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

# 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 компьютерные 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

## 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

### 1.4 Структура работы

## 2 Обзор литературы

### 2.1 Вычислительная модель

Для моделирования работы кластера и алгоритмов планирования требуется модель рабочей нагрузки. Standard Workload Format(SWF)[13] 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[14]).

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[8]. 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[1] и позволяет разработчикам интегрировать алгоритмы на нескольких языках на основе inter-process communication. Подробное сравнение SimGrid и DSLab можно найти в документации по DSLab[3]. 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).

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

```
"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

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[9]. 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 тим 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[11] 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 глубокого обучения, для которых он и предназначен. Кроме этого, в Elastisim поддерживатеся 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 поддержать 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

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 использовать исследователями соответствующих предметных областей, например, новый симулятор использует готовые модули вычислений, сети и диска (подробнее об этом в главе 4.2).

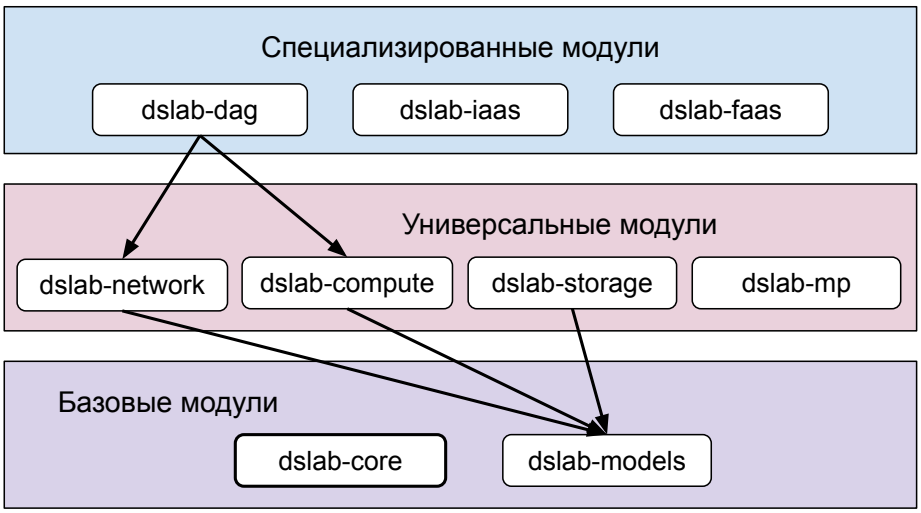


Рис. 3.1: Архитектура `DSLab`

Описание архитектуры основано на официальной документации [2]. Реализация всех модулей опубликована в GitHub-репозитории[6].

## 3.1 Асинхронное управление событиями

В предыдущей работе была добавлена возможность управлять событиями `DSLab` асинхронно[10], 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

```
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.

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`[12]), после этого ядро симуляции 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 JSON-файла.

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

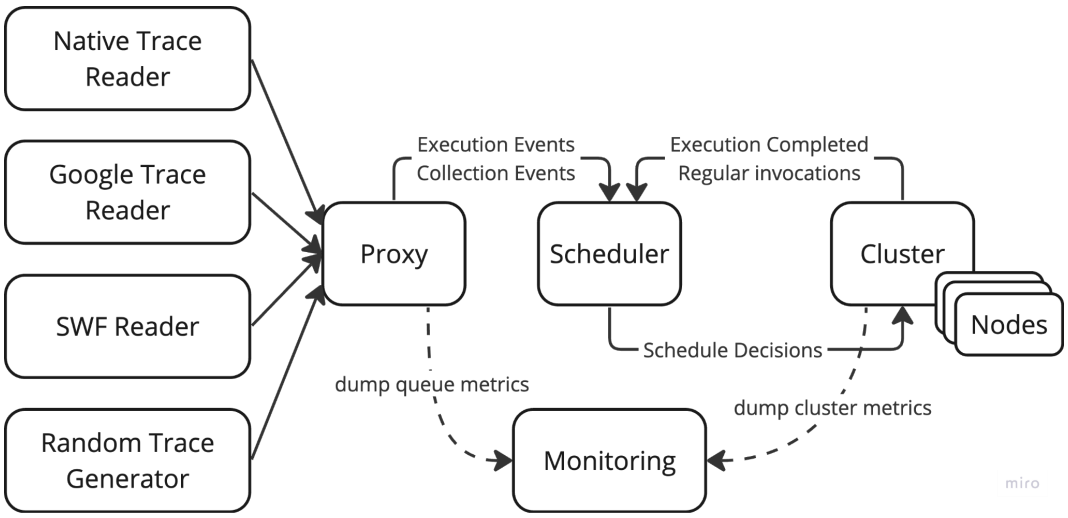


Рис. 4.1: Архитектура симулятора

анализа работы алгоритмов и сравнения их эффективности. За это отвечает отдельный компонент – `Monitoring`.

## 4.2 Модель кластера

Вычислительный кластер – это набор отдельных серверов, которые могут быть связаны между собой моделируемой сетью. 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

Подробнее с описанием этих модулей и примерами использования можно ознакомиться в официальной документации DSLab[4][5][7].

## 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[14], а `Execution` 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 представленны в таблицах 4.2 и 4.3.

| **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 |

Таблица 4.2: Описание `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 приоритет задания/коллекции. Если у задачи указан собственный приоритет, то предполагается, что он переопределяет данный приоритет. |

Таблица 4.3: Описание `CollectionRequest`

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`:

```
pub trait ExecutionProfile {
  async fn run(self: Rc<Self>, processes: &Vec<HostProcessInstance>);
}
```

Рис. 4.2: 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 работу. Сама работа описывается в терминах асинхронного программирования на Rust и использует предоставленный интерфейс взаимодействия с кластером:

```
async fn sleep(&self, time: f64);
async fn run_flops(&self, flops: f64);
async fn transfer_data(&self, size: f64, dst_process: ProcessId);
async fn write_data(&self, size: u64);
async fn read_data(&self, size: u64);
```

Рис. 4.3: Интерфейс взаимодействия задачи с кластером

## 4.4 Модель планирования заданий

## 4.5 Входные данные симуляции

## 4.6 Выходные данные симуляции

## 4.7 Конфигурация симуляции

Вся настройка параметров симуляции происходит через конфигурационный файл в формате YAML. Пример конфигурации представлен на рисунке 4.4.

```
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. 4.4: Пример конфигурации симуляции

# 5 Эксперименты и тестирование

## 5.1 Производительность

## 5.2 Честность

# 6 Заключение

# Список литературы


[1] 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.

[2] *DSLab architecture*. URL: https://docs.google.com/document/d/12CcGpdulqMJppAYoNrOdpIWZYEa3flXn9JMX5fdyAnY/edit.

[3] *DsLab comparison to other simulators*. URL: https://docs.google.com/document/d/1H2711nGiS6m7QTo4pMfmrIBmg4llLKUhQzDiFNIw-UU/edit.

[4] *DSLab compute module documentation*. URL: https://osukhoroslov.github.io/dslab/docs/dslab_compute/index.html.

[5] *DSLab network module documentation*. URL: https://osukhoroslov.github.io/dslab/docs/dslab_network/index.html.

[6] *DsLab repository*. URL: https://github.com/osukhoroslov/dslab.

[7] *DSLab storage module documentation*. URL: https://osukhoroslov.github.io/dslab/docs/dslab_storage/index.html.

[8] 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].

[9] 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.

[10] *Implementation of Asynchronous Programming Support for the DSLab Framework*. URL: https://nogokama.github.io/ThirdYearCoursework/report.pdf.

[11] 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. URL: https://doi.org/10.1145/3545008.3545046.

[12] *Rust crate «futures»*. URL: https://crates.io/crates/futures.

[13] *The Standard Workload Format Specification*. URL: https://www.cs.huji.ac.il/labs/parallel/workload/swf.html.

[14] 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.