AleaNG

Job Scheduling Simulator

This simulator builds upon previous Alea simulator.
My work was kindly supported by:
CESNET, zájmové sdružení právnických osob
Generála Píky 430/26
160 00 Praha 6
Czech Republic

Dalibor Klusáček
klusacek@cesnet.cz

Table of Contents

Downloading the AleaNG

AleaNG is distributed on GitHub: https://github.com/aleasimulator/AleaNG

You can either fork it or download it as a zip archive. In the following text, we assume the latter (beginner guide scenario).

Compilation

Compilation on Linux systems

On Linux-like systems, extract the downloaded zip folder and run the following commands:

```
mkdir -p out

javac -encoding UTF-8 -cp "lib/simjava.jar:lib/gridsim.jar:lib/*" -d out
$(find src -name "*.java")
```

The `mkdir -p out` will create `./out` folder for the compiled bytecode. Next, `javac...` command will compile all Java classes of AleaNG using the modified `simjava.jar` and `gridsim.jar` libraries (**their ordering is important, so keep it this way**).

Compilation on Windows systems

On Linux-like systems, extract the downloaded zip folder and run following commands (in **PowerShell**: `powershell.exe`):

```
mkdir out

$files = Get-ChildItem -Recurse -Filter *.java src | ForEach-Object {
$_.FullName }

javac -d out -cp "lib\simjava.jar;lib\gridsim.jar;lib\*;." $files
```

It is important to **run these commands in PowerShell**, since the standard cmd.exe uses a different syntax.

Running Alea

Running AleaNG on Linux systems

On Linux-like systems, navigate to the folder where you have AleaNG (and the newly created `./out` folder) and execute the following command (i.e., call java with the correct classpath and your main class `ExperimentSetup`):

```
java -cp "out:lib/simjava.jar:lib/gridsim.jar:lib/*"
xklusac.environment.ExperimentSetup
```

Running AleaNG on Windows systems

To run your program, you just need to call java with the correct classpath and your main class name (`ExperimentSetup`).

```
java -cp "out;lib\simjava.jar;lib\gridsim.jar;lib\*"
xklusac.environment.ExperimentSetup
```

Simulation Start

If sucessfull, java command shown above will start the AleaNG simulator. AleaNG will read `configuration.properties` file to setup the simulation. By default, it will read **Example 1** stored in the `./examples/Example1/` folder. You should see something like this in the output:

```
-----------------------------------
|    Alea NG (Next Generation)    |
-----------------------------------
Working directory: /home/klusacek/AleaNG-compile
result root: results/2025-11-04-10-58-06
Initialising SimJava2 modified by Dalibor Klusacek (xklusac@fi.muni.cz).
-------------------------------------
Starting Queue Loader ...
...
-------------------------------------
Creating cluster cl_dedicated with 4 nodes. Each node has 1 CPUs and 4
GPUs.
Creating cluster cl_normal_1 with 64 nodes. Each node has 1 CPUs and 4
GPUs.
-------------------------------------
...
Starting simulation using Alea NG (Next Generation)
Starting GridSim version 5.0
Entities started.
...
Start time of workload is UnixStartTime: 1735689600, [01:00 01-01-2025]

================== !!! WARNING !!! ==================
AleaNG will read all 1943 jobs from workload file.
(Shortening the number specified in config: 10000 jobs)
================== !!! WARNING !!! ==================
```

```
>>> 10 arrived, in queue/schedule 4 jobs, requiring 61 CPUs, held jobs 0
running 6 jobs, free CPUs 3, free RAM 17408 GB, free GPUs 272, tried jobs
0, Day: 01-01-2025 [01:00] SimClock: 19
...
```

Visualization Caveats

By default, simulator will try to draw charts once the simulation completes. Therefore, this feature must be turned off if AleaNG is to be executed in a non-graphical environment (e.g., using a (remote) text terminal (putty) or in a batch-mode on a computing cluster). To turn visualization off, check `configuration.properties`: and make sure that `draw_chart` is set to false:

```
draw_chart=false
```

Memory Requirements

If you plan to run large simulations, make sure to specify enough Java heap space for the JVM. To do so, just add, e.g., `-Xms1024m -Xmx4096` to your `java` command.
Example for Linux system:

```
java -Xms1024m -Xmx4096m -cp "out:lib/simjava.jar:lib/gridsim.jar:lib/*"
xklusac.environment.ExperimentSetup
```

Simulation Setup

AleaNG will read `configuration.properties` file to setup the simulation. By default, it reads **Example 1** stored in the `./examples/Example1/` folder. **If you want to try Example 2, 3 or Example 4**, all you need to do is rewrite the `configuration.properties` file with the one provided in each `./examples/ExampleXY/` directory.

Once more familiar, you can **edit the configuration file**, changing the algorithms, workloads and simulation setup according to your needs. Each parameter is explained in the `configuration.properties` file, which looks like this:

```
#---------------------------------------
# DATA_SETS (workloads). Use a comma to separate two or more different
workloads
data_sets=Example1-load_50%-urgent_10%-instance_0.swf, Example1-load_75%-
urgent_10%-instance_0.swf, Example1-load_100%-urgent_10%-instance_0.swf
# Data set directory
data_set_dir=/examples/Example1/
# number of gridlets (jobs) to be expected/simulated from the workload file
(specify for each data set written above)
total_gridlet=10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,1
0000,10000,10000

#---------------------------------------
# SCHEDULING ALGORITHMS
# 0 = fair_Strict_Ordering (FCFS-like, no backfill)
# 1 = fair_Aggressive_Backfilling (no reservation)
# 2 = fair_EASY_Backfilling
algorithms = 1, 2

#---------------------------------------
# CHART SETUP (how often is the state of the system sampled)
# how often the results are sampled for charts
sample_tick=600
# if true, charts are shown at the end of the experiment
draw_chart=true
...
```

Configuration

The simulator can be configured by the `./configuration.properties` file located in the main directory. Here you can specify:

- the names of the workloads to be used
- the number of jobs to be used from each such workload
- the type of scheduling algorithm(s) to be used. Some basic scheduling algorithms are already provided such as:
    - 0 – Strict Ordering
    - 1 – Aggressive Backfilling
    - 2 – Easy Backfilling
    - and many more (including those that can build schedule (execution plan)

and various other scheduling –related parameters. For example, you can specify whether:

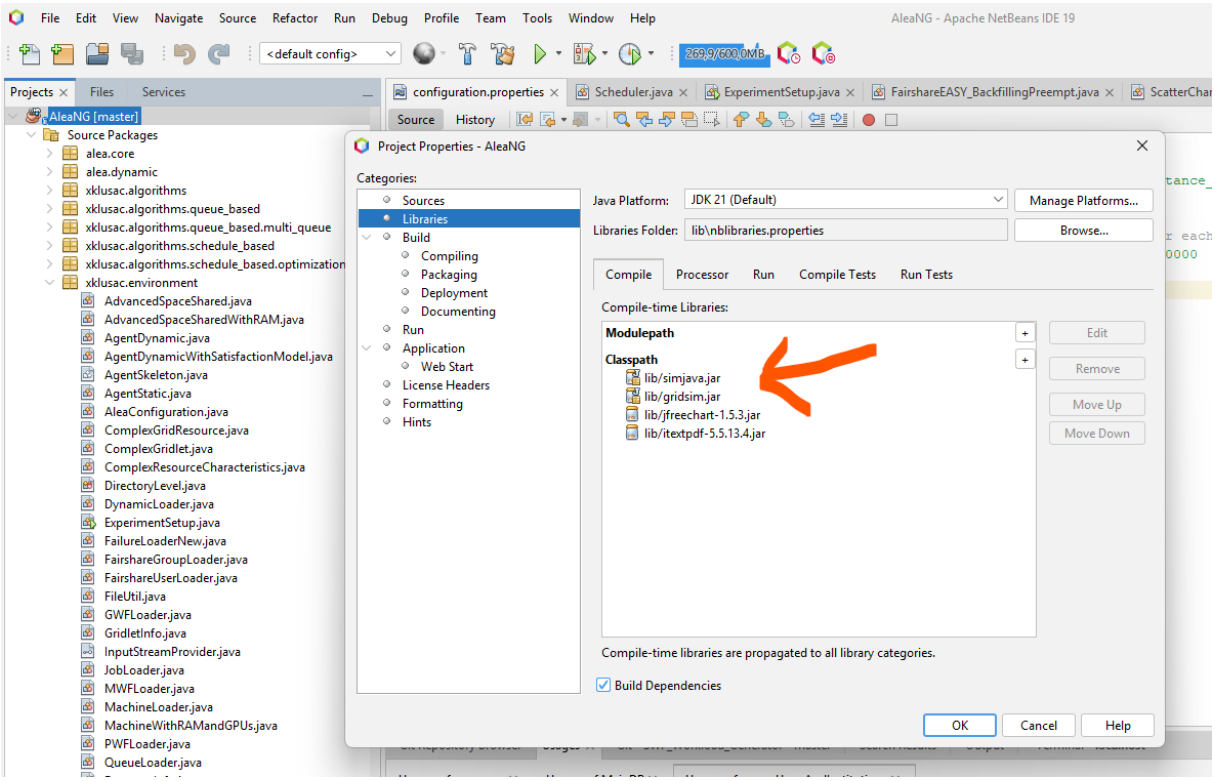
- job runtime estimates will be used and if so, which type of estimate will be used (user-provided or “artificial”).
- fair-sharing should be used and if so, whether decay should be applied and how often.
- visualization should be used, showing either general metrics like utilization and waiting/running jobs

Extending Alea

AleaNG is distributed as a NetBeans project, so the most convenient way is to download NetBeans IDE (https://netbeans.org/) and open AleaNG there. After opening, it will probably be necessary to "resolve reference problem" by selecting simjava.jar and gridsim.jar libraries, which are required to run the simulator. Those are distributed along with the AleaNG in the ./lib directory.

Once libraries are loaded (**it is important to load simjava.jar as the first library** – see the screenshot), you can modify, compile, and run the simulator. You can also run the simulator without the IDE by executing the Alea.jar file located in the ./dist folder. Use the command when located in the main AleaNG folder:

```
java -jar "./dist/AleaNG.jar"
```



Workloads

AleaNG needs a valid data set (workload) to be located in the specified data-set directory.

Each workload needs at least two files – a job description and a machine description file. More complex simulations also require queue, group, and user description files.

The format of the workload file is typically identical to the SWF format (http://www.cs.huji.ac.il/labs/parallel/workload/swf.html), while the machine-description file in AleaNG uses its own format.

The machine description file is stored in a file with `.machines` filename extension. The file format specifies one computer cluster per each line, attributes are separated by TAB space. Examples of workload files can be found in `./examples/` directory. Each file has a header, where field are described.

Synthetic Workload Generator

You can also use the provided synthetic workload generator available at: https://github.com/aleasimulator/SWF_Workload_Generator

The generator creates an extended, **SWF-like workload format** and additional files needed by the **AleaNG simulator** to perform detailed job scheduling simulations. Notably, the generator creates:

1. *job file* (in extended SWF format, see SWF description: https://www.cs.huji.ac.il/labs/parallel/workload/swf.html)
2. *machine file* (describing clusters)
3. *queue file* (describing queues and their limits and priorities)
4. *group file* (describing user-groups and their limits)
5. *user file* (describing users, their groups, their relative shares and limits)

Main Classes of AleaNG simulator

`ExperimentSetup.java` reads the `configuration.properties` file and starts the experiment by creating all important entities such as job workload loader and `Scheduler.java`

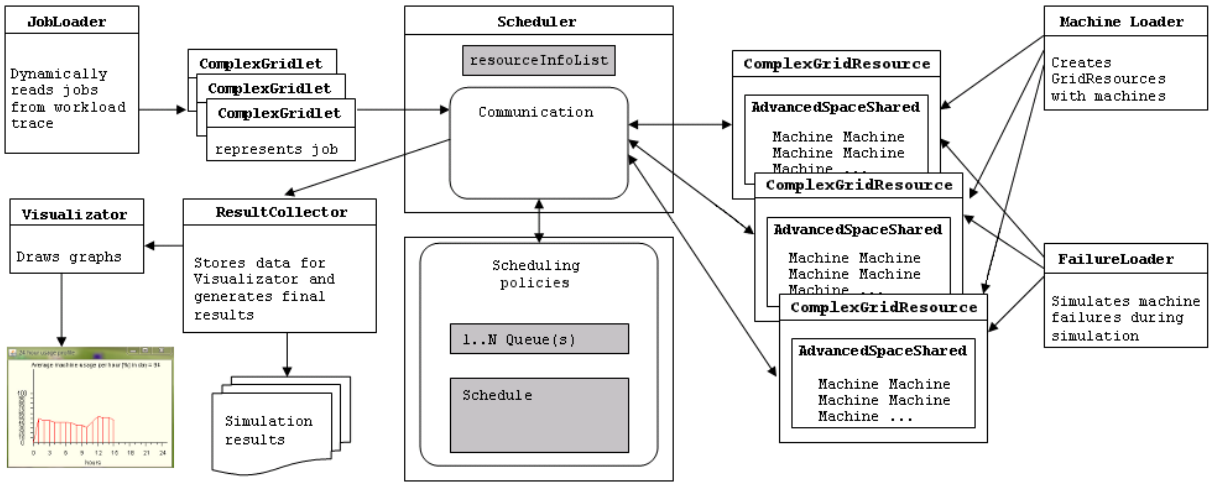
`Scheduler.java` is the main class responsible for handling events related to job scheduling (e.g., job submission (arrival), job completion, etc.). It is the right place to add new event/time-driven handlers. For example, if you want to develop a routine that reacts to a certain event (e.g., job submission) or is executed periodically (e.g., some type of result collection or visualization), `Schedler.java` is the right place to put such event/time handling code.

`JobLoader.java` is used to parse the workload. Typically, Standard Workload Format (SWF) files are used, thus a special `SWFLoader.java` is then responsible for parsing SWF file, creating jobs to be processed by the simulator (so called gridlets). These are passed on to the `Scheduler.java`

`GridletInfo.java` is an encapsulation of the standard `Gridlet.java` class and it stores important information about job being processed in the AleaNG. Importantly, it contains the `getJobRuntime` function that is used to tell the simulator what is the (expected) job runtime. Using various configuration options in `configuration.properties` file, you can define whether this function return the exact runtime (perfect estimate) or some other type of estimate, e.g., user provided or artificially generated using some form of a predictor.

Dynamic information about resources (clusters) are stored in instances of `ResourceInfo.java` class (one per each cluster). Here you can find the schedule and various useful methods such as `printSchedule`, which prints the current schedule of planned jobs along with their expected start times. `Scheduler.java` holds the appropriate references to the instances of `ResourceInfo` classes in its `resourceInfoList` structure.

Once jobs are selected by a chosen scheduling algorithm, they are sent to the cluster where they are executed. The class responsible for execution on a cluster is called `ComplexGridResource.java` and it uses space-sharing job allocation policy where each job is guaranteed to get all the resources it requires (no time-sharing or overcommitting is involved). Once completed (the proper amount of simulation time has passed), jobs are returned to the scheduler marked as “completed”.



General overview of some of the most important entities is shown in the figure above.

Enabling Complex Simulations

To enable more complex simulation, familiarize yourself with configuration.properties and available options. You can also use the provided synthetic workload generator available at: https://github.com/aleasimulator/SWF_Workload_Generator

Job Runtime Estimates

AleaNG supports various forms of job runtime estimates. By default, it supports exact estimates (i.e., actual runtime = estimate), user-provided estimates as specified in SWF workload, or "refined estimates" using some arbitrary predictor. There are some "prebuilt" predictors in Alea, that use previous job runtimes (of a given user) to determine the probable runtime of a new job. Certainly, these predictors can be modified/replaced to fit your needs. The right place to look at is the `getJobRuntime` function in `GridletInfo.java` class.

Scheduling Algorithms

AleaNG supports both queue-based and schedule-based (plan-based) scheduling algorithms. Each algorithm is placed in a separate class that implements the `SchedulingPolicy.java` interface.

Results

Results are generated as the simulation proceeds and are collected in a complex folder structure in the `./results` directory.

Each simulation gets a distinctive folder where the simulation configuration file is copied as well. Beside some general preprocessed results allowing for quick comparison, this folder contains one folder per each workload and per each algorithm.

Within each such folder a complete list of processed jobs (and their parameters) and a list of users is kept, allowing to perform “post mortem” analysis of simulation performance (using your own job parser).

Also, graphical outputs are stored there (if enabled in `configuration.properties`) as PNG and PDF (see an example below).