Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Кислов Евгений Витальевич

Разработка подхода к реализации альтернативного планировщика для менеджера ресурсов "SLURM"

Научный руководитель:
в.н.с. НИВЦ МГУ, к.ф-м.н
Жуматий Сергей Анатольевич

Москва, 2022

Содержание

1. Аннотация

Низкая эффективность использования ресурсов является одной из основных проблем в области высокопроизводительных вычислений. Поэтому на суперкомпьютере необходимо иметь планировщик задач – программу, которая даст возможность пользователям ставить задачи в очередь на исполнение и выполнять их, учитывая приоритеты и лимиты каждого пользователя, а также максимизируя плотность ресурсов суперкомпьютера. В данный момент на суперкомпьютере Ломоносов-2 используется менеджер ресурсов SLURM и его встроенный планировщик задач backfill.

В данной работе предлагается подход к реализации альтернативного внешнего планировщика задач, который будет работать совместно с SLURM и дополнять функционал, который не поддерживается в SLURM и его планировщике задач - возможность задавать альтернативные алгоритмы или цепочки алгоритмов, задавать произвольные лимиты пользователям при исполнении задач, а также возможности расширенного сбора статистики по задачам.

В результате был разработан внешний планировщик, удовлетворяющий требованиям, описанным выше и апробирован на реальных задачах, взятых с суперкомпьютера Ломоносов-2 за февраль 2022 года. В результате апробации было показано, что алгоритм, разработанный к внешнему планировщику работает более эффективно, чем алгоритмом backfill, встроенный в SLURM.

2. Введение

В настоящее время суперкомпьютеры используются во многих областях науки – в медицине, квантовой физике, химии, биологии. Во всех этих предметных областях требуется применение высокопроизводительных вычислений для проведения масштабных расчетов. Это приводит ко всей большей массовости суперкомпьютерных технологий, поскольку увеличивается число пользователей таких систем. Повышение массовости данной области приводит к тому, что появляется все больше пользователей, которые являются экспертами в своих областях, однако не обладают профессиональными навыками создания эффективных параллельных приложений. Удовлетворение всех запросов пользователей в режиме реального времени в рамках мощностей суперкомпьютера является обязательной задачей такой системы.

На суперкомпьютере Ломоносов-2 ежедневно выполняется несколько сотен задач, которые ставят десятки разных пользователей. Каждый пользователь принадлежит определенным группам, которые имеют свои привилегии и ограничения. Диапазон задач, которые ставят пользователи, сильно колеблется - некоторые задачи занимают пару минут, а некоторые могут занимать недели, некоторые требуют один вычислительный узел, другие - сотни вычислительных узлов. В связи с этим необходимо максимально эффективно использовать ресурсы системы, уделяя их в равной степени задачам различных масштабов.

На данный момент момент на суперкомпьютере Ломоносов-2 используется менеджер ресурсов и планировщик задач SLURM(Simple Linux Utility for Resource Management) [1], который является одним из самых распространенных менеджеров ресурсов для суперкомпьютера. Среди его преимуществ стоит отметить масштабируемость и использование по умолчанию алгоритма планирования Backfill, который позволяет значительно увеличить плотность используемых ресурсов.

При всех вышеописанных достоинствах менеджера ресурсов SLURM, в его конфигурации и алгоритмах нет возможности использовать более широкий набор параметров, что, как следствие, ограничивает функционал, который мог бы значительно повысить эффективность процесса планирования задач. Например, в SLURM отсутствует возможность лимитирование ресурсов по разным признакам, например, группе пользователя - в SLURM лимиты только на число задач на счёте и в очереди, максимальное число процессоров на одну задачу и на время работы задачи.

Также нет возможности выставления приоритетов задач в зависимости от группы пользователей, очереди и других признаков. Кроме этого, отсутствует функционал, который необходим для администрирования и анализа работы планировщика задач – расширенных обработчиков событий(например, отказ вычислительного узла - такие события могут быть важны для администратора суперкомпьютера) и расширенного сбора статистики.

Задача данной работы заключается в разработке подхода к реализации альтернативного планировщика задач, который будет работать совместно с менеджером ресурсов SLURM и дополнять его возможности. Для это необходимо исследовать ряд вопросов. Первый из них – каким способом реализовать планировщик: как дополнение к коду SLURM или как отдельную программу, взаимодействующую с SLURM по сетевому интерфейсу.

Особое внимание было уделено архитектуре разрабатываемого подхода. Важно было учесть то, что другие разработчики будут в будущем использовать разработанный планировщик для добавления новых алгоритмов планирования, которые смогут использовать любые параметры в своей работе и не будут ограничены в используемых методах и средствах. Необходимо было разработать такой подход, который позволил бы указывать явно последовательность, в которой алгоритмы будут исполняться, а также гарантировать то, что все заданные алгоритмы не будут конфликтовать между собой.

При анализе работ по похожей тематике были найдены проекты [3, 5, 7]. Среди них [3, 7] не поддерживаются в течение последних десяти лет, проект [5] стал коммерческим и поэтому был исключен из рассмотрения. Кроме этого, работы [5, 7] не имели необходимого в рамках данной работы функционала.

3. Постановка задачи

В данной работе необходимо разработать подход к реализации планировщика для менеджера ресурсов SLURM, позволяющего:

- Добавлять алгоритмы планирования или последовательность разных алгоритмов планирования и выбирать, какие алгоритмы использовать для определенной очереди.
- Задавать в алгоритмах планирования приоритеты и лимиты по любым параметрам, например по идентификаторам пользователей, групп пользователей, разделов.
- Предоставлять выделенной группе пользователей возможность назначать лимиты и приоритеты на постановку задач, а также просматривать статистику очереди за любой промежуток времени.
- Добавлять обработчики на произвольные события планировщика.
- Использовать внутренние механизмы SLURM для исполнения задач.

4. Описание менеджера ресурсов SLURM

Так как данная работа зависит от SLURM, далее будет описана организация данного менеджера ресурсов.

SLURM - это отказоустойчивая система управления кластерами и планирования заданий для больших и малых кластеров Linux. SLURM не требует модификации ядра для своей работы и относительно самодостаточен. Как диспетчер рабочих нагрузок кластера SLURM выполняет следующие функции - во-первых, он предоставляет пользователям доступ к ресурсам на определенный период времени, чтобы они могли выполнять работу, во-вторых, он обеспечивает основу для запуска, выполнения и мониторинга работы на наборе выделенных узлов.

SLURM состоит из демонов slurmd, работающих на каждом вычислительном узле, и центрального демона slurmctld, работающего на управляющем узле (с дополнительным резервным двойником). Демоны slurmd обеспечивают отказоустойчивую иерархическую связь. Также, в случае использования базы данных в качестве хранилища конфигурации планировщика и информации о задачах, необходимо запустить демон slurmdbd, который будет обеспечивать хранение данных в базе данных MySQL.

Пользователь может взаимодействовать со SLURM посредством CLI команд, таких как

squeue(просмотр очереди заданий), sbatch(постановка задачи в очередь), scontrol(изменение конфигурации кластера).

Иерархия демонов и их взаимодействие посредством команд показано на рисунке 1.

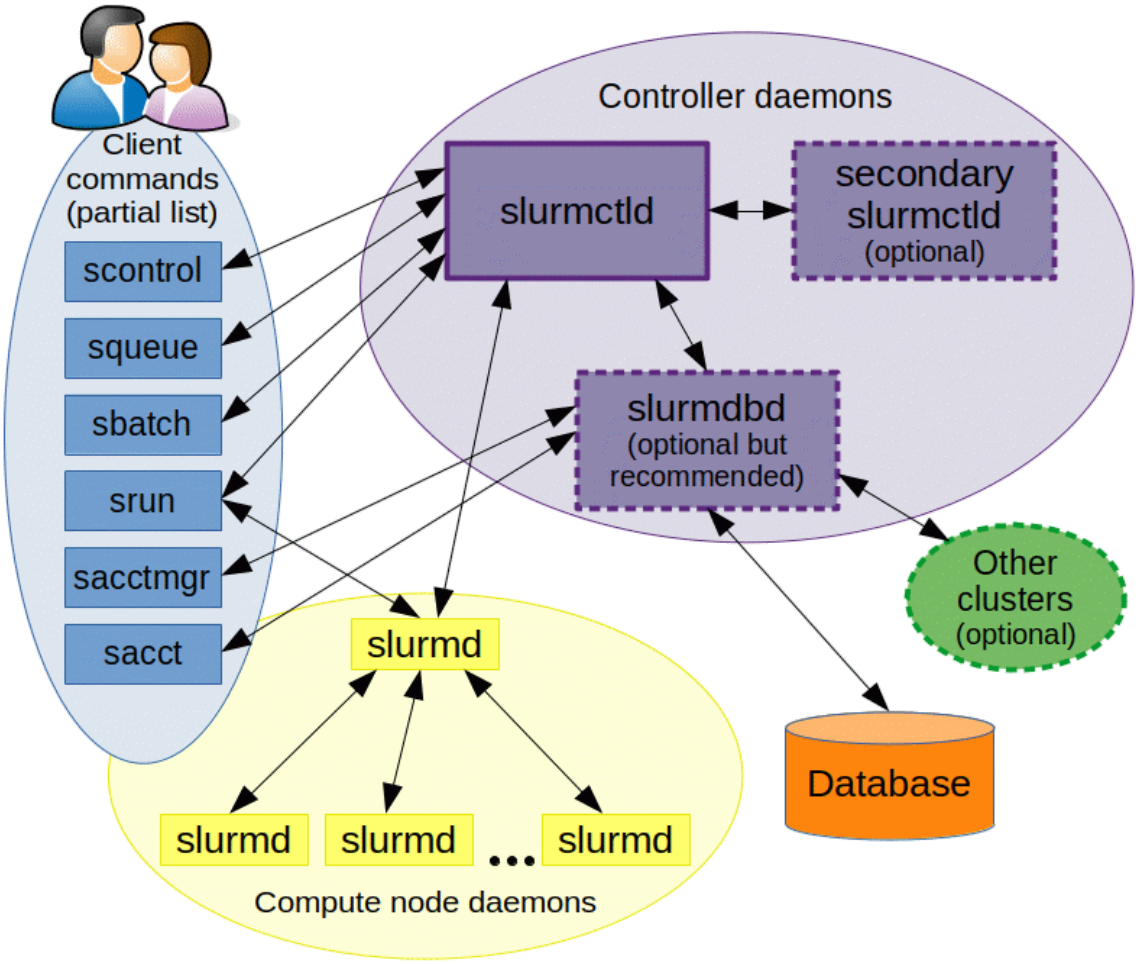

Рис. 1. Иерархия демонов SLURM

SLURM спроектирован для гетерогенных кластеров с допустимым количеством процессоров до 10 миллионов. Одно из основных преимуществ SLURM — модульность: ему доступны десятки дополнительных плагинов. Написаны стандартные планировщики, один из которых, backfill, используется для оптимизации работы суперкомпьютера «Ломоносова». При возникновении ситуации, когда стандартные алгоритмы планирования полностью не устраивают системного администратора, можно легко ввести в эксплуатацию собственный плагин, используя стандартные интерфейсы (wiki и wiki2), которые предоставляет SLURM

4.1. Описание протокола wiki2

SLURM может поддерживать взаимодействие с внешними планировщиками посредством интерфейса wiki2(работающего поверх TCP). В ходе работы он был

подробно изучен и использован. Команды данного протокол имеют вид: *AUTH=slurm DT=SC=-300 TS=<TIMESTAMP> CMD=<CMD> ARG=<ARGS>*. Ниже приведен список основных команд(их необходимо отправлять по сокету, открытому на порту 7321, который используется по умолчанию в интерфейсе wiki2):

1. *GETJOBS* - получение списка всех задач в очереди. Доступные параметры - ID, COMMENT, DPROCS, GNAME, MAXNODES, NAME, NODES, QUEUETIME, RCLASS, RMEM, STARTDATE, STARTTIME, STATE, TASKS, UNAME, UPDATETIME, WCLIMIT.
2. *GETNODES* - получение списка вычислительных узлов системы и их конфигурацию
3. *STARTJOB* - постановка задачи на выполнение по ее идентификатору
4. *CANCELJOB* - отмена выполнения задачи по ее идентификатору
5. *SUSPENDJOB* - приостановление задачи по ее идентификатору(при этом задача остается в памяти)
6. *RESUMEJOB* - возобновление задачи по ее идентификатору.

Также SLURM уведомляет по wiki2 о наступлении двух событий:

1. Задача была поставлена в очередь или закончена. В этом случае внешний планировщик получает сообщение “1234” по TCP соединению.
2. Был запущен демон slurmctld. В этом случае внешний планировщик получает сообщение “1235” по TCP соединению.

4.2. Описание REST API

SLURM имеет поддержку REST API с версии 20.2, которое позволяет получить конфигурацию планировщика, список задач и прочую информацию. Однако отсутствует возможность запуска задачи и возможность подписки на события менеджера ресурсов SLURM, поэтому данный интерфейс не подходит для реализации внешнего планировщика.

5. Существующие планировщики задач к менеджеру ресурсов SLURM

Кроме двух стандартных планировщиков, встроенных в SLURM - builtin и backfill, были исследованы существующие open-source решения среди планировщиков задач к менеджеру ресурсов SLURM.

1. Система плагинов SLURM spank plugins [3, 4], которая является частью менеджера ресурсов SLURM, однако не имеет достаточного функционала для планирования задач.
2. Планировщик задач MAUI [5, 6]. Его разработка началась в 1995 году, в ходе которой были добавлены плагины для различных менеджеров ресурсов, в том числе и для SLURM, с которым MAUI взаимодействует в качестве внешнего планировщика по протоколу wiki2. Однако в 2005 данное решение стало позиционироваться, как коммерческое, под названием MOAB, поэтому данное решение не подходит из-за высокой стоимости.
3. IpSched [7], перестал поддерживаться в 2011 году и, кроме того, не поддерживал нужного функционала.

Таким образом, в результате исследования существующих планировщиков не было найдено аналогичных решений, которые реализуют весь необходимый функционал и активно поддерживаются.

6. Сравнение двух подходов: внешнего и внутреннего планировщика к SLURM

Перед началом разработки архитектуры планировщика, необходимо было выбрать один из двух принципиально разных подходов - писать новый плагин в существующий код внутри менеджера ресурсов SLURM или писать независимое приложение, которое будет взаимодействовать со SLURM по протоколу wiki2, описанному выше.

Первый подход имеет существенные недостатки: SLURM постоянно обновляется и любые изменения в ядре приведут к тому, что при каждом обновлении нужно будет отслеживать каждое из таких изменений. Это значит, что такой код будет тяжело поддерживаем и подвержен ошибкам, что недопустимо при использовании на суперкомпьютере. Как следствие, многие внутренние механизмы SLURM, например методы манипуляции хранилищем конфигурации, будут недоступны, что лишает данный подход его основного преимущества. Кроме этого, писать код на C к существующему проекту [1] на несколько десятков тысяч строк, сильно опирающемуся на такие механизмы, как многопоточность, в разы сложнее, чем писать код на таком высокоуровневом языке, как python.

С учетом всех недостатков первого подхода, в данной работе был выбран второй подход, то есть отдельное приложение, которое будет взаимодействовать с SLURM через сетевой интерфейс.

7. Предлагаемый подход

Как было написано выше, в рамках данной работы требуется разработать подход к реализации внешнего планировщика, который будет взаимодействовать с SLURM, причем через сетевой интерфейс, как было выяснено в разделе 6.

Среди прочих требований, подход должен сохранить тот цикл работы менеджера ресурсов SLURM, который предусмотрен изначально с учетом встроенного планировщика задач, а также сохранить те средства постановки задачи, просмотра статуса задачи, просмотра состояния узлов и очередей, которые были изначально.

Таким образом, разрабатываемый планировщик задач имеет ограниченный список допустимых действий, который можно описать так:

1. Получать список задач.
2. Получать список узлов, очередей и прочую информацию, представляющую собой состояние SLURM.
3. Ставить задачи на выполнение, отменять выполнение задач, ставить выполнение задач на паузу.

Нужно заметить, что данное ограничение в используемых действиях по отношению к менеджеру ресурсов SLURM не должно накладывать ограничения на тот функционал, который реализуется в самом внешнем планировщике. К примеру, требование оставить без изменений средства SLURM получения состояния задач не накладывает ограничение на разработку собственных средств получения информации о задачах.

Разрабатываемый планировщик естественно представить в виде цикла, в начале каждой итерации которого проверяется, нет ли изменений в списке задач по сравнению с прошлой итерацией и, если есть, то выполнить определенные действия по отношению к задачам, изменившим свое состояние.

С учетом наложенных на разрабатываемый планировщик ограничений, постановки задачи(подробнее см. раздел 3) и используемого сетевого интерфейса wiki2(подробнее см. раздел 4.1), каждую итерацию цикла внешнего планировщика можно представить псевдокодом, представленным на рисунке 2.

```
get event from wiki2

if event type is "job queued" or "job completed":
    get SLURM state and jobs from wiki2
    for event handler in event handlers list:
        send event to event handler
    for scheduling component in scheduling components list:
        run scheduling algorithm
    if runnable jobs list is not empty:
      for job in runnable jobs list:
        run job
    update scheduler state
```

Рис 2. Псевдокод итерации цикла внешнего планировщика задач.

Также стоит отметить, что во внешнем планировщике должна быть предусмотрена фаза инициализации и добавления компонентов планирования и обработчиков событий.

Далее будет описана архитектура системы, которая будет решать поставленную задачу и удовлетворять всем описанным требованиям.

7.1. Архитектура системы

В результате был разработан внешний планировщик задач, архитектура которого представлена на рисунке 3:

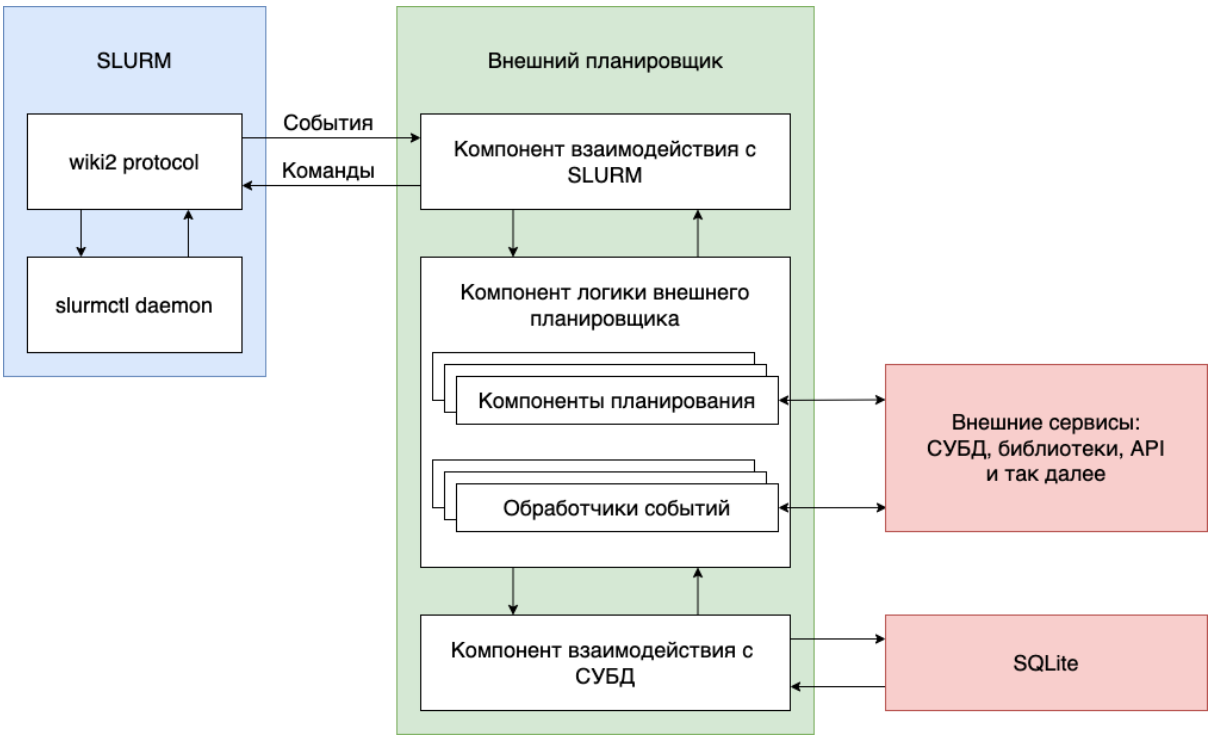

Рис. 3. Архитектура разработанного планировщика

Для того, чтоб реализация внешнего планировщика была более поддерживаемой, было решено логически выделить три компонента разрабатываемой системы:

1. Компонент логики внешнего планировщика.
2. Компонент взаимодействия с менеджером ресурсов SLURM.
3. Компонент взаимодействия с СУБД.

Далее будут описаны первые два компонента.

7.2. Компонент логики внешнего планировщика

Компонент логики внешнего планировщика принимает на вход событие от компонента взаимодействия с SLURM(подробнее см. раздел 7.3), применяет систему компонентов планирования(подробнее см. раздел 7.2.1), а также систему обработчиков событий(подробнее см. раздел 7.2.2). В ходе работы данный компонент вызывает функции компонента взаимодействия с SLURM, например постановка задачи на выполнение.

7.2.1. Система компонентов планирования

Для того, чтоб в будущем разрабатывать новые алгоритмы, которые могут использовать любые параметры на входе и любые средства в ходе работы, задавать последовательность, в которой их исполнять, а также по мере востребования включать или выключать алгоритмы, необходимо разработать отдельный модуль - систему компонентов планирования.

С точки зрения логики внешнего планировщика компонент планирования представляет из себя абстракцию, которая реализуется каждым конкретным алгоритмом. Каждый компонент планирования будет принимать на вход два параметра:

- Базовое состояние системы. Включает в себя состояние, полученное из менеджера ресурсов SLURM в момент запуска последовательности алгоритмов и содержит такую информацию, как список вычислительных узлов и список очередей. Также включает в себя состояние, которое хранит сам внешний планировщик. В основном это те переменные, которые нет возможности получить из SLURM через сетевой интерфейс, например количество задач, выполняемых в каждой очереди. Данное состояние здесь называется базовым, так как подразумевается, что каждый алгоритм будет иметь свое специфическое состояние.
- Полный список задач в очереди, включая все данные о каждой задаче - список узлов, очередь, идентификатор пользователя и его группы и так далее.

Также каждый компонент планирования имеет фазу инициализации для того, чтоб заранее подготовить все необходимые ресурсы для будущих вызовов. Например, это может быть установка соединения с базой данных, инициализация необходимых файлов в файловой системе, подгрузка необходимой конфигурации, настраивание системы журналирования, загрузка и инициализация необходимых библиотек и многое другое.

Упрощенная работа цепочки компонентов планирования приведена на рисунке 4:

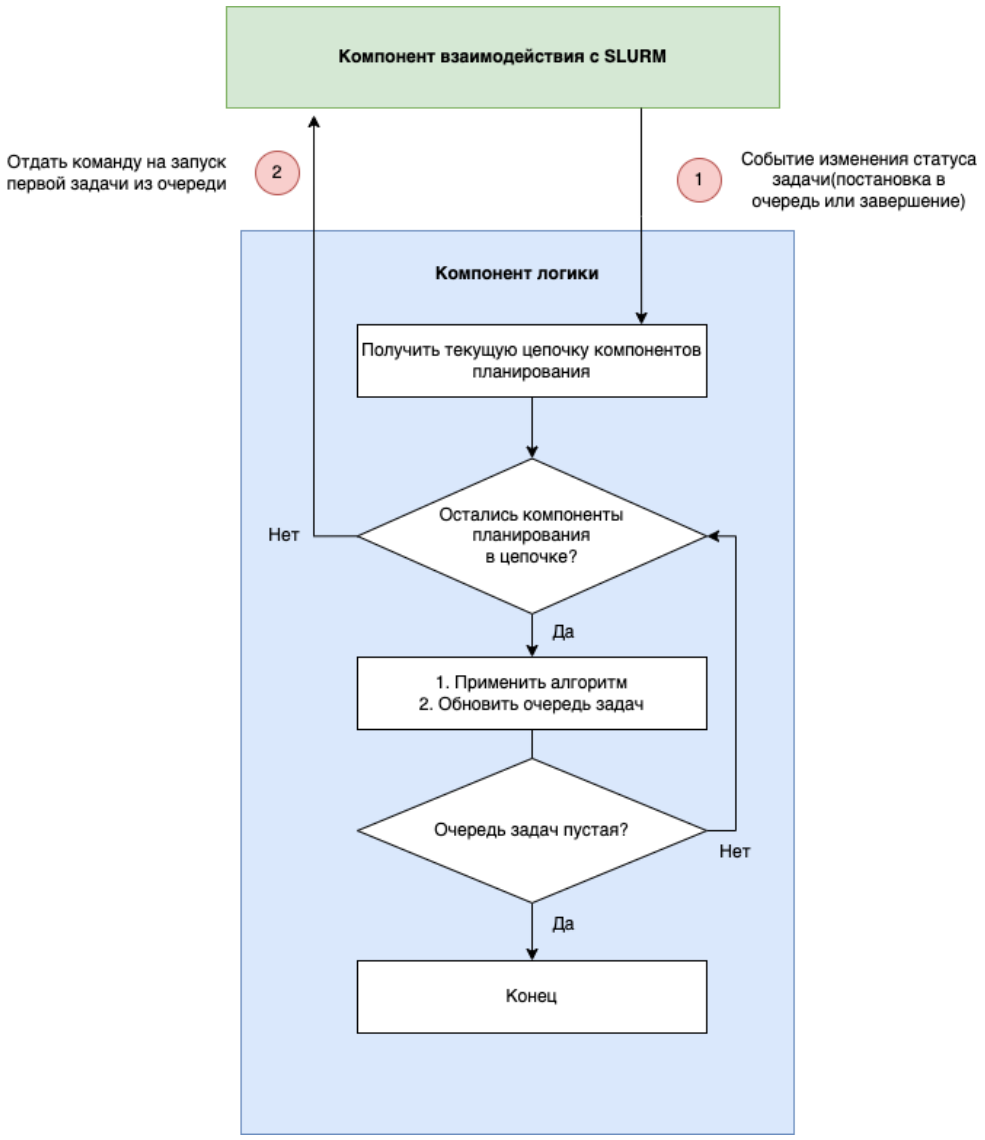

Рис. 4. Блок-схема, описывающая работу цепочки алгоритмов

7.2.2. Система обработчиков событий

Во внешнем планировщике необходима система обработчиков событий, которая позволит системным администратором и аналитикам собирать полную информацию о работе планировщика, включая состояния узлов в определенный момент времени, состояние очереди и прочую информацию, которая в совокупности из себя представляет полный снимок системы в определенный момент времени.

Система обработчиков событий, так же, как и система алгоритмов, имеет абстракцию “обработчик событий”, которая реализуется каждый раз под конкретный обработчик событий.

На вход обработчик событий принимает один параметр - событие, который содержит название события и полную информацию о нем. Например в случае

завершения задачи это будет идентификатор задачи, время ожидания задачи в очереди, время выполнения задачи, статус завершения и так далее.

Так же, как и алгоритм, обработчик событий имеет фазу инициализации, в которую будут заранее инициализированы все необходимые средства, впоследствии использованные в момент обработки события.

7.3. Компонент взаимодействия с SLURM

Данный компонент отвечает за взаимодействие с менеджером ресурсов SLURM и предоставляет интерфейс остальным компонентам системы для подписки на события, постановки задачи в очередь и многих других действий. Важно отметить, что предоставляется именно интерфейс, а не конкретная реализация взаимодействия с SLURM. Это позволяет изменять способы взаимодействия с SLURM, не меняя других компонентов системы.

Описанный подход позволит в случае необходимости независимо от других компонентов внешнего планировщика реализовать метод взаимодействия с SLURM через собственный протокол и заменить им протокол wiki2, который используется в данный момент.

На данный момент описываемый компонент реализован под протокол wiki2 и представляет из себя цикл с поллингом событий от SLURM через неблокирующий сокет-сервер, который для каждого события подготавливает нужные данные - информацию о задаче и текущее состояние планировщика SLURM и передает эти данные а компонент логики внешнего планировщика.

7.4. Выбранные инструменты

В качестве языка программирования был выбран язык python. Выбор обусловлен лаконичностью языка и высоким уровнем конструкций, а также тем, что в данном случае не было необходимо использовать низкоуровневые конструкции - основную часть кода составляет взаимодействие со SLURM посредством сокетов, обращение к базе данных и непосредственная логика планировщика.

В качестве хранилища данных использовалась встраиваемая СУБД SQLite.

8. Предлагаемый алгоритм планирования

Для того, чтоб к реализованному внешнему планировщику задач разработать алгоритм, который будет сопоставим по эффективности с алгоритмом backfill, используемом в менеджере ресурсов SLURM, были изучены похожие работы [9, 11-13], в которых описываются различные подходы, используемые в данном алгоритме, а также изменения, которые позволяют достичь большей эффективности.

В результате был разработан модифицированный алгоритм backfill, который ставит задачу исходя из приоритета, вычисленного следующим образом:

```
nodes_coeff = 1
queued_time_coeff = 2.5
max_time_coeff = 1

nodes_fit = 100.0 * task_nodes / nodes_available
queued_time = current_time - task_queue_time

priority =
  nodes_coeff * nodes_fit +
  queued_time_coeff * nodes_fit +
  max_time_coeff + max_time
```

Рис 5. Формула вычисления приоритета задачи.

Такой приоритет вычисляется каждый раз для всех задач из очереди, когда планировщик задач проверяет, есть ли доступные по текущим ресурсам задачи. Далее, список задач очереди сортируется по приоритету и на выполнение ставится задача, имеющая наивысший приоритет. Стоит также отметить, что приоритет определенной задачи не связан явным образом с приоритетом других задач.

9. Апробация на данных с суперкомпьютера Ломоносов-2

9.1. Условия апробации

Для апробации необходимо было максимально приблизить условия и набор задач суперкомпьютера Ломоносов-2. Для этого была использована архивная выборка задач с суперкомпьютера за февраль 2022 года - 7500 задач с количеством занимаемых узлов от 1 до 50 штук и временем выполнения от нескольких секунд до суток. Распределение занимаемого количества узлов и времени выполнения показано на рисунках 6 и 7.

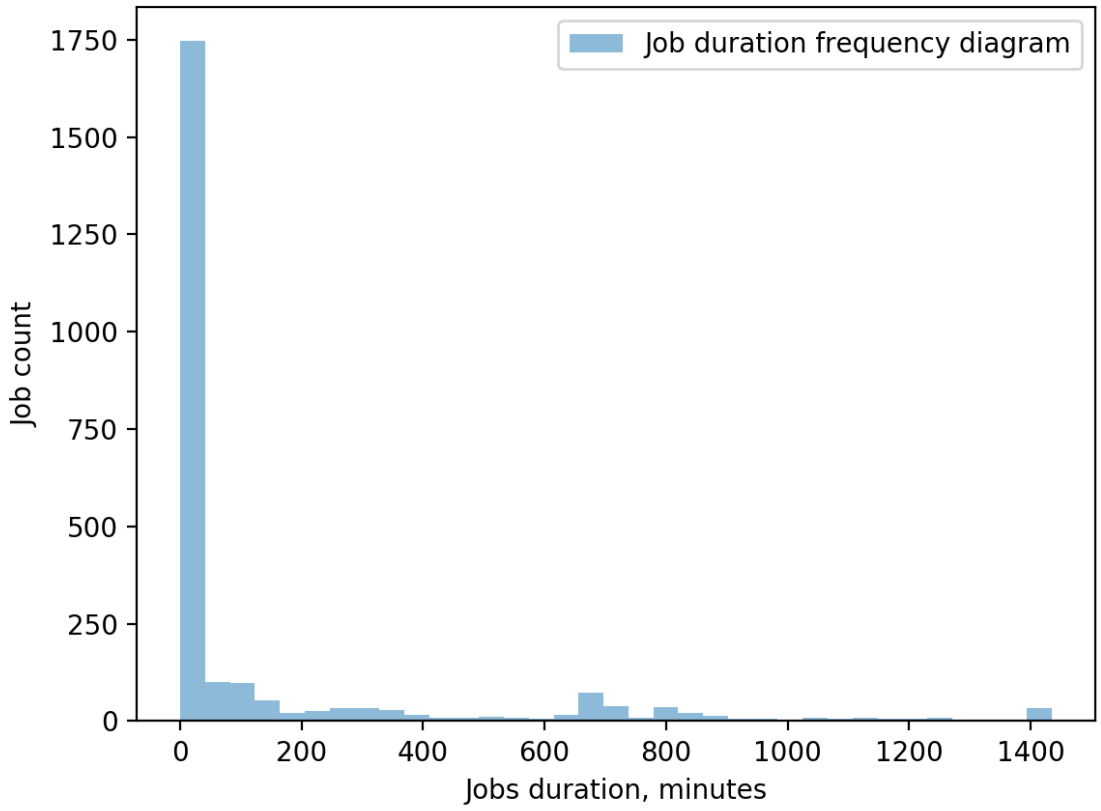

Рис 6. Распределение времени выполнения задачи.

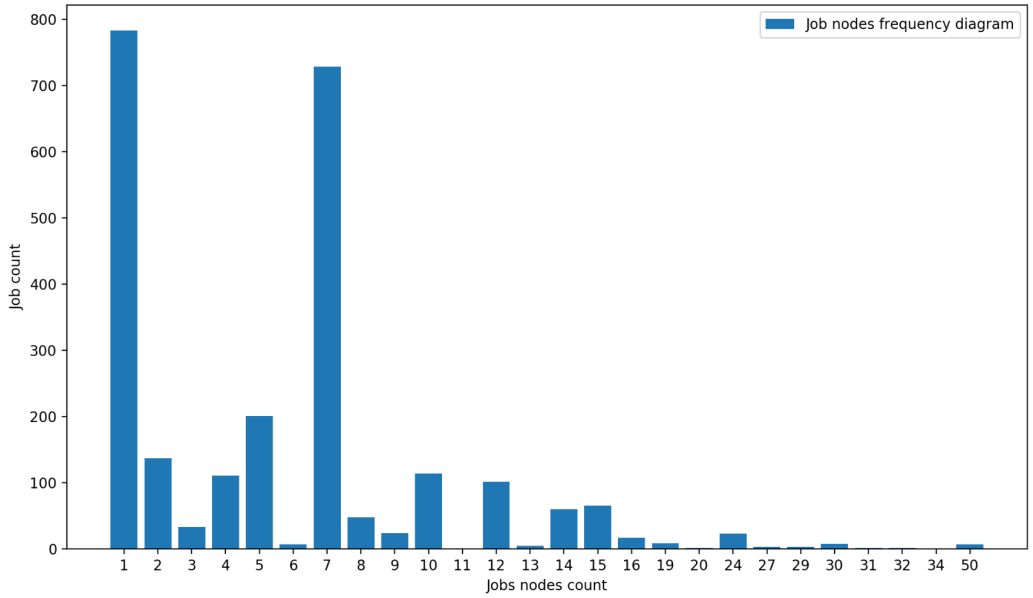

Рис 7. Распределение количества занимаемых задачей узлов.

Для того, чтоб как можно сильнее приблизить набор задач для апробации к реальным задачам, было использовано именно это распределение. Исключение составило только то, что из распределения по времени выполнения были взяты задачи, выполнявшиеся до 400 минут. Это было сделано с целью более быстрого тестирования.

Таким образом параметрами для задач были:

- Количество занимаемых узлов - от 1 до 50 включительно
- Время запуска - от 1 секунды до 400 минут

Для того, чтоб эмулировать различное время запуска, использовался bash-скрипт, вызывающий функцию sleep [8] из стандарта POSIX с параметром времени запуска задачи.

Было решено запустить 280 задач в течение 1.5 часа. Этот выбор был обусловлен тем, что на используемой исторической выборке среднее количество задач в день было примерно равно 280. Задачи запускались с равным интервалом в 20 секунд. Такие условия моделировали пиковую нагрузку на суперкомпьютер в течение одного дня.

Для генерации и запуска задач была отдельно реализована программа,

которая

- Заранее генерировала единые параметры задач для всех последующих запусков. Таким образом два сравниваемых планировщика, встроенный в SLURM планировщик backfill и разработанный внешний планировщик, получали идентичные наборы задач, что позволило более точно сравнивать результаты запуска в будущем.
- Запускала задачи на исполнение с равными промежутками времени.

Апробация проводилась на виртуальном сервере с выделенными ресурсами. Аппаратные характеристики - CPU, Intel x86, 756636 kB RAM.

На узел был установлен SLURM версии 15.08 и настроена симуляция 128 вычислительных узлов.

9.2. Результаты апробации

В результате апробации на встроенном в SLURM планировщике задач backfill были получены следующие результаты по времени ожидания задачи в очереди:

- Среднее время ожидания в очереди: 42 секунды.
- Медиана времени ожидания в очереди: 26 секунд.
- 95-й перцентиль ожидания в очереди: 83 секунды.

В то же время, для разработанного внешнего планировщика задач и разработанного к нему алгоритма были получены следующие результаты:

- Среднее время ожидания в очереди: 29 секунд.
- Медиана времени ожидания в очереди: 29 секунд.
- 95-й перцентиль ожидания в очереди: 66 секунд.

Сравнительная диаграмма частот по времени ожидания задачи в очереди показана на рисунке 8:

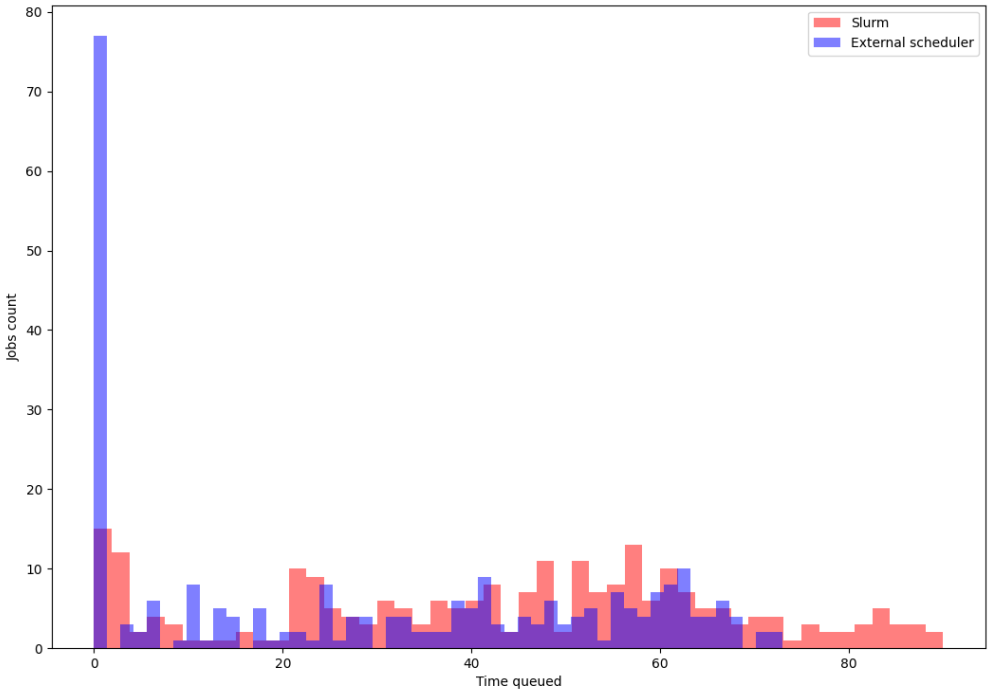

Рис 8. Сравнительная диаграмма частот по времени ожидания задачи в очереди.

Таким образом, по таким характеристикам, как среднее время ожидания и по сравнению графика частот времени ожидания задачи в очереди можно сделать вывод, что разработанный к внешнему планировщику задач алгоритм backfill является более оптимальным, чем встроенный в SLURM алгоритм backfill.

Алгоритм является более оптимальным из-за того, что среди прочих параметров задачи, таких, как количество занимаемых узлов и время выполнения, учитывается время ожидания в очереди с помощью подсчета приоритета задачи, в котором для времени ожидания выставлен более высокий коэффициент относительно остальных параметров.

10. Результаты

В результате было выполнено следующее:

1. Разработан подход к реализации планировщика, использующий внутренние механизмы менеджера ресурсов SLURM, позволяющий добавлять произвольные алгоритмы планирования и обработчики событий.
2. На базе реализованного подхода разработан модифицированный алгоритм планирования backfill.
3. Прототип апробирован на исторических данных, взятых с суперкомпьютера Ломоносов-2.

Список литературы

1. Slurm, описание и документация [Электронный ресурс]. - Режим доступа: https://slurm.schedmd.com/, свободный. (Дата обращения: 01.03.2022).
2. Slurm, исходный код [Электронный ресурс]. - Режим доступа: https://github.com/SchedMD/slurm, свободный. (Дата обращения: 01.03.2022).
3. Slurm spank plugins, описание и документация [Электронный ресурс]. - Режим доступа https://slurm.schedmd.com/spank.html, свободный. (Дата обращения: 01.03.2022).
4. Slurm spank plugins, исходный код [Электронный ресурс]. - Режим доступа: https://github.com/grondo/slurm-spank-plugins, свободный. (Дата обращения: 01.03.2022)
5. Планировщик задач MOAB [Электронный ресурс]. - Режим доступа: https://adaptivecomputing.com/, свободный. (Дата обращения: 01.03.2022).
6. Maui administrator guide [Электронный ресурс]. - Режим доступа: http://docs.adaptivecomputing.com/maui/pdf/mauiadmin.pdf, свободный. (Дата обращения: 01.03.2022).
7. IpSched scheduler, исходный код [Электронный ресурс]. - Режим доступа: https://github.com/serensoner/slurm-ipsched, свободный. (Дата обращения: 01.03.2022).
8. sleep(3) Linux manual page [Электронный ресурс]. - Режим доступа: https://man7.org/linux/man-pages/man3/sleep.3.html, свободный. (Дата обращения: 01.01.2022).
9. Comparison of backfilling algorithms for job scheduling in distributed memory parallel system [Электронный ресурс]. - Режим доступа: https://peer.asee.org/comparison-of-backfilling-algorithms-for-job-scheduling-in-distributed-memory-parallel-system.pdf, свободный. (Дата обращения: 01.05.2021).
10. Job scheduling with the SLURM resource manager [Электронный ресурс]. - Режим доступа: https://is.muni.cz/th/173052/fi_b_b1/thesis.pdf, свободный. (Дата обращения: 01.11.2020).
11. Sergei Leonenkov, Sergey Zhumatiy (2015). “Introducing New Backfill-Based Scheduler for SLURM Resource Manager”. Procedia Computer Science. 66, p.661-669.

12. R. Baraglia (2008). “Backfilling Strategies for Scheduling Streams of Jobs on Computational Farms”. Making Grids Work. II, p. 103-115.
13. D. Jackson (2007). Core Algorithms Of The Maui Scheduler. Brigham Young University, Proto, Utah.
14. Исходный код проекта GLURMO [Электронный ресурс]. - Режим доступа: https://github.com/ew-kislov/GLURMO, свободный. (Дата обращения: 01.10.2020).