

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

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Программный проект на тему:

Разработка симулятора вычислительного кластера

Выполнил студент:

группы БПМИ206, 4 курса

Макогон Артём Аркадьевич

Принял руководитель ВКР:

Сухорослов Олег Викторович

Доцент, к.т.н.

Факультет компьютерных наук НИУ ВШЭ

Москва 2024

Содержание

1	Введение	6
1.1	Актуальность и значимость	6
1.2	Цели и задачи ВКР	6
1.3	Основные результаты работы	7
1.4	Структура работы	7
2	Обзор литературы	8
2.1	Вычислительная модель	8
2.2	Аналоги	8
2.2.1	AccaSim	9
2.2.2	BatSim	9
2.2.3	IRMaSim	10
2.2.4	Elastisim	10
2.3	Выводы	11
3	Фреймворк DSLab	11
3.1	Архитектура DSLab	11
3.2	Асинхронное управление событиями	12
4	Реализация симулятора	13
4.1	Архитектура симулятора	13
4.2	Модель кластера	14
4.2.1	Модуль вычислений	14
4.2.2	Модуль сети	15
4.2.3	Модуль хранилища	16
4.3	Вычислительная модель	17

4.3.1	Деление на Collection и Execution	17
4.3.2	ExecutionProfile	17
4.3.3	Описание нагрузки через входной файл	20
4.3.4	Декларативное создание профилей нагрузки через файл	20
4.4	Модель планирования заданий	21
4.5	Входные данные симуляции	24
4.5.1	Трейс нагрузки	24
4.5.2	Конфигурация кластера и сети	25
4.6	Выходные данные симуляции	26
4.7	Конфигурация симуляции	26
5	Эксперименты и тестирование	27
5.1	Производительность	27
5.2	Честность	28
5.3	Детерминированность симуляции	31
6	Заключение	32
	Список источников	33

Аннотация

Алгоритмы планирования задач на вычислительных кластерах являются предметом активных исследований. Проведение таких исследований на настоящем кластере долго и дорого, поэтому для сравнения различных гипотез используются симуляторы. Качество получаемых результатов во многом зависит от точности используемой модели. Большинство современных симуляторов используют простые модели, которых не всегда достаточно. Эта работа сосредоточена на разработке симулятора вычислительного кластера на базе фреймворка DSLab, который позволит очень точно и гибко моделировать сценарии вычислительной нагрузки с использованием примитивов асинхронного программирования. DSLab предоставляет массу возможностей как для написания новых алгоритмов, так и для использования готовых модулей различных компонент распределенной системы. Таким образом, новый симулятор получит основные преимущества DSLab, которые сделают его еще более привлекательным для исследователей сложных алгоритмов.

Ключевые слова

Распределенные системы, вычислительный кластер, алгоритмы планирования, симуляция, DSLab

Annotation

Algorithms for scheduling tasks on compute clusters are the subject of active research. It is quite expensive to conduct such studies using real cluster, so simulators are used to compare different hypotheses. The quality of the results obtained largely depends on the accuracy of the model used. Most modern simulators use simple compute models, which are not always enough. Our work is focused on developing a simulator of a compute cluster based on the DSLab framework, which will allow very accurate and flexible modeling of computational load scenarios using asynchronous programming primitives. DSLab provides a lot of opportunities both for writing new algorithms and using ready-made modules of various distributed system components. Thus, the new simulator will receive the basic advantages of DSLab, which will make it even more attractive for researchers of complex algorithms.

Keywords

Distributed systems, compute cluster, scheduling algorithms, simulations, DSLab

1 Введение

Вычислительные кластеры, состоящие из набора серверов с различными ресурсами (процессор, память, диск), широко применяются для проведения сложных расчетов и обработки больших объемов данных в науке и бизнесе. На большом промышленном кластере одновременно могут выполняться тысячи заданий (batch jobs) различных пользователей, а часть заданий может находиться в очереди в ожидании выделения ресурсов. Запуском заданий и распределением ресурсов между ними управляет менеджер кластера или планировщик. В основе работы менеджера кластера лежат алгоритмы планирования задания (job scheduling), основной целью которых является максимально эффективное использование ресурсов кластера. Данные алгоритмы являются предметом активных исследований. Проводить такие исследования на реальном кластере долго и дорого, поэтому необходима компьютерная модель (симулятор), позволяющая быстро проверить гипотезу или провести сравнительное тестирование разных алгоритмов на некоторой истории нагрузки кластера. Подобный симулятор также может использоваться в учебном процессе для знакомства студентов с такими системами и возникающими в них задачами.

1.1 Актуальность и значимость

Алгоритм планирования задач – ключевая часть архитектуры всех вычислительных кластеров. Поиск оптимальных алгоритмов очень актуален, поскольку повышение утилизации имеющихся ресурсов позволит снизить затраты на оборудование кластера и ускорить его работу. Существующие симуляторы в основном предлагают простые модели задач или углубляются в конкретную область (например, глубинное обучение).

Новый симулятор, разработанный на базе фреймворка DSLab позволит исследователям подбирать алгоритмы под конкретные сценарии нагрузки, используемые в их кластере. Таким образом, симулятор позволит более точно предсказывать поведение настоящего кластера.

1.2 Цели и задачи ВКР

Целью ВКР является разработка симулятора вычислительного кластера на базе фреймворка DSLab.

Для достижения этой цели были поставлены следующие задачи:

1. Изучить литературу по теме и существующие симуляторы, подготовить обзор с анализом их преимуществ и недостатков.
2. Поэтапно реализовать компоненты симулятора, покрыть их тестами и снабдить комментариями.
3. Подготовить и провести эксперименты, демонстрирующие работоспособность симулятора и достижение всех требований.
4. Подготовить документацию пользователя.

1.3 Основные результаты работы

В результате ВКР был разработан эффективный и многофункциональный симулятор вычислительного кластера на базе фреймворка DSLab, были реализованы несколько популярных алгоритмов планирования задач и проведены тесты на данных трейсов различных компаний, а также синтетических данных.

1.4 Структура работы

Дальнейшая структура работы следующая:

- Глава 2 содержит обзор литературы и существующих симуляторов.
- Глава 3 описывает фреймворк DSLab и его асинхронное расширение, на базе которого был разработан симулятор.
- Глава 4 описывает реализацию симулятора, основные компоненты и используемые модели.
- Глава 5 содержит описание проведенных экспериментов на разработанном симуляторе и их анализ.
- Глава 6 подводит итоги работы и предлагает дальнейшие направления разработки и улучшения симулятора.

2 Обзор литературы

2.1 Вычислительная модель

Для моделирования работы кластера и алгоритмов планирования требуется модель рабочей нагрузки. Standard Workload Format(SWF)[20] широко используется в литературе и поддерживается многими симуляторами. Однако, SWF предоставляет довольно мало информации о рабочей нагрузке. Ключевая информация о единице рабочей нагрузки «Job» включает время прибытия, время выполнения, приоритет и потребность в ресурсах. Единственное предположение, которое мы можем сделать относительно рабочей нагрузки заключается в том, что «Job» потребляет все заданные ресурсы во время выполнения. Наиболее популярные трейсы рабочей нагрузки представлены в аналогичном формате (например, Google trace[22]).

Однако, существует и более сложный подход к моделированию задач на кластере, при котором время выполнения задачи рассчитывается на основе структуры рабочей нагрузки и доступных ресурсов. Основное преимущество этого подхода заключается в том, что больше деталей задачи могут быть учтены и использованы при планировании. Кроме того, появляется возможность с большей точностью рассчитать загрузку кластера и поведение его компонент. Основной недостаток – для такого моделирования довольно сложно найти данные из открытых источников, поскольку информация о детальной структуре задач обычно конфиденциальна и не публикуется.

2.2 Аналоги

Существующие симуляторы относятся к одной из двух категорий: симуляторы, созданные с нуля, и основанные на готовой платформе. Другой способ группировки симуляторов – по типу рабочей нагрузки, которую они поддерживают: симуляторы, которые работают с SWF или аналогичными форматами, и поддерживающие сложное описание рабочей нагрузки.

2.2.1 AccaSim

Характерным примером такого симулятора «с нуля» является AccaSim[11]. Он основан на дискретно-событийном моделировании и содержит предопределенные алгоритмы планирования. Он довольно прост в использовании, но не поддерживает сложные конфигурации. В отличие от AccaSim, наш симулятор сможет моделировать произвольные сценарии рабочей нагрузки, которые будут определены пользователями. Кроме того, наш симулятор является модульным, поскольку основан на DSLab. Пользователи смогут подключать модули, написанные другими исследователями в рамках платформы DSLab, и использовать их в своих экспериментах. Например, DSLab предоставляет модели сети, которые могут точно симулировать различные топологии.

2.2.2 BatSim

Симулятор Batsim основан на SimGrid[3] и позволяет разработчикам интегрировать алгоритмы на нескольких языках на основе inter-process communication. Подробное сравнение SimGrid и DSLab можно найти в документации по DSLab[6]. Ключевое ограничение BatSim – из возможных ресурсов кластера он поддерживает моделирование только «вычисления», «передачу данных» и «операции с распределенной файловой системой». Batsim предоставляет интерфейс для настройки пользовательских типов рабочей нагрузки, но они ограничены несколькими заданными паттернами и возможными последовательными комбинациями из них (см. рис. 2.1).

```
"jobs": [
  {"id":"simple", "subtime":1, "res": 4, "profile": "simple"},
],
"profiles": {
  /* workload profile definition */
  "simple": {
    "type": "parallel",
    "cpu": [5e6, 0, 0, 0],
    "com": [5e6, 0, 0, 0,
            5e6,5e6, 0, 0,
            5e6,5e6, 0, 0,
            5e6,5e6,5e6, 0]
  },
}
```

Рис. 2.1: Пример описания нагрузки в BatSim

Так же из рис. 2.1 можно увидеть, что потребность в ресурсах задачи выражается в

количестве выделенных «ресурсов» (одно число – количество серверов), и задача размещается на них целиком. Таким образом, существенно ограничивается планирование задач с разным профилем требований к ресурсам (задача упаковки). Стоит отметить, что каждый сервер имеет свою вычислительную мощность, а так же каким-то образом размещен в топологии сети, поэтому проблема размещения задач оптимальным образом на кластере не сводится к тривиальной. Необходимость учитывать топологию сети в такой модели – сложная проблема планирования. Однако, проблема упаковки задач на несколько ресурсов в такой модели не рассматривается.

Наш симулятор расширит подход BatSim для определения произвольной рабочей нагрузки, но при этом предоставит больше возможностей для управления ресурсами (процессор, память, диск), а так же предоставит еще больше возможностей для описания сценария нагрузки с использованием примитивов асинхронного программирования.

2.2.3 IRMaSim

Дальнейшим развитием BatSim стал симулятор IRMaSim[14]. В нем отмечаются недостатки BatSim в точности моделирования вычислений и предоставляются намного более точные модели процессоров и добавляются различные модели оперативной памяти. В этом симуляторе возможно точно указать тип оперативной памяти и ее скорость, и все это будет учитываться в симуляции. Так же этот симулятор поддерживает сценарии работы с задачами глубинного обучения (Deep Reinforcement Learning). Однако, ключевое ограничение остается то же – потребность задач в ресурсах указывается одним числом – количеством «ресурсов» (серверов). По аналогии с BatSim, этот симулятор фокусируется на потреблении энергии в процессе работы кластера и позволяет разрабатывать алгоритмы планирования, оптимизирующие энергопотребление.

2.2.4 Elastisim

Симулятор Elastisim[16] также основан на SimGrid и ориентирован на гибкие рабочие нагрузки (malleable workloads). Это позволяет моделировать динамические изменения количества ресурсов, выделяемых на исполнение задач. Одна из важных особенностей Elastisim – высокая точность моделирования задач глубинного обучения, для которых он

и предназначен. Кроме этого, в Elastisim поддерживается моделирование передачи данных (I/O) по сети. Авторы обращают особое внимание на то, что операции I/O могут стать узким местом, поскольку объем передаваемых данных постоянно увеличивается. Выполнение каждой задачи состоит из нескольких этапов и может быть настроено с помощью файла JSON путем объединения предварительно определенного набора операций: CPU, GPU, операций I/O и дисковых операций. Это самый продвинутый симулятор из всех рассмотренных источников. Мы будем стремиться предоставить еще больше возможностей для подробного описания выполняемых задач, однако наш симулятор будет поддерживать более распространенную модель вычислений, при которой каждой задаче выделяется фиксированная аллокация ресурсов на время выполнения.

2.3 Выводы

Рассмотренные симуляторы решают разные задачи симуляции: симуляция упаковки задач на кластер в условиях нескольких ресурсов (AccaSim), и точная симуляция вычислительного процесса (BatSim, IRMaSim, Elastisim). При этом ни в одном симуляторе эти задачи не пересекаются. Наш симулятор разработан в более общей модели, что позволяет поддерживать задачу упаковки и точное моделирование вычислений одновременно. Более того, одной из тем исследований планирования задач на кластерах является честность (в условиях, когда кластер делится между несколькими пользователями). Задача честности не рассматривалась в приведенных симуляторах, однако наш симулятор поддерживает и эту возможность.

3 Фреймворк DSLab

3.1 Архитектура DSLab

В качестве платформы для реализации симулятора выбран DSLab – модульный фреймворк для моделирования и тестирования распределенных систем. Модули можно разделить на 3 группы: базовые, универсальные и специализированные (см рисунок 3.1). За основу взят подход дискретно-событийного моделирования, при котором компоненты симуляции обмениваются событиями через базовый низкоуровневый модуль `dslab-core`. Соответственно, пользовательская логика каждого компонента состоит из обработки входящих событий

от других компонент и отправки новых. Универсальные модули могут использоваться исследователями соответствующих предметных областей, например, новый симулятор использует готовые модули вычислений, сети и диска (подробнее об этом в главе 4.2).

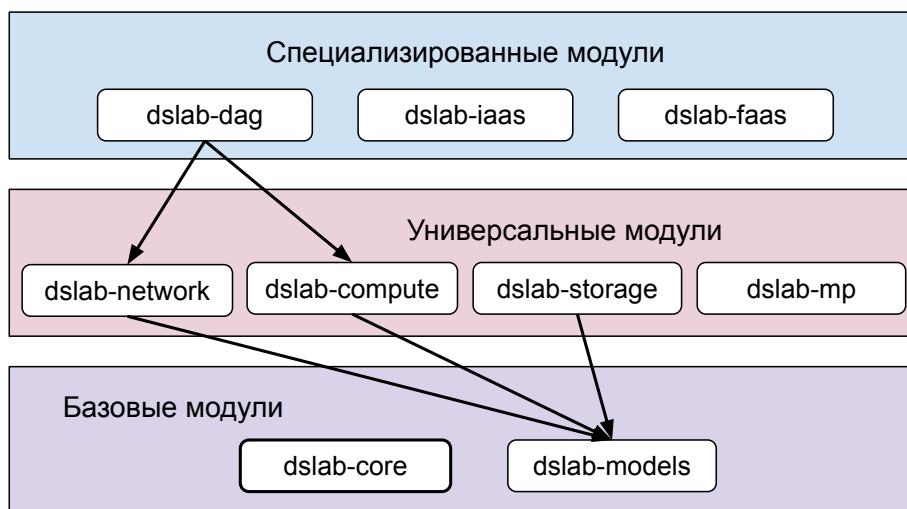


Рис. 3.1: Архитектура DSLab

Описание архитектуры основано на официальной документации [4]. Реализация всех модулей опубликована в GitHub-репозитории[9].

3.2 Асинхронное управление событиями

В предыдущей работе была добавлена возможность управлять событиями DSLab асинхронно[15], что существенно расширило возможности написания компонент и сделало возможным ключевую особенность нового симулятора – описание профиля нагрузки задач через асинхронные примитивы языка Rust (подробнее об этом в главе 4.3). Ключевая особенность такого расширения – возможность спрятать от пользователя явную работу с событиями, предоставив ему высокоуровневые асинхронные функции, в реализации которых зашито асинхронное ожидание внутренних событий. Например, при работе с записью на диск можно предоставить пользователю следующую асинхронную функцию:

В этом примере используется детализированное ожидание события по ключу. Исполнение симуляции прерывается до того момента, пока компонент (в данном случае – вычислительный сервер, на котором запущена задача) не получит событие от собственного диска о том, что запись завершилась (либо успешно, либо с ошибкой – для альтернативного ожидания используется макрос `select!` из библиотеки `futures`[18]), после этого ядро симуляции

```

pub async fn write_data(&self, size: u64) -> Result<(), String> {
    let req_id = self.disk.write(size, /*requester=*/self.ctx.id());

    select! {
        _ = self.ctx.recv_event_by_key::<DataWriteCompleted>(req_id).fuse() => {
            Result::Ok(())
        }
        failed = self.ctx.recv_event_by_key::<DataWriteFailed>(req_id).fuse() => {
            Result::Err(failed.data.error)
        }
    }
}

```

Рис. 3.2: Пример асинхронной функции записи данных на диск.

продолжит исполнение и результат вернется пользователю.

Таким образом, пользователю не нужно будет самостоятельно следить за событиями компонент и можно будет разделить инструкции выполнения задач от конкретных компонент, предоставив лишь «сценарий работы» в виде асинхронных функций.

Точно таким же образом реализованы и другие примитивы: вычисления и работа с сетью.

4 Реализация симулятора

4.1 Архитектура симулятора

По аналогии с самим фреймворком DSLab симулятор модульный, его архитектура представлена на рисунке 4.1.

Симулятор может работать в нескольких режимах, принимая на вход несколько видов входных данных. Это может быть как готовый трейс нагрузки из открытых источников (например, Google Trace), так и пользовательский ввод задач, описанный в виде YAML-файла (подробнее в разделе 4.5.1).

Собирая информацию с компонентов **Proxy** и **Cluster** можно получить статистику о работе кластера и алгоритмов планирования. Например, насколько загружен кластер, насколько хорошо утилизируются его ресурсы. Такие выходные данные могут быть использованы для анализа работы алгоритмов и сравнения их эффективности. За это отвечает отдельный компонент – **Monitoring**(подробнее в разделе 4.6).

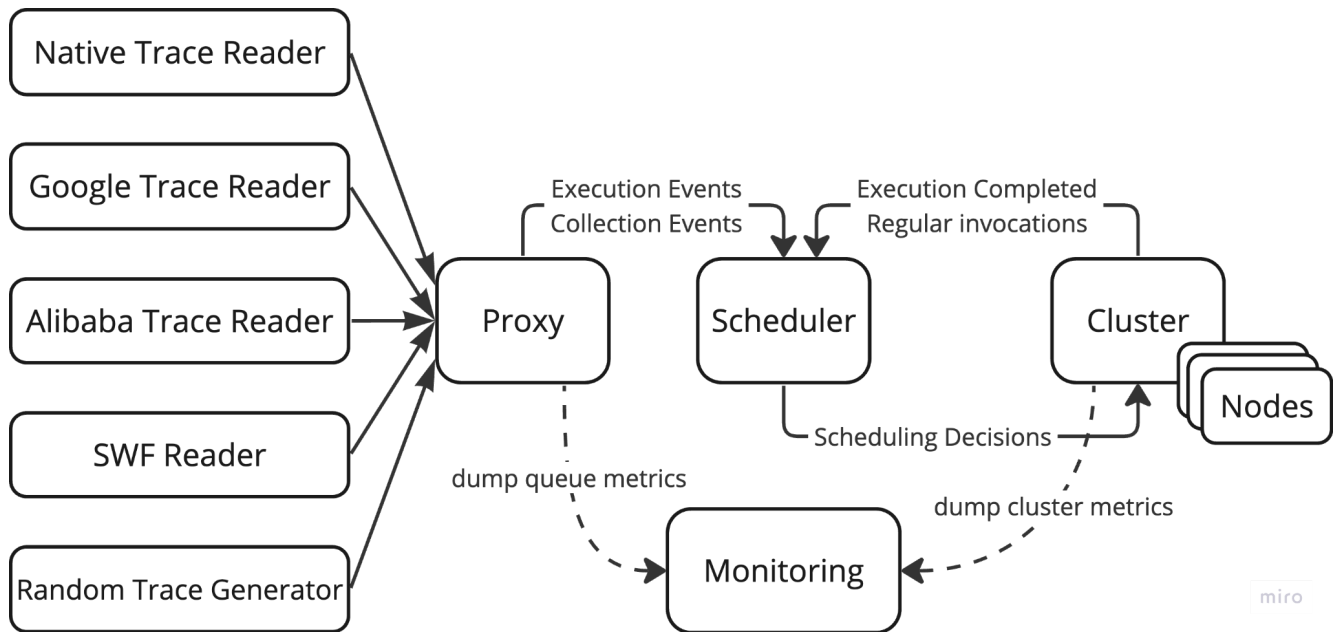


Рис. 4.1: Архитектура симулятора

4.2 Модель кластера

Вычислительный кластер – это набор отдельных серверов, которые могут быть связаны между собой моделируемой сетью (подробнее в разделе 4.2.2). В симуляторе используются готовые модули DSLab, из которых собирается каждый сервер (см. таблицу 4.1).

Название модуля	Используемый модуль	Обязательный
compute	dslab_compute::multicore::Compute	да
network	dslab_network::Network	нет
disk	dslab_storage::disk::Disk	нет

Таблица 4.1: Модули каждого сервера в кластере

Все эти модули конфигурируются при создании кластера. В случае, если хост запланировано выходит из строя или начинает работу после начала симуляции, его модули все равно создаются сразу как компоненты симуляции DSLab. Это необходимо, поскольку каждый сервер и каждый модуль сервера является обработчиком событий в симуляции и должен быть соответствующим образом зарегистрирован.

4.2.1 Модуль вычислений

Для вычислений используется модуль `Compute` с фиксированными слотами – «аллокациями», которые состоят из некоторого количества ядер процессора и объема выделенной оперативной памяти. В текущей используемой модели аллокации не могут пересекаться, однако могут

быть заняты не полностью, что скажется на утилизации кластера. Т.е. на данном этапе планировщик не имеет возможность выделить больше ресурсов чем есть на сервере в надежде, что запущенные задачи используют меньше, чем просят.

Поскольку некоторые трейсы нагрузки не содержат точную информацию о конфигурации кластера, единицы измерения «ядер процессора» и размера оперативной памяти могут быть относительны. Например, в Google Trace[22] размер оперативной памяти и количества ядер процессора указаны как доля от максимального значения в кластере. Аналогичным образом нормированы и требования задач к ресурсам. Такой политики компании придерживаются с целью не раскрывать реальные конфигурации своих серверов и требования к задачам. Поскольку модуль **Compute** принимает на вход целые числа (количество ядер и объем памяти), то при создании кластера в симуляции данные в трейсе доли умножаются на большие константы таким образом, чтобы получились целые числа и погрешность была небольшой (очевидно, что при этом теряются единицы измерения). Все константы задаются пользователем при конфигурации симуляции.

Подробнее описание модуля **Compute** и примеры использования можно посмотреть в официальной документации DSLab[7].

4.2.2 Модуль сети

Модуль сети **Network** содержит несколько моделей на выбор, которыми можно воспользоваться в симуляции:

1. **ConstantBandwidthNetworkModel** – самая простая модель, которая предполагает, что все сервера в кластере соединены сетью с постоянной пропускной способностью. Нагрузка сети одними компонентами никак не влияет на другие. Подходит для простой и быстрой симуляции сетевого взаимодействия.
2. **SharedBandwidthNetworkModel** – модель, предполагающая наличие одного сетевого канала, общего для всех участников сети. Может быть достаточно точной в ситуации, когда все компоненты сети подключены к одному центральному коммутатору и пропускная способность сети много превосходит пропускную способность центрального коммутатора, являющегося узким местом. Однако, в реальности для такого случая можно применить модель **Star-Topology** и получить существенно большую точность с

небольшим штрафом в производительности. Поэтому и эту модель используют с целью более быстрого моделирования сети в ущерб точности.

3. **TopologyAwareNetworkModel** – гибко настраиваемая и достаточно точная модель сети с возможностью задания топологии. В симуляторе пользователь может самостоятельно настроить и передать симулятору сеть, которая наиболее точно отражает желаемую ситуацию. Однако, по умолчанию предоставляется вариант **Fat-Tree Topology**, который на практике чаще всего используется в датацентрах и кластерах, поскольку является эффективным и отказоустойчивым[17]. Схематично топология сети «Fat-Tree» представлена на рисунке 4.2. Серверы объединяются в стойки (L1 Switch), которые связаны между собой коммутаторами (L2 Switch). Все пропускные способности и задержки между узлами, а также количество серверов в стойках и количество верхнеуровневых коммутаторов задаются пользователем при конфигурации симуляции.

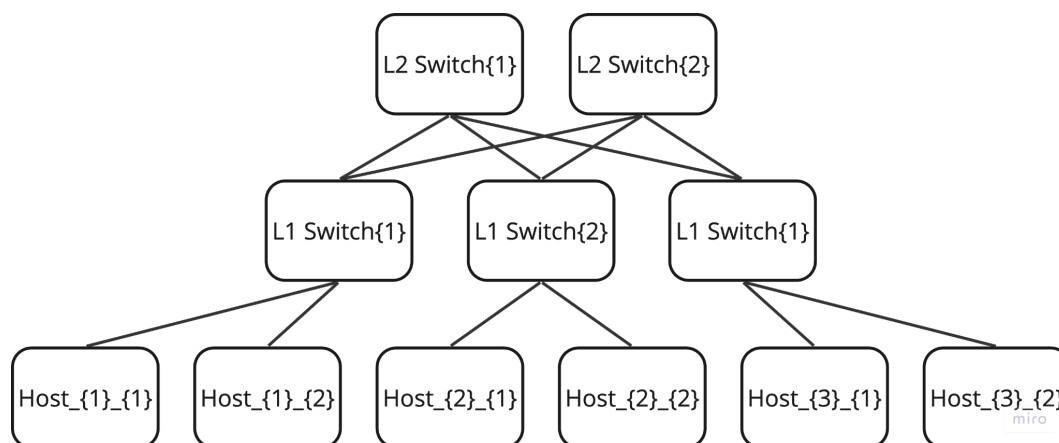


Рис. 4.2: Пример топологии сети «Fat-Tree» на 6 серверов по 2 сервера в стойке

Подробнее описание модуля **Network** и примеры использования можно посмотреть в официальной документации DSLab[8].

4.2.3 Модуль хранилища

Модуль диска используется для симуляции операция чтения-записи в хранилище. Модель предоставляет возможность гибко настроить деградацию производительности диска в зависимости от количества параллельных операций (особенно актуально для HDD операций). Также модель предоставляет возможность реализовать пользовательский планировщик

дисковых операций, но по умолчанию используется классический «FIFO»-планировщик (First-In-First-Out).

Подробнее описание модуля `Storage` и примеры использования можно посмотреть в официальной документации DSLab[10].

4.3 Вычислительная модель

4.3.1 Деление на `Collection` и `Execution`

Поскольку симулятор может поддерживать несколько разных вычислительных моделей, для описания нагрузки был выбран обобщенный вариант: пользователь описывает задачи (`Execution`), которые непосредственно запускаются на кластере и потребляют ресурсы, и эти задачи могут быть связаны между собой заданием или коллекцией (`Collection`). Термин `Collection` происходит из аналогичного определения задания в последней версии Google Trace[22]. `Execution` же является минимальной единицей планирования и может из себя представлять как простые вычисления, так и сложную программу, которой для запуска нужно одновременно выделить несколько серверов. Структуры данных представлены в таблицах 4.2 и 4.3.

Описание требуемых ресурсов задачи состоит из требования к количеству хостов, а так же опционально к количеству ядер процессора и памяти на хостах. Таким образом, сохраняется возможность поддержать модель планирования аналогичной `BatSim`, `IRMaSim` и `Elastisim`, но и не упускается возможность для моделирования упаковки задач на кластерах (при указывании количества хостов больше одного, есть возможность указать требования к ресурсам каждого хоста отдельно или всем сразу).

4.3.2 `ExecutionProfile`

Задачи, которые непосредственно выполняются на кластере, представляются в виде объектов, для которых реализован интерфейс `ExecutionProfile`:

Метод `run` может быть реализован самим пользователем, но так же могут использоваться заготовки для простых сценариев работы. Такая задача получает в свое «владение» от планировщика набор выделенных аллокаций на узлах кластера и выполняет некоторую

Поле	Тип	Описание
id	u64	Уникальный идентификатор задачи на всю симуляцию
collection_id	Option<u64>	Индекс задания или коллекции задач
execution_index	Option<u64>	Индекс задачи внутри задания/коллекции
time	f64	Время, когда задача становится доступной для планирования
schedule_after	Option<f64>	Время, когда задача становится доступной для исполнения на кластере. Поле введено для поддержки совместимости с трейсом нагрузки из Google.
resources	ResourceRequirements	Структура требуемых ресурсов для исполнения задачи
profile	Rc<dyn ExecutionProfile>	Профиль нагрузки (см секцию 4.3.2)
wall_time_limit	Option<f64>	Максимальное время на выполнение задачи, после которой она автоматически будет отменена
priority	Option<u64>	Приоритет задачи
start_after	Option<Vec<u64>>	Список индексов задач внутри коллекции, после которых можно начать выполнение этой задачи (должен быть указан collection_id)

Таблица 4.2: Описание структуры данных `ExecutionRequest`

Поле	Тип	Описание
id	u64	Уникальный идентификатор задания/коллекции на всю симуляцию.
time	f64	Время, когда задание/коллекция становится известной планировщику.
user	Option<String>	Пользователь, который запускает эту коллекцию/задание. Используется для тестирования алгоритмов честного разделения кластера.
priority	Option<u64>	Приоритет задания/коллекции. Если у задачи указан собственный приоритет, то предполагается, что он переопределяет данный приоритет.
start_after_collection_ids	Option<Vec<u64>>	Список уникальных идентификаторов заданий/коллекций, после которых можно начать выполнение задач этой коллекции.

Таблица 4.3: Описание структуры данных `CollectionRequest`

работу. Сама работа описывается в терминах асинхронного программирования на Rust и использует предоставленный интерфейс взаимодействия с кластером:

Поскольку симулятор поддерживает несколько моделей вычислений, то интерфейс `ExecutionProfile` принимает на вход несколько процессов (гарантированно хотя бы один).

```
#[async_trait(?Send)]
pub trait ExecutionProfile {
    async fn run(self: Rc<Self>, processes: &Vec<HostProcessInstance>);
}
```

Рис. 4.3: Интерфейс ExecutionProfile

```
async fn sleep(&self, time: f64);
async fn run_compute_work(&self, compute_work: f64);
async fn transfer_data_process(&self, size: f64, dst_process: ProcessId);
async fn transfer_data_component(&self, size: f64, dst_component: Id);
async fn write_data(&self, size: u64);
async fn read_data(&self, size: u64);
```

Рис. 4.4: Интерфейс взаимодействия задачи с кластером

Для простых ситуаций (например, симуляции нагрузки по Google Trace) используются тривиальные имплементации ExecutionProfile, которые рассчитаны на выполнение на одном сервере.

Для более сложного моделирования работы задачи можно использовать весь спектр асинхронных инструментов из библиотеки futures[18]. Например, паттерн работы «Master-Workers» можно реализовать как показано на примере 4.5.

```
pub struct MasterWorkersProfile {
    pub master_compute_work: f64,
    pub worker_compute_work: f64,
    pub data_transfer_size: f64,
}

#[async_trait(?Send)]
impl ExecutionProfile for MasterWorkersProfile {
    async fn run(self: Rc<Self>, processes: &Vec<HostProcessInstance>) {
        let master = &processes[0];
        let workers = &processes[1..];

        futures::future::join_all(workers.iter().map(|p| async {
            master.transfer_data_process(self.data_transfer_size, p.id).await;
            p.run_compute(self.worker_compute_work).await;
        })))
        .await;

        master.run_compute(self.master_compute_work).await;
    }
}
```

Рис. 4.5: Пример реализации интерфейса ExecutionProfile

4.3.3 Описание нагрузки через входной файл

Пользователь может самостоятельно реализовывать свои паттерны нагрузки на кластер, тем самым получая высокую степень точности симуляции. Для использования этих профилей из необходимо зарегистрировать в симуляции перед ее началом, и тогда можно будет подавать на вход задачи, использующие этот профиль. Например, после регистрации профиля `MasterWorkersProfile` можно описать задачу (`ExecutionRequest`) следующим образом с помощью YAML-файла:

```
- submit_time: 10.0
  resources:
    nodes_count: 1
    cpu_per_node: 2
    memory_per_node: 4
  profile:
    type: master-workers
    args:
      master_compute_work: 100.
      worker_compute_work: 2000.
      data_transfer_size: 100.
```

Рис. 4.6: Пример описания задачи с использованием профиля `MasterWorkersProfile`

Во время симуляции будет создан объект задачи с переданными параметрами (из секции `args`) и передан в планировщик для размещения на кластере. Как только планировщик найдет для задачи место, будет выполнен метод `run` из реализации `MasterWorkersProfile` (см. рисунок 4.5).

4.3.4 Декларативное создание профилей нагрузки через файл

Помимо реализации интерфейса `ExecutionProfile`, есть и другой способ описания профилей нагрузки – через YAML-файл. Такой способ описания позволяет быстрее создавать новые профили нагрузки без необходимости писать код, но с меньшими функциональными возможностями. Пользователю предоставляется два основных комбинатора профилей: параллельный и последовательный, которые могут быть вложены друг в друга. Пример описания профиля нагрузки представлен на рисунке 4.7.

Можно видеть, что комбинаторы могут быть вложены, позволяя создавать сложные сценарии. По умолчанию предоставляются базовые профили, описанные в таблице 4.4. Эти профили покрывают функциональность профилей из `BatSim`[2] и позволяют создавать

```

compute-simple:
  type: compute-homogenous
  args:
    compute_work: 1000.0
compute-hard:
  type: compute-homogenous
  args:
    compute_work: 2000.0
net-all-to-all-simple:
  type: communication-homogenous
  args:
    size: 500.0
disk-simple-read:
  type: disk-read
  args:
    size: 1000
disk-simple-write-part:
  type: disk-write
  args:
    size: 600

complex_execution_profile:
  type: sequence
  args:
    repeat: 2
  profiles:
    - disk-simple-read
    - compute-simple
    - parallel
  args:
    profiles:
      - compute-hard
      - net-all-to-all-simple
    - disk-simple-write-part
    - parallel
  args:
    profiles:
      - compute-simple
      - net-all-to-all-simple
    - disk-simple-write-part

```

(a) Пример описания базовых профилей (b) Пример описания профиля через комбинаторы

Рис. 4.7: Пример описания профилей нагрузки через YAML-файл

разнообразные сценарии нагрузки на кластер, кроме этого с помощью комбинаторов можно сделать более сложные вложенные профили.

4.4 Модель планирования заданий

Симулятор поддерживает несколько основных моделей планирования:

1. Планировщик принимает решение о планировании на основе всей информации (выбирает подходящую задачу и подходящий хост). В таком случае планировщик может вызываться периодически и по событиям (например, по получению новых задач и завершению прошлых). В такой модели у планировщика есть набор задач, которые в данный момент наиболее приоритетны (обычно это некоторая очередь), и планировщик выбирает под эти задачи наиболее подходящие серверы (или части серверов) кластера. Такая модель используется в одном из наиболее популярных планировщиков для кластеров **Slurm**^[19](Simple Linux Utility for Resource Management). В этой модели под задание (job) с помощью отдельного плагина **Node Selection** выбирается набор серверов, на которые впоследствии распределяются задачи (tasks).
2. Вычислительные серверы периодически сообщают планировщику о своих свободных

Профиль	Аргументы	Описание
compute-homogenous	compute_work: f64	Выполнить <code>compute_work</code> работы на всех переданных процессах.
compute-matrix	matrix: Vec<Vec<f64>>	Выполнить вычисления на процессах согласно матрице <code>matrix</code> , например: [10, 20] – первый процесс выполняет работу 10, а второй – 20.
communication-homogenous	size: f64	Передать <code>size</code> данных между всеми переданными процессами.
communication-matrix	matrix: Vec<Vec<f64>>	Передать данные между процессами согласно матрице <code>matrix</code> , например: [0, 20, 10, 0] – первый процесс передает второму данные размера 20, а второй первому – 10. Аналогично с определением матрицы передачи данных из BatSim[2].
communication-src-dst	src: Vec<usize> dst: Vec<usize> size: f64	Передать <code>size</code> данных от процессов с индексами из <code>src</code> к процессам с индексами из <code>dst</code> .
communication-external	processes: Vec<usize> input_size: f64 output_size: f64 component_name: String	Передать данные между процессами и другими компонентами сети, например, между процессами и внешним хранилищем. Для этого внешнее хранилище должно быть предварительно зарегистрировано в сети и настроено.
disk-read	processes: Vec<usize> size: f64	Прочитать <code>size</code> данных с диска у данных процессов. Если процессы не указаны, чтение просиходит у всех.
disk-write	processes: Vec<usize> size: f64	Записать <code>size</code> данных на диск у данных процессов. Если процессы не указаны, запись на диск происходит у всех.

Таблица 4.4: Описание предоставляемых по умолчанию профилей нагрузки

ресурсах, и планировщик принимает решение о подходящих задачах (которые могут аналогичным образом браться из очереди) конкретно под сервер, который сделал запрос. Так же серверы информируют планировщик об изменении свободных ресурсов в связи с завершением текущих задач. Такая модель используется в планировщике кластера Yт в Яндексе[23].

Поддерживая обе эти модели получается, что симулятор способен воспроизводить работу разных планировщиков, что делает его более универсальным. Однако, из-за этого размывается четкое разделение рабочей нагрузки на задание (job) и задачи (tasks). В такой ситуации

получается, что `Execution` (см. раздел 4.3.1) может выступать как простейшая задача продолжительностью несколько минут (из модели планировщика `YT`, `Google` и др.), так и сложная много-серверная программа, внутри которой есть своя собственная логика разделения ресурсов (из модели планировщика `Slurm`). В последнем случае дополнительную логику «планирования» и разделения большого объема выделенных ресурсов реализует как раз сам пользователь внутри `ExecutionProfile`, эта деталь отличает симулятор от планировщика `Slurm`, где такая логика реализована внутри главного планировщика.

Реализуя свой планировщик пользователь может реализовать интерфейс планировщика `Scheduler` (см. рисунок 4.8).

```
pub trait Scheduler {  
    fn on_host_added(&mut self, ctx: &Context, host: HostConfig);  
    fn on_execution_request(&mut self, ctx: &Context, req: ExecutionRequest);  
    fn on_collection_request(&mut self, ctx: &Context, req: CollectionRequest);  
    fn on_execution_finished(&mut self, ctx: &Context, execution_id: u64);  
    fn on_host_notification(&mut self, ctx: &Context, host_id: Id, resources:  
        ResourcesPack);  
}
```

Рис. 4.8: Интерфейс планировщика `Scheduler`

Для более точного управления моделью планировщика можно запустить симуляцию, передав ей компонент отдельный симуляции `DSLlab`, который будет получать сырые события от компонент `Proxy` и `Cluster`. Тогда у пользователя будет еще больше контроля над моделью планирования: компонент сможет отправлять себе произвольные события, использовать асинхронные функции из асинхронного расширения ядра `DSLlab` и т.д.

С помощью контекста (который передается в каждый метод интерфейса) планировщик управляет симуляцией: иметь доступ к генератору псевдослучайных чисел, ставить на исполнение задачи, выбрав для них сервера, а так же частично отменять их. Полная отмена в данный момент не полностью реализована, например невозможно отменить операцию передачи данных по сети, это связано с внутренними ограничениями модуля `dslab_network`, которые в будущем могут быть доработаны.

4.5 Входные данные симуляции

4.5.1 Трейс нагрузки

В качестве входных данных симуляции выступает трейс нагрузки на кластер. Трейс представляет собой набор задач, которые поступают на кластер в течение симуляции. Как описывалось в разделе 4.3.1, они могут быть связаны некоторым общим заданием/коллекцией. В данный момент симулятор поддерживает несколько видов входной нагрузки:

- **Native** – трейса нагрузки в формате YAML, описанного в секции 4.3. Состоит из нескольких файлов: файл с описанием задач (`ExecutionRequest`-ы), файл с описанием коллекций (`CollectionRequest`-ы) и файл с описанием профилей нагрузки (`ExecutionProfile`).
- **Google** – трейс нагрузки кластера Google[22]. Версия 2019 года представлена в виде таблицы `BigQuery`, из которой достаточно сложно скачать большой объем данных (из-за ограниченных квот), однако версию 2011 года можно скачать в виде `CSV`-файла. Подробное описание и сравнение этих трейсов представлено в статье `Borg: Next Generation`[21] от Google. Для простоты в симуляторе предполагается, что все задачи потребляют все выделенные ресурсы на протяжении всего времени выполнения (заполняются тривиальными заглушками `ExecutionProfile`), однако в трейсе 2019 года присутствует подробная информация о профиле загрузки задачи, которую можно использовать для более точного моделирования в последующей работе. Сам трейс представлен в виде событий, поэтому его инкрементальное прочтение сделать сложно (для того, чтобы выяснить время выполнения задачи, необходимо прочитать трейс до события окончания задачи), из-за этого с трейсом удобнее работать, преобразовав его в формат **Native**. Поскольку данные о требованиях к ресурсам задач представлены в нормированном виде (числами с плавающей точкой от 0 до 1, где 1 – максимальное возможное значение ресурса среди всех доступных серверов), то при чтении этого трейса необходимо указать константу, на которую умножаются эти значения, чтобы получить целочисленные значения CPU и памяти.
- **Alibaba** – Трейс нагрузки кластера Alibaba 2018 года[1]. Состоит из двух файлов: файл `batch_tasks.csv` описывает зависимости между выполняемыми задачами (`instances`), информация о которых содержится в `batch_instances.csv`. Информация о задачах по

аналогии с Google Trace переводится в тривиальные профили нагрузки. Однако, этот трейс можно читать инкрементально, что упрощает работу с ним.

- **Standard Workload Format Reader** – Трейс нагрузки в формате SWF[20]. Транслируется в симулятор по аналогии с Alibaba Trace, т.к. может быть в исходном виде прочитан инкрементально.
- **Random Trace Generator**. Детерминированный генератор синтетической нагрузки на кластер. С помощью встроенного в симуляцию DSLab генератора случайных чисел создает заданное количество задач с заданными требованиями к ресурсам. Время выполнение (либо необходимая процессорная работа) генерируется из нормального распределения с заданными параметрами среднего и дисперсии (см. пример в секции 5.2).

Все перечисленные генераторы нагрузки реализуют интерфейс `WorkloadGenerator` (см. 4.9). Поскольку количество задач может быть очень большим (входные файлы могут иметь размеры десятки гигабайт) и не поместится в оперативную память сразу, то генераторы нагрузки должны уметь читать данные инкрементально, выдавая задачи для планировщика пачками не больше указанного лимита. Предполагается, что задачи в трейсе упорядочены по времени поступления на кластер.

```
pub trait WorkloadGenerator {  
    fn get_workload(  
        &mut self, ctx: &SimulationContext, limit: Option<u64>,  
    ) -> Vec<ExecutionRequest>;  
  
    fn get_collections(&self, ctx: &SimulationContext) -> Vec<CollectionRequest>;  
}
```

Рис. 4.9: Интерфейс `WorkloadGenerator`

Пользователь может реализовать этот интерфейс самостоятельно и тогда у него появится возможность использовать свой собственный формат трейса нагрузки. Как альтернативный вариант, можно конвертировать имеющийся трейс в один из поддерживаемых форматов.

4.5.2 Конфигурация кластера и сети

Существует несколько способов задать конфигурацию серверов в кластере:

1. Через конфигурационный файл в формате YAML. Пример конфигурации представлен на рисунке 4.10. Есть возможность задать группы серверов с одинаковой конфигурацией, а затем получать статистику утилизации ресурсов в серверах по группам (подробнее см. раздел 4.6). Составные части каждого сервера описаны в разделе 4.2. Через конфигурацию так же можно указать вид сети, используемой в кластере, и ее параметры (например, указать распределение хостов по стойкам при использовании топологии **Fat-Tree**) (описана в разделе 4.2.2).
2. Через чтение конфигурации кластера из **Google-trace**.
3. Через чтение конфигурации кластера из **Alibaba-trace**.

Стоит отметить, что использование трейса нагрузки из какого-либо источника (**Google trace** или **Alibaba trace**) не обязывает использовать конфигурацию кластера из этого же источника и наоборот. Однако, учитывая, что в обоих трейсах информация о ресурсах является нормированной (в целях скрыть данные о реальных мощностях кластера и объеме выполняемых задач), использовать их как-то иначе представляется практически невозможным.

4.6 Выходные данные симуляции

Выходными данными симуляции являются файлы с метриками загруженности кластера и очереди задач. В процессе работы отдельный компонент **Monitoring** собирает данные о мгновенной загрузке серверов и размере очереди, и агрегирует их, считая среднее значение на заданных временных промежутках. Пользователь может настроить интервал сжатия данных в конфигурационном файле симуляции (см. конфигурацию 4.10). В разделе честности распределение ресурсов по умолчанию собираются метрики использования доминантной доли ресурса различными пользователями[12]. Подробнее это рассмотрено в примере использования симулятора 5.2.

4.7 Конфигурация симуляции

Вся настройка параметров симуляции происходит через конфигурационный файл в формате YAML. Пример конфигурации представлен на рисунке 4.10.

```

workload:
  - type: Native
    path: /path/to/trace
    options:
      profiles_path: /path/to/profiles

hosts:
  - count: 5
    cpus: 20
    memory: 20
    name_prefix: c
  - count: 5
    cpus: 5
    memory: 60
    name_prefix: m

monitoring:
  host_load_compression_time_interval: 100
  scheduler_queue_compression_time_interval: 100
  display_host_load: false
  collect_user_queues: true

scheduler:
  hosts_invoke_interval: 10

```

Рис. 4.10: Пример конфигурации симуляции

В такой симуляции входная нагрузка будет в формате **Native** с указанными дополнительными профилями нагрузки. Используются два типа серверов: 5 серверов с 20 ядрами и 20 Гб памяти и 5 серверов с 5 ядрами и 60 Гб памяти. Сбор статистики загрузки кластера ведется только по группам и целиком на кластер, а так же собираются метрики честности планирования (по пользователям). Метрики загрузки и размера очереди сжимаются на отрезках времени в 100 единиц и берется среднее на каждом отрезке. Каждый хост сообщает о свободных ресурсах планировщику каждые 10 единиц времени (единицы измерения могут быть любыми, но по умолчанию подразумеваются секунды).

5 Эксперименты и тестирование

5.1 Производительность

В этом разделе будет проведен замер производительности при симуляции Alibaba Trace на 8 дней (входной файл 120GB). TODO

5.2 Честность

В качестве тестирования симулятора были проведены эксперименты на известный алгоритм честного распределения кластера между несколькими пользователями. Были реализованы алгоритмы DRF[12] и Tetris[13].

Алгоритм DRF (Dominant Resource Fairness) основан на понятии доминантного ресурса пользователя. Если пользователь потребляет $1/10$ всей памяти в кластере, $1/15$ всех ядер процессора, то для него доминантной долей будет считаться число $1/10$ (в ситуации, когда распределяется два доступных ресурса – CPU и RAM). Смысл алгоритма состоит в том, чтобы каждый пользователь получал равную доминантную долю ресурсов (в случае, когда квоты пользователей равны, иначе доля считается от выделенной на кластере квоты). Поэтому из всех задач, которые готовы к исполнению планировщик в каждый момент времени выбирает самого обделенного пользователя и запускает его задачу.

Такой подход позволяет обеспечить честное распределение ресурсов, однако упаковка задач на серверах может быть неоптимальной. Для борьбы с этим был предложен алгоритм Tetris[13]. На вход принимается произвольная функция честности (в нашем случае это DRF), а также «коэффициент честности» $f \in [0, 1]$. В каждый момент принятия решения, планировщик сортирует задачи по приоритету их размещения на кластере с точки зрения честности, а затем выбирает наилучшую задачу с точки зрения упаковки среди $(1 - f)$ доли очереди. В качестве эвристической функции упаковки Tetris использует скалярное произведение вектора ресурсов задачи на вектор свободных ресурсов сервера. Чем оно больше, тем лучше задача упаковывается на сервер. В таком виде приоритет явно отдается более тяжелым задачам, чтобы этого избежать, можно разделить результат на нормы векторов ресурсов.

Таким образом, конфигурируя алгоритм через параметр f , можно добиться баланса между честностью и плотностью упаковки задач. При $f = 0$ на каждом шаге планировщик выбирает наиболее подходящую задачу с точки зрения упаковки (т.к. выбирает среди всей доступной очереди), а при $f = 1$ выбирает наиболее обделенного пользователя (т.к. может взять только первую задачу из очереди). Стоит отметить, что при любых значениях $f < 1$ честность распределения ресурсов не гарантируется и существует возможность развития событий, при котором один из пользователей оказывается полностью обделен ресурсами на кластере, т.к.

его задачи всегда подходят хуже других. На практике (при значении f достаточно близком к 1) такое случается крайне редко из-за очень большого количества задач с различными требованиями.

Для такой симуляции был выбран кластер с распределением ресурсов по серверам, указанными на графике 5.1.

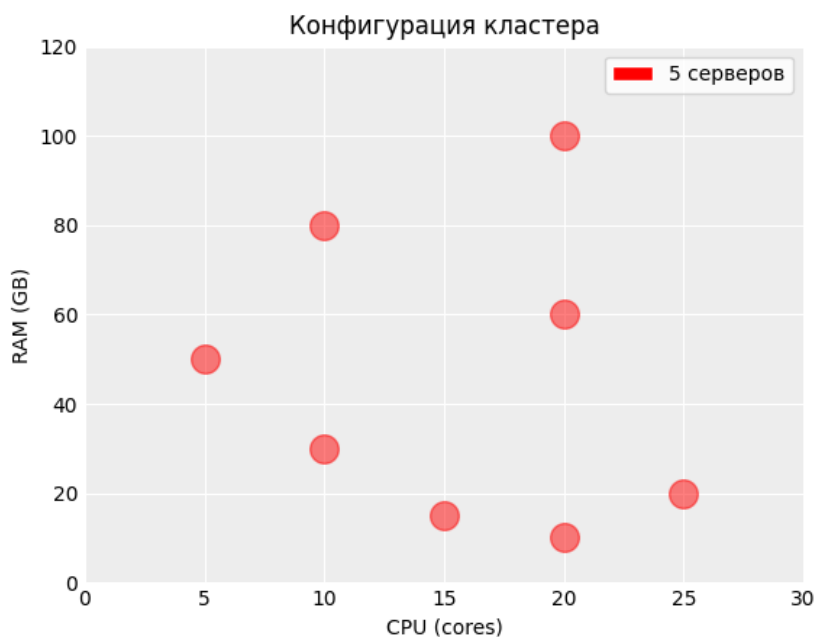


Рис. 5.1: Распределение ресурсов на серверах кластера

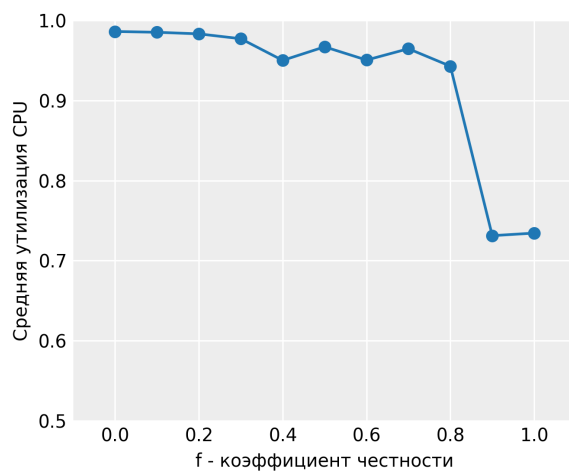
В качестве синтетической нагрузки были сгенерированы 10 пользователей с равной долей доступа к кластеру с запросами на задачи, представленными в таблице 5.1. Каждый пользователь непрерывно поставлял задачи с требованиями, указанными в таблице. Продолжительность задачи генерировалась из нормального распределения с параметрами среднего (`duration mean`) и дисперсии (`duration dev`) для каждого пользователя.

При симуляции с разными параметрами честности производились замеры средней утилизации кластера от момента полной загрузки кластера до момента, когда у какого-либо пользователя заканчивались задачи. Результаты представлены на графиках 5.2.

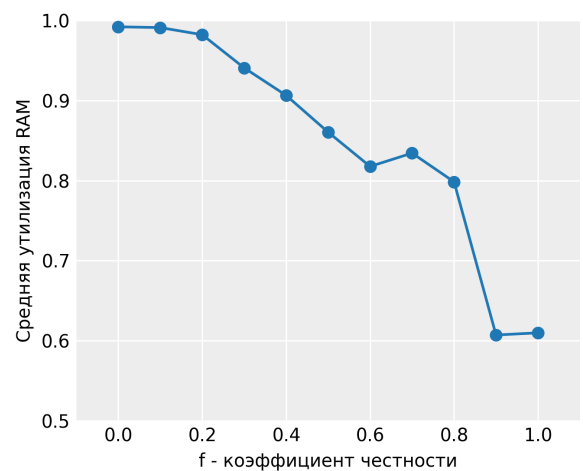
Из графиков видно, что качественный скачок в утилизации происходит при значении параметра честности $f = 0.8$. Это можно объяснить тем, что ровно при этом значении параметра, алгоритм **Tetris** получает возможность выбирать из двух задач при планировании (т.к. всего пользователей 10). Как только появляется такая возможность, качество упаковки задач на кластере резко возрастает.

Пользователь	CPU	RAM	Количество задач	Duration Mean	Duration Dev
user0	1	5	1500	250	40
user1	1	10	800	400	40
user2	1	15	800	300	30
user3	2	5	1300	250	30
user4	2	6	1200	250	30
user5	2	7	800	300	40
user6	5	4	1000	180	30
user7	5	5	700	160	10
user8	7	5	200	800	200
user9	10	2	400	220	30

Таблица 5.1: Описание синтетической нагрузки на кластер



(а) Утилизация CPU в зависимости от f

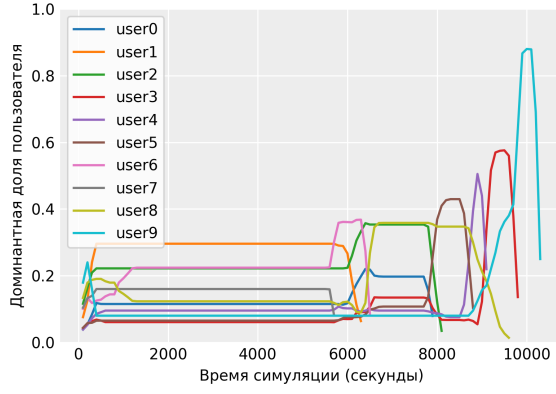


(б) Утилизация памяти в зависимости от f

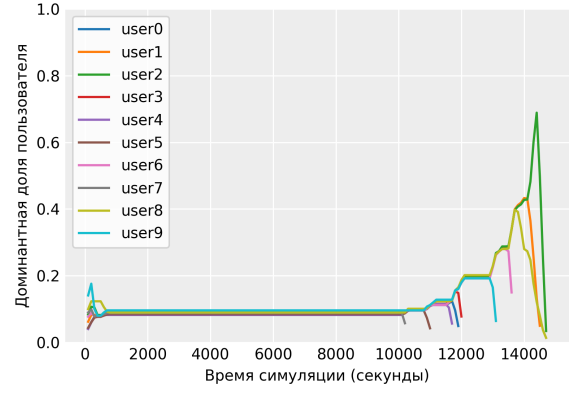
Рис. 5.2: Утилизация кластера в зависимости от параметра честности

Чтобы сравнить честность распределения, посмотрим на графики 5.3 доминантных долей пользователей в течение времени симуляции. Можно отметить, что при коэффициенте $f = 0.6$ (график 5.3d) честность страдает не сильно, особенно пока у всех пользователей есть задачи, а при $f = 0.5$ (график 5.3c) уже можно видеть, что два пользователя существенно обделяются (получают долю в два раза меньше, чем получает большинство), еще два других пользователя обделяются не так сильно. На обоих графиках видно, что самым обделенным пользователем является **user9** – это можно объяснить тем, что у него профиль нагрузки задач отличается больше всех от других: он потребляет больше всех CPU и меньше всех RAM, поэтому его задачи хуже всего подходят в освободившиеся слоты от других пользователей.

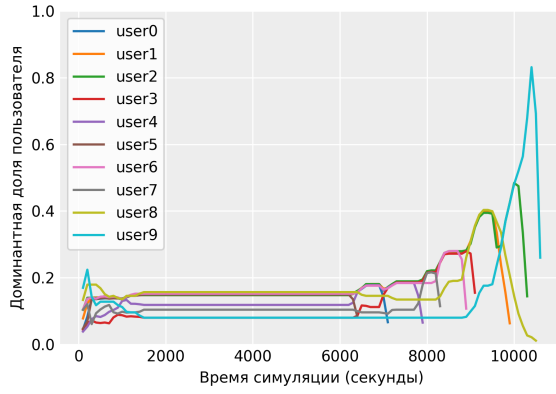
Ожидаемо можно отметить, что при $f = 1$ доли у всех пользователей абсолютно одинаковые. В случае, когда у какого-то пользователя заканчиваются задачи, освободившиеся ресурсы



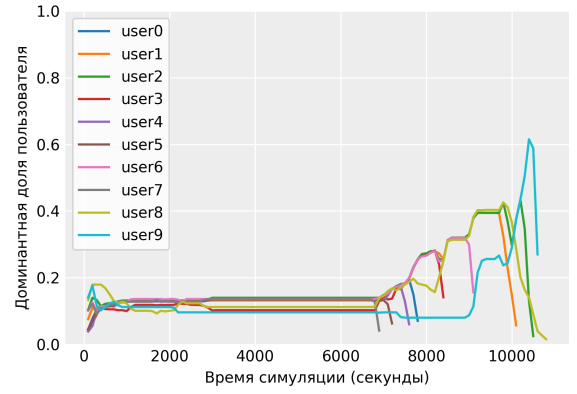
(a) $f = 0.0$



(b) $f = 1.0$



(c) $f = 0.5$



(d) $f = 0.6$

Рис. 5.3: Распределение доминантных долей использования кластера пользователями при различных коэффициентах честности f

равномерно распределяются по оставшимся пользователям. Но по времени симуляции, и по утилизации (графики 5.2) видно, что это не самый лучший вариант занять кластер.

Таким образом можно сделать вывод, что для такой конфигурации кластера и такой нагрузки оптимальный параметр честности для классического алгоритма **Tetris** будет $f = 0.6$. Именно с проверки подобных гипотез можно использовать симулятор вместо того, чтобы запускать все эти эксперименты на реальном кластере, т.к. для проведения подобного эксперимента в симуляторе потребуется несколько секунд времени.

5.3 Детерминированность симуляции

Важным аспектом любой симуляции является воспроизводимость результатов. Фреймворк **DSL** предоставляет универсальный интерфейс доступа к генератору случайных чисел в симуляции, который зависит только от переданного в симуляцию сеида. Реализованный симу-

лятор полностью использует эту возможность и является детерминированным при условии детерминированности алгоритма планирования. Очень важным является использование генератора случайных чисел из переданного в планировщик контекста (см. интерфейс 4.8), однако в стандартных структурах языка **Rust** используется другой генератор случайных чисел, и это необходимо учитывать. Например, при итерации по ключам стандартной хэш-таблицы они будут возвращаться в разных порядках в зависимости от запуска программы.

6 Заключение

В рамках ВКР удалось реализовать многофункциональный и производительный симулятор вычислительного кластера. Код симулятора опубликован в открытом доступе на GitHub[5]. Как результат работы можно выделить основное достижение: возможность пользователям исключительно гибко (по сравнению со всеми другими симуляциями) настраивать профиль загрузки сервера. Использование асинхронных примитивов языка **Rust** дает много возможностей для точно описания нагрузки. При этом, поддерживаются и более простые сценарии использования симулятора.

С помощью симулятора проведены несколько экспериментов, которые показывают его применимость на больших симуляциях и в разных моделях планирования: честного распределения ресурсов между пользователями и оптимальной упаковки задач на кластере.

В качестве дальнейшего развития симулятора можно модифицировать модули **DSLlab**, сделав возможной более полную отмену задач (добавить такой функционал в модулях **dslab-network** и **dslab-disk**).

Список литературы

- [1] *Alibaba cluster-trace-v2018*. Technical Report. Posted at https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace_2018.md. 2018.
- [2] *BatSim profile types overview*. URL: <https://batsim.readthedocs.io/en/latest/input-workload.html#profile-types-overview>.
- [3] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson и Frédéric Suter. “Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms”. В: *Journal of Parallel and Distributed Computing* 74.10 (июнь 2014), с. 2899—2917. DOI: 10.1016/j.jpdc.2014.06.008. URL: <https://inria.hal.science/hal-01017319>.
- [4] *DSLAb architecture*. URL: <https://docs.google.com/document/d/12CcGpdulqMJppAYoNrOdpIWZYEa3flXn9JMX5fdyAnY/edit>.
- [5] *DsLab Cluster Scheduling Simulator repository*. URL: <https://github.com/nogokama/dslab-cluster-simulator>.
- [6] *DsLab comparison to other simulators*. URL: <https://docs.google.com/document/d/1H2711nGiS6m7QTo4pMfmrIBmg41lLKUhQzDiFNIw-UU/edit>.
- [7] *DSLAb compute module documentation*. URL: https://osukhoroslov.github.io/dslab/docs/dslab_compute/index.html.
- [8] *DSLAb network module documentation*. URL: https://osukhoroslov.github.io/dslab/docs/dslab_network/index.html.
- [9] *DsLab repository*. URL: <https://github.com/osukhoroslov/dslab>.
- [10] *DSLAb storage module documentation*. URL: https://osukhoroslov.github.io/dslab/docs/dslab_storage/index.html.
- [11] Cristian Galleguillos, Zeynep Kiziltan, Alessio Netti и Ricardo Soto. *AccaSim: a Customizable Workload Management Simulator for Job Dispatching Research in HPC Systems*. 2018. arXiv: 1806.06728 [cs.DC].
- [12] Ali Ghodsi, Matei A. Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker и Ion Stoica. “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types”. В: *Symposium on Networked Systems Design and Implementation*. 2011. URL: <https://api.semanticscholar.org/CorpusID:6693615>.

- [13] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao и Aditya Akella. “Multi-resource packing for cluster schedulers”. в: *SIGCOMM Comput. Commun. Rev.* 44.4 (абр. 2014), с. 455—466. ISSN: 0146-4833. DOI: [10.1145/2740070.2626334](https://doi.org/10.1145/2740070.2626334). URL: <https://doi.org/10.1145/2740070.2626334>.
- [14] Adrián Herrera, Mario Ibáñez, Esteban Stafford и Jose Luis Bosque. “A Simulator for Intelligent Workload Managers in Heterogeneous Clusters”. в: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2021, с. 196—205. DOI: [10.1109/CCGrid51090.2021.00029](https://doi.org/10.1109/CCGrid51090.2021.00029).
- [15] *Implementation of Asynchronous Programming Support for the DSLab Framework*. URL: <https://nogokama.github.io/ThirdYearCoursework/report.pdf>.
- [16] Taylan Özden, Tim Beringer, Arya Mazaheri, Hamid Mohammadi Fard и Felix Wolf. “ElastiSim: A Batch-System Simulator for Malleable Workloads”. в: *Proceedings of the 51st International Conference on Parallel Processing*. ICPP ’22. Bordeaux, France: Association for Computing Machinery, 2023. ISBN: 9781450397339. DOI: [10.1145/3545008.3545046](https://doi.org/10.1145/3545008.3545046). URL: <https://doi.org/10.1145/3545008.3545046>.
- [17] F. Petrini и M. Vanneschi. “k-ary n-trees: high performance networks for massively parallel architectures”. в: *Proceedings 11th International Parallel Processing Symposium*. 1997, с. 87—93. DOI: [10.1109/IPPS.1997.580853](https://doi.org/10.1109/IPPS.1997.580853).
- [18] *Rust crate «futures»*. URL: <https://crates.io/crates/futures>.
- [19] *Slurm CPU Management User and Administrator Guide*. URL: https://slurm.schedmd.com/cpu_management.html.
- [20] *The Standard Workload Format Specification*. URL: <https://www.cs.huji.ac.il/labs/parallel/workload/swf.html>.
- [21] Muhammad Tirmazi, Adam Barker, Nan Deng, Md Ehtesam Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter и John Wilkes. “Borg: the Next Generation”. в: *EuroSys’20*. Heraklion, Crete, 2020.
- [22] John Wilkes. *Google cluster-usage traces v3*. Technical Report. Posted at <https://github.com/google/cluster-data/blob/master/ClusterData2019.md>. Mountain View, CA, USA: Google Inc., апр. 2020.

- [23] Устройство планировщика Yandex YT. Запись с конференции *HighLoad*. URL: https://www.youtube.com/watch?v=Uv-IcGZSRpk&ab_channel=HighLoadChannel.