar{y}_k = \frac{1}{n_k}\sum_{t\in S_k} y_t, \quad \bar{\hat{y}}_k = \frac{1}{n_k}\sum_{t\in S_k} \hat{y}_t.$$

Преимущество этой ошибки в том, что она позволяет оценивать вклад каждого предсказанного сегмента, вместо нахождения ошибки на каждом предсказании или на их среднем.

Модели будут обучаться на центральном процессоре, так как планируется, что TaskShift будет выполняться на главном узле кластера, где недоступен графический ускоритель. В тестировании использовался процессор Apple silicon m4.

5.4.2 Результаты сравнения

Проведя эксперименты по заданной модели, были получены следующие результаты ошибки, отображенные в таблице 5.1.

Таблица 5.1: Сравнение моделей по метрике MSSE и времени обучения

Модель	**MSSE**	**Время обучения**
NeuralProphet	7.37	4 мин. 24 сек.
Prophet	579.81	3 мин. 17 сек.
SARIMAX	16.71	41 мин. 9 сек.
TBATS	28.77	1 час. 38 мин.

Результаты показывают преимущество NeuralProphet не только в качестве оценки, но еще в 10-25 раз меньшее время обучения, чем у SARIMAX и TBATS. Prophet, хоть и показал самое быстрое время, но из-за того, что он не учитывает последние значения, показал огромную ошибку. Это связано с тем, как уже упоминалось, что Prophet не имеет авторегрессии, а, следовательно, не может подстраиваться под аномалии. Можно рассмотреть пример: 1 и 2 января явно аномальные дни из-за низкой загрузки, но модель, не зная, что в последнее время нагрузка только снижалась, сделает предсказание выше, чем должно быть.

SARIMAX и TBATS хотя и показали хороший результат в оценивании, обучались слишком долго, что не делает возможным их частое переобучение, например, в стыке между субботой и воскресеньем, или в пятницу вечером.

Исходя из всего вышеперечисленного, лучшим вариантом является NeuralProphet. Эта модель сочетает в себе достаточную точность, быстрое время обучения и хорошую интерпретируемость. Более того, она проста в использовании, и не требует серьезных знаний в глубинном обучении, как этого хотят нейросетевые модели или модели на основе деревьев.

5.5 Архитектура системы

Архитектура проекта изначально должна была включать в себя один монолитный модуль, который бы запрашивал данные из HPC TaskMaster, а далее передавал бы запрос о запуске напрямую в SLURM, но этот подход был пересмотрен.

В итоге, архитектура сервиса включает в себя четыре модуля:

- модуль конфигураций,
- модуль агрегирования данных,
- модуль предсказания временных рядов,
- модуль анализа загруженности и запуска задач.

Данная структура позволяет быстро менять и дополнять функционал при необходимости, а также разграничивает зоны ответственности.

Модуль агрегирования данных необходим для того, чтобы обучать модель предсказания на новых данных. Он включает в себя работу с базой данных временных рядов и передачу данных в модуль предсказания. Модуль должен позволять передавать и сохранять данные.

Модуль предсказания временных рядов нужен для того, чтобы предсказывать, какую нагрузку стоит ждать на выходных. Предсказание необходимо для определения порога, до которого кластер будет загружен, а из этого можно понять, какие задачи влезут в доступные ресурсы.

Модуль анализа и запуска получает текущую информацию о кластере из системы HPC TaskMaster. В информацию входят занятые и свободные ресурсы, общая загруженность и задачи, находящиеся в очереди. После этого модуль определяет короткие задачи, которые завершатся до понедельника, определяет, доступны ли ресурсы, необходимые для выполнения, и после этого высчитывает нагрузку, которая будет после запуска этой задачи. Если все три пункта проходят, то модуль отдает приказ о запуске задачи в обход очереди SLURM.

Графическая схема взаимодействия сервиса с внешними системами и внутренними модулями в нотации C4 [19] изображена на схеме 5.1:

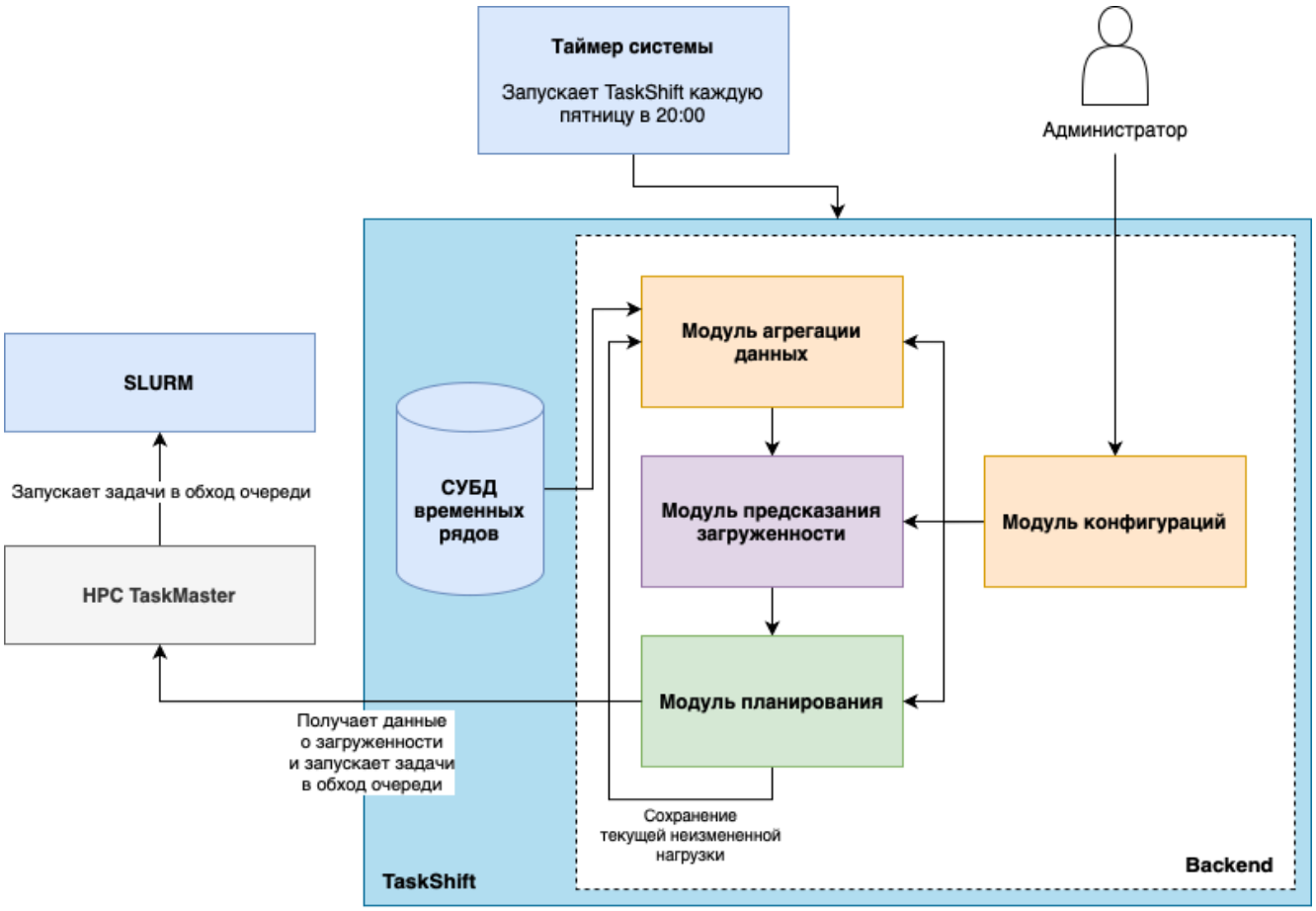


Рис. 5.1: Архитектура TaskShift в нотации C4

Архитектура в нотации C4 описывает внешний контекст, такой как администратор суперкомпьютерного комплекса или HPC TaskMaster, и описывает внутренние контейнеры: backend часть, состоящую из 4 уже ранее описанных компонентов, и разворачиваемую отдельно СУБД временных рядов.

Полная и релевантная версия архитектуры, включающая в себя описание стека технологий и написанная с помощью инструмента likeC4 [10], находится в репозитории TaskShift по пути *docs/scheme.c4*.

6 Реализация

6.1 Язык программирования

Выбор основного языка для разработки - непростая задача. Изначально планировалось писать сервис на Go, но из-за того, что код должен быть понятен для наибольшего количества разработчиков, был выбран язык Python. Более того, Python был выбран в том числе из-за поддержки пакетов предсказаний. Он позволяет писать легкий и понятный мо-

дульный код, где быстродействие не является приоритетной задачей.

6.2 Хранение данных

Поскольку система должна агрегировать большой массив данных временных рядов, вместе с этим сохраняя текущие значения, нужно было выбрать подход, который одновременно простой и функциональный.

Для не слишком частой записи в сервисе TaskShift, которая, по предположению, должна была записывать данные только в пятницу, мог бы подойти подход с использованием JSON-файлов. Но сервис, используя JSON-хранилище, теряет гибкость и возможности для масштабирования. Если, например, предположить, что сервис мог бы сохранять не измененные собой данные о загруженности каждые 15 минут, то это была бы большая нагрузка на подобный метод хранения данных, и это стало бы узким местом. Поэтому преимущественно рассматривались полноценные СУБД.

Выбор СУБД для временных рядов весьма ограничен: на рынке не так много открытых профильных систем, а обычные SQL и NoSQL СУБД плохо подходят из-за неоптимального хранения таких данных. Из доступных систем для временных рядов есть QuestDB, InfluxDB, Prometheus и TimescaleDB, но в итоге была выбрана InfluxDB v2 - самая популярная СУБД для хранения временных рядов с открытым исходным кодом. Кроме этого, она уже используется в HPC TaskMaster, что упростит интеграцию и поддержку.

6.3 Реализация модулей

6.3.1 Модуль конфигураций

Статическая конфигурация

Одно из главных условий разработки сервиса - простота его дальнейшего использования, поэтому было принято решение написать гибкий модуль конфигурации, который бы настраивал не только ключи доступа, но еще и параметры модели и планировщика.

Конфигурирование ключей доступа к базе данных и API HPC TaskMaster происходит просто - с помощью переменных окружения, записанных в *.env* файл. При каждом запуске сервиса они выгружаются из файла и заносятся как переменные окружения с помощью библиотеки *dotenv* [5]. Это делает их доступными только во время рантайма программы, что сильно повышает безопасность сервиса.

Конфигурация планировщика и модели предсказания происходит с помощью *.yml* файлов. Для этого была написана собственная функция, читающая и обрабатывающая *.yml*

формат. Эта функция используется при создании отдельных классов конфигурации, класс конфигурации кластера написан в листинге 2. Такой подход позволяет инкапсулировать и упростить доступ к переменным конфигурации.

```
class ClusterConfig(ConfigurationBase):
    def __init__(self, file=cluster_config_file):
        super().__init__(file=file, service_name="cluster configuration file")

    def get_nodes_info(self):
        return self.config["nodes"]

    def get_cluster_info(self):
        return self.config["cluster"]

    def get_devices_count(self):
        cpu_count, gpu_count = 0, 0

        for node in self.config["nodes"]:
            node_count = node["count"]
            cpu_count += (
                node["cpu"]["sockets"] * node["cpu"]["cores_per_cpu"] * node_count
            )

            if node["gpu"] is not None:
                gpu_count += node["gpu"]["count"] * node_count

        return cpu_count, gpu_count
```

Листинг 2: Базовый класс конфигурации

Динамическая конфигурация

Поскольку сервис работает с задачами и планировщиком SLURM, его внутренний модуль планирования должен иметь возможность конфигурироваться и управляться администратором во время его работы. Для этого был написан REST API интерфейс с помощью библиотеки *FastAPI* [6].

Он позволяет во время работы изменять политику запуска задач, ограничивая доступную нагрузку, или задать определенные параметры задач в очереди. Кроме этого, предусмотрена возможность отключать или запускать модуль планирования. Пример REST API запроса продемонстрирован в листинге 3:

```
PATCH /scheduler/config/setmaxload/ HTTP/1.1
Host: localhost
Content-Type: application/json
Accept: application/json
{
  "max_cpu_load": 85.5,
  "max_gpu_load": 90.0
}
```

Листинг 3: Пример REST API запроса

6.3.2 Модуль агрегации данных

Модуль агрегации данных тесно связан с СУБД для временных рядов, поэтому InfluxDB была развернута перед началом разработки модуля. Для этого используется официальный докер-образ для быстрого запуска. Для взаимодействия использовалась библиотека influxdb-client-python [7]. Она позволяет быстро интегрировать почти любую логику взаимодействия с СУБД. Таким образом, в модуле есть две функции: одна отвечает за сохранение данных в нужную БД, а вторая отвечает за чтение данных загруженности CPU и GPU, объединяя два измерения в одну таблицу с помощью *pivot*:

```
    query = """from(bucket: "cluster_load")
    |> range(start: 1970-01-01T00:00:00Z)
    |> filter(fn: (r) => r._measurement == "gpu_load" or r._measurement
       == "cpu_load")
    |> pivot(
        rowKey: ["_time"],
        columnKey: ["_measurement"],
        valueColumn: "_value"
    )
    """
```

Листинг 4: Запрос в InfluxDB с pivot

6.3.3 Модуль предсказания временных рядов

В главе 5.4.2 было определено, что самая подходящая библиотека для предсказаний временных рядов - **NeuralProphet**. Еще одним плюсом для архитектуры TaskShift является то, что библиотека написана на Python, поэтому модуль не придется выносить отдельным сервисом на другом языке, а можно интегрировать во весь остальной код.

Модуль предсказаний получает сырые данные из СУБД с помощью модуля агрегации данных, после чего конвертирует их в пригодный для NeuralProphet формат. Для него колонка времени должна быть названа *ds*, а колонка данных *y*. Кроме этого, не должно быть

никаких пропусков и дубликатов, поэтому все данные еще раз форматируются под формат измерений раз в 15 минут, а все пропуски заменяются с помощью интерполяции.

Для удобного взаимодействия с другими частями сервиса, модель предсказаний была вынесена в отдельный класс, который обучает ее на всем датасете в момент создания, а также предоставляет интерфейс для получения данных о предсказаниях. Кроме этого, реализованы отдельные функции, предоставляющие быстрый доступ к нужным в планировщике данным, таким как среднее за определенный день или среднее за все время. Примеры этих функций показаны в листинге 5:

```
...
def get_avg_window(start=0, end=None, df_list = []):
    if not df_list:
        df_list = list(model.get_forecasts())

    avg_list = []

    for df in df_list:
        if end is None:
            end = len(df)

        avg_list.append(np.mean(df[start:end]))

    return avg_list

def get_saturday_avg():
    return get_avg_window(end=FORECASTS_IN_ONE_DAY)
```

Листинг 5: Интерфейс взаимодействия с ForecastModel

6.3.4 Модуль планирования

Модуль планирования, как и упоминалось ранее, должен определять, какие задачи из ожидающих запуска вместятся в свободные ресурсы и запускать их.

Сами задачи в двух состояниях: «ожидающие запуска» и «в работе», сервис получает из TaskMaster через REST API со специальным токеном. После этого задачи формируют очередь. Потом, каждые 5 минут планировщик выбирает следующую возможную задачу и запускает ее. Поскольку делать ACK трудозатратно, был предложен другой подход. Вместо синхронного подтверждения запуска, TaskShift сверяет список задач в очереди при каждом обновлении. Если запущенная ранее задача не запустилась, задача отправляется в конец очереди, чтобы в конечном итоге попробовать еще раз ее запустить. Но, поскольку с каждым новым запросом список задач изменяется, их порядок также в конечном итоге изменяется. Поэтому было решено использовать очередь, которая бы сохраняла порядок уже существу-

ющих элементов между изменениями.

Вместе с тем планировщик должен проверять наличие уже запущенной задачи в очереди. Если бы существование элемента проверялось только в *deque*, сложность операции составляла $\mathcal{O}(n)$. К тому же из-за этого сложность функции *rebuild*, которая бы перестраивала очередь на основании уже имеющихся задач, была бы $\mathcal{O}(n^2)$. В результате этого вместе с очередью используется *set*. Совместив все вышесказанное, для сервиса была написана собственная реализация очереди, показанная в листинге 6:

Итоговая сложность функций класса очереди:

Метод	**Временная сложность**
`__init__`	$\mathcal{O}(n)$
`add_elements`	$\mathcal{O}(m)$
`rebuild`	$\mathcal{O}(n+m)$
`put`	$\mathcal{O}(1)$
`pop`	$\mathcal{O}(1)$
`empty`	$\mathcal{O}(1)$
`__contains__`	$\mathcal{O}(1)$

Таблица 6.1: Асимптотическая сложность методов класса `UniqueQueue`

```
class UniqueQueue:
    def __init__(self, initial=None):
        self._dq = deque(initial or [])
        self._set = set(self._dq)

    def rebuild(self, elements):
        elements_set = set(elements)
        self._dq = deque(x for x in self._dq if x in elements)
        self._set = set(self._dq)

        for item in elements_set - self._set:
            self._dq.append(item)
            self._set.add(item)

    def put(self, item):
        if item not in self._set:
            self._dq.appendleft(item)
            self._set.add(item)

    def pop(self):
        item = self._dq.pop()
        self._set.remove(item)
        return item

    ...
```

Листинг 6: Интерфейс взаимодействия с моделью предсказаний

Графическое описание работы очереди с сохранением порядка между вызовами показано на схеме 6.1:

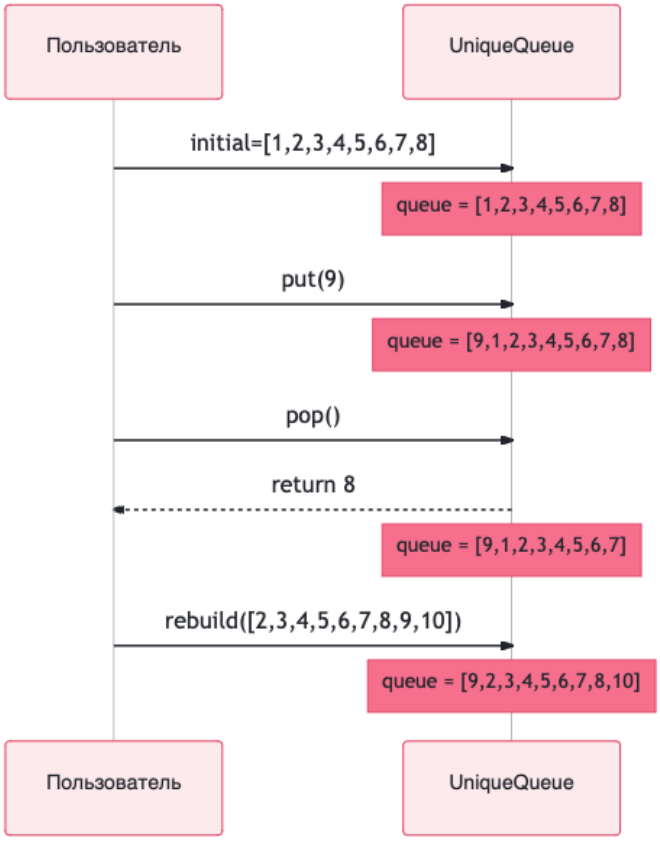


Рис. 6.1: Жизненный цикл UniqueQueue

Для запуска планирования в определенное время модуль использует библиотеку APScheduler [2]. Она позволяет написать подобие cron таймера и запускать задачи в определенные дни с определенными интервалами. Благодаря этому логика модуля выглядит так:

1. Планировщик получает прогноз на выходные в пятницу в 23:50.
2. Планировщик каждые 5 минут в следующие два дня запускает функцию проверки задач *task_scheduler*, которая получает все задачи в очереди и, следуя политике, их обрабатывает.

Task_scheduler работает следующим образом: функция получает прогноз и, исходя из него, определяет свободные ресурсы на следующий день. Затем получает лист ожидания и с помощью него перестраивает очередь, и по построенной очереди пытается запустить каждую вмещающуюся в свободные ресурсы задачу, если ее лимит времени вмещается в оставшееся время до следующего понедельника 0 часов 0 минут. Если в следующий запуск через 5 минут планировщик снова обнаружит уже запущенную им задачу в листе ожидания, он перекинет эту задачу в конец очереди, чтобы дойти до нее последней. Таким образом *task_scheduler* не пытается запустить невозможные задачи, а также не пытается по несколько раз запускать задачи, в запуске которых отказал SLURM.

6.4 Логирование

Поскольку планировщик cron находится прямо внутри системы, система должна быть всегда активной. Но во время работы могут происходить инциденты, и администратору нужно понимать, в каком состоянии сейчас находится система.

Для этого во всей подсистеме используются логеры, написанные с помощью библиотеки *loguru* [11]. Это позволило относительно просто внедрить по всей системе сразу несколько уровней детализации событий. Используется сразу несколько уровней логирования: от trace до critical. Самый низкий уровень, например, используется при анализе работы очереди с сохранением порядка.

6.5 Тестирование

Так как TaskShift в конечном итоге будет введена в промышленную эксплуатацию, она должна всегда гарантировать стабильную работу. Для этого должно быть подробное тестирование каждого написанного модуля и отдельно всей системы. Поэтому с помощью библиотеки pytest [13] для тестирования были написаны следующие типы тестов:

1. unit-тесты,
2. интеграционные тесты,
3. функциональные тесты.

Unit-тесты были написаны для всех модулей и находятся в папке *tests units*. Интеграционные тесты проверяют работу API TaskMaster и находятся в папке *tests integrations*. Функциональные тесты оценивают эффективность системы на уже заранее сгенерированном наборе задач. В итоге общее покрытие, посчитанное с помощью *pytest-cov* составило 95%.

Файл	**Покрытие**
source/configs/config.py	95%
source/data_agregator/client.py	92%
source/data_agregator/core.py	95%
source/forecaster/core.py	100%
source/forecaster/model.py	98%
source/scheduler/core.py	95%
source/scheduler/integration.py	90%
Итого	**95%**

Таблица 6.2: Покрытие тестами исходного кода проекта

6.6 Развертывание приложения

Поскольку система должна представлять собой готовый к использованию продукт, важным требованием является простота ее развертывания и запуска. Поэтому система была упакована в Docker-контейнер [4]. Отдельным контейнером также запускается InfluxDB, что минимизирует затраты на настройку и запуск СУБД. Для одновременного запуска обоих контейнеров был подготовлен файл конфигурации docker-compose.

7 Результаты

Все поставленные задачи были выполнены, включая: анализ предметной области, анализ данных, проектирование и программную реализацию. Разработанная подсистема в реальном времени способна анализировать загруженность кластера, делать прогноз на основе нейросетевой модели и, в периоды сезонной пониженной загрузки суперкомпьютера в выходные, запускать без очереди задачи, которые помещаются в свободные ресурсы суперкомпьютера и успевают выполниться до окончания выходных. По итогу работы была опубликована научная статья и сделан доклад на Всероссийской конференции "Параллельные вычислительные технологии 2025 проходившей 8-10 апреля 2025г. в Москве. Апробация на конференции позволила найти новые идеи и развить подсистему. Например, в системе появился отдельный API для внешнего управления и сохранение неизмененных данных напрямую из планировщика TaskShift.

Работу системы можно кратко описать следующим образом. Каждую пятницу в 23:50 cron-таймер запускает функцию прогнозирования. Она запрашивает все данные о загружен-

ности суперкомпьютера за прошедшую неделю и на основе этих данных начинает обучение модели предсказания временных рядов NeuralProphet. После этого, с 0:00 в субботу, cron-таймер модуля планирования циклично запускает функцию планировщика. Она запрашивает все запущенные и ожидающие запуска в данный момент вычислительные задачи, сортирует задачи в очереди по лимиту времени и находит те, которые можно запустить. Критерии запуска следующие:

- Задача помещается в свободные пространство загрузки, определяемое как $100 - avg()$.
- Лимит времени задачи успеет закончиться до понедельника 12 часов ночи.
- В данный момент есть свободные ресурсы, требующиеся для работы задаче.

Последний критерий определяется с помощью анализа запущенных задач. После этого, подходящей под критерии задаче TaskShift увеличивает приоритет и снимает ограничения пользователя на квоты. В следующем цикле через минуту система проверяет, осталась ли задача в очереди. Если да, то задача считается неуспешной и отправляется в конец очереди. В обоих случаях из очереди вновь берется первая задача и проверяется на критерии. Если задача не подходит, то она отправляется в конец очереди и берется следующая. И так далее.

Благодаря разработанной системе TaskShift [1] на модельных данных удалось запустить 12 ожидающих задач, тем самым смоделировав загруженность суперкомпьютера в выходные в 97% вместо ожидаемых 89% без работающей системы. Тем самым определяется реальная практическая значимость предложенного решения, ведь оно позволяет выполнять большее количество задач за один и тот же промежуток времени.

Разработанная подсистема внедрена в тестовую эксплуатацию на суперкомпьютере НИУ ВШЭ. Собираются логи и рассчитывается эффективность предложенного подхода. В следующем году, в рамках выполнения дипломной работы, подсистема TaskShift будет переведена в промышленную эксплуатацию.

Проект будет продолжен в рамках дипломной работы. Будет производиться расширение прогнозирования на весь год, в частности, для повышения загрузки суперкомпьютера в каникулы, новогодние и майские праздники. Планируется внедрить ансамблирование нейросетевых моделей для прогнозирования загрузки каждого типа вычислительных узлов суперкомпьютера НИУ ВШЭ.

[1] https://github.com/sssciel/TaskShift

Список литературы

[1] *A beginner's guide to SLURM*. URL: https://github.com/influxdata/influxdb-client-python.

[2] *APScheduler documentation*. URL: https://apscheduler.readthedocs.io/en/3.x/.

[3] Thierry Blu, Philippe Thévenaz и Michael Unser. "Linear interpolation revitalized". В: *IEEE Transactions on Image Processing* 13.5 (2004), с. 710—719.

[4] Inc Docker и др. "Docker". В: *línea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker* (2020).

[5] *DotEnv GitHub repository*. URL: https://github.com/theskumar/python-dotenv.

[6] *FastAPI documentation*. URL: https://fastapi.tiangolo.com/.

[7] *influxDB python client*. URL: https://github.com/influxdata/influxdb-client-python.

[8] Pavel Kostenetskiy, Roman Chulkevich и Viachesvlav Kozyrev. "HPC Resources of the Higher School of Economics". В: *Journal of Physics: Conference Series* 1740.1 (янв. 2021), с. 012050. DOI: 10.1088/1742-6596/1740/1/012050. URL: https://dx.doi.org/10.1088/1742-6596/1740/1/012050.

[9] Pavel Kostenetskiy, Artemiy Shamsutdinov, Roman Chulkevich, Vyacheslav Kozyrev и Dmitriy Antonov. "HPC TaskMaster–Task Efficiency Monitoring System for the Supercomputer Center". В: *International Conference on Parallel Computational Technologies*. Springer. 2022, с. 17—29.

[10] *likeC4 framework official site*. URL: https://likec4.dev/.

[11] *Loguru GitHub repository*. URL: https://github.com/Delgan/loguru.

[12] Hoang Minh Nguyen, Gaurav Kalra и Daeyoung Kim. "Host load prediction in cloud computing using long short-term memory encoder–decoder". В: *The Journal of Supercomputing* 75.11 (2019), с. 7592—7605.

[13] Brian Okken. *Python Testing with pytest*. Pragmatic Bookshelf, 2022.

[14] *SLURM Workload Manager*. URL: https://slurm.schedmd.com.

[15] Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Nikolay Laptev, Christoph Bergmeir и Ram Rajagopal. "Neuralprophet: Explainable forecasting at scale". В: *arXiv preprint arXiv:2111.15397* (2021).

[16] Oskar Triebe, Nikolay Laptev и Ram Rajagopal. “Ar-net: A simple auto-regressive neural network for time-series”. В: *arXiv preprint arXiv:1911.12436* (2019).

[17] Supercomputer Modeling Unit. *HPC TaskMaster repository GitLab*. URL: https://git.hpc.hse.ru/open-source/hpc-taskmaster (дата обр. 03.02.2025).

[18] Supercomputer Modeling Unit. *HSE University HPC Cluster cHARISMa*. URL: https://hpc.hse.ru/hardware/hpc-cluster (дата обр. 27.01.2025).

[19] Andrea Vázquez-Ingelmo, Alicia García-Holgado и Francisco J García-Peñalvo. “C4 model in a software engineering subject to ease the comprehension of uml and the software”. В: *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE. 2020, с. 919—924.

[20] Vadim Voevodin, Roman Chulkevich, Pavel Kostenetskiy, Vyacheslav Kozyrev, Anton Maliutin, Dmitry Nikitenko, Sergey Rykovanov, Artemiy Shamsutdinov, Yuriy Shkandybin и Sergey Zhumatiy. “Administration, Monitoring and Analysis of Supercomputers in Russia: a Survey of 10 HPC Centers”. В: *Supercomputing Frontiers and Innovations* 8.3 (окт. 2021), с. 82—103. DOI: 10.14529/jsfi210305. URL: https://superfri.org/index.php/superfri/article/view/397.

[21] Leland Wilkinson. “The grammar of graphics”. В: *Handbook of computational statistics: Concepts and methods*. Springer, 2011, с. 375—414.

[22] Сергей Анатольевич Жуматий и Константин Сергеевич Стефанов. *Суперкомпьютеры: администрирование*. ООО «МАКС Пресс», 2018.

[23] П.С. Костенецкий, А.Б. Шамсутдинов, Р.А. Чулкевич и В.И. Козырев. “Разработка подсистемы анализа эффективности использования вычислительных ресурсов для системы HPC TaskMaster”. В: *Параллельные вычислительные технологии (ПаВТ’2023)*. 2023, с. 155—161.