



Разработка симулятора вычислительного кластера

Выполнил: Макогон Артём Аркадьевич, БПМИ206

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

5 июня 2024 г.



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.



- > Использовать реальный кластер долго и дорого.
- > Необходимо обеспечивать воспроизводимость результатов.



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.



- > Использовать реальный кластер долго и дорого.
- > Необходимо обеспечивать воспроизводимость результатов.

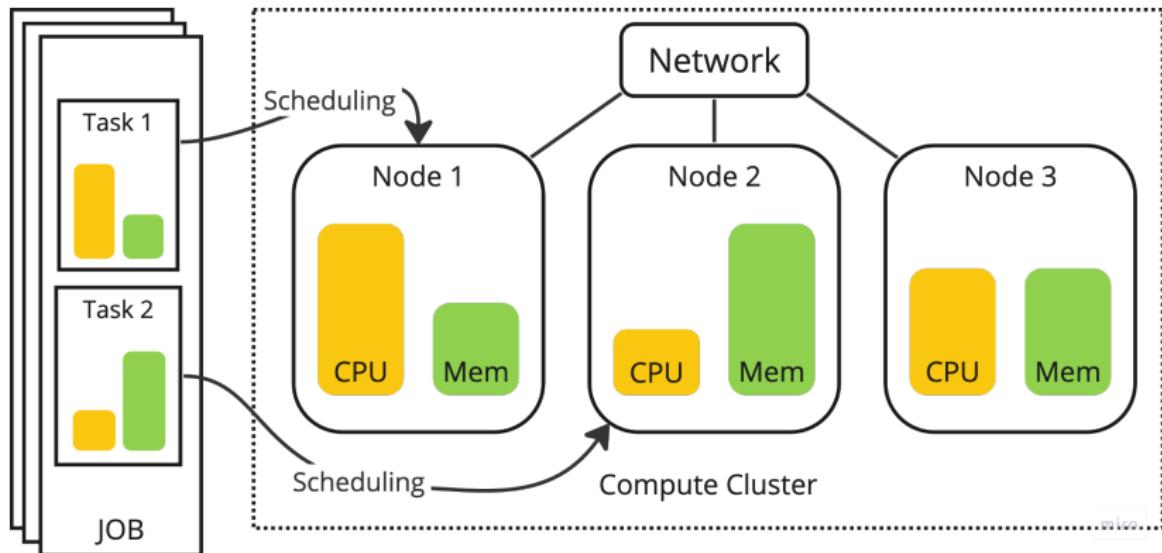


- ✓ Используются симуляторы.



Введение

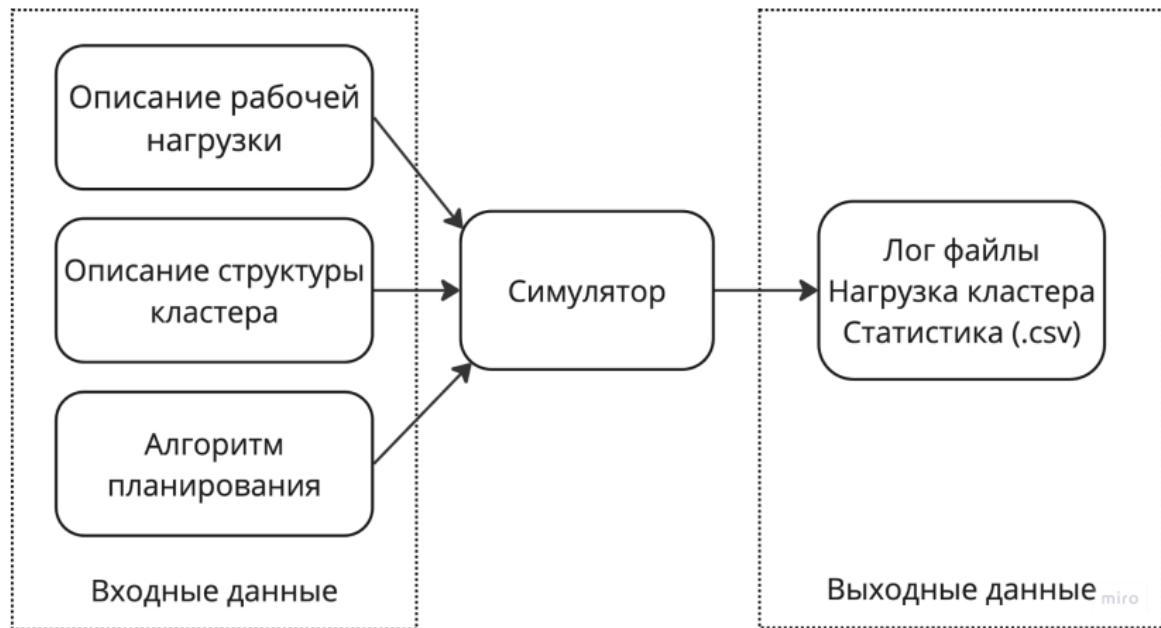
Архитектура кластера



Модель архитектуры кластера

Введение

Симуляция кластера



Входные и выходные данные симулятора



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 требования 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 требования к ресурсам

Требования к планированию:

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



Введение

Виды входных данных для симулятора

Standard Workload Format (SWF)

Custom workloads

- CPU/RAM
- 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 trivialально

Custom workloads

- CPU/RAM/disk/network
- Даны 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 trivialально
- Используется в популярных трейсах (Google, Alibaba, и т.д.)

Custom workloads

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



Введение

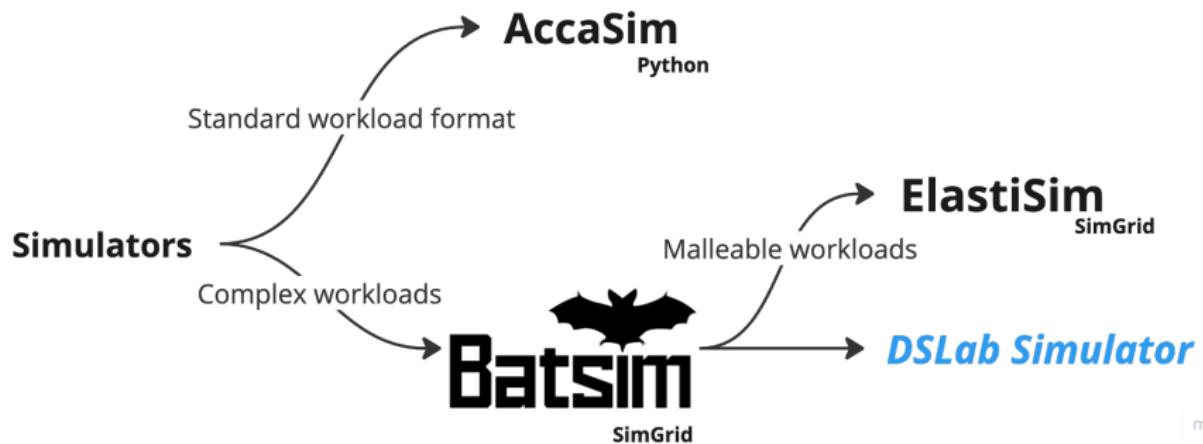
Постановка задачи

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

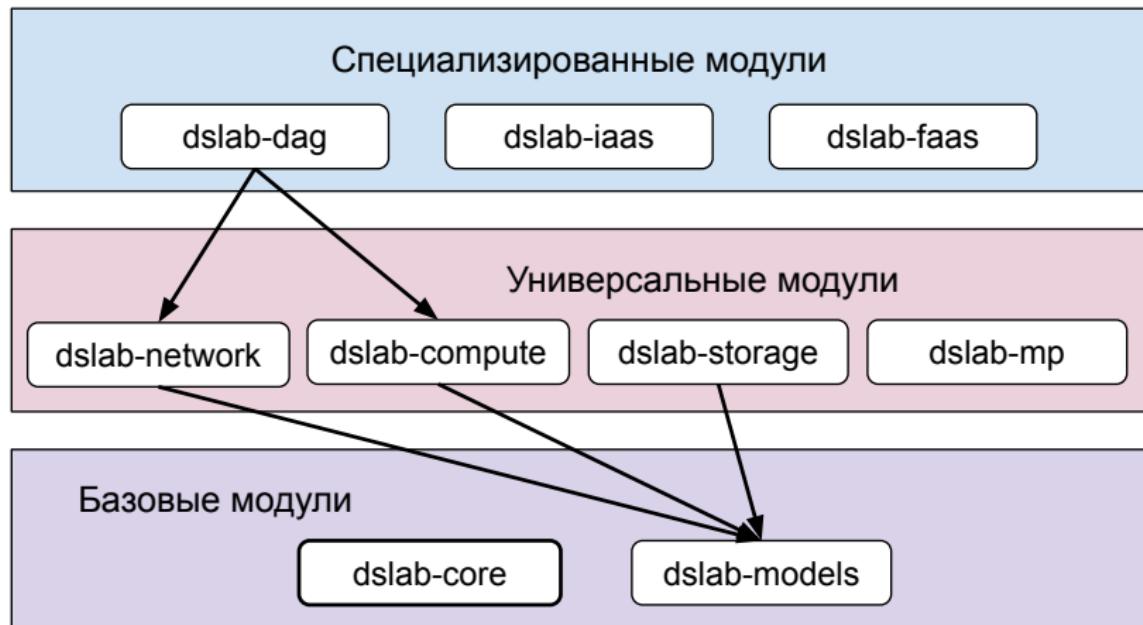


Существующие симуляторы



Реализация симулятора

Архитектура DSLab





Реализация симулятора

Асинхронное управление событиями. Комбинаторы futures

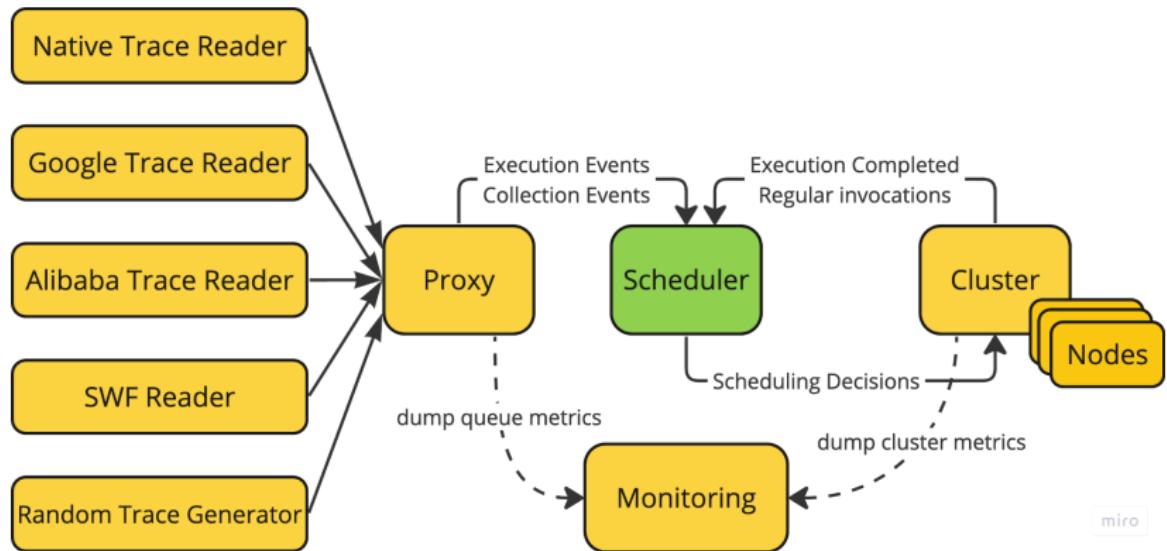
```
async fn process_task(&self, req:  
    TaskRequest) {  
let mut task = TaskInfo {req};  
  
self.download_data(&task).await;  
self.read_data(&task).await;  
self.run_task(&task).await;  
self.write_data(&task).await;  
self.upload_result(&task).await;  
}  
  
async fn run(&self, args: JobArgs){  
futures::join!(  
    self.download_data(args.node_1)  
    self.download_data(args.node_2)  
)}
```

Пример последовательных задач.

Пример параллельных задач.

Реализация симулятора

Архитектура симулятора



Архитектура симулятора



Реализация симулятора

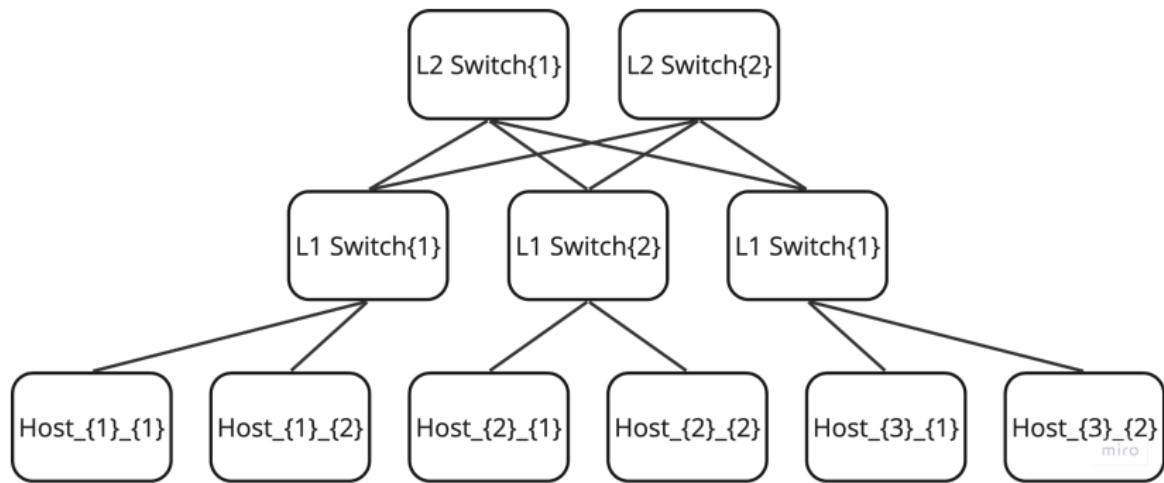
Модель кластера

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

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).



Реализация симулятора

Trait ExecutionProfile

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

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: 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

```
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

```
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

```
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

```
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 описания рабочей нагрузки через файл YAML

```
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

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
```

Пример описания базовых профилей

Пример описания профиля через комбинаторы



Сравнение с BatSim

Модель рабочей нагрузки

```
"jobs": [
    {"id": "job1", ... "res": 4, "profile": "sequence"},  
],  
  
"profiles": {  
    "homogeneous": {  
        "type": "parallel_homogeneous",  
        "cpu": 10e6,  
        "com": 1e6  
    },  
    "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]  
    },  
    "sequence": {  
        "type": "composed",  
        "repeat": 4,  
        "seq": ["simple", "homogeneous", "simple"]  
    },  
},
```

Сравнение с BatSim



Скорость

Входная задача:

- Параллельные вычисления и коммуникация всех со всеми на 5 серверах кластера.
- Кластер из 30 серверов.

Симулятор	Время работы симуляции (сек)		
	2100 задач	4300 задач	8500 задач
DSLab-Sim	2.49	5.1	9.8
BatSim	16.5	53	130

Сравнение скорости работы симуляторов



Сравнение с BatSim

Итоги сравнения

Наш симулятор:

- Предоставляет базовые профили, покрывающие функционал BatSim



Сравнение с BatSim

Итоги сравнения

Наш симулятор:

- Предоставляет базовые профили, покрывающие функционал BatSim
- Позволяет более гибко настраивать нагрузку через асинхронные функции и комбинаторы



Сравнение с BatSim

Итоги сравнения

Наш симулятор:

- Предоставляет базовые профили, покрывающие функционал BatSim
- Позволяет более гибко настраивать нагрузку через асинхронные функции и комбинаторы
- Поддерживает моделирование упаковки задач и более гибкую настройку кластера



Итоги сравнения

Наш симулятор:

- Предоставляет базовые профили, покрывающие функционал BatSim
- Позволяет более гибко настраивать нагрузку через асинхронные функции и комбинаторы
- Поддерживает моделирование упаковки задач и более гибкую настройку кластера
- Поддерживает моделирование диска



Сравнение с BatSim

Итоги сравнения

Наш симулятор:

- Предоставляет базовые профили, покрывающие функционал BatSim
- Позволяет более гибко настраивать нагрузку через асинхронные функции и комбинаторы
- Поддерживает моделирование упаковки задач и более гибкую настройку кластера
- Поддерживает моделирование диска
- Более удобен в использовании.



Сравнение с BatSim

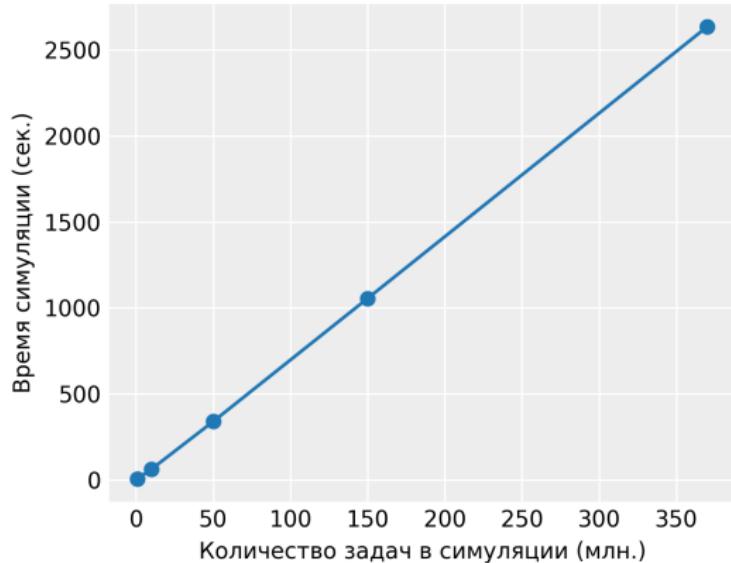
Итоги сравнения

Наш симулятор:

- Предоставляет базовые профили, покрывающие функционал BatSim
- Позволяет более гибко настраивать нагрузку через асинхронные функции и комбинаторы
- Поддерживает моделирование упаковки задач и более гибкую настройку кластера
- Поддерживает моделирование диска
- Более удобен в использовании.
- Существенно превосходит BatSim по скорости.

Эксперименты: скорость

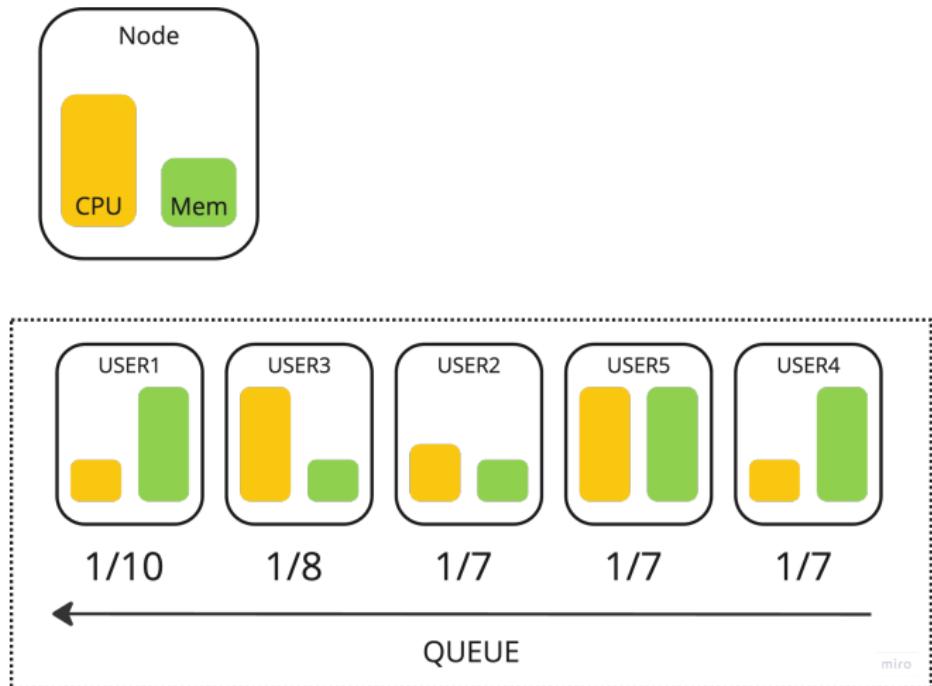
Производительность на Alibaba Trace (2018 г.)



Время работы симуляции в зависимости от количества задач

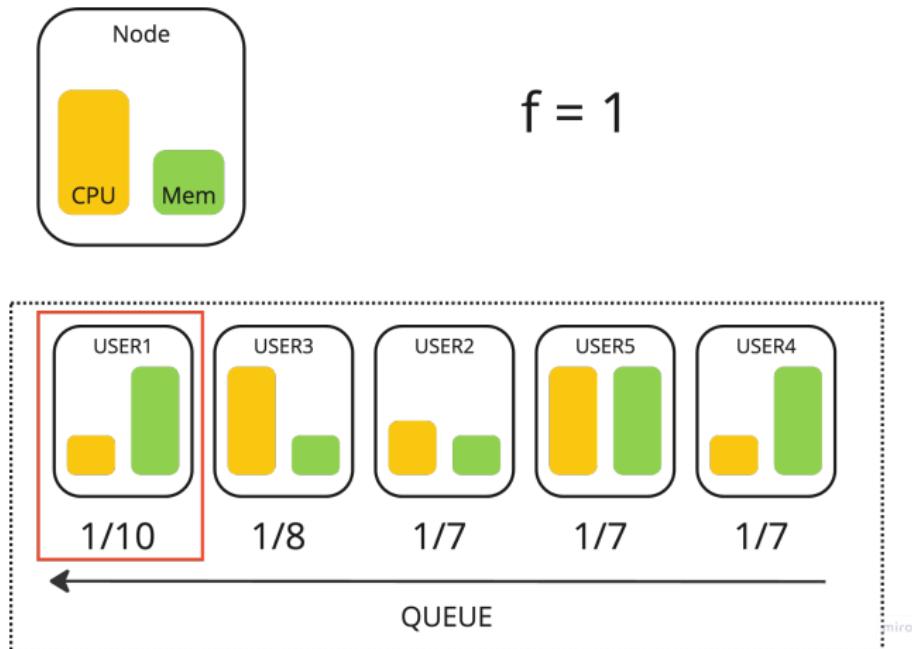
- На трейсе длиной 2.5 дня (350 млн задач, 25GB) время работы 45 минут
- Ускорение в 85 раз относительно реальной работы.
- Необходимо сортировать трейс по времени.
- Без инкрементального чтения невозможно обработать трейс.

Эксперименты: справедливость DRF & Tetris (Grandl и др., 2014).



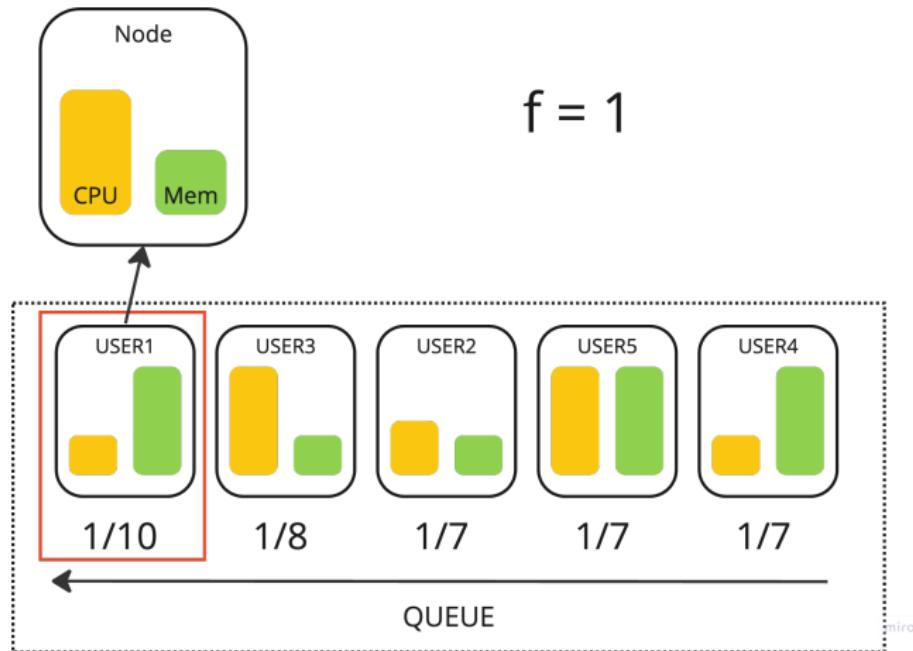
Работа Tetris. Коэффициент справедливости f .

Эксперименты: справедливость DRF & Tetris (Grandl и др., 2014).



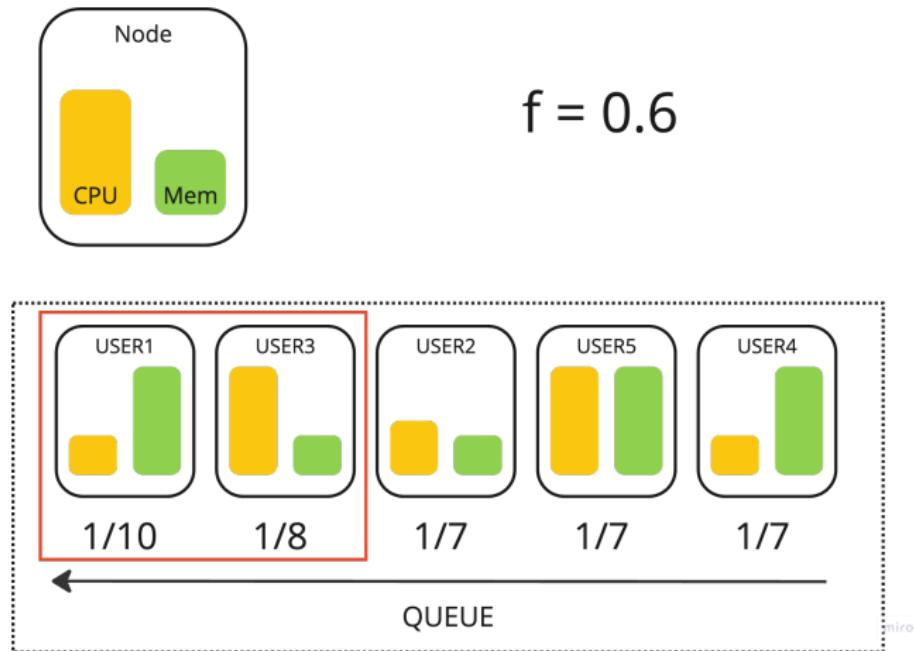
Работа Tetris. Коэффициент справедливости f .

Эксперименты: справедливость DRF & Tetris (Grandl и др., 2014).



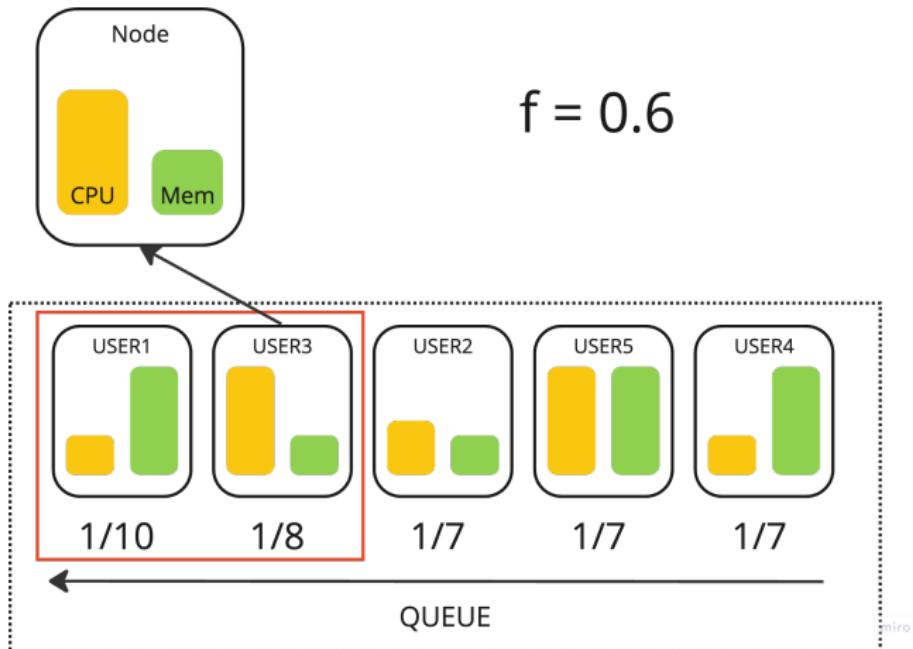
Работа Tetris. Коэффициент справедливости f .

Эксперименты: справедливость DRF & Tetris (Grandl и др., 2014).



Работа Tetris. Коэффициент справедливости f .

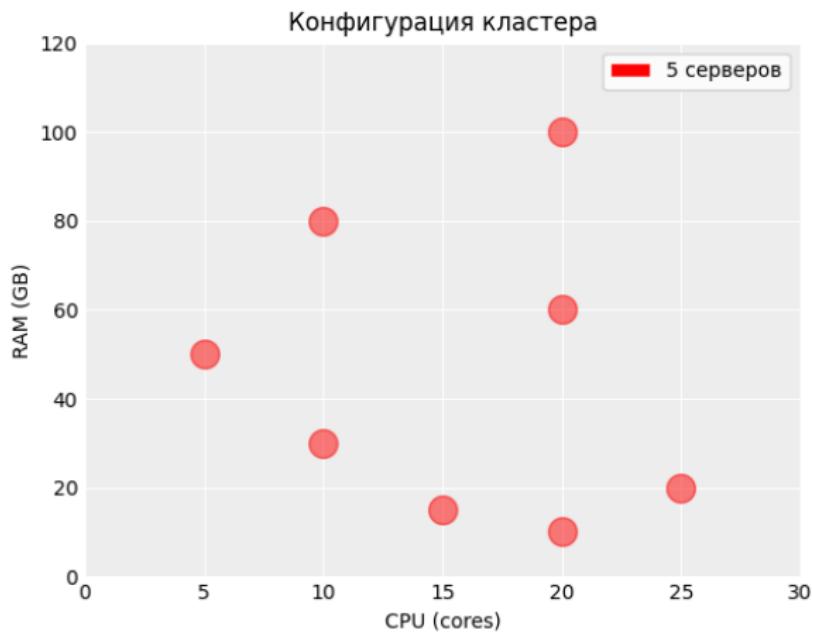
Эксперименты: справедливость DRF & Tetris (Grandl и др., 2014).



Работа Tetris. Коэффициент справедливости f .

Эксперименты: справедливость

Набор серверов



Распределение ресурсов на серверах кластера



Эксперименты: справедливость

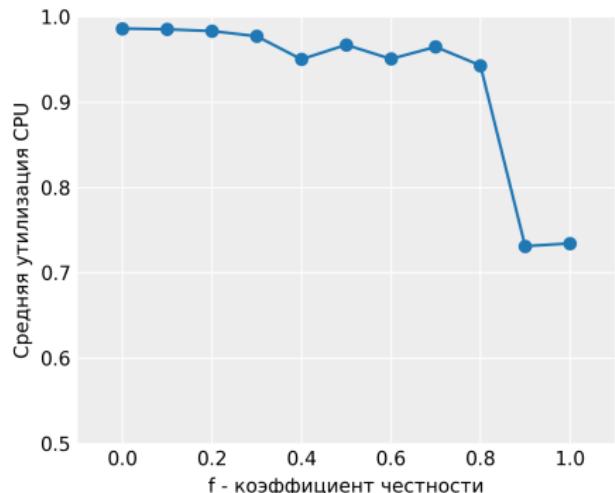
Конфигурация пользователей

Имя	CPU	RAM	Кол-во 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

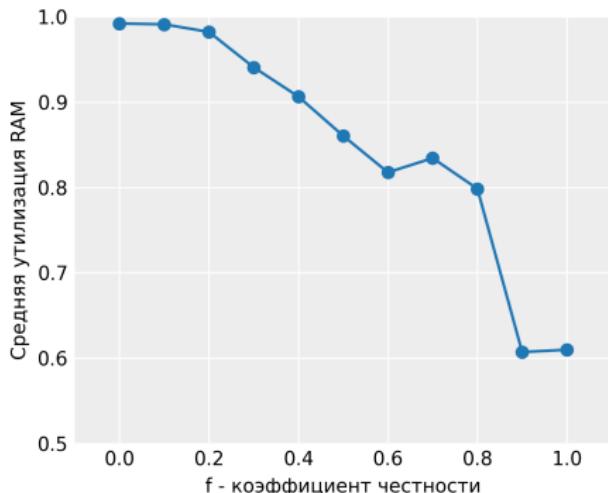
Описание синтетической нагрузки на кластер

Эксперименты: справедливость

Результаты утилизации



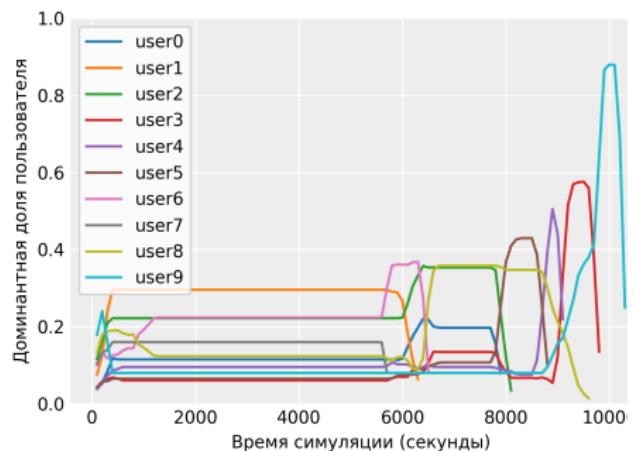
Утилизация CPU при разных f



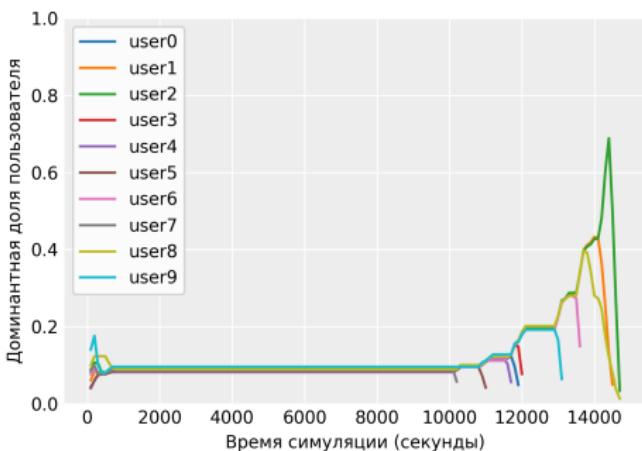
Утилизация RAM при разных f

Эксперименты: справедливость

Результаты справедливости



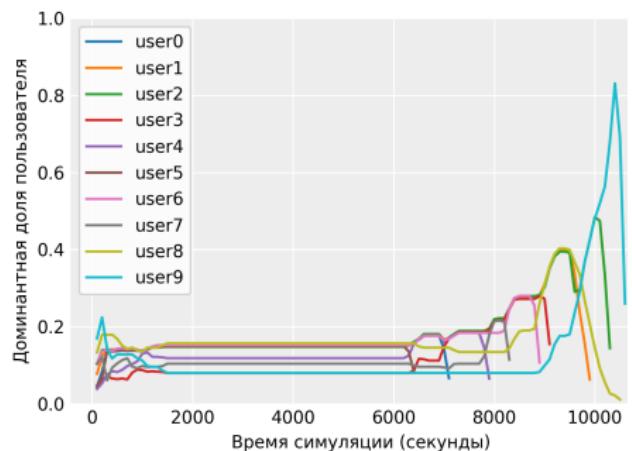
$$f = 0$$



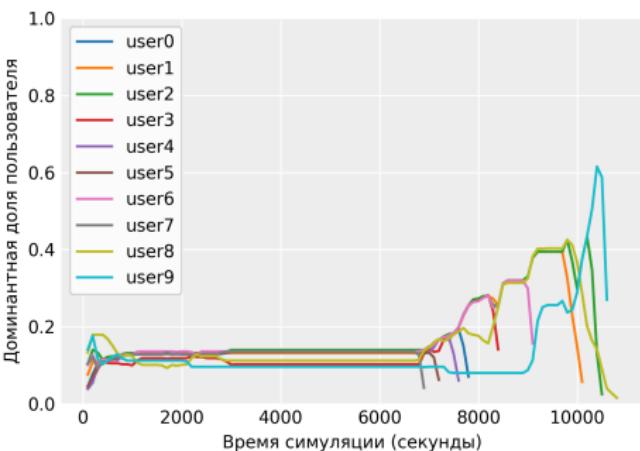
$$f = 1$$

Эксперименты: справедливость

Результаты справедливости



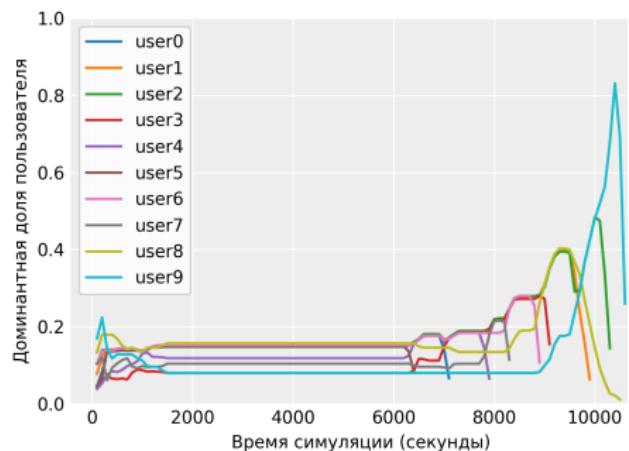
$$f = 0.5$$



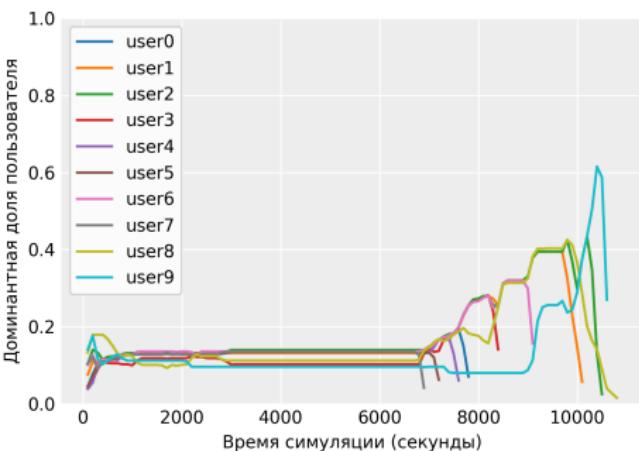
$$f = 0.6$$

Эксперименты: справедливость

Результаты справедливости



$$f = 0.5$$



$$f = 0.6$$

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).



- ✓ 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.



1. Алгоритмы справедливости. DRF & Tetris
2. Профили нагрузки по умолчанию
3. Выходные данные и monitoring
4. Ускорение async-dslab-core в 2 раза
5. Как устроено 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

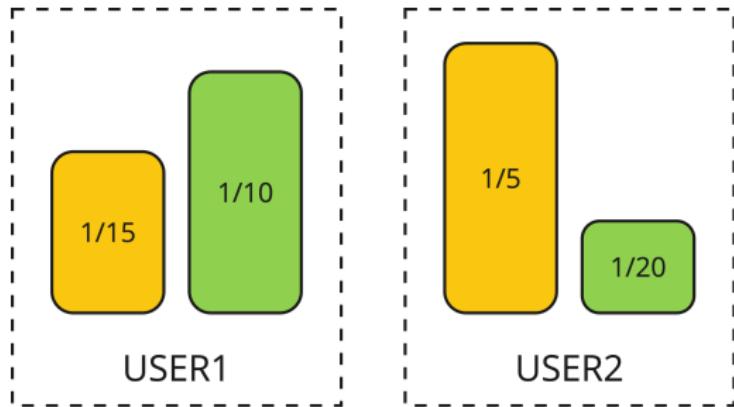
```
"jobs": [
    {"id": "job1", ... "res": 4, "profile": "sequence"}, ],
}

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



Приложения

Алгоритмы справедливости. DRF & Tetris

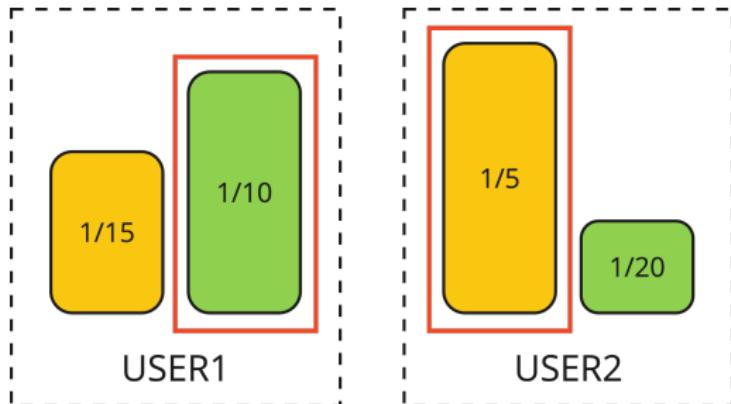


Потребление ресурсов пользователями



Приложения

Алгоритмы справедливости. DRF & Tetris



Dominant shares:

- $\text{USER1} = 1/10$
- $\text{USER2} = 1/5$

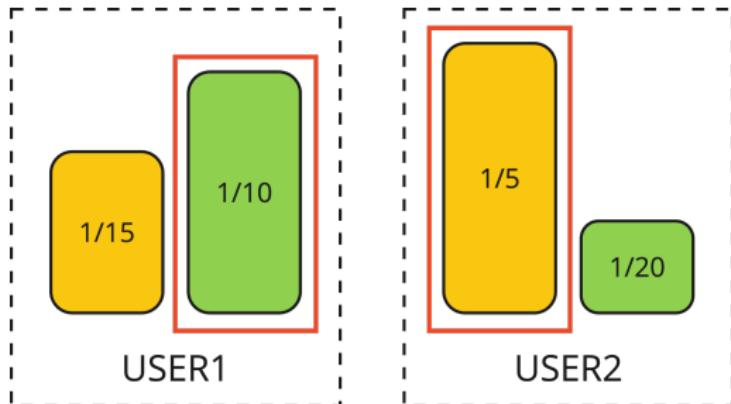
miro

Потребление ресурсов пользователями



Приложения

Алгоритмы справедливости. DRF & Tetris



Dominant shares:

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

miro



Приложения

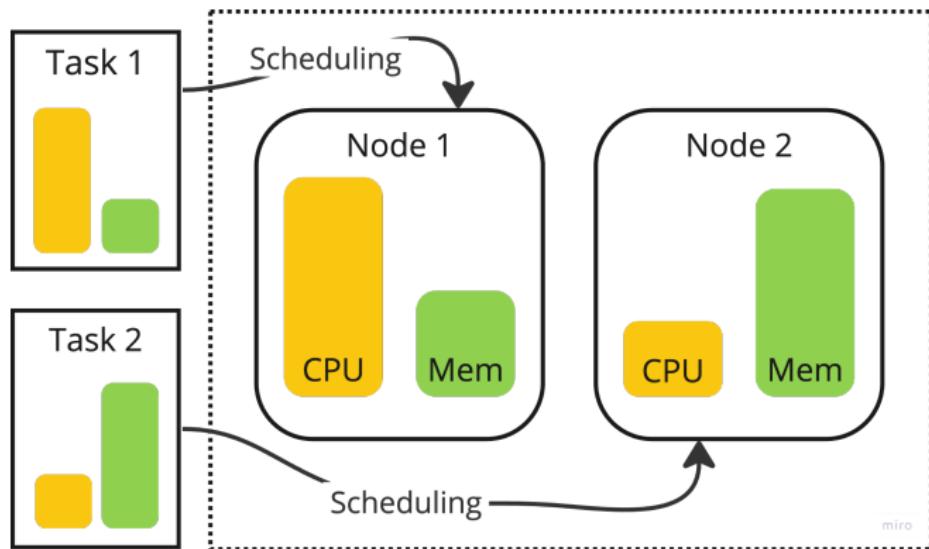
Алгоритмы справедливости. DRF & Tetris

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

Приложения

Алгоритмы справедливости. DRF & Tetris

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



Пример работы алгоритма Tetris



Приложения

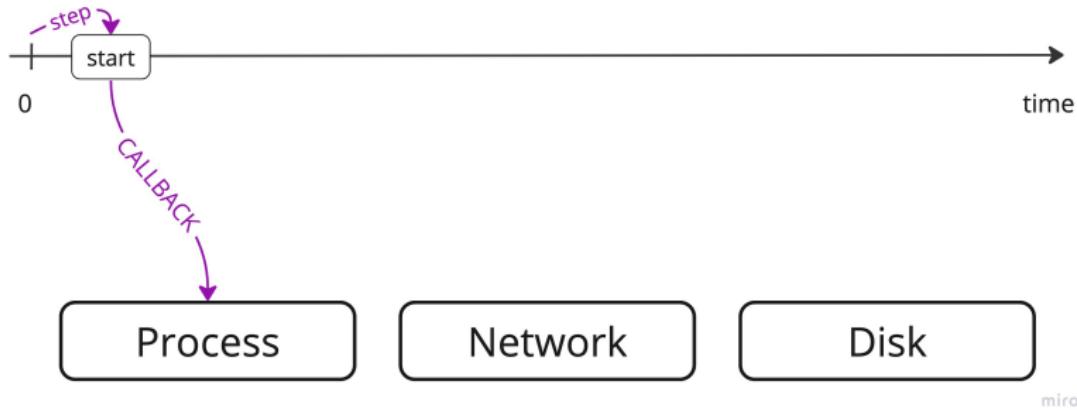
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

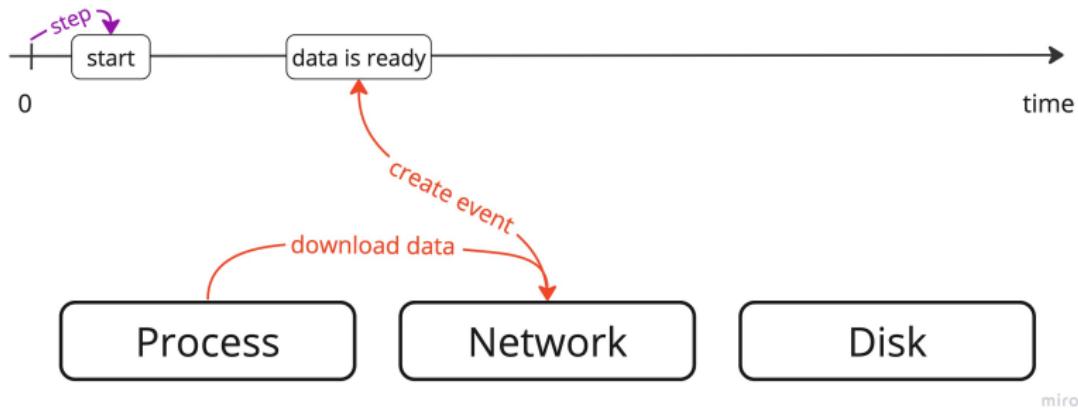
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

Как устроено дискретно-событийное моделирование

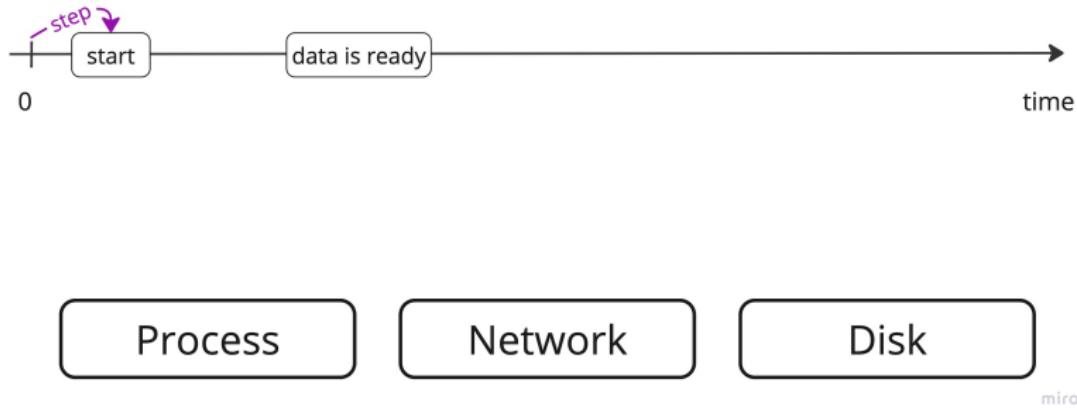


Исполнение симуляции



Приложения

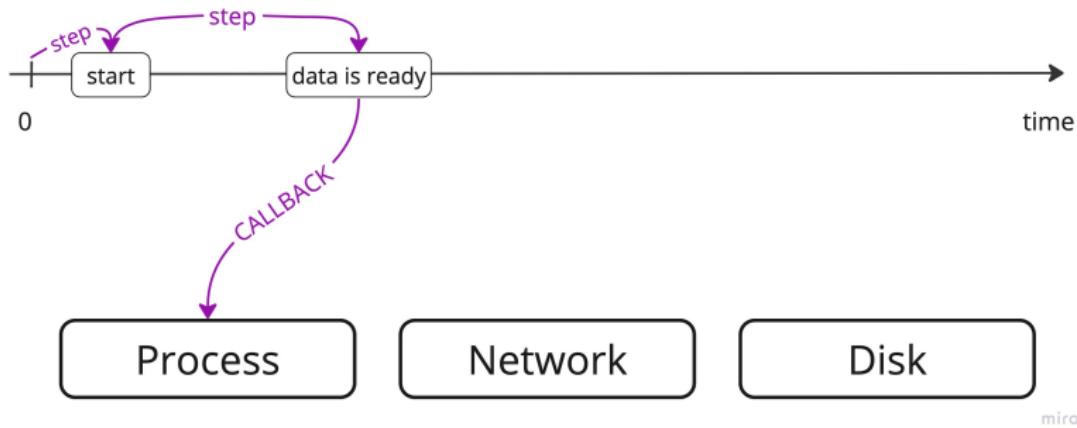
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

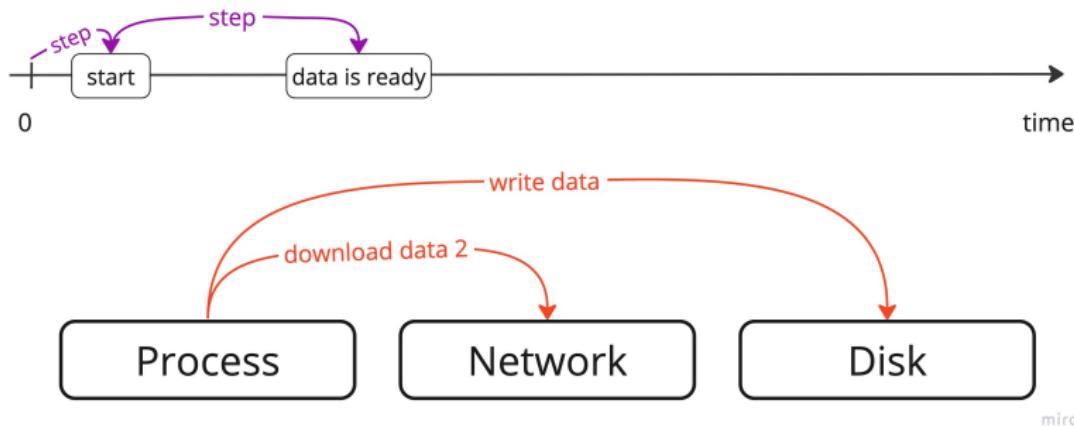
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

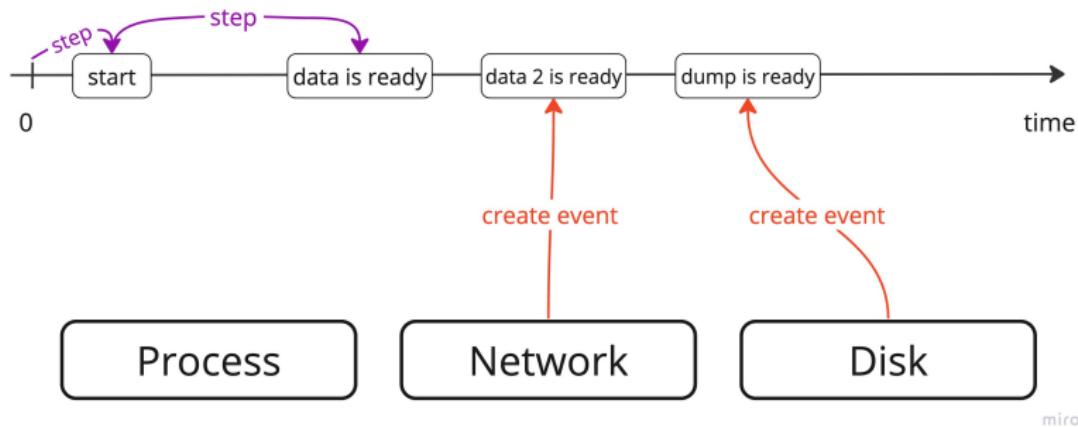
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

Как устроено дискретно-событийное моделирование

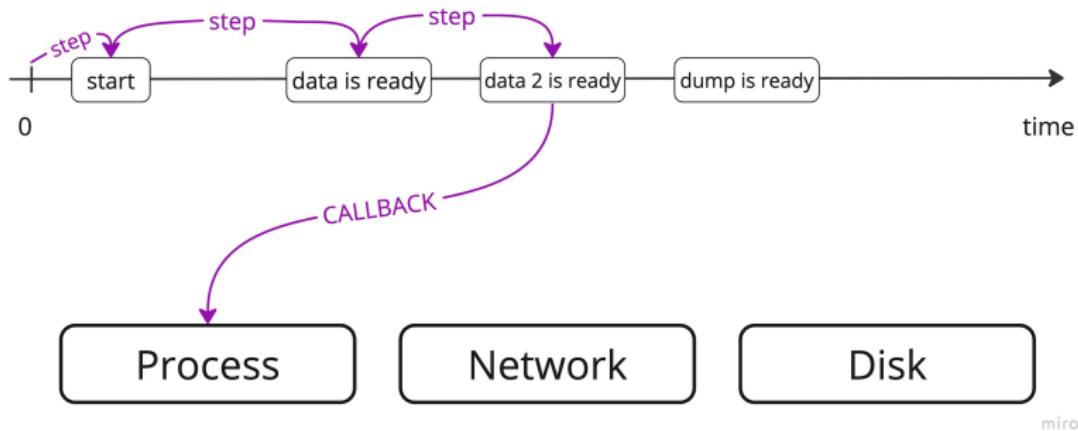


Исполнение симуляции



Приложения

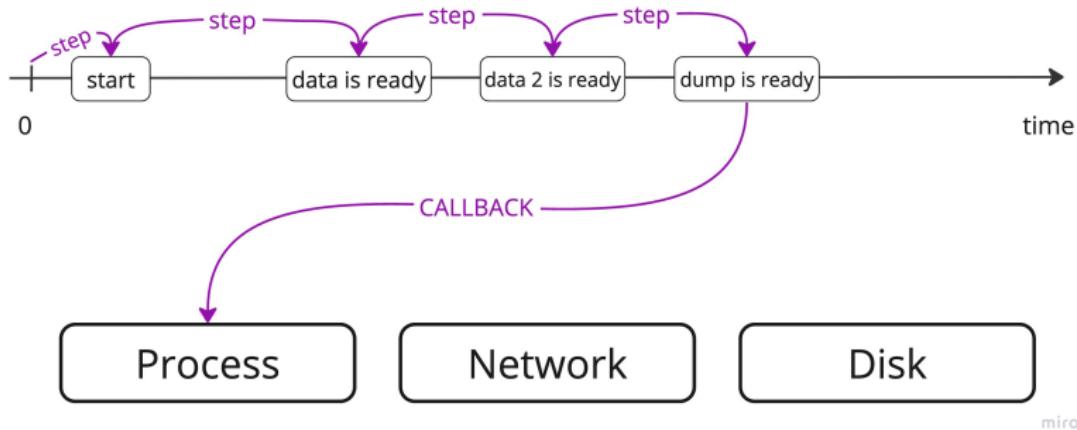
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

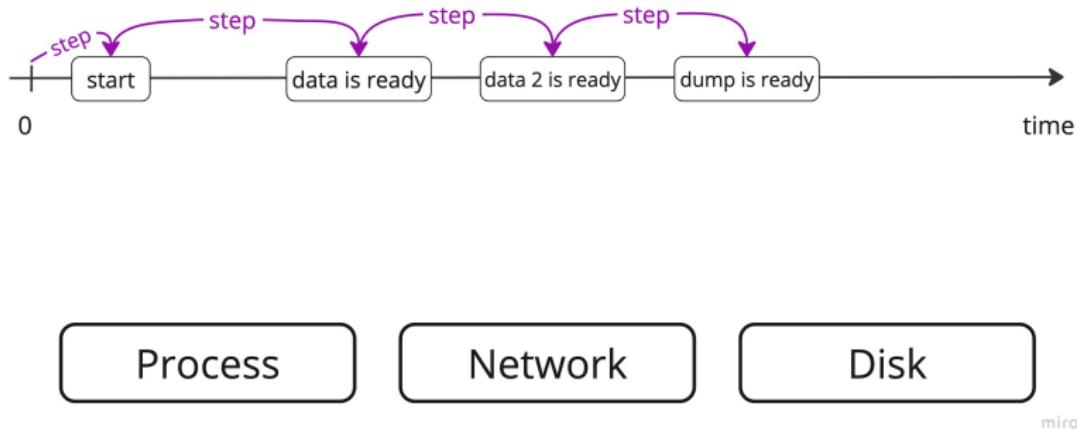
Как устроено дискретно-событийное моделирование



Исполнение симуляции

Приложения

Как устроено дискретно-событийное моделирование



Исполнение симуляции