# Development Of Compute Cluster Simulator

Artem Makogon
*Faculty of Computer Science*
*National Research University Higher School of Economics*
Moscow, Russia
aamakogon@edu.hse.ru

***Abstract*—Algorithms for scheduling tasks on compute clusters are the subject of active research. It is quite expensive to conduct such studies, 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**

## I. Introduction

Compute clusters consisting of a set of servers with various resources (processor, memory, disk) are widely used for complex calculations and processing large amounts of data in science and business. On a large industrial cluster, thousands of batch jobs from different users can be running simultaneously, and some jobs may be in a queue waiting for resources to be allocated. The cluster manager or scheduler manages the launch of tasks and the allocation of resources between them. The cluster manager's work is based on job scheduling algorithms, the main purpose of which is to maximize the efficient use of cluster resources. These algorithms are the subject of active research. It is long and expensive to conduct such studies on a real cluster, so a computer model (simulator) is needed to quickly test a hypothesis or conduct comparative testing of different algorithms on some cluster load history. An important feature of such simulators is the ability to flexibly adjust the cluster load model in order to get the most realistic results. Such a simulator can also be used in the educational process to familiarize students with such systems and the tasks arising in them.

In research and development environments, simulators offer a cost-effective way of evaluating new scheduling algorithms and strategies before deploying them in production clusters. Researchers can test various workload scenarios, fine-tune parameters, and assess algorithmic performance under different conditions. This iterative process enables researchers to refine their algorithms and identify optimal configurations for specific use cases.

The new simulator will be developed based on DSLab[1], which is one of the most advanced simulators[2][3]. There are already developed models of processors, disks and network that can be used as the basis of a new simulator.

## II. Literature Review

### A. Workload Model

In order to simulate cluster workload and test scheduling algorithms, the workload model is required. The standard workload format (SWF)[4] is widely used in the literature and is supported by many simulators. The SWF provides quite a few information about workload. The key information about single workload unit «Job» includes arrival time, execution time, priority and resource requirement. The only assumption we can make about the workload is that job consumes all the given resources during the execution time. Most popular workload traces are provided in similar format (for example Google trace[5]).

Another approach to workload modeling is based on more complex simulation, where job time is calculated based on workload structure and available resources. The main advantage of this way of simulating workload is that all the details of the job execution can be taken into account and used while scheduling. Also, the cluster utilization can be calculated with more precision. The main disadvantage is that it is quite difficult to find open-source data for such simulations because job details are usually confidential and never published.

### B. Simulators

Numerous simulators to evaluate cluster scheduling algorithms were developed in previous work, and most of them fall into one of the two following categories: (1) simulators built from scratch, and (2) simulators based on platform simulation framework. Another way of grouping simulators is by the workload type they support: (1) simulators that support workload traces in the SWF or similar formats, and (2) simulators that support complex workload description.

Simulators built from scratch are usually designed for one purpose and can hardly be applied in different scenarios. The representative example of such simulator is AccaSim[6]. It is based on discrete event modeling and contains some pre-defined resource managers and scheduling policies. It is quite easy to use, but does not support complex configurations. In contrast to AccaSim, our simulator will be able to simulate arbitrary workload scenarios, that will be defined by users. In addition to this, our simulator is modular as it is based on DSLab. Users will be able to connect modules written by other researchers within the DSLab framework and use them in their experiments.

For example, DSLab provides shared network models, that can simulate the variety of network topologies.

Simulators based on platform simulation framework are much more flexible. The most popular framework is SimGrid[7]. The detailed comparison between SimGrid and DSLab can be found in the DSLab documentation[3].

The batch-system simulator Batsim[8] is based on SimGrid and allows developers to integrate algorithms in several languages based on inter-process communication. Batsim provides the interface to configure custom workload types and scenarios, but they are limited by several given options and possible sequential combinations of them. Our simulator will extend this approach for the arbitrary workload definition using asynchronous programming primitives. In addition to this we will aim to achieve higher performance, because our simulator will be implemented as Rust crate and native Rust integration is supposed to be faster than inter-process communication.

The Elastisim[9] simulator is also based on SimGrid and focused on malleable workloads. It allows to simulate the dynamic changes in the number of resources allocated to a job. The key feature of Elastisim is high precision of the simulated deep learning workload, which it is designed for. In addition to compute simulation Elastisim is also focused on providing simulation for I/O operations. Authors pay special attention to the fact that I/O may become a bottleneck as the amount of transferred data is constantly increasing. The structure of Elastisim job is quite complicated. Execution consists of several phases and can be configured using JSON file by combining the pre-defined set of operations: cpu, gpu, I/O and disk operations. This is the most advanced workload simulator of all the reviewed sources. We will aim to provide even more opportunities for detailed workloads description.

## III. Methods

### A. DSLab

DSLab is a software framework for simulation and testing of distributed systems. The framework is organized as a set of loosely coupled software modules with an API. This approach allows users of the framework (researchers, developers, teachers) to flexibly assemble solutions from modules for their own purposes. The framework is written in Rust, which provides high performance and safety.

### B. Compute Cluster Model

As a basis for the cluster model, we will use the existing models of processors, disks and network from DSLab. The cluster contains the set of nodes (or servers), which are connected by a network. The cluster scheduler is responsible for placing workload units (jobs) on the nodes. Jobs can take either a fixed amount of resources or a variable amount within a certain range.

### C. Simulator Architecture

The simulator will be able to accept a variety of different inputs, such as SWF, Google Trace or more complex workload scenarios (described in the next section). The architecture of the simulator is shown in the Figure 1:

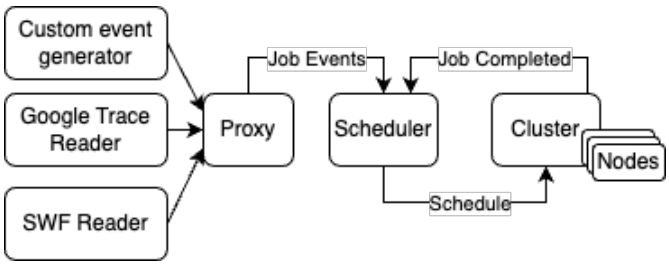


Figure 1: Simulator architecture

The workload generator is responsible for parsing the input data and pass it to a `Proxy`. The `Scheduler` component is responsible for placing jobs on the servers. It is implemented by the user. The cluster component is responsible for simulating the behavior of servers and network.

By combining the information gathered on `Proxy` and `Cluster` components, the simulator will be able to provide a detailed report on the cluster load and the execution of jobs. This report can be used to evaluate the performance of the scheduling algorithms. For example, the following metrics can be calculated: the average time of job execution, the average time of job waiting in the queue, the average cluster utilization, etc.

### D. Asynchronous events control

DSLab is based on discrete event simulation, which is a well-known approach to modeling distributed systems. Furthermore, DSLab provides a core feature to manage events asynchronously[10] using Rust async primitives, thereby making possible to write expressive code within the simulation. This feature is about to get released as stable, so new simulator will rely on it.

The user will be able to write a task using Rust code, which will consist of several asynchronous operations. For example, the following code snippet demonstrates how to write a task that consists of sequential asynchronous operations:

```rust
async fn run(&self, args: JobArgs) {
    self.download_data(&args).await;
    self.read_data(&args).await;
    self.run_task(&args).await;
    self.write_data(&args).await;
    self.upload_result(&args).await;
}
```

Figure 2: Example of sequential tasks execution

On contrast with the existing simulators, where the workload is described using JSON, DSLab simulator will allow using combiners from the `futures` crate[11]. The following code snippet demonstrates how to write a task that consists of parallel asynchronous operations:

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

Figure 3: Example of parallel tasks execution

The combination of given methods will allow to simulate

complex workloads, which are not supported by the existing simulators. This will be achieved by implementing Rust trait (code snippet 4) separately for different types of Jobs.

```
pub trait Job {
    async fn run(self: Box<Self>,
        job_context: JobContext);
}
```

Figure 4: Job trait description

In this case `JobContext` will contain all the necessary functions to interact with cluster and other assigned nodes for the jobs (in order to exchange data or to perform some operations). The semantics and the structure of the `JobContext` will be refined during the development of the simulator.

## IV. Expected Results

The main result of the work will be a high-performance simulator that will allow to configure arbitrary workload scenarios and evaluate the performance of scheduling algorithms. Users will be able to choose between several approaches for describing cluster and workloads: SWF, Google Trace or custom workload scenarios.

Due to its flexibility, the new simulator can be customized individually for an existing cluster and will allow researchers to have high confidence in the results obtained in the simulation.

## V. Conclusion

Since the number of calculations performed on clusters is continuously increasing, special attention is paid to the effectiveness of planning algorithms. The potential improvement of cluster utilization by units of percent will save huge amounts of money. The development of a simulator for compute clusters is an important task, as it allows to test the performance of scheduling algorithms without the need to use a real cluster.

## References


[1] *Dslab repository*. [Online]. Available: https://github.com/osukhoroslov/dslab.

[2] *Dslab architecture*. [Online]. Available: https://docs.google.com/document/d/12CcGpdulqMJppAYoNrOdpIWZYEa3flXn9JMX5fdyAnY/edit.

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

[4] *The standard workload format specification*. [Online]. Available: https://www.cs.huji.ac.il/labs/parallel/workload/swf.html.

[5] J. Wilkes, "Google cluster-usage traces v3", Google Inc., Mountain View, CA, USA, Technical Report, Apr. 2020, Posted at https://github.com/google/cluster-data/blob/master/ClusterData2019.md.

[6] C. Galleguillos, Z. Kiziltan, A. Netti, and R. Soto, *Accasim: A customizable workload management simulator for job dispatching research in hpc systems*, 2018. arXiv: 1806.06728 [cs.DC].

[7] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms", *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. DOI: 10.1016/j.jpdc.2014.06.008. [Online]. Available: https://inria.hal.science/hal-01017319.

[8] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator", in *20th Workshop on Job Scheduling Strategies for Parallel Processing*, Chicago, United States, May 2016. [Online]. Available: https://hal.science/hal-01333471.

[9] T. Özden, T. Beringer, A. Mazaheri, H. M. Fard, and F. Wolf, "Elastisim: A batch-system simulator for malleable workloads", in *Proceedings of the 51st International Conference on Parallel Processing*, ser. ICPP '22, Bordeaux, France: Association for Computing Machinery, 2023, ISBN: 9781450397339. DOI: 10.1145/3545008.3545046. [Online]. Available: https://doi.org/10.1145/3545008.3545046.

[10] *Implementation of asynchronous programming support for the dslab framework*. [Online]. Available: https://nogokama.github.io/ThirdYearCoursework/report.pdf.

[11] *Rust crate «futures»*. [Online]. Available: https://crates.io/crates/futures.


Word Count: 1715