Разработка симулятора вычислительного кластера

Выполнил: Макогон Артём Аркадьевич, БПМИ206

Руководитель: Сухорослов Олег Викторович, к.т.н, доцент НИУ ВШЭ

June 2, 2024

Введение

Описание предметной области

- Вычислительные кластеры активно используются в исследованиях и компаниях.
- От качества алгоритма планирования задач зависит эффективность использования ресурсов.
- Такие алгоритмы являются предметом активных исследований.

Введение

Описание предметной области

- Вычислительные кластеры активно используются в исследованиях и компаниях.
- От качества алгоритма планирования задач зависит эффективность использования ресурсов.
- Такие алгоритмы являются предметом активных исследований.

$$\Downarrow$$

> При разработке алгоритмов их необходимо тестировать.

> Использовать реальный кластер долго и дорого.

Введение

Описание предметной области

- Вычислительные кластеры активно используются в исследованиях и компаниях.
- От качества алгоритма планирования задач зависит эффективность использования ресурсов.
- Такие алгоритмы являются предметом активных исследований.

$$\Downarrow$$

> При разработке алгоритмов их необходимо тестировать.

> Использовать реальный кластер долго и дорого.

$$\Downarrow$$

✓ При разработке и тестировании используют **симуляторы**.

Введение

Модель рабочей нагрузки на кластер

Рабочая нагрузка состоит из заданий (job), которые делятся на задачи (tasks).

Введение

Модель рабочей нагрузки на кластер

Рабочая нагрузка состоит из заданий (job), которые делятся на задачи (tasks).

Структуры заданий:

- `rigid` – фиксированные требования к ресурсам
- `moldable` – адаптивные требования к ресурсам
- `malleable` – гибкие требование к ресурсам

Введение

Модель рабочей нагрузки на кластер

Рабочая нагрузка состоит из заданий (job), которые делятся на задачи (tasks).

Структуры заданий:

- `rigid` – фиксированные требования к ресурсам
- `moldable` – адаптивные требования к ресурсам
- `malleable` – гибкие требование к ресурсам

Требования к планированию:

- Раздельное планирование (MapReduce)
- Комплектное планирование (Gang Scheduling, Slurm)

Введение

Виды входных данных для симулятора

`Standard Workload Format (SWF)`

- CPU/RAM

`Custom workloads`

- CPU/RAM/disk/network

Введение

Виды входных данных для симулятора

`Standard Workload Format (SWF)`

- CPU/RAM
- Дано время исполнения и требования к ресурсам
- Нагрузка вычисляется тривиально

`Custom workloads`

- CPU/RAM/disk/network
- Дана нагрузка и ресурсы
- Время вычисляется с помощью моделей

Введение

Виды входных данных для симулятора

`Standard Workload Format (SWF)`

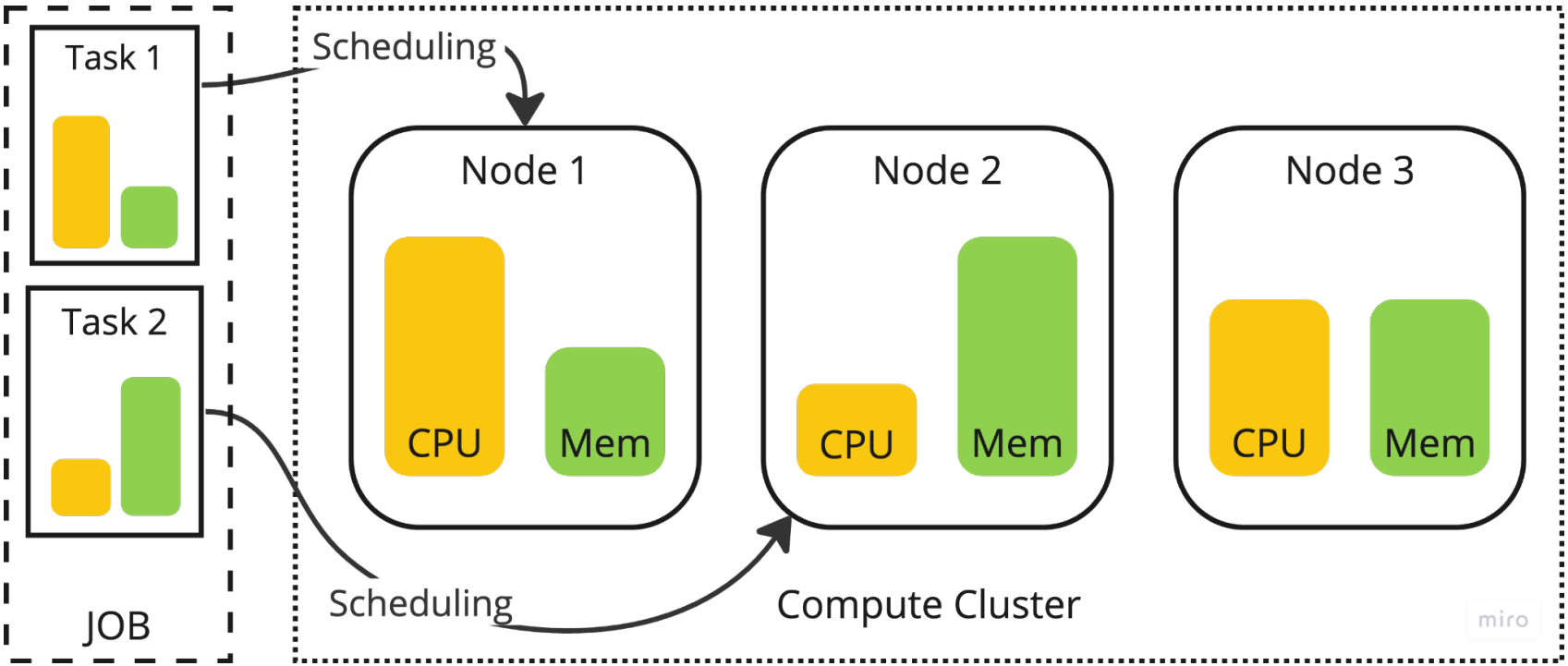
- CPU/RAM
- Дано время исполнения и требования к ресурсам
- Нагрузка вычисляется тривиально
- Используется в популярных трейсах (Google, Alibaba, и т.д.)

`Custom workloads`

- CPU/RAM/disk/network
- Дана нагрузка и ресурсы
- Время вычисляется с помощью моделей
- Обычно NDA

Введение

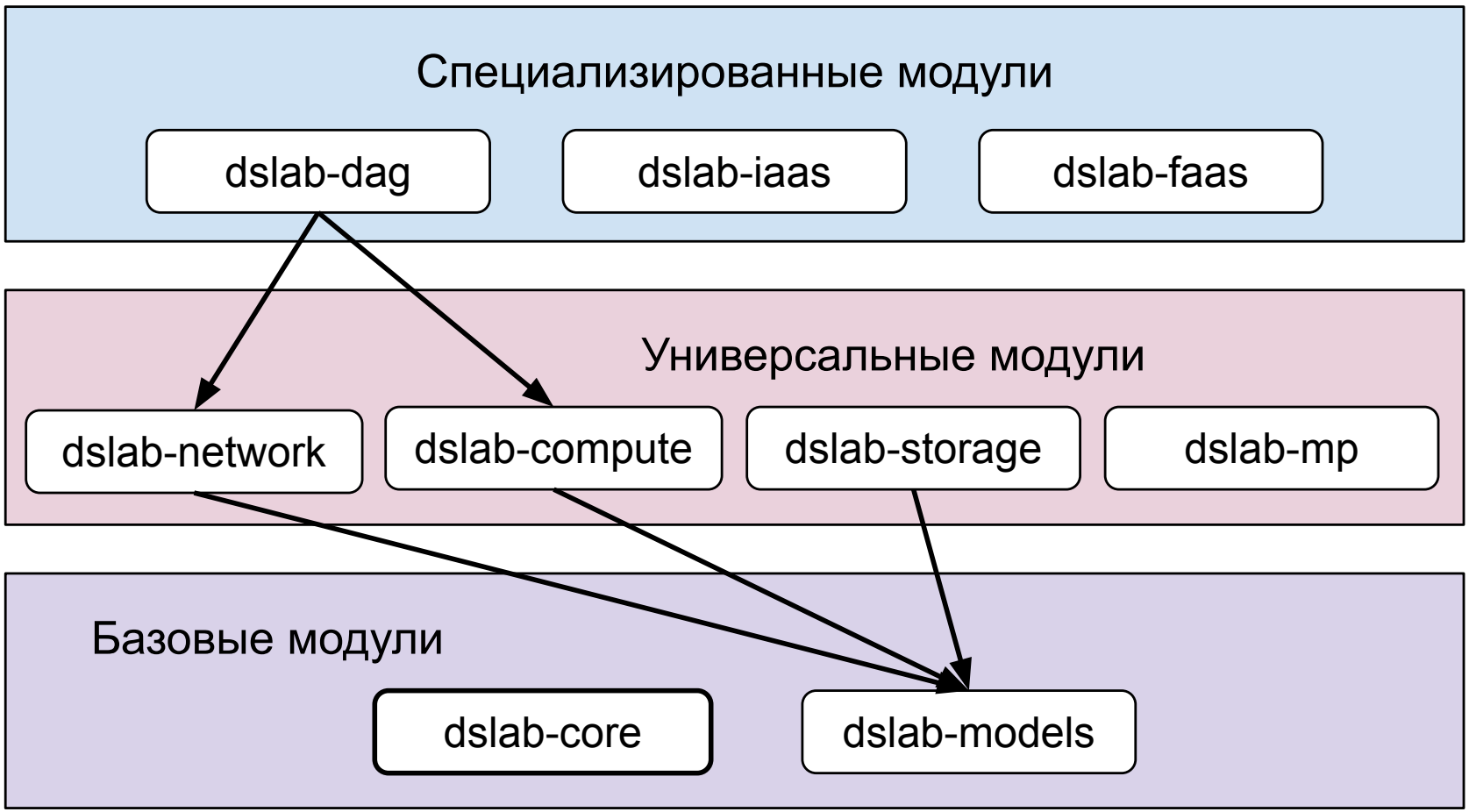
Архитектура кластера



Простая модель архитектуры кластера

Введение

Архитектура DSLab



Архитектура фреймворка DSLab

Введение

Асинхронное управление событиями. Комбинаторы futures

```
async fn process_task(&self, req:
    TaskRequest) {
  let mut task = TaskInfo {req};

  self.download_data(&task).await;
  self.read_data(&task).await;
  self.run_task(&task).await;
  self.write_data(&task).await;
  self.upload_result(&task).await;
}
```

Пример последовательных задач.

```
async fn run(&self, args: JobArgs){
  futures::join!(
    self.download_data(args.node_1)
    self.download_data(args.node_2)
  )
}
```

Пример параллельных задач.

Введение

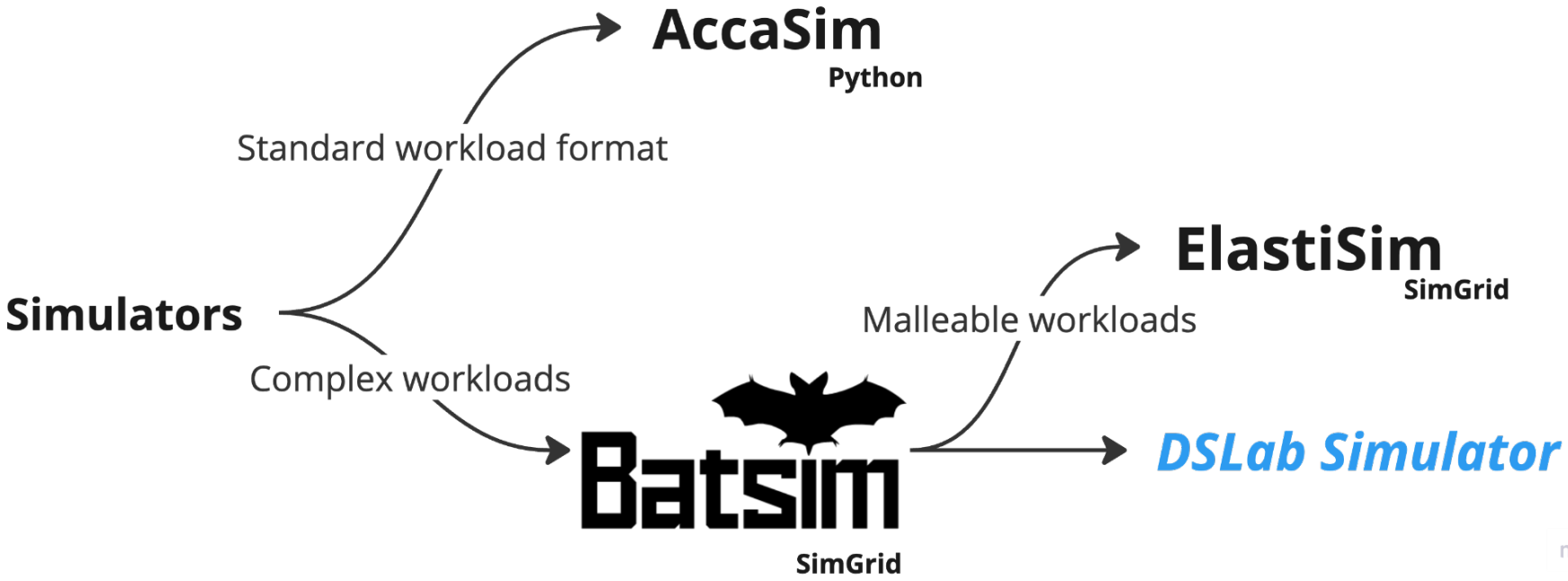
Постановка задачи

Цель проекта – разработать симулятор вычислительного кластера на базе фреймворка DSLab. Для этого необходимо:

- Изучить литературу по теме и существующие симуляторы, подготовить обзор с анализом их преимуществ и недостатков.
- Реализовать компоненты симулятора.
- Написать документацию и тесты.
- Подготовить и провести эксперименты, демонстрирующие работоспособность симулятора.

Обзор литературы

Существующие симуляторы



Существующие симуляторы

Обзор литературы

BatSim

- Разработан на базе фреймворка SimGrid.
- Поддерживает SWF и пользовательские профили нагрузки через JSON-файл.
- Алгоритмы планирования подключаются к симулятору через IPC (inter-process-communication).

Обзор литературы

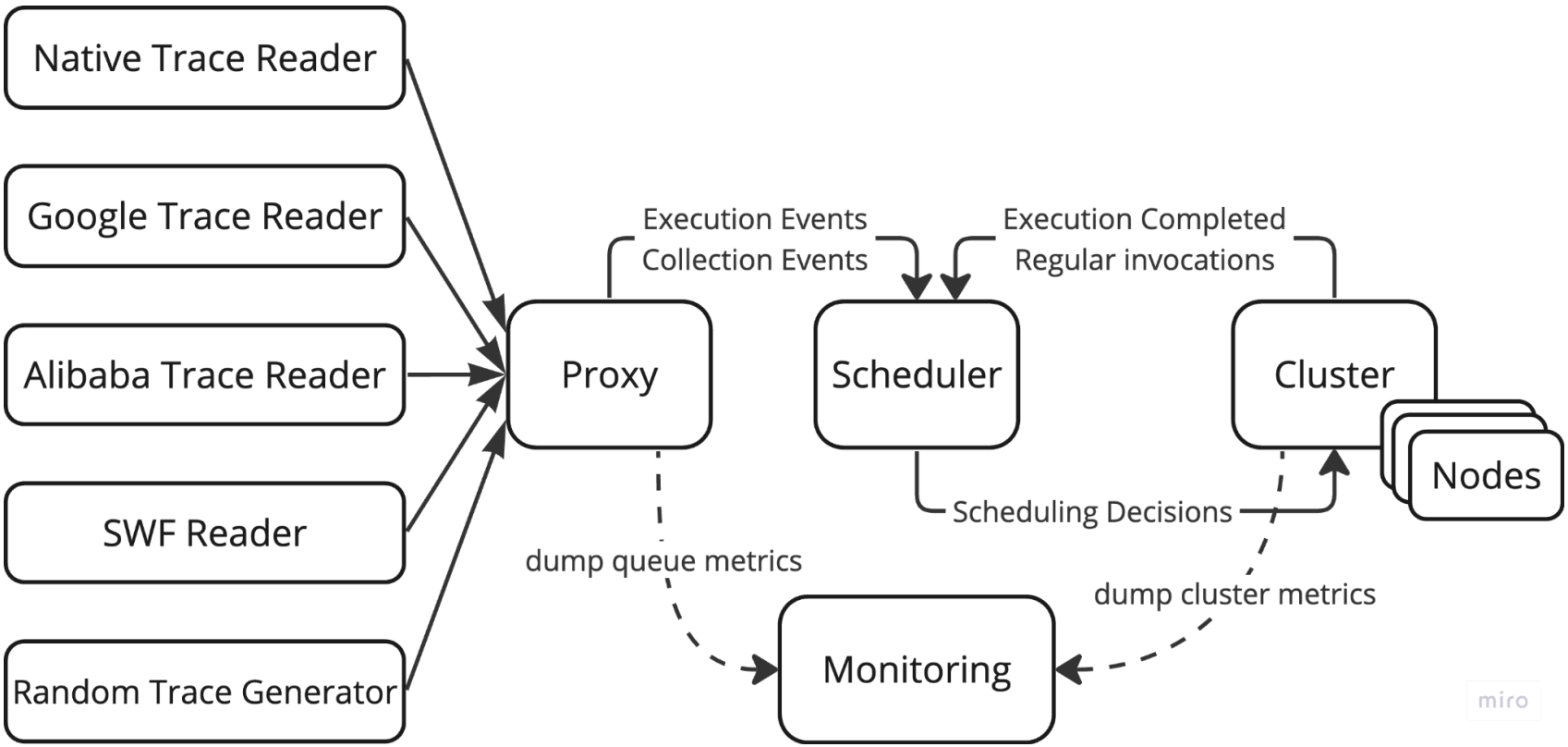
Профиль нагрузки в BatSim

```
"jobs": [
  {"id": "job1",  ...  "res": 4, "profile": "sequence"},
],

"profiles": {
  "homogeneous": {
    "type": "parallel_homogeneous",
    "cpu": 10e6,
    "com": 1e6
  },
  "sequence": {
    "type": "composed",
    "repeat" : 4,
    "seq": ["simple","homogeneous","simple"]
  },
}
```

Реализация симулятора

Архитектура симулятора



Архитектура симулятора

Реализация симулятора

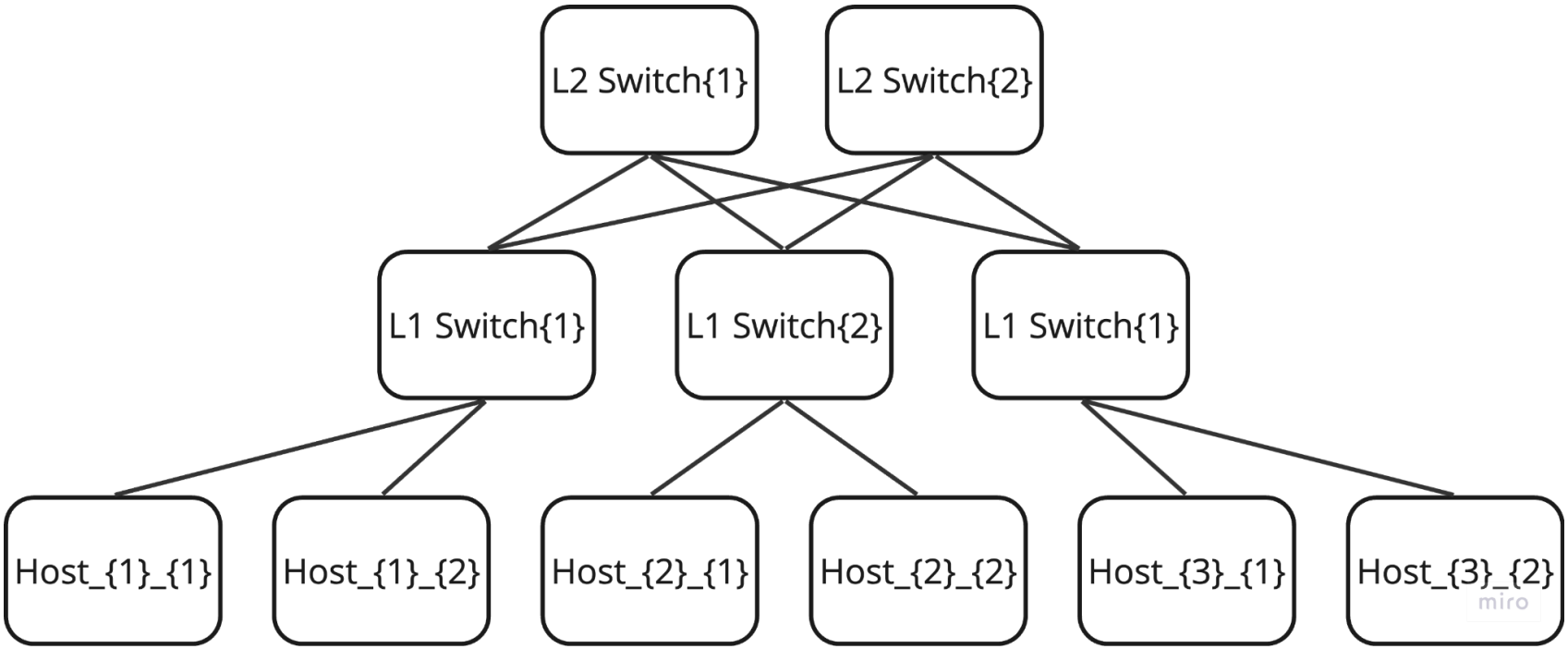
Модель кластера

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

Модули каждого сервера в кластере

Реализация симулятора

Модель сети Fat-Tree-Topology



Модель сети Fat-Tree-Topology в кластере

Реализация симулятора

Деление на `Collection` и `Execution`

Поле	Тип	Описание
id	`u64`	Уникальный идентификатор задания
user	`Option<String>`	Пользователь, который запускает задание.
priority	`Option<u64>`	Приоритет задания.
...	...	...

Описание структуры данных `Collection`

Поле	Тип	Описание
id	`u64`	Уникальный идентификатор задачи
collection_id	`Option<u64>`	Индекс задания
time	`f64`	Время, когда задача становится доступной для планирования
resources	`ResourceRequirements`	Структура требуемых ресурсов
profile	`Rc<dyn ExecutionProfile>`	Профиль нагрузки
...	...	...

Описание структуры данных `Execution`

Реализация симулятора

Trait ExecutionProfile

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

Интерфейс ExecutionProfile

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

Интерфейс взаимодействия процессов с кластером

Реализация симулятора

Пример реализации ExecutionProfile

```rust
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;
  }
}
```

Пример реализации интерфейса ExecutionProfile

Эксперименты

Производительность на Alibaba Trace

Эксперименты

Алгоритмы честности

Заключение

- ✓ `SWF`-based simulation
- Custom workload simulation
- Support reading workload traces from popular sources (Google, Alibaba)
- Support for collecting metrics during the simulation and writing them to a file with the results
- High performance, support for cluster modeling from 1-10K servers

Основные источники

1. The standard workload format specification. https://www.cs.huji.ac.il/labs/parallel/workload/swf.html
2. Dslab repository. https://github.com/osukhoroslov/dslab
3. BatSim docs. https://batsim.readthedocs.io/en/latest/