Alea Job Scheduling Simulator

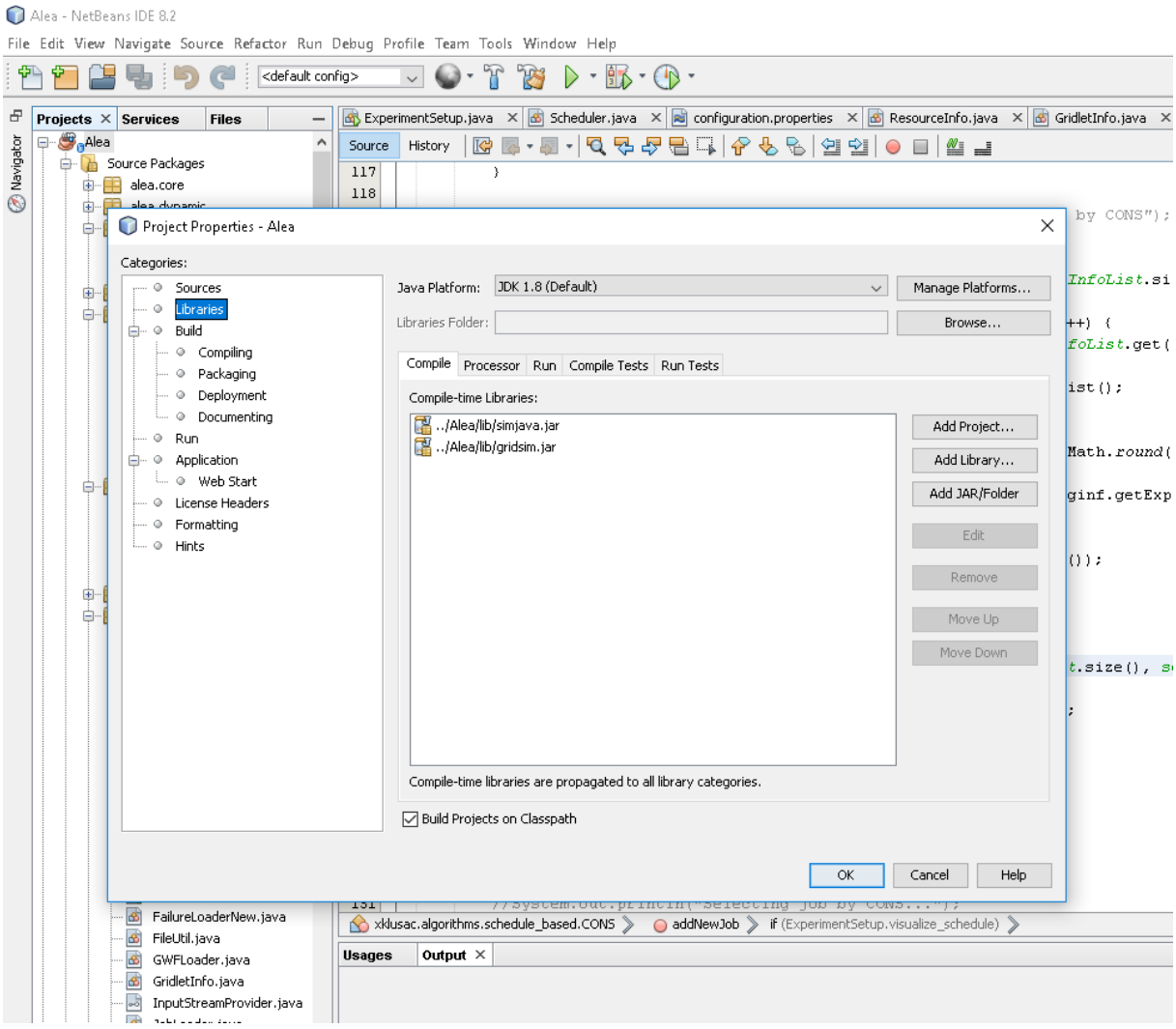
Downloading the Alea

Ale ais distributed on GitHub: https://github.com/aleasimulator/alea

Running Alea

Ale ais distributed as a Netbeans Project, so the most convenient way is to download Netbeans IDE (https://netbeans.org/) and open Alea there. After opening, it will be probably necessary to “resolve reference problem” by selecting simjava.jar and gridsim.jar libraries, that are required to run the simulator. Those are distributed along with the Alea in the ./lib directory.

Once libraries are loaded (it is important to load simjava.jar as the first library – see the screenshot), you can 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 Alea folder: java -jar "./dist/Alea.jar"



Workloads

Alea needs a valid data set (workload) to be located in the ./data-set directory. Each workload needs two files – job description and machine description file. The format of the workload file is typically identical to SWF format (http://www.cs.huji.ac.il/labs/parallel/workload/swf.html), while the machine-description file in Alea uses its own format. 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. One line has the following attributes:

cluster_id <tab> cluster_name <tab> number_of_nodes <tab> CPUs_per_node <tab> CPU_speed <tab> RAM_per_node_in_KB

So, in the following example we have two clusters (zewura/zegox), having 20/48 nodes, each having 80/12 CPUs per node (with default speed 1) and approx. 500/90 GB of RAM per each node:

```
0	zewura	20	80	1	529426432
1	zegox	48	12	1	94035968
```

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 – FCFS
  - 2 – Easy Backfilling
  - 4 – Conservative Backfilling
  - 6 – Shortest Job First
  - 7 – Fair-Share FCFS
  - 9 – Fair-share Conservative Backfilling
  - 22 – Fair-share Easy Backfilling

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 what type of resource tracking metric will be used.
- “on the fly” visualization should be used, showing either general metrics like utilization and waiting/running jobs or prepared schedule. (Schedule visualization is however only possible for planning-based algorithms such as Conservative backfilling and similar. It will not work for pure queue-based
- algorithms since there is no schedule to visualize).

Main Classes of Alea 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`

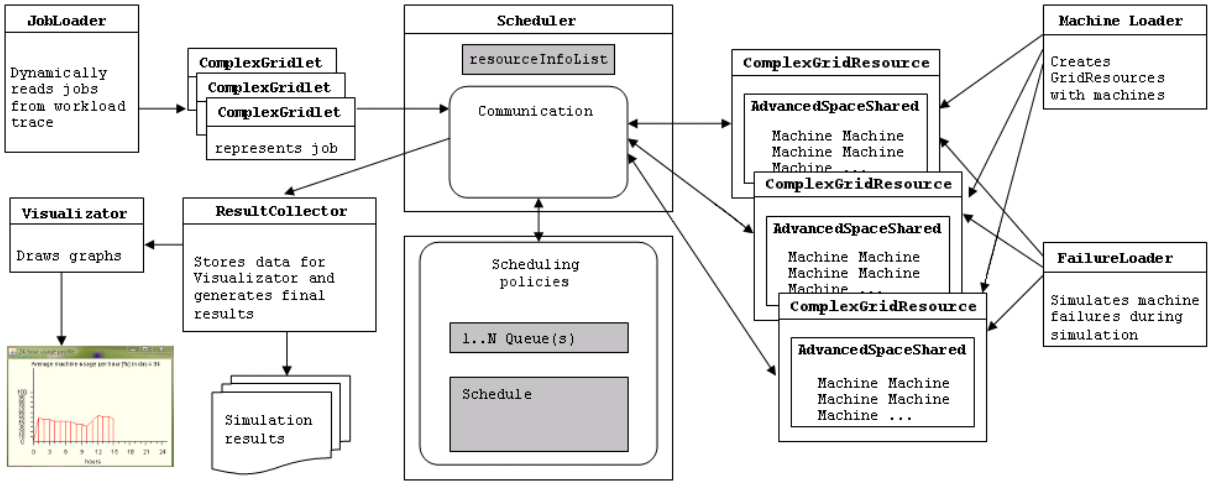
`Scheduler.java` is the main class responsible for handling events related to job scheduling (e.g., job submission (arrival), job completion, etc.). It is the right place to add new event/time-driven handlers. For example, if you want to develop a routine that reacts to a certain event (e.g., job submission) 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 Alea. 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.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 that 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.

Job Runtime Estimates

Alea 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

Alea 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).