Разработка симулятора вычислительного кластера

Выполнил: Макогон Артём Аркадьевич, БПМИ206

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

June 3, 2024

Введение

Описание предметной области

- Вычислительные кластеры активно используются в исследованиях и компаниях.
- От качества алгоритма планирования задач зависит эффективность использования ресурсов.
- При разработке алгоритмов их необходимо тестировать.

Введение

Описание предметной области

- Вычислительные кластеры активно используются в исследованиях и компаниях.
- От качества алгоритма планирования задач зависит эффективность использования ресурсов.
- При разработке алгоритмов их необходимо тестировать.

$$\Downarrow$$

> 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.
- При разработке алгоритмов их необходимо тестировать.

$$\Downarrow$$

> Использовать реальный кластер долго и дорого.

$$\Downarrow$$

✓ Используются **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)`

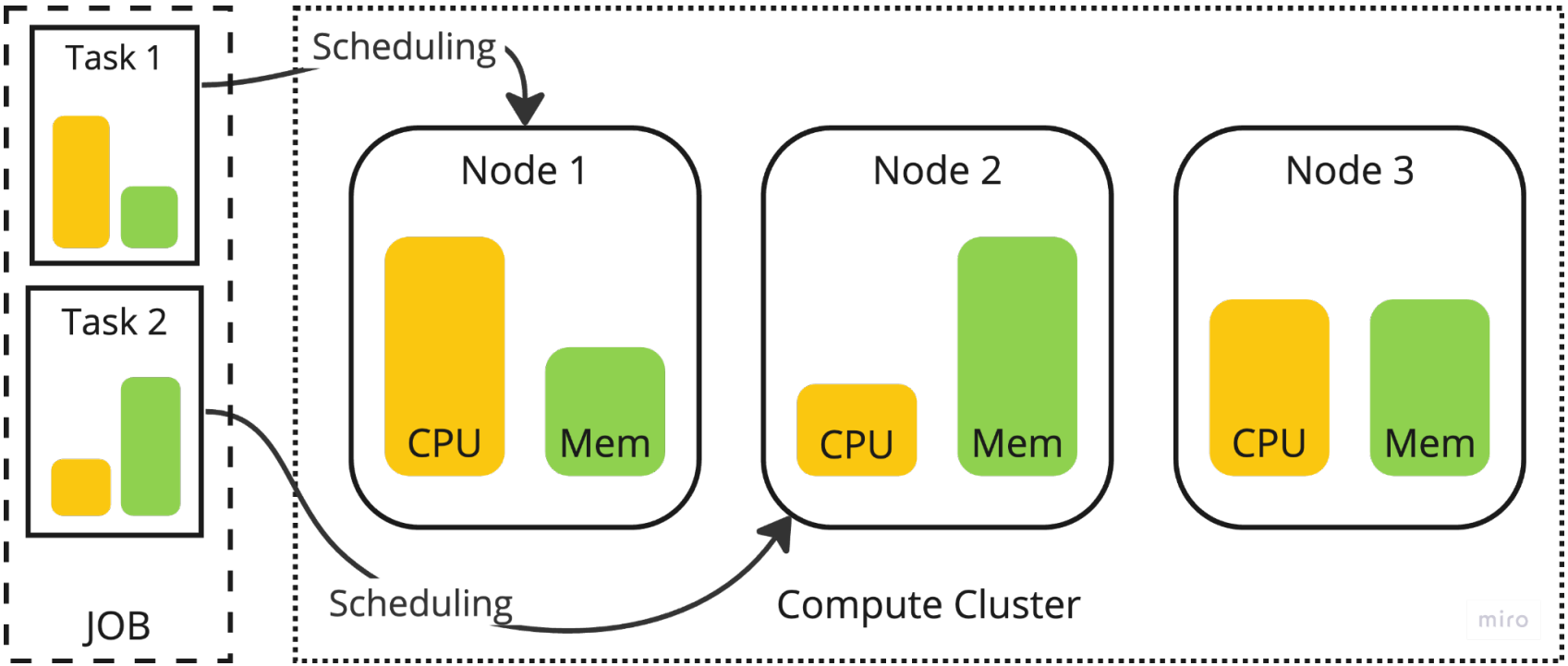
- CPU/RAM
- Дано время исполнения и требования к ресурсам
- Нагрузка вычисляется тривиально
- Используется в популярных трейсах (Google, Alibaba, и т.д.)

`Custom workloads`

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

Введение

Архитектура кластера



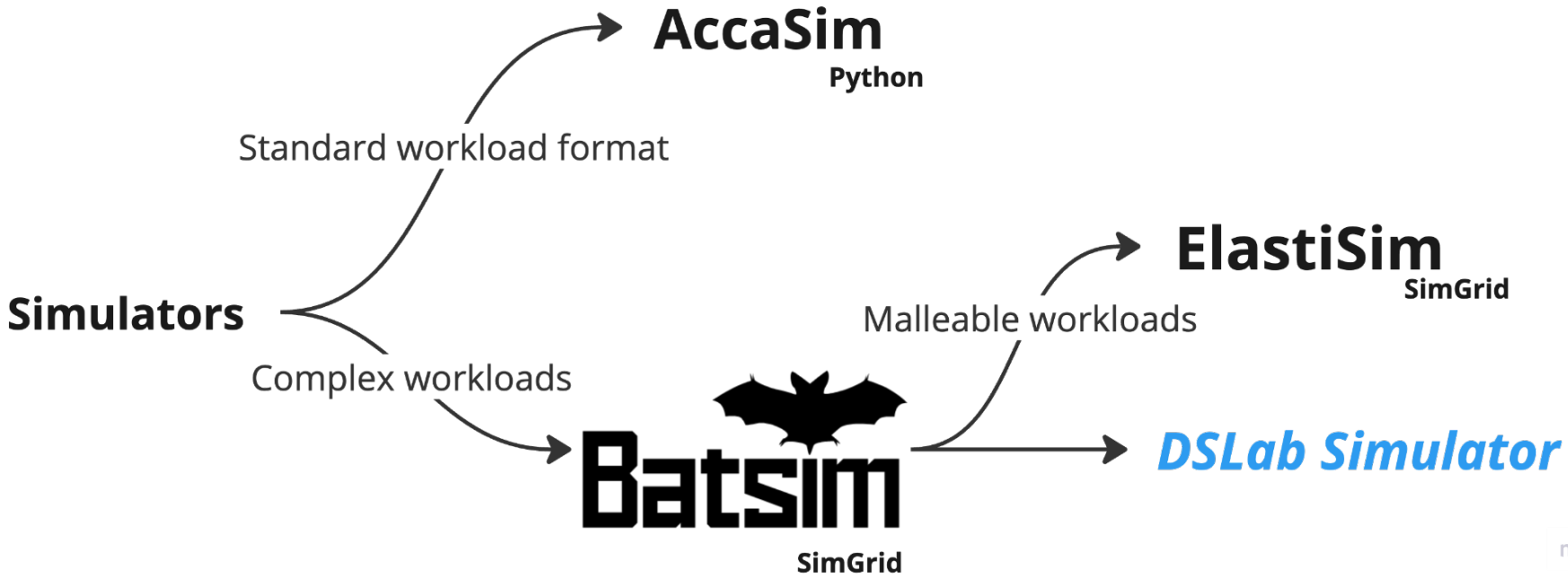
Простая модель архитектуры кластера

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

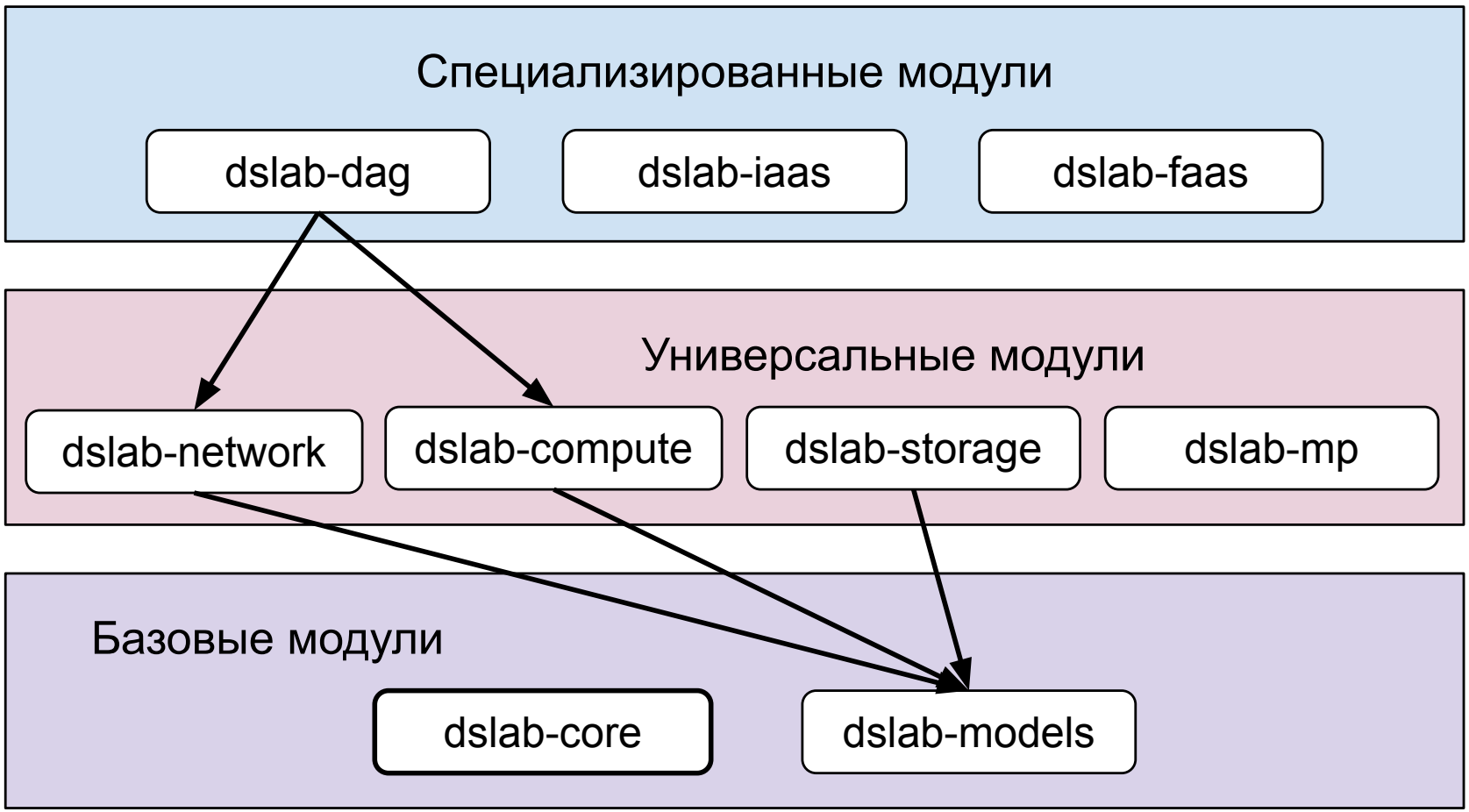
Существующие симуляторы



Существующие симуляторы

Реализация симулятора

Архитектура DSLab



Архитектура фреймворка DSLab

Реализация симулятора

Асинхронное управление событиями. Комбинаторы `futures`

```rust
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;
}
```

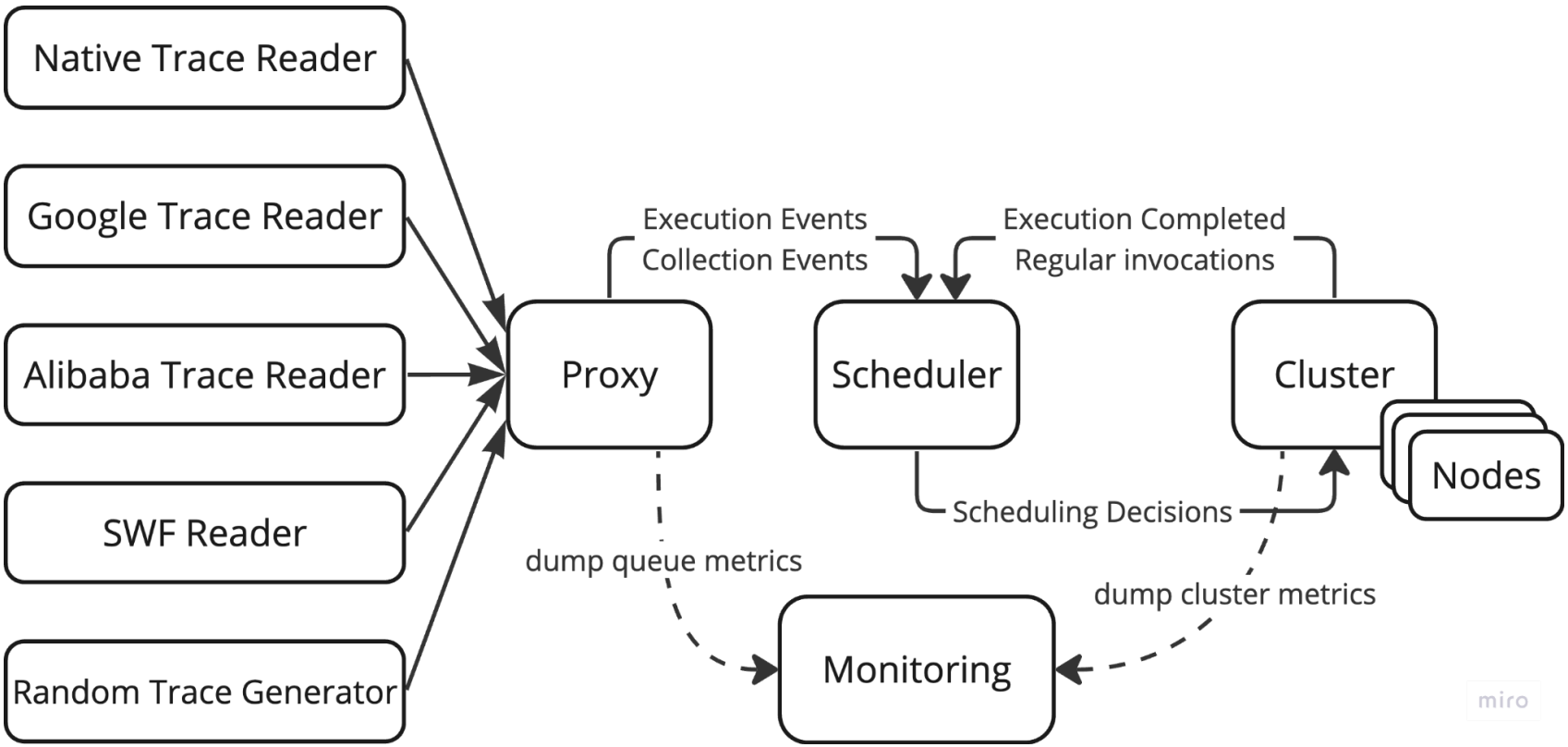
Пример последовательных задач.

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

Пример параллельных задач.

Реализация симулятора

Архитектура симулятора



Архитектура симулятора

Реализация симулятора

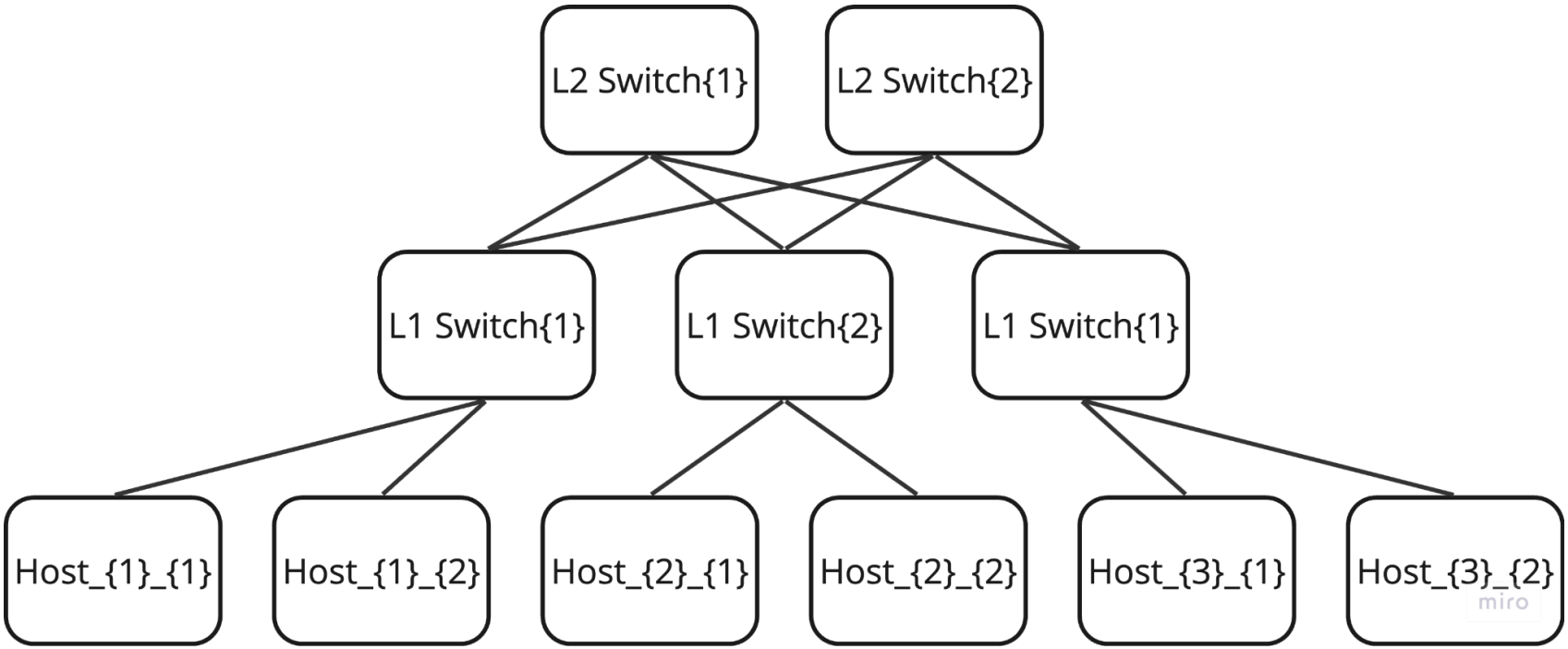
Модель кластера

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;
  }
}
```

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;
  }
}
```

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;
  }
}
```

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;
  }
}
```

Реализация симулятора

Входные данные

1. `Native` – нагрузка в формате `YAML`

Реализация симулятора

Входные данные

1. `Native` – нагрузка в формате `YAML`
2. `Google Trace`
3. `Alibaba Trace`

Реализация симулятора

Входные данные

1. `Native` – нагрузка в формате `YAML`
2. `Google Trace`
3. `Alibaba Trace`
4. `SWF` – Standard Workload Format

Реализация симулятора

Входные данные

1. `Native` – нагрузка в формате `YAML`
2. `Google Trace`
3. `Alibaba Trace`
4. `SWF` – Standard Workload Format
5. `Synthetic Trace Generator`

Реализация симулятора

Модель планировщика

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

Интерфейс планировщика Scheduler

Реализация симулятора

Модель планировщика

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

Интерфейс планировщика `Scheduler`

- Возможности контекста:
  1. Запланировать/отменить задачу (полная отмена недоступна)

Реализация симулятора

Модель планировщика

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

Интерфейс планировщика `Scheduler`

- Возможности контекста:
  1. Запланировать/отменить задачу (полная отмена недоступна)
  2. Доступ к генератору случайных чисел симуляции

Реализация симулятора

Модель планировщика

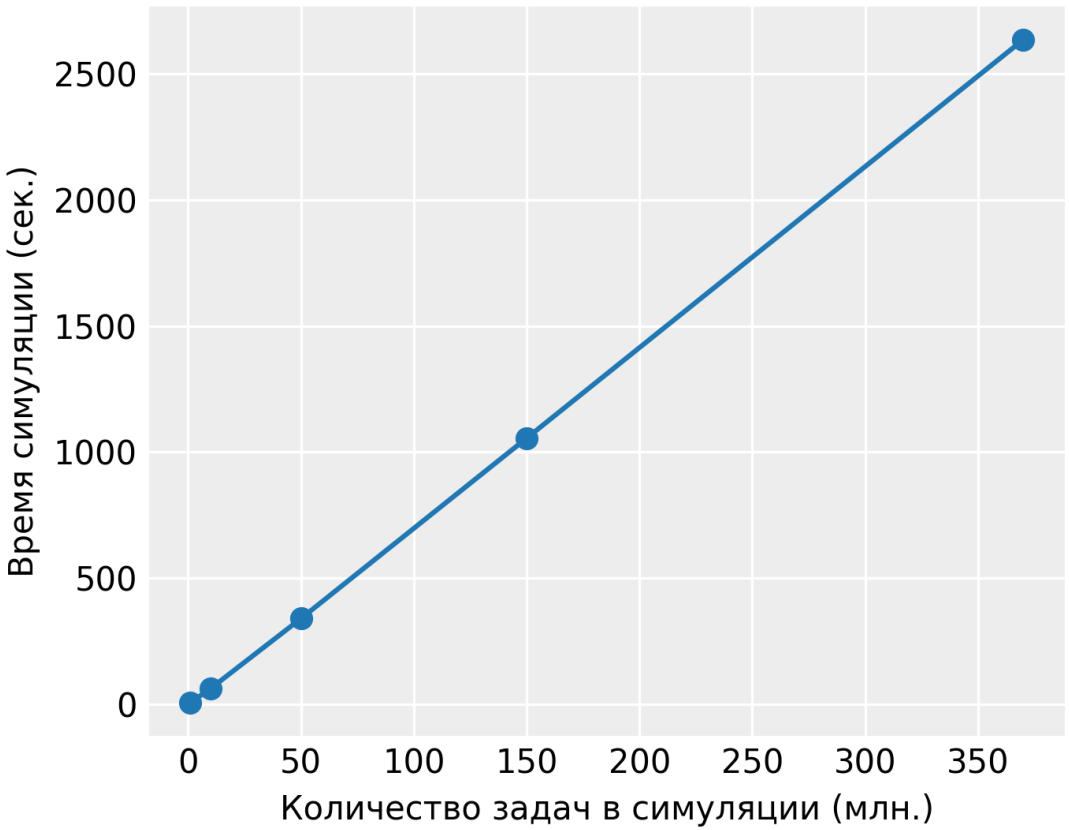
```
pub trait Scheduler {
  fn on_host_added(&mut self, ctx: &Context, host: HostConfig);
  fn on_execution_request(&mut self, ctx: &Context, req: Execution);
  fn on_collection_request(&mut self, ctx: &Context, req: Collection);
  fn on_execution_finished(&mut self, ctx: &Context, execution_id: u64);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

Интерфейс планировщика `Scheduler`

- Возможности контекста:
  1. Запланировать/отменить задачу (полная отмена недоступна)
  2. Доступ к генератору случайных чисел симуляции
- Можно реализовать планировщик как компонент симуляции и получить больше возможностей (например, асинхронность).

Эксперименты

Производительность на `Alibaba Trace`

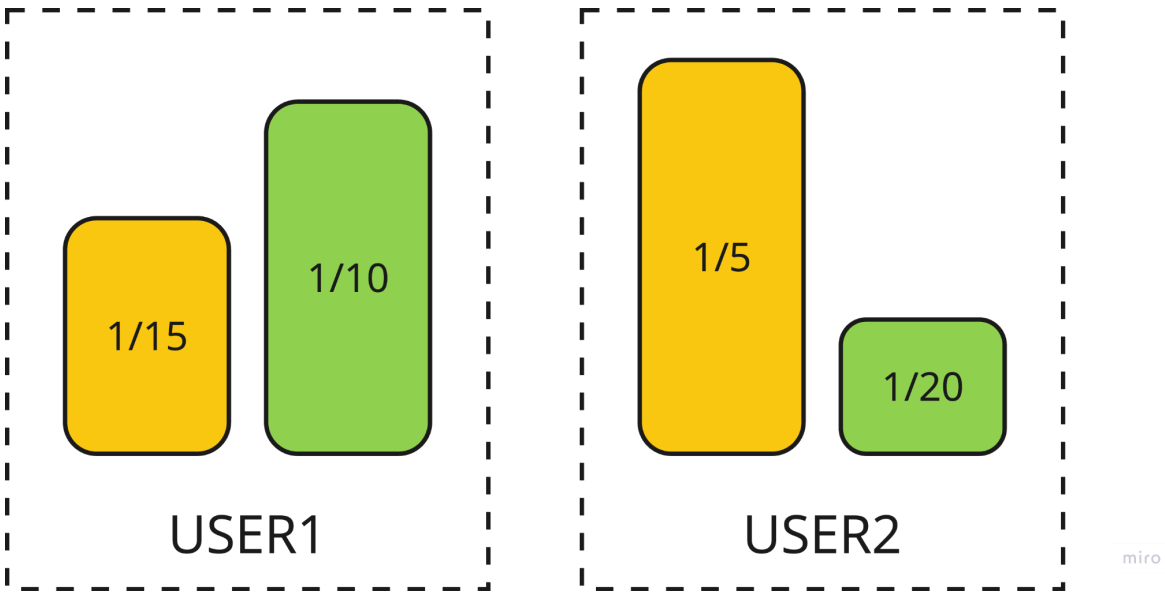


Время работы симуляции в зависимости от количества задач

- На трейсе длиной 2.5 дня (350 млн задач, 25GB) время работы 45 минут
- Ускорение в 85 раз относительно реальной работы.
- Необходимо сортировать трейс по времени.

Эксперименты

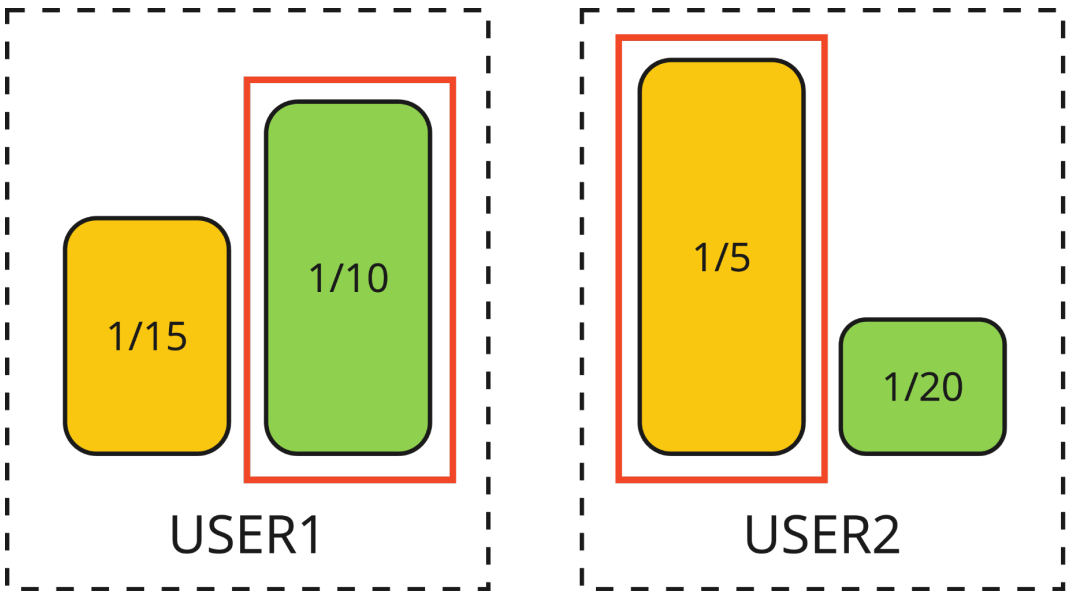
Алгоритмы честности. DRF & Tetris



Потребление ресурсов пользователями

Эксперименты

Алгоритмы честности. DRF & Tetris



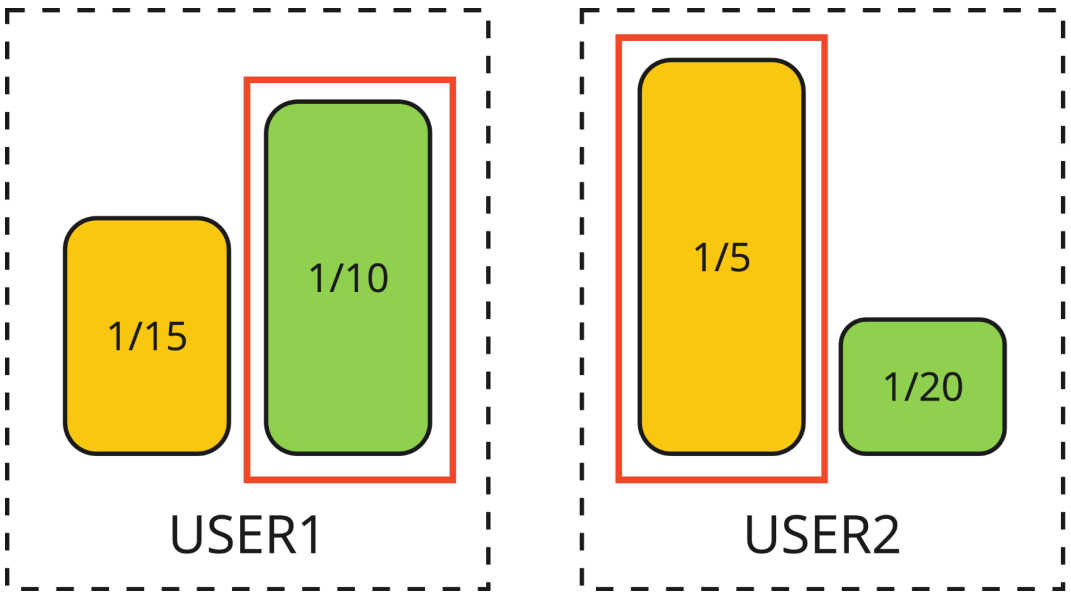
Потребление ресурсов пользователями

Dominant shares:

- USER1 = 1/10
- USER2 = 1/5

Эксперименты

Алгоритмы честности. DRF & Tetris



Потребление ресурсов пользователями

Dominant shares:

- USER1 = 1/10
- USER2 = 1/5
- USER1 первый получит ресурсы

Эксперименты

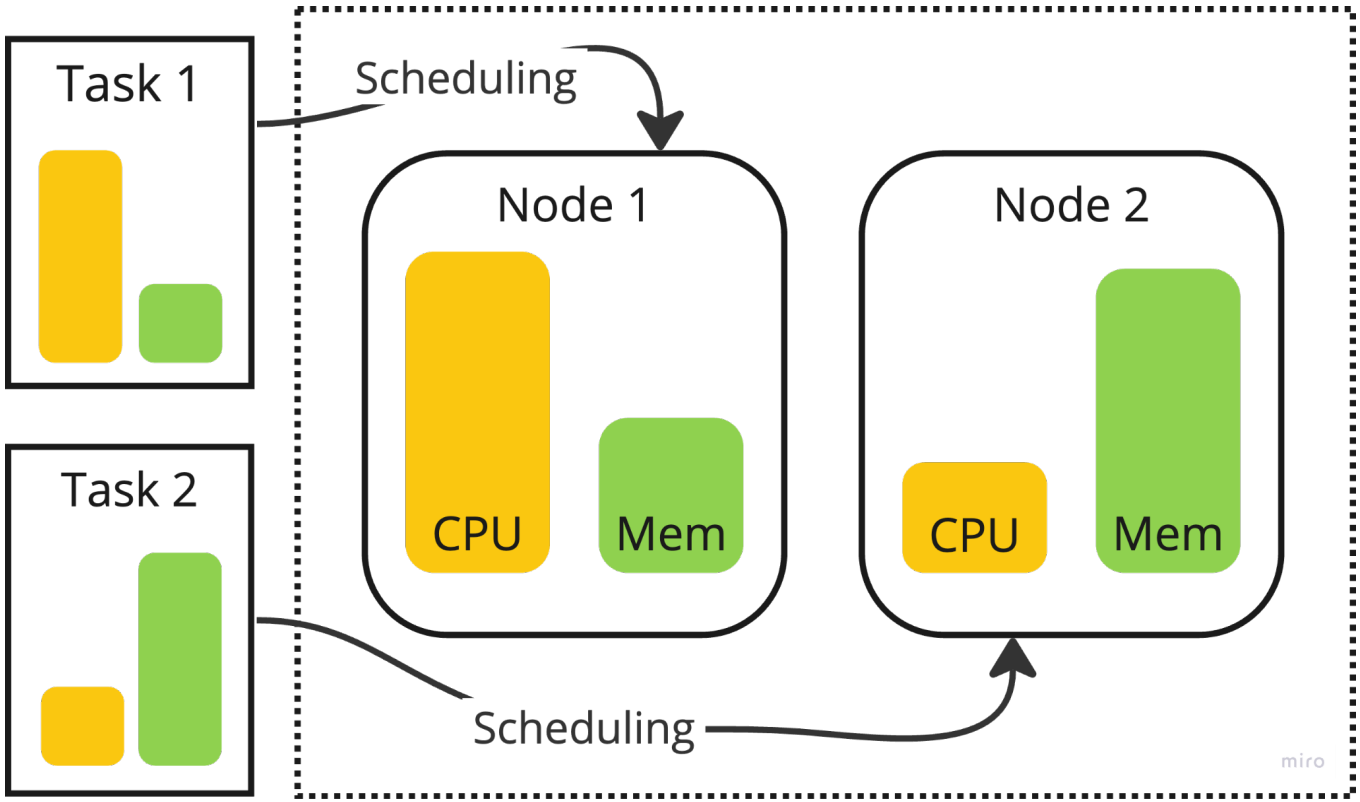
Алгоритмы честности. DRF & Tetris

Метрика «упаковки»: $H_{angle} = 1 - \cos(\measuredangle(Task_{res}, Server_{res}))$

Эксперименты

Алгоритмы честности. DRF & Tetris

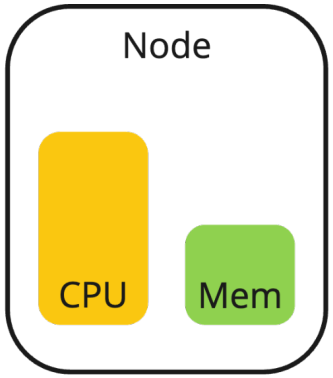
Метрика «упаковки»: $H_{angle} = 1 - \cos(\measuredangle(Task_{res}, Server_{res}))$

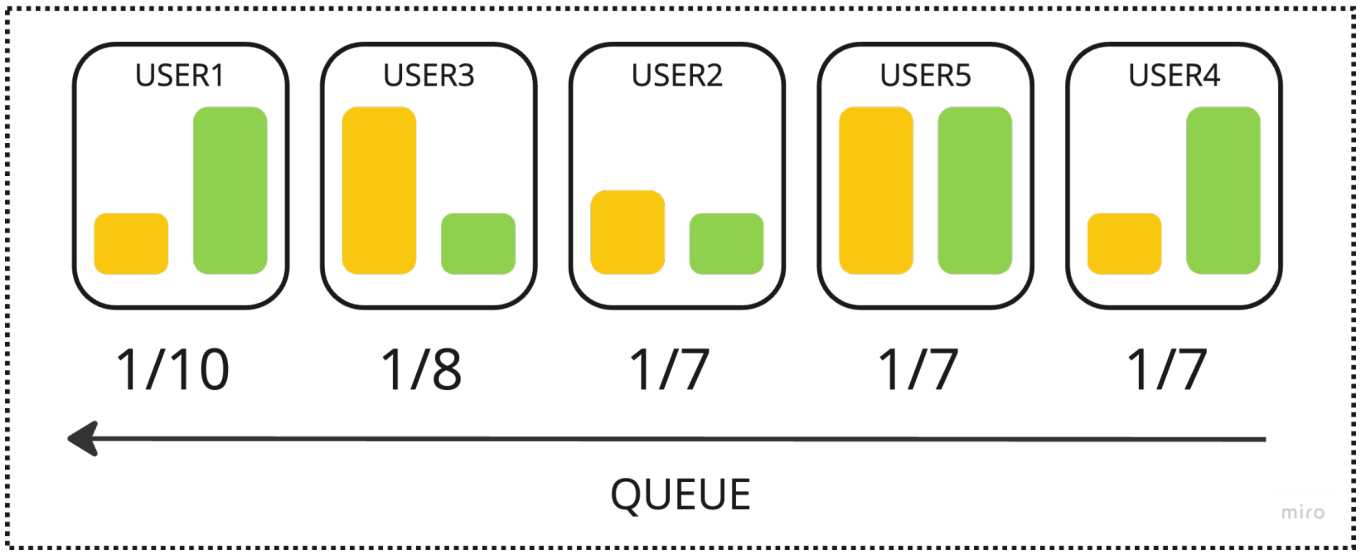


Пример работы алгоритма Tetris

Эксперименты

DRF & Tetris. Коэффициент честности f

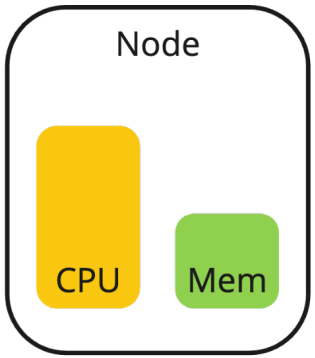




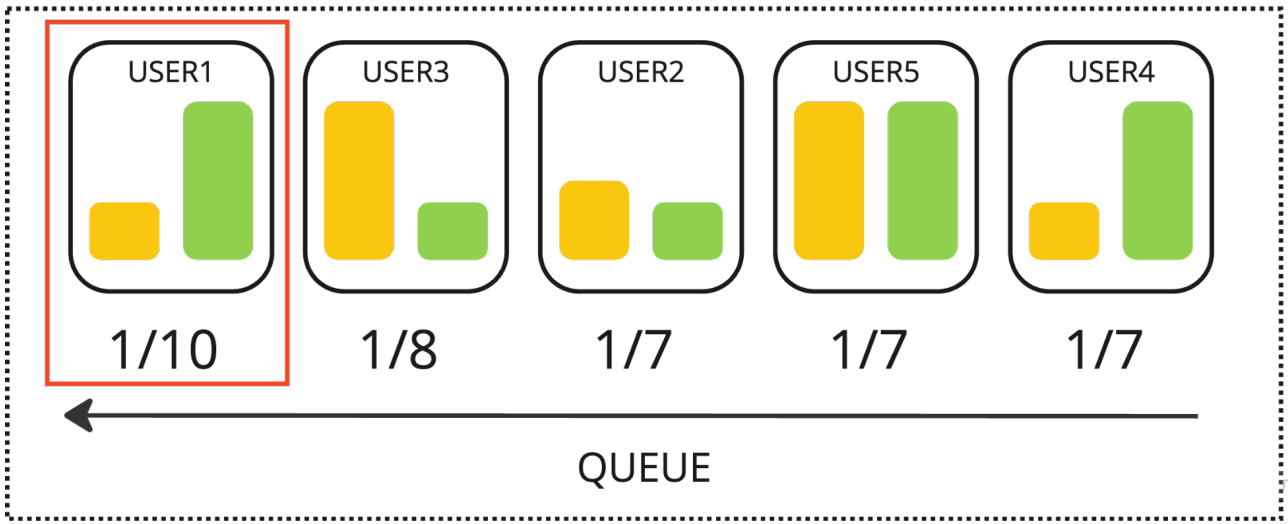
Процесс принятия решения о планировании

Эксперименты

DRF & Tetris. Коэффициент честности f



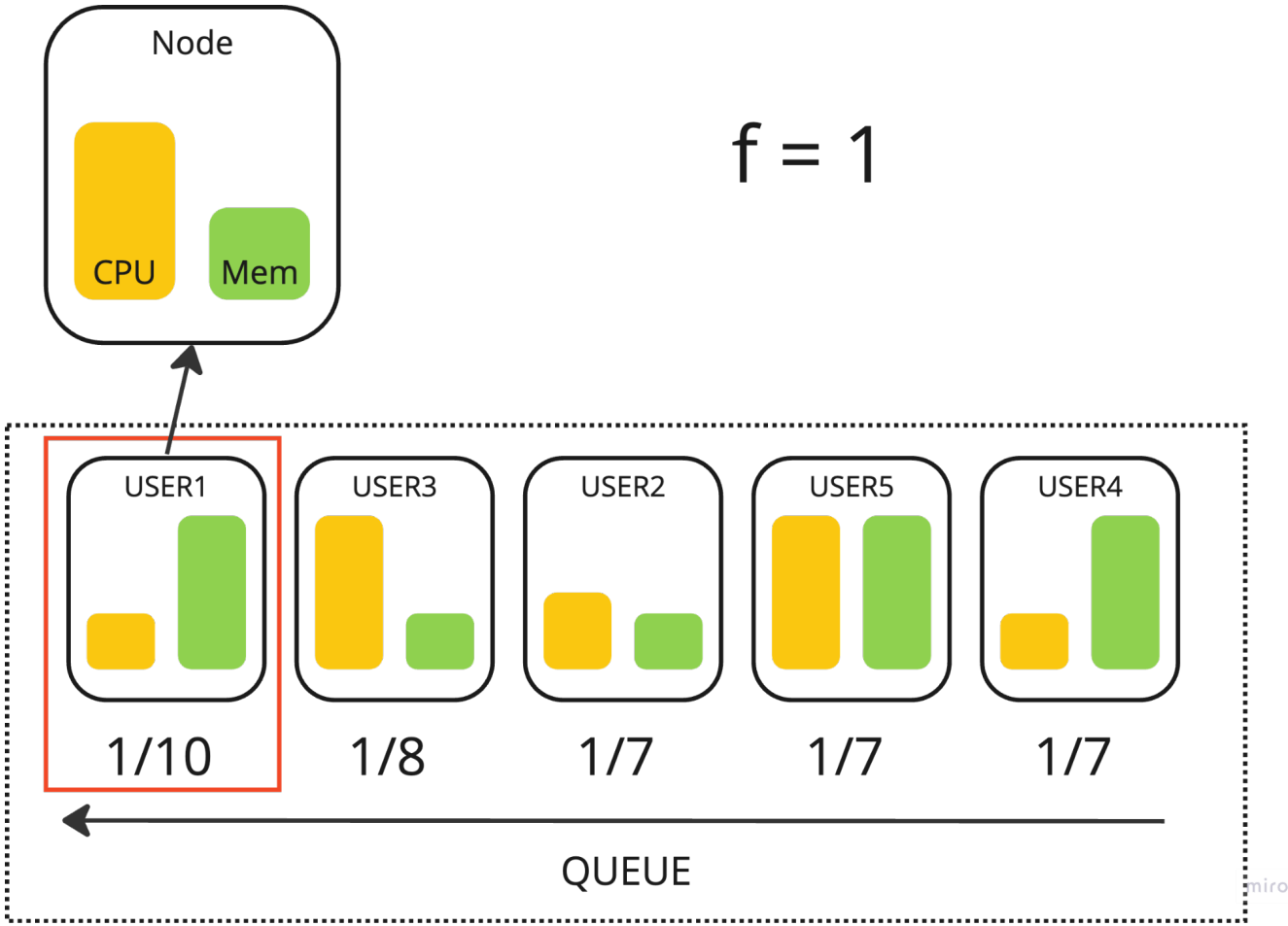
$$f = 1$$



Процесс принятия решения о планировании

Эксперименты

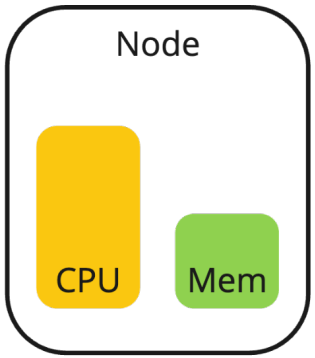
DRF & Tetris. Коэффициент честности f



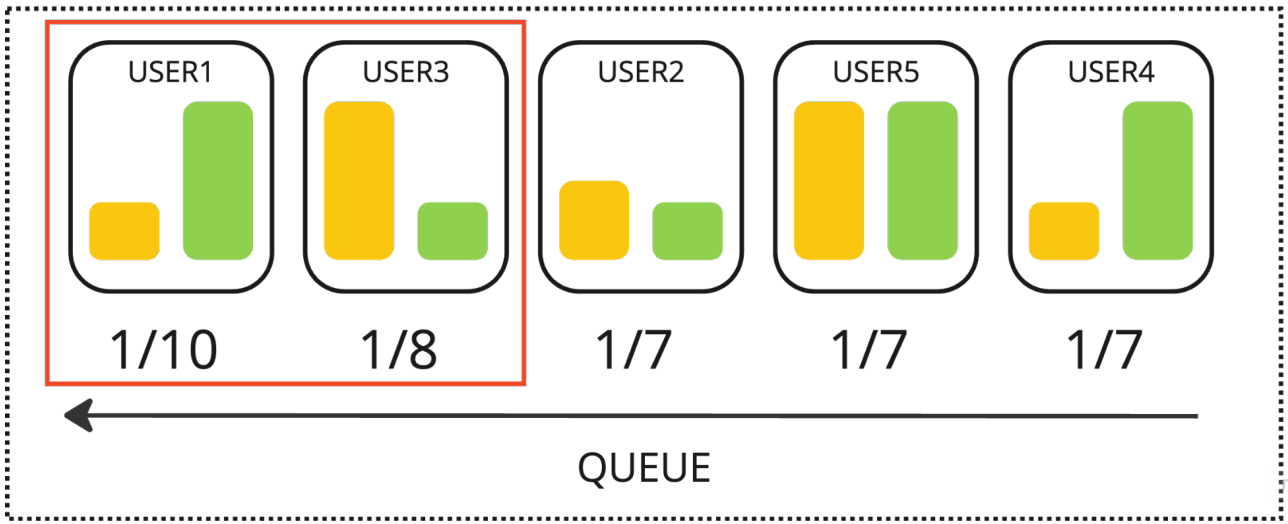
Процесс принятия решения о планировании

Эксперименты

DRF & Tetris. Коэффициент честности f



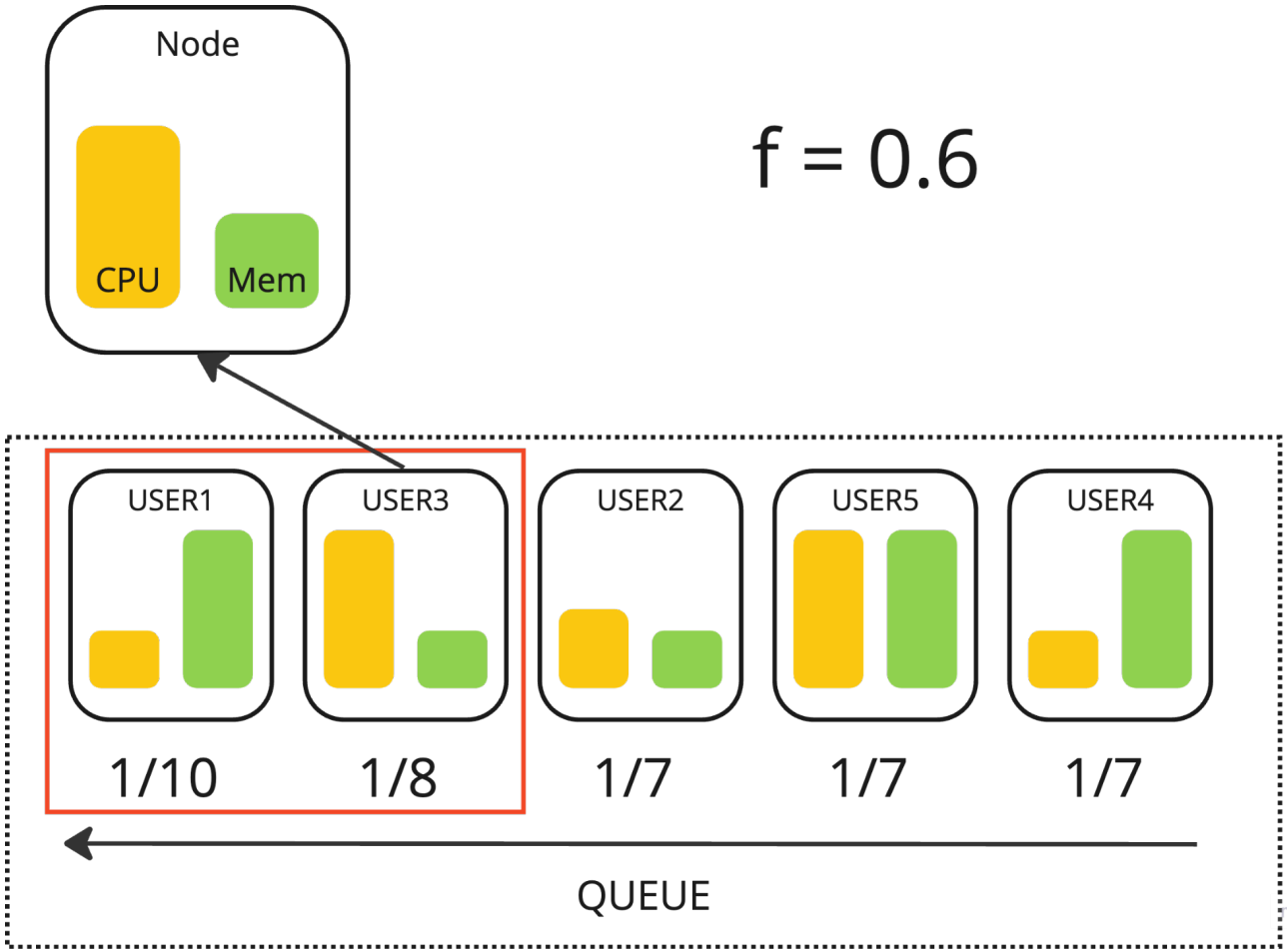
$$f = 0.6$$



Процесс принятия решения о планировании

Эксперименты

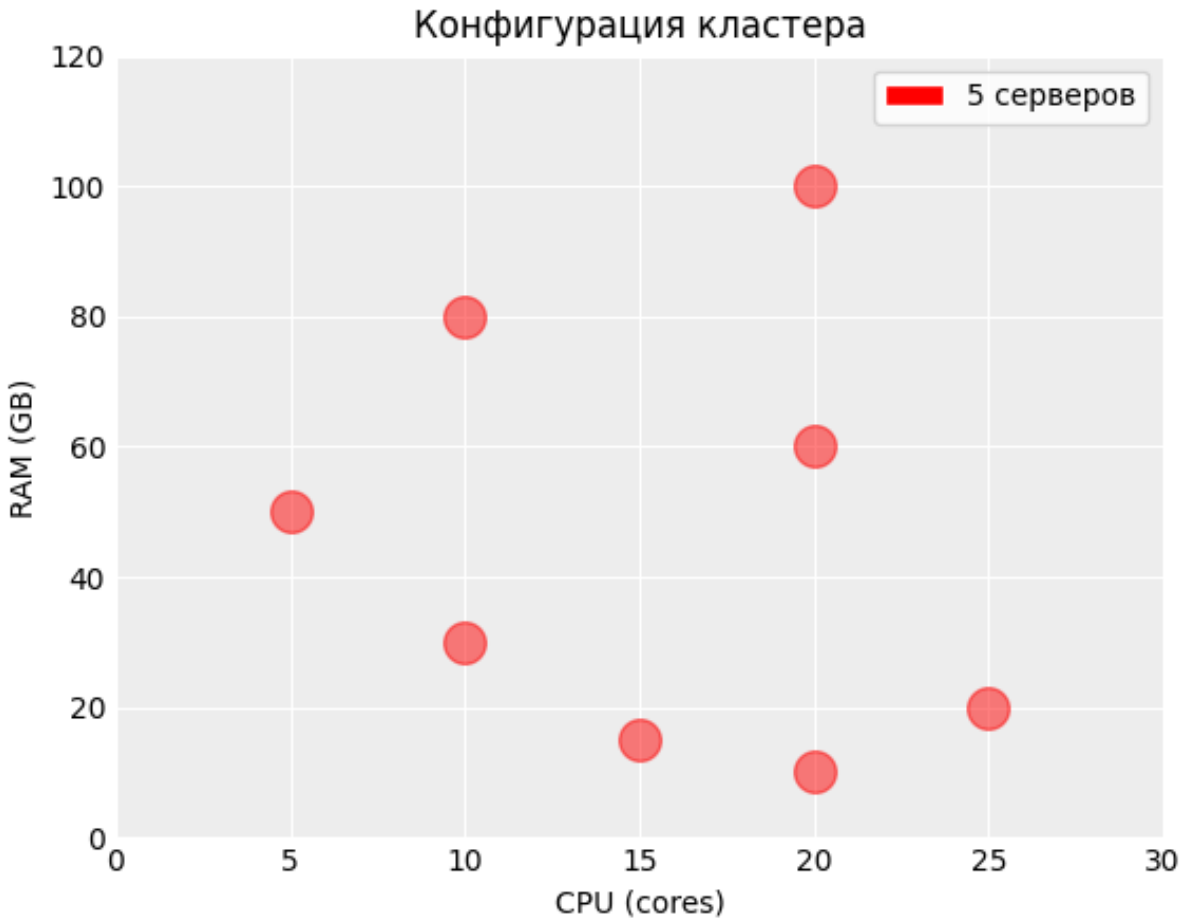
DRF & Tetris. Коэффициент честности f



Процесс принятия решения о планировании

Эксперименты

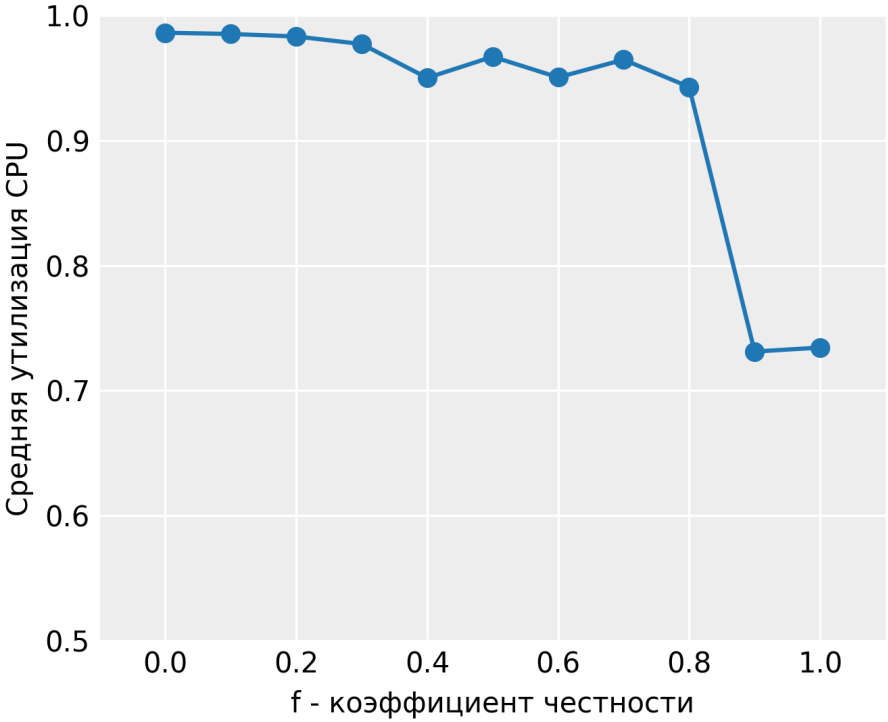
Кластер для экспериментов на честность



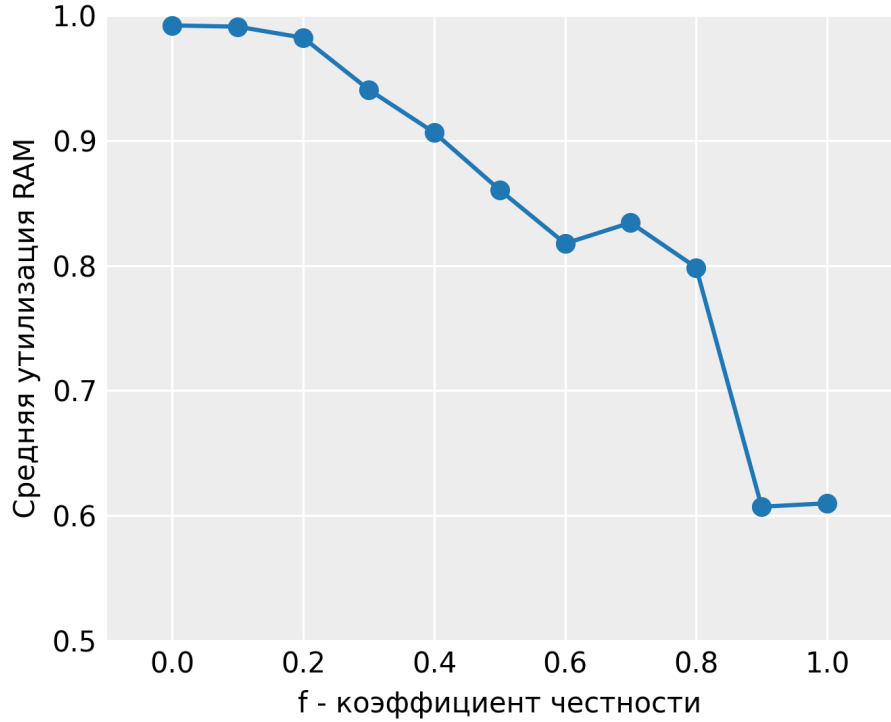
Распределение ресурсов на серверах кластера

Эксперименты

Результаты утилизации



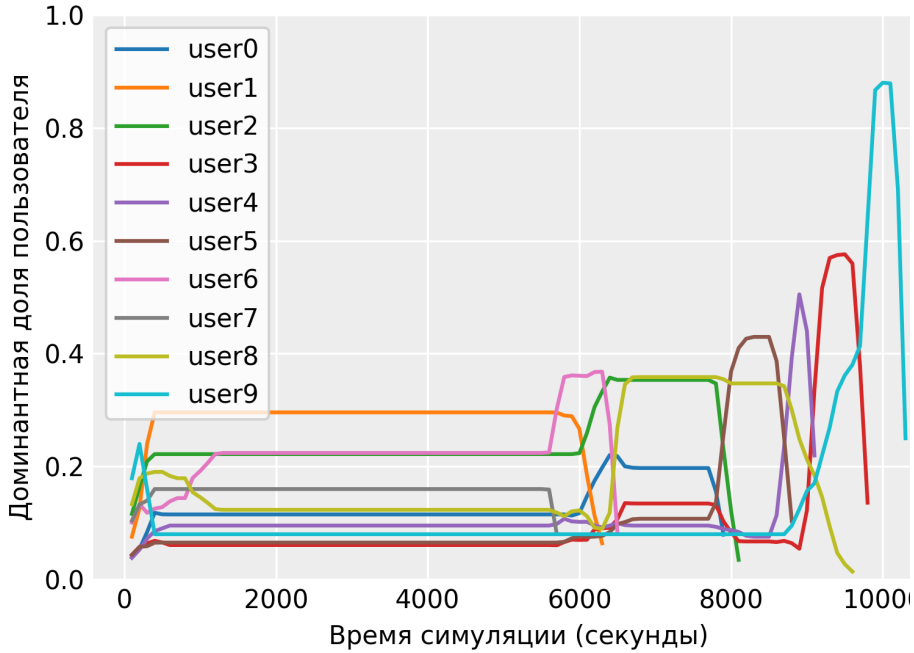
Утилизация CPU при разных f



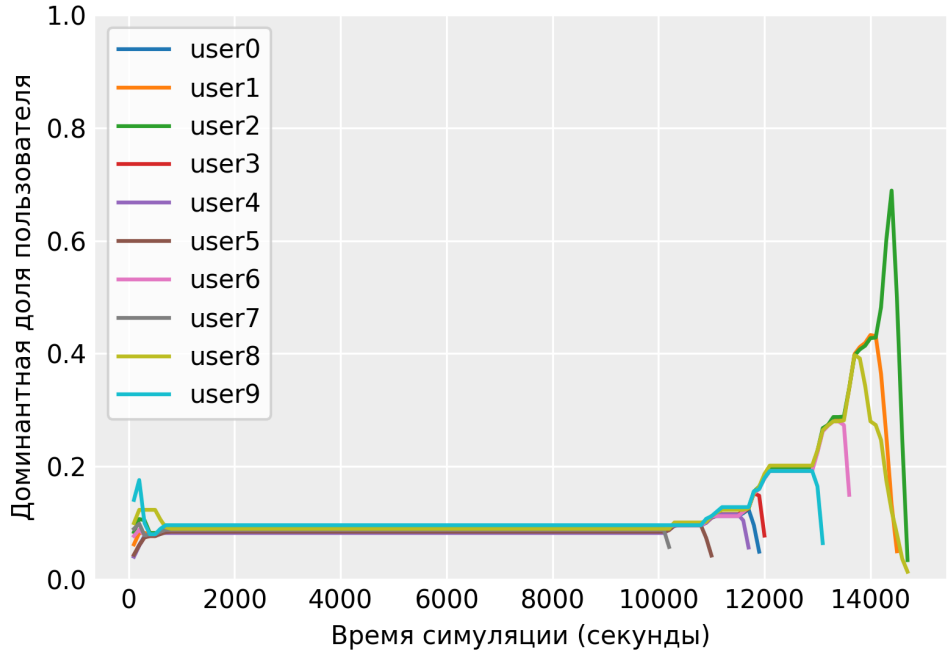
Утилизация RAM при разных f

Эксперименты

Результаты честности



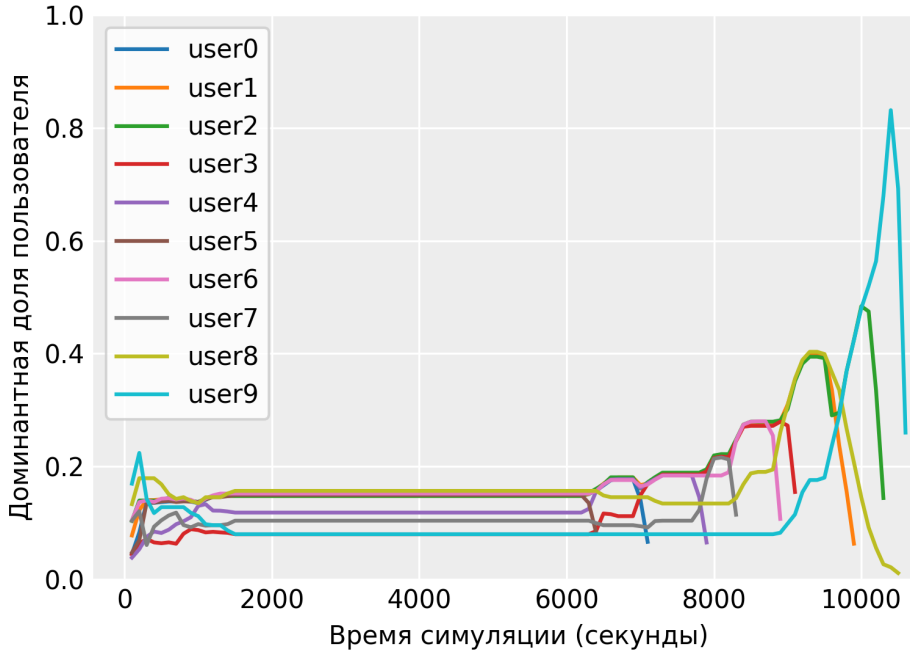
$f = 0$



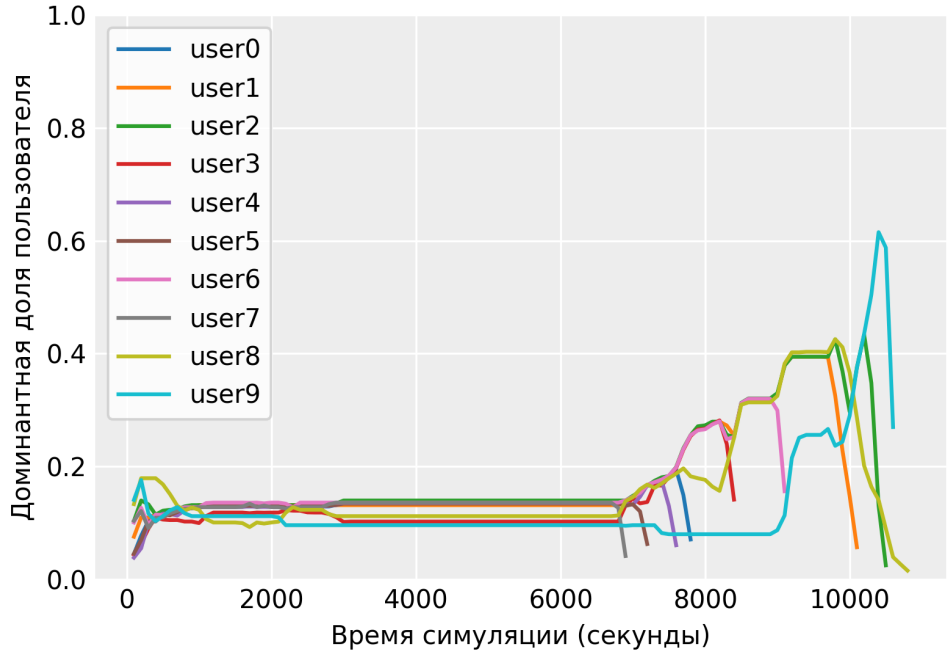
$f = 1$

Эксперименты

Результаты честности



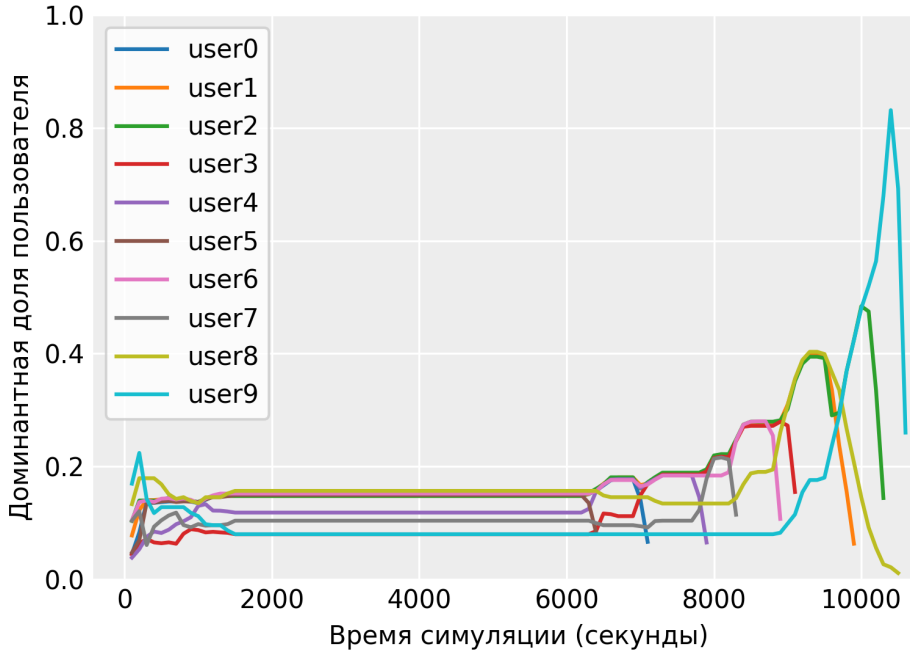
$f = 0.5$



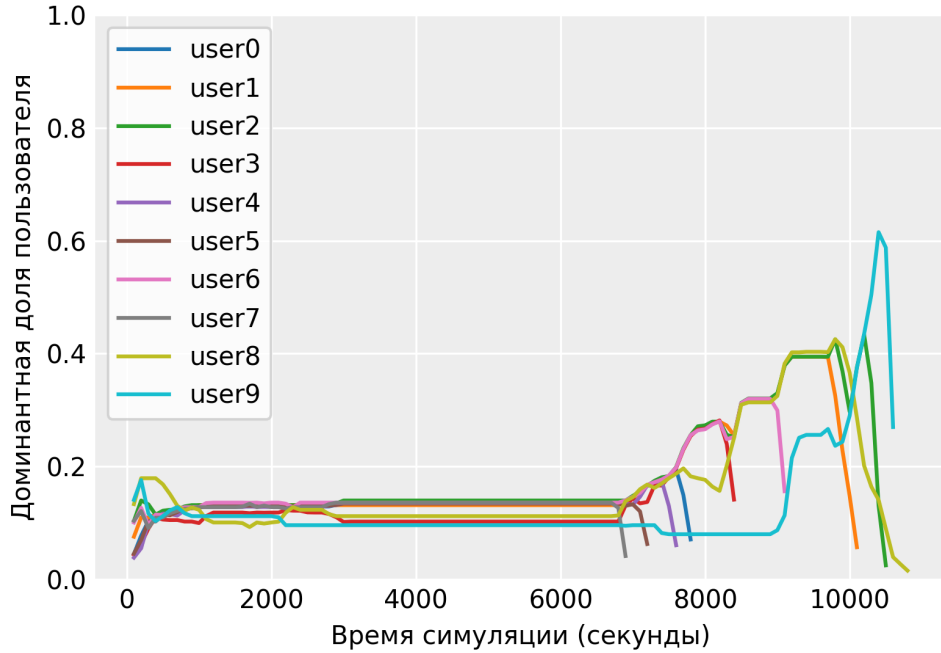
$f = 0.6$

Эксперименты

Результаты честности



$f = 0.5$



$f = 0.6$

Вывод: $f = 0.6$ – оптимальный параметр честности для такой конфигурации кластера и задач пользователей.

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.
- ✓ Поддержать несколько моделей рабочей нагрузки и планирования заданий.

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.
- ✓ Поддержать несколько моделей рабочей нагрузки и планирования заданий.
- ✓ Поддержать чтение трейсов нагрузки компаний (Google, Alibaba).

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.
- ✓ Поддержать несколько моделей рабочей нагрузки и планирования заданий.
- ✓ Поддержать чтение трейсов нагрузки компаний (Google, Alibaba).
- ✓ Поддержать адаптивный сбор статистики для анализа результатов (нагрузка на кластер, честность, статистика очереди)

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.
- ✓ Поддержать несколько моделей рабочей нагрузки и планирования заданий.
- ✓ Поддержать чтение трейсов нагрузки компаний (Google, Alibaba).
- ✓ Поддержать адаптивный сбор статистики для анализа результатов (нагрузка на кластер, честность, статистика очереди)
- ✓ Реализовать несколько популярных алгоритмов (FCFS, DRF, Tetris).

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.
- ✓ Поддержать несколько моделей рабочей нагрузки и планирования заданий.
- ✓ Поддержать чтение трейсов нагрузки компаний (Google, Alibaba).
- ✓ Поддержать адаптивный сбор статистики для анализа результатов (нагрузка на кластер, честность, статистика очереди)
- ✓ Реализовать несколько популярных алгоритмов (FCFS, DRF, Tetris).
- ✓ Доработать реализацию асинхронного ядра `dslab-core` и ускорить его работу в 2 раза.

Заключение

- ✓ Реализовать эффективный симулятор вычислительного кластера.
- ✓ Поддержать несколько моделей рабочей нагрузки и планирования заданий.
- ✓ Поддержать чтение трейсов нагрузки компаний (Google, Alibaba).
- ✓ Поддержать адаптивный сбор статистики для анализа результатов (нагрузка на кластер, честность, статистика очереди)
- ✓ Реализовать несколько популярных алгоритмов (FCFS, DRF, Tetris).
- ✓ Доработать реализацию асинхронного ядра `dslab-core` и ускорить его работу в 2 раза.

Главное достижение: предоставить возможность описывать рабочую нагрузку в виде асинхронных примитивов языка `Rust`.

Вопросы

1. Сравнение с симулятором BatSim
2. Входные профили нагрузки через yaml-файл
3. Профили по умолчанию
4. Выходные данные и monitoring
5. Ускорение async-dslab-core в 2 раза
6. Модель мгновенной честной доли с отменой задач
7. Дискретно-событийное моделирование

Приложения

Сравнение с симулятором 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).

Приложения

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"]
  },
}
```