Обработка потока задач с зависимостями на нескольких вычислительных кластерах

Тимофей Владимирович Санников, Алексей Николаевич Сальников

Московский государственный университет им. М.В. Ломоносова
timohaj1@yandex.ru, alexey.salnikov@gmail.com

Аннотация В работе рассматривается проблема планирования задач с зависимостями по данным между вычислительными кластерами. Исследуется разработка алгоритмов планирования задач: модификация жадного алгоритма backfill, GRAPHENE и Spear. Включает экспериментальное исследование на вычислительных кластерах. Ожидаемый результат — создание эффективных алгоритмов планирования и инструментария для оценки их производительности.

Ключевые слова: планирование · backfill · жадный алгоритм · grid

Введение

В современном мире перед исследовательскими группами встают задачи, требующие значительных вычислительных ресурсов. Для обработки таких сложных и ресурсоемких вычислений применяются вычислительные кластеры — группы компьютеров, объединенных высокоскоростной сетью с низкими задержками.

Основными характеристиками вычислительного кластера являются его вычислительная мощность, измеряемая в FLOPS, а также количество одновременно задействованных вычислительных устройств: процессорных ядер, графических сопроцессоров и других компонентов.

Примеры задач, требующих значительных вычислительных ресурсов:

- **Исследования в медицине (например, изучение работы мозга)** [6]. В рамках этих исследований регистрировались электрокардиограммы (ЭКГ) у 30 испытуемых в различных условиях. Обработку полученных данных требовалось выполнить с высокой скоростью и точностью.
- **Моделирование физических процессов**. В этой области вычислительные кластеры используются для симуляции сложных явлений, таких как турбулентность в жидкостях, распространение электромагнитных волн и прочие задачи, требующие высокой точности расчетов. Примером такой задачи является [7], в рамках которой происходил модельный эксперимент, требующий большого количества вычислительных ресурсов.
- **Обучение нейросетей**. Для обучения многих нейросетей, необходимо использовать вычислительные класты. Обучение в исследованиях зачастую происходит на различных данных, соотвественно появляются множество задач на обучение, которые необходимо распределить по различным вычислительынм кластерам [1].

Для работы с вычислительными кластерами пользователи получают удаленный доступ к узлам системы через протокол SSH (Secure Shell). Это позволяет выполнять вычисления, запускать программы и управлять задачами, находясь на другом компьютере. Через SSH можно подключаться как к управляющему узлу, который распределяет задачи между вычислительными узлами, так и непосредственно к отдельным узлам, если это требуется. Такой подход обеспечивает безопасный и эффективный способ взаимодействия с кластерными ресурсами.

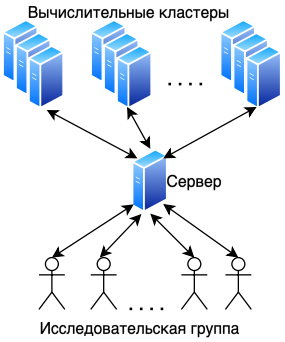
Помимо классических вычислительных кластеров, существуют Grid-системы, которые позволяют объединять распределенные вычислительные ресурсы, находящиеся в разных географических зонах. В отличие от традиционных кластеров, где задачи исполняются на локально связанных узлах, Grid-системы используют более гибкий подход, объединяя мощность нескольких независимых кластеров и серверов. Однако большинство существующих Grid-систем обладают высокой сложностью в установке, настройке и использовании, требуя значительных временных и технических затрат. Кроме того, их использование зачастую требует куда большего доступа, чем тот, который предоставляется по средствам протокола SSH, что ограничивает их применение для широкого круга пользователей.

Дополнительной сложностью является наличие зависимостей между задачами. В реальных сценариях выполнения вычислений часто требуется, чтобы одна задача начиналась только после завершения другой. Это приводит к необходимости управления процессами таким образом, чтобы учитывать порядок выполнения, минимизировать задержки и оптимально использовать вычислительные ресурсы.

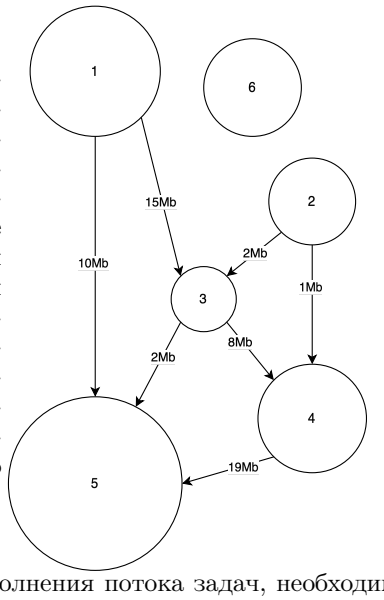
Таким образом, возникает необходимость в создании инструмента, который позволит упростить работу с Grid-системами и автоматизировать управление зависимостями между задачами. Такой инструмент должен снижать сложность конфигурации, быть адаптированным под работу через протокол SSH и предоставлять эффективные механизмы распределения вычислительных ресурсов.

1 Описание проблемы использования нескольких кластеров

Нами рассмотрен только частный случай, когда группа ученых устанавливает у себя в организации сервер, на который приходят запросы на выполнение задач со стороны клиентов (рабочих мест сотрудников). Сервер определяет, на каком вычислительном кластере будет выполнена задача, и отправляет необходимые данные и команду для постановки задачи в очередь на соответствующем кластере.



Поток задач — набор задач, которые поступают на сервер от исследовательской группы. У каждой задачи есть необходимые ей ресурсы (количество потоков), максимальное время выполнения задачи, необходимые для ее выполнения файлы и функция для вычисления времени выполнения задачи на выбранном кластере (опционально). Для представления потока задач используется представление направленного графа в виде файла формата json, который загружается в систему и привязывается к полю topology базы данных задач.



Для достижения меньшего времени выполнения потока задач, необходимо учитывать скорость соединения между сервером и кластером, так как перед постановкой задачи в очередь на кластере возможна передача файлов.

В рамках данной работы будем минимизировать время выполнения всего потока. Для оценки времени выполнения конкретной задачки с момента ее постановки будем использовать следующую функцию (1):

$$\min_{j\ in\mathcal{J}} \left(T^j_{\text{total}} \right) \tag{1}$$

$$T^j_{\text{total}} = \max_{i\in\mathcal{T}} \left(t^i_{\text{all}} \right), \tag{2}$$

где для каждой задачи i:

$$t^i_{\text{all}} = t_{\text{start}} + t^i_{\text{tr}} + t^i_{\text{que}} + t^i_{\text{exec}},$$
$$t^i_{\text{tr}} = \frac{S_i}{v} + t^i_{\text{tr\_prev}}.$$

Ограничения:

- Порядок выполнения, определяемый зависимостями: $j \prec i \Rightarrow t^j_{\text{all}} \leq t^i_{\text{start}}$.
- Запрет на превышение ресурсов кластера: $\sum_{i\in C_k} r_i \leq R_k$, $\forall k$.
- Запрет гранулярности выполнения задачи: $x_{ik} \in \{0, 1\}$.

Выполнение на разных кластерах возможно только при наличии всех необходимых файлов и ПО для обмена текущим состоянием задачи, включая оперативную память. Учитывая ограниченный доступ к кластеру без возможности просмотра памяти и необходимость передачи файлов, это увеличит время, необходимо выполнять задачу только на одном кластере (в обзоре назовем этот пункт целостностью).

Обозначение	**Описание**
$\mathcal{J}$	Множество расписаний
$\mathcal{T}$	Множество всех задач
T^j_{total}	Общее время выполнения потока при расписании j
t^i_{all}	Полное время выполнения задачи i
t^i_{tr}	Время передачи данных для задачи i
t^i_{exec}	Время выполнения задачи i на кластере
t^i_{que}	Время ожидания задачи i в очереди
S_i	Объем данных задачи i (бит)
v	Скорость передачи данных (бит/с)
$t^i_{\text{tr\_prev}}$	Время окончания передачи данных предыдущих задач на кластер
$j \prec i$	Задача i зависит от задачи j
R_k	доступные ресурсы кластера k
r_i	ресурсы, требуемые задачей i
x_{ik}	Задача i выполняется на вычислительном кластере k

Для достижения эффективного распределения ресурсов необходимо учитывать:

- зависимости между задачами;
- время выполнения задачи;
- время передачи данных, необходимых для выполнения задачи;
- время в очереди, с учетом других задач;
- целостность задачи.

2 Обзор существующих алгоритмов планирования в вычислительных кластерах

2.1 Описание алгоритмов планирования

Federated Scheduling Алгоритм, который описан в статье [4], разделяет задачи на две группы: быстрозавершаемые и долгосрочные. Это позволяет уменьшить задержки выполнения и эффективно распределять ресурсы кластера. Планирование происходит относительно локальных анализаторов каждого кластера. Данный метод снижает конкуренцию между разными типами задач и увеличивает общую пропускную способность системы.

Spear Scheduling Алгоритм из статьи [3] использует нейросетевую модель, которая обучается в процессе работы и адаптируется к текущему состоянию кластера. Такой метод позволяет предсказывать оптимальное распределение задач и минимизировать задержки, возникающие из-за неоптимального выбора узлов.

Bucket Algorithm Метод планирования, описанный в статье [5], предполагает перераспределение задач через регулярные промежутки времени с учетом изменяющихся параметров системы. Такой подход особенно эффективен в средах с высокой изменчивостью нагрузки и позволяет минимизировать простои вычислительных ресурсов.

GRAPHENE Алгоритм статьи [2] сочетает часть принципов Federated Scheduling и Bucket Algorithm, а также учитывает специфику кеширования данных. В контексте вычислительных кластеров оптимизирует процесс передачи файлов между сервером и узлами, что позволяет значительно повысить эффективность работы системы.

Модифицированный Backfill Алгоритм статьи [8] является усовершенствованной версией классического Backfill, включающей учет времени передачи данных и времени в очереди. В отличие от стандартного Backfill, он анализирует не только свободные временные интервалы в расписании, но и ожидаемые задержки, связанные с передачей файлов и завершением зависимых задач. Такой подход позволяет более точно прогнозировать время начала выполнения задач, минимизировать простои и увеличить загрузку вычислительных узлов. Но в нем нет учета зависимостей между задачами.

2.2 Сравнение алгоритмов планирования

Таблица 1: Соответствие алгоритмов поставленным требованиям

Название алгоритма	Учет зависимостей между задачами	Учет времени работы	Учет времени передачи	Учет времени в очереди	Учет других задач	Целость задачи	Реализуем для нашего случая
Federated Scheduling	Да	Да	Нет	Нет	Да	Да	Да
Spear Scheduling	Да	Да	Нет	Нет	Да	Да	Да
Bucket Algorithm	Да	Да	Нет	Нет	Да	Да	Да
GRAPHENE	Да	Да	Нет	Нет	Да	Да	Да
Modified Backfill	Нет	Да	Да	Да	Да	Да	Да

Из таблицы 1 видно, что большинство алгоритмов не учитывают время передачи данных. Все алгоритмы можно модифицировать так, чтобы они учитывали время передачи данных. Модифицированный BackFill не готов к использованию при наличии потока задач и нацелен на минимизацию времени выполнения конкретной задачи, а не всего потока. Для решения этих проблем, необходимо объединить его с идеей алгоритма Bucket о перераспределении задач, а также добавить учет зависимостей между задачами.

Для реализации были выбраны алгоритмы:

- GRAPHENE, из-за преимуществ в скорости расстановки по сравнению с Bucket. Дополнительной модификацией будет учет времени перердачи данных и времени в очереди;
- Spear, в котором при проходе по дереву поиска будет дополнительно учитываться время передачи и время в очереди. A DRL будет оценивать текущее состояние кластеров и, основываясь на этом, изменять шансы на выбор кластера;

- Модифицированный Backfill, с учетом новых модификаций, позволяющих делать расстановку с учетом зависимостей.

3 Обход графа потока задач

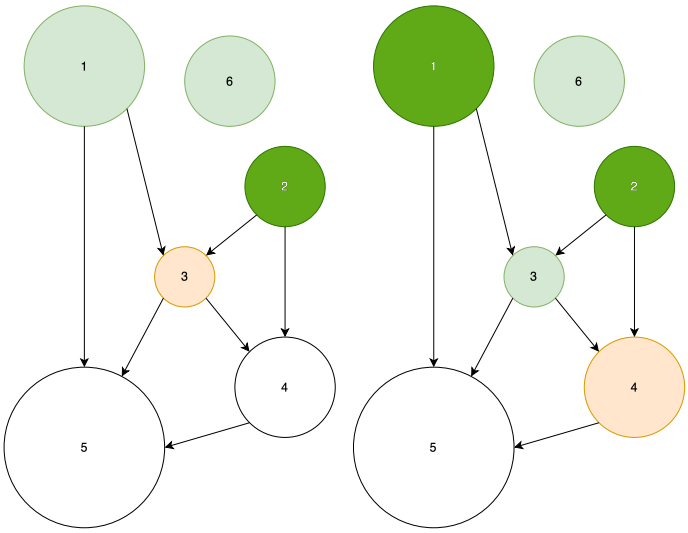
В нашем случае обход графа осуществляется посредством изменения статуса задач. На графиках 1:

- **Темнозеленый** – задача выполнена;
- **Светлозеленый** – задача готова к выполнению и соответственно распределению на кластер;
- **Оранжевый** – задача учитывается при построении расписания, но не ставится на выполнение (в случае возможности отправки данных для задачи, отправка данных происходит).

Идея заключается в том, что при ярусном обходе графа количество «светлозеленых» и «оранжевых» задач достаточно ограничено, чтобы производить полный перебор вариантов расстановки среди них. Причем приоритет выполнения всегда у «светлозеленых» задач. То есть в случае рисунка 1b будут рассматриваться только 3 варианта постановки задач в планировщик:

- 6→3→4;
- 3→6→4;
- 3→4→6.

В последующих главах темнозеленый и светлозеленый урои светлозеленый ярусы будут называться 1 и 2 ярусами соответственно.



(a) (b)

Рис. 1: Графики обработки вычислений

4 Новый алгоритм, основанный на BackFill

Модифицированный алгоритм Backfill включает в себя несколько улучшений для учета времени передачи файлов и времени нахождения задач в очереди.

4.1 Учет времени передачи необходимых файлов

- Перед постановкой задачи в очередь оценивается **время передачи файлов**, необходимых для выполнения этой задачи.
- Пока файлы не переданы, задача не может быть поставлена в очередь на выполнение.
- В алгоритме появляется **минимальное время начала выполнения задачи** – она не может стартовать раньше, чем все ее данные будут доступны.
- После определения этого минимального времени применяется стандартный алгоритм **Backfill**, но с учетом найденного ограничения.
- Если файлы уже находятся в нужном вычислительном кластере, задача может быть распределена сразу, как в классическом **Backfill**.

4.2 Учет времени в очереди

Введем обозначения:

- $t_{\min}$ – минимальное время, после которого задача может быть запланирована;
- t_{tr} – время окончания передачи файлов;
- $t_{\text{que this}}$ – время нахождения задачи в очереди (без учета задач, которые еще не завершили передачу файлов).

После завершения передачи всех файлов задача ставится во **внутреннюю очередь кластера**. Минимальное время начала выполнения определяется как:

$$t_{\min} = t_{\text{tr}} + t_{\text{que this}}. \tag{3}$$

Это позволяет **точнее оценить** время выполнения задачи, учитывая как передачу данных, так и загрузку вычислительных ресурсов. В результате выполняется **двухуровневое планирование**:

1. сначала на сервере, где выбирается подходящий кластер;
2. затем внутри самого вычислительного кластера.

4.3 Применение алгоритма для нескольких вычислительных кластеров

В случае, если задействовано несколько вычислительных кластеров, для каждого из них создается отдельный **планировщик**. Планировщики работают по улучшенному алгоритму **Backfill** и управляются с единого сервера.

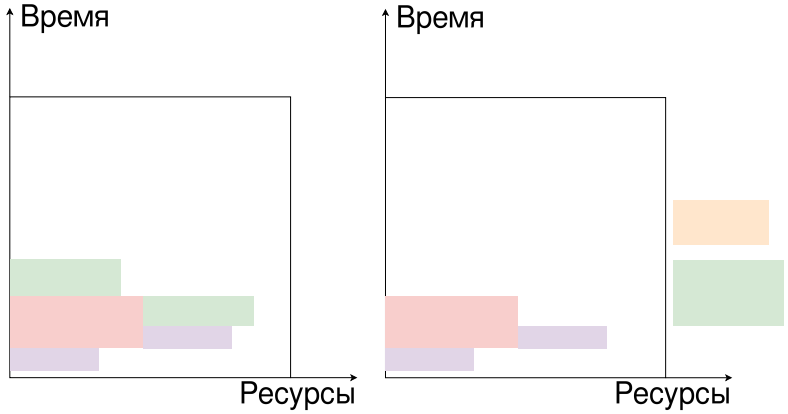
Для каждой задачи проверяется, через сколько времени она сможет стартовать на каждом кластере. Задача отправляется в тот кластер, где **время старта будет минимальным**.

4.4 Минимизация времени выполнения потока и учет зависимостей между задачами

За основу возмем идею алгоритма Bucket, раз в определенный промежуток времени будем перераспределять задачи по различным кластерам.

4.5 Демонстрация работы алгоритма

Продемонстрируем работу алгоритма в ситуации, если поставленная зеленая задача, имела зависимости в момент, когда ее поставили. Тогда сразу после выполнения одной из других задач(не обязательно на этом кластере) или поступлении новой задачи, происходит перераспределение задач по кластерам. Причем, если данные уже были отправлены, у нее больший шанс попасть на определенный ранее кластер из-за отсутствия времени передачи данных, необходимых для ее выполнения.



5 GRAPHENE

5.1 Доработанный алгоритм Graphene

Алгоритм **Graphene** предназначен для эффективного управления потоками задач в вычислительных кластерах. Основная идея Graphene заключается в том, чтобы минимизировать время ожидания задач за счет:

- динамического прогнозирования доступности ресурсов;
- разделения задач по их вычислительным характеристикам;
- анализа зависимостей между задачами и передачи необходимых файлов перед их выполнением.

5.2 Основные принципы работы алгоритма Graphene

В отличие от Backfill, алгоритм Graphene не просто заполняет пробелы в расписании, а активно формирует **граф зависимостей** задач и ресурсов. Основные этапы работы:

1. **Анализ входящих задач**: оценивается объем вычислений и потребность в ресурсах.
2. **Разделение задач на группы**: задачи классифицируются на **большие** и **малые** (см. раздел 5.3).
3. **Определение зависимостей**: формируется граф, в котором учитываются очередность выполнения задач и их потребность в данных.
4. **Прогнозирование выполнения**: оценивается, когда ресурсы станут доступными для каждой задачи.
5. **Планирование с учетом передачи данных**: задачи ставятся в очередь на выполнение с учетом времени их передачи и доступности вычислительных мощностей.

5.3 Разделение задач на большие и малые

Важной особенностью Graphene является **разделение задач** на две группы:

- **Большие задачи** – требующие значительных вычислительных мощностей и большого объема данных.
- **Малые задачи** – менее ресурсоемкие, но потенциально более частые.

Критерии разделения:

- Большая задача:

$$\alpha \cdot t_{\text{exec}_i} + \beta \cdot R_i > T_{\text{threshold}} \qquad w_i = \begin{cases} \alpha \cdot \left(t_{\text{exec}_i} + \frac{S_i}{v} \right) + \beta \cdot R_i, & \text{большая} \\ \gamma \cdot \left(\frac{S_i}{v} + \delta \cdot D_i \right), & \text{малая} \end{cases}$$

- Малая задача: иначе

Переменная	**Описание**
w_i	Вес задачи i
t_{exec_i}	Время выполнения (с)
S_i	Объем данных (бит)
v	Скорость передачи (бит/с)
D_i	Число зависимостей
R_i	Ресурсы (ядра/память)

α, β, γ, δ — константы.

Такой подход необходим по следующим причинам:

1. **Эффективность использования ресурсов**: крупные задачи могут блокировать очередь, а малые задачи позволяют заполнять промежутки между большими вычислениями.
2. **Минимизация задержек**: мелкие задачи выполняются быстрее, не мешая работе крупных задач.
3. **Гибкость планирования**: зная, какие задачи являются приоритетными, алгоритм может оптимально распределять вычислительные мощности.

Приоритет выполнения задач определяется на основе анализа графа зависимостей. Если малые задачи не мешают выполнению крупных, они могут быть запущены параллельно и на одном вычислительном кластере.

5.4 Применение алгоритма для нескольких вычислительных кластеров

Graphene позволяет управлять задачами на нескольких вычислительных кластерах. Процесс организации планирования основан на анализе текущей загруженности кластеров:

1. Каждый локальный планировщик анализирует загруженность своего кластера и прогнозирует доступность ресурсов.
2. Глобальный координатор собирает информацию о загруженности кластеров и принимает решение о назначении задачи.
3. Если требуется передача данных между кластерами, алгоритм заранее оценивает ее длительность и учитывает при принятии решений.

То есть на сервере не хранится уже составленное расписание, а распределение происходит в зависимости от текущего состояния кластеров.

6 Spear

6.1 Общая концепция

Модифицированный алгоритм Spear предназначен для оптимального планирования задач в распределенной вычислительной среде с учетом зависимостей между задачами и неоднородных требований к ресурсам. В основе алгоритма лежит комбинация *Монте-Карло Дерева Поиска (MCTS)* и *Глубинного обучения с подкреплением (DRL)*, что позволяет динамически адаптироваться к изменяющимся условиям кластера и минимизировать время выполнения всего задания (makespan).

6.2 Процесс планирования

Процесс планирования задач состоит из следующих этапов:

1. Формирование состояния кластера

На основе текущего состояния вычислительных узлов и DAG-графа формируется пространство решений, в котором учитываются доступные ресурсы, загруженность системы и незавершенные задачи.

2. Монте-Карло Дерево Поиска (MCTS)

MCTS используется для поиска наиболее оптимального расписания выполнения задач:

- **Выбор (Selection)**: выбор наиболее перспективного узла дерева решений на основе эвристики *Upper Confidence Bound (UCB)*.
- **Расширение (Expansion)**: создание новых возможных состояний системы путем назначения очередной задачи на вычислительный узел.
- **Симуляция (Simulation)**: оценка возможного времени завершения всего задания путем моделирования выполнения оставшихся задач.

– **Обратное распространение (Backpropagation)**: обновление значений узлов дерева на основе полученных данных.

3. Использование глубинного обучения с подкреплением (DRL) Чтобы ускорить процесс планирования, в Spear используется обученная нейросетевая модель, предсказывающая наиболее выгодные ходы при выборе задач. DRL позволяет:

– оптимизировать процесс поиска решений за счет предсказания вероятностного распределения возможных действий;
– улучшить симуляцию MCTS, заменяя случайный выбор решений на более осмысленный, основанный на обученных данных;
– минимизировать время вычислений, сокращая число ненужных проверок и переборов.

6.3 Адаптация под вычислительные кластеры

В модифицированной версии Spear учтены особенности работы в распределенной системе:

– Учитываются неоднородные требования к ресурсам.
– Планирование выполняется в реальном времени, минимизируя задержки при принятии решений.
– Алгоритм адаптируется к разным нагрузкам за счет динамической настройки параметров поиска.
– Включены дополнительные эвристики для предотвращения «голодания» задач, зависимых от узких мест в кластере.
– Для передачи данных для кластера и времени в очереди создается отдельный параметр(минимальное время старта), который участвует в рассчетах MCTS, а DRL, получает дополнительным параметром вектор значений времен передачи всех необходимых задаче данных для каждого кластера. Из-за чего происходит дополнительная корреляция, позволяющая влиять времени передачи данных на предсказание лучшего распределения.

7 Программная реализация

7.1 Инструмент постановки задачи на выполнение

Для работы со всеми алгоритмами была разработана система постановки задач в очередь, которая производит отправку необходимых файлов на вычислительные кластеры, ставит задачи во внутренние очереди вычислительных кластеров и обновляет статусы задачи (ожидает выполнения → отправляется на кластер → в очереди на выполнение → выполняется → завершена).

Хранение всех необходимых данных происходит в базе данных PostgreSQL (см. рис. 2).

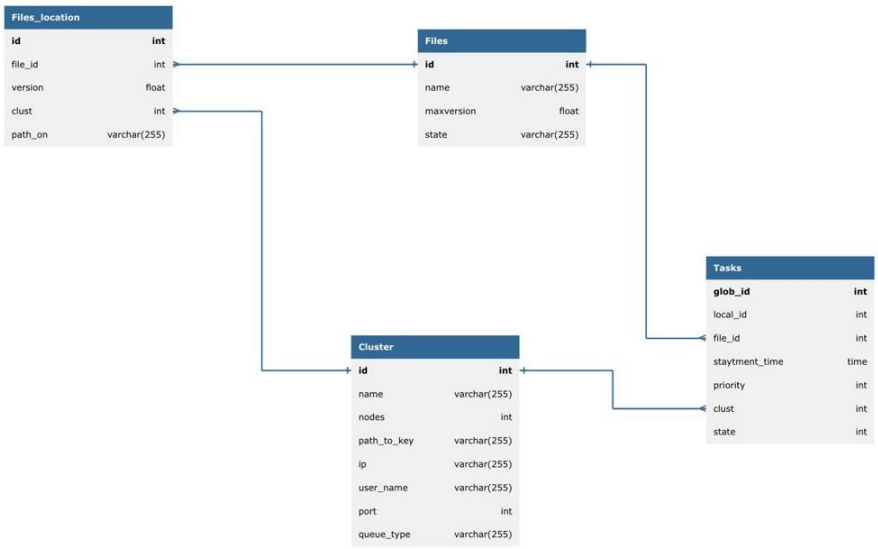


Рис. 2: Схема базы данных

Инструмент для работы с кластерами был написан на языке python, работа с кластером происходит через протокол SSH с использованием библиотеки paramiko.

7.2 Планировщики

Для оптимизации времени обработки потока, написание поланировщиков производилось на языке C++. Планировщики вызываются из Python через **библиотеку ctypes**, обеспечивая минимальные накладные расходы на межъязыковое взаимодействие.

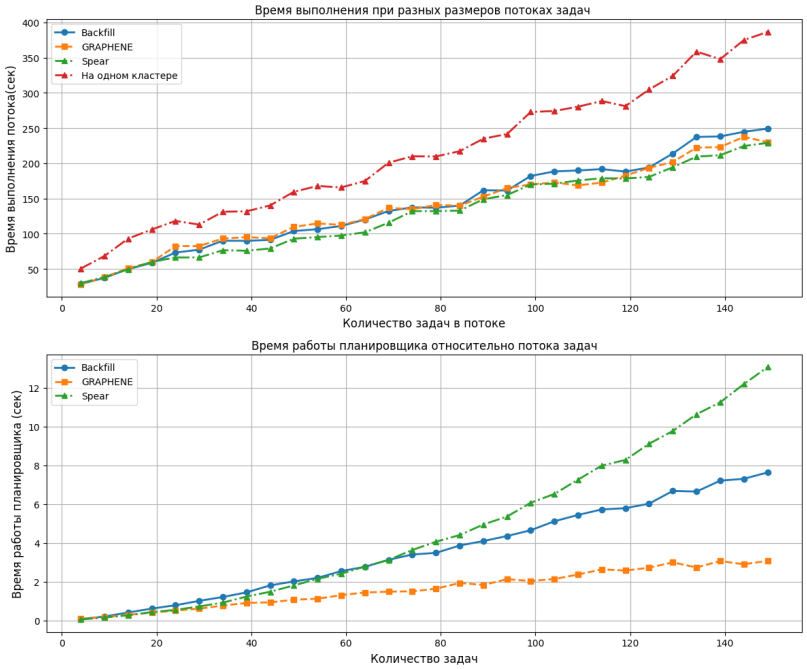
Обмен данными реализован через protobuf, что позволяет значительно уменьшить потери производительности при передаче данных задач и состояний кластеров.

8 Результаты тестирования

Для тестирования эффективности алгоритмов был написан генератор задач. Производилась релаксация матриц разного размера до достижения определенного качества.

Генерировались потоки задач, связанных друг с другом файлом вывода, в который отгружались рассчитанные матрицы. Таким образом можно было проверить корректность работы алгоритма.

Планировщики тестировались в одинаковых условиях, на одинаковых потоках задач. Для тестирования были запущены два компьютера с включенным slurm в режиме отладки, что позволило выставить необходимое число потоков, не смотря на их физическое отсутствие.



В результате тестирования было выявлено, что в большинстве ситуаций алгоритм spear показывает себя лучше всего, но тратит наибольшее количество времени на вычисления. Так что в некоторых ситуациях следует задуматься об использовании алгоритма GRAPHENE.

9 Заключение

В данной работе были рассмотрены вопросы планирования вычислительных задач в среде распределенных вычислительных кластеров. Проведен анализ существующих алгоритмов, таких как Federated Scheduling, Spear Scheduling, Bucket Algorithm, GRAPHENE и модифицированный Backfill, выявлены их преимущества и недостатки.

Для решения проблемы минимизации времени выполнения потока задач с зависимостями предложены модификации алгоритмов. В частности, был разработан новый алгоритм, основанный на Backfill, с учетом времени передачи файлов и времени в очереди на выполнение. Также была доработана схема планирования в алгоритме GRAPHENE и разработана новая формула оценки веса задачи, учитывающая необходимые данные. Кроме того, адаптация алгоритма Spear позволила использовать методы Монте-Карло

Дерева Поиска и глубинного обучения с подкреплением для динамической оптимизации планирования так, чтобы в них учитывались важные параметры при планировании между несколькими вычислительными кластерами.

Лучше всех в плане эффективности расстановки показал себя алгоритм Spear, по времени работы алгоритма лидером стал GRAPHENE, Backfill показал баланс между эффективностью и скоростью работы планировщика. Так что каждый из реализованных алгоритмов может оказаться лучшим для минимизаци и времени выполнения потока в зависимости от самого потока задач и зарактеристик сервера.

Таким образом, реализации предложенных алгоритмов позволили повысить эффективность использования вычислительных ресурсов, учесть зависимости между задачами и минимизировать задержки, связанные с передачей данных и ожиданием в очереди на выполнение.

Список литературы

1. Chan, E.R., Monteiro, M., Kellnhofer, P., Wu, J., Wetzstein, G.: pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis (2014). Stanford University
2. Grandl, R., Kandula, S., Rao, S., Akella, A., Kulkarni, J.: Graphene: Packing and dependency-aware scheduling for data-parallel clusters. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). USENIX Association (2016)
3. Hu, Z., Tu, J., Li, B.: Spear: Optimized dependency-aware task scheduling with deep reinforcement learning. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), p. 201. IEEE, Dallas, TX, USA (2019). doi:10.1109/ICDCS.2019.00201
4. Liu, H., Zhou, H., Chen, H., Yan, Y., Huang, J., Xiong, A., Yang, S., Chen, J., Guo, S.: A federated learning multi-task scheduling mechanism based on trusted computing sandbox. Sensors **23**(4), 2093 (2023). doi:10.3390/s23042093. URL https://doi.org/10.3390/s23042093
5. Stephens, D., Bennett, J., Zhang, H.: Implementing scheduling algorithms in high-speed networks. IEEE Journal on Selected Areas in Communications **17**(6), 1145–1158 (1999). doi:10.1109/49.772449
6. Васильев А.Н., Маковская А.Е., Каплан А.Я.: Динамика сенсомоторного ритма ЭЭГ при мысленном повторении за наблюдаемым движением. Журнал высшей нервной деятельности им. И. П. Павлова **73**(4) (2023)
7. Комаров А.С., Чертов О.Г., Быховец С.С.: Воздействие осиновых плантаций с коротким оборотом рубки на биологический круговорот углерода и азота в лесах бореальной зоны: модельный эксперимент. Математическая биология и биоинформатика **10**(2), 398–415 (2015)
8. Санников Т.В., Сальников А.Н.: Двухуровневое планирование потока задач между несколькими вычислительными кластерами. In: Сборник РИНЦ «Программные Системы и Инструменты», pp. 60–73 (2023)