The McMaster Grid Scheduling Testing Tool

User Manual

1 Preparation of Servers

In this phase the execution layer should be prepared. The *Puller.jar* executable should be running, by executing the following command: *java -jar Puller.jar.*

2 User Interface

The User interface of the software is divided into the tool bar (where the most used actions have short cuts), the menu bar (where system functions can be invoked) and the main tabs. Each main tab is responsible for one phase of the test or a particular functionality and has several sub tabs. The remaining sections discuss these tabs in detail.

3 Definition Phase

This phase is done through the main tab labelled *Definitions.* In this phase the tester should define the parameters of the system, including:

- General Parameters (e.g time units in minutes or scheduling policy to be used)
- Job classes
- Servers
- Server Availability

3.1 General Parameters

These parameters are accessed under the main tab *Definitions* and the sub tab *General*.

- Time Unit in Minutes: this parameter defines the length of the time unit used in a test in minutes. A time unit is a hypothetical time unit used as the unit of all time quantities in the system. (e.g. The units of execution rates and arrival rates are task per time unit)
- Mean Time to Repair: this parameter defines the mean length of the failure periods for all the servers when the artificial failures option is enabled.
- Mean Time to Failure: this paramter defines the mean length of the up-time periods for all the servers when the artificial failures option is enabled.
- Mapping Scheme: this parameter determines the scheduling policy used in a test.
- Time Resolution: This parameter is T_{system}. Please refer to section 8.1.2 of the thesis.
- Artificial Failures and T/O (Time-out): Artificial failures can be simulated to study the effects of failures. This parameter determines whether the artificial failures option is enabled or not.
- Time-out: Every job has an estimated execution time. If a server failure occurs while executing a job, the completion notification will not reach the mapper. The mapper waits

for that job to be completed for *n* times the expected execution time, where *n* is the "Time-out after" parameter. Then the mapper invokes the *handleTimeOut* method of the active scheduling scheme.

After the user has set all of the parameters, they should click on the Apply button.

3.2 Job Classes

The job classes are defined in this phase. Every job class has an ID, execution time and arrival rate. The ID of a job class is the number of the column which represents this job class in the mue matrix (Section 3.1 of the thesis). The execution time of a job class is the mean time in milliseconds in which the job will be executed. The arrival rate is the mean number of jobs that will arrive to the system per time unit under an exponential interarrival time distribution. To add a job, click on the plus button and fill the iterations and the arrival rate. Then click on the Submit button (Figure B.2). The classes will be added in order. To delete a job, select it and then click the minus button.

3.3 Servers

The servers are defined in this phase. There are many ways that this can be done. Obviously, the servers to be defined should be the ones set up in the Preparation of Servers phase. One way of adding servers is to click the plus button and insert the information related to the server. The full canonical hostname should be inserted as the hostname. The servers will be added in order. The ID of a server is its order in the mue matrix (Section 3.1 of the thesis).

For convenience, a file can be prepared where each line corresponds to the hostname of one server. The file extension should be *srs*.

To delete a server, it should be selected and then the minus button should be clicked.

Processing Rates

To modify or view the processing rates of a server, the user should select the server by clicking on it. The *Processing Rates* tab will appear on the right side. The real rates (second column) are those which the software estimated using the execution times given by the tester. The assumed rates (third column) can be changed. The *Assumed Rate* column for a server with ID *i* tis the i^{th} column in the mue matrix. To change an assumed rate, the user has to click on the appropriate cell and type a new number and press enter. For convenience, the tester can import all of the processing rates of a setting using the *Import Mue* button. The file should be a text file with the extension *MUE*. The file format should be similar to the mue matrix but entries are separated by commas.

ex.

```
2.2, 7, 10.25, 1
1.95, 7.05, 9.78, 0.95
 2, 7.25, 10.02, 0.98
```

Sets the assumed rates for three job classes(3 lines) on four servers(4 rates per line).

Failure Periods

In the case that the artificial failure option is enabled, the artificial failure of a server can be viewed by clicking on a server, and then selecting the *Failure Periods* sub tab. To generate new failure traces for all the servers, the button *Fill Traces* should be clicked. The actual mean up-time and the mean failure period of a server are viewed at the bottom. To change these values for each server individually, the user has to change values in the text fields and click on *Apply*.

Availability

Every machine has a puller module running on it. To set up the module a message has to be sent to it. In this phase the messages (and hence the settings) of the puller modules (i.e servers) are prepared and sent.

One or more servers are selected from the table on the left, then the properties are set in the right side. The properties are:

- Availability is the a_j value (Section 3.2 of the thesis).
- Availability Mode is what method of availability prediction is used (Section 3.2 of the thesis). There are three different modes. Choosing different modes will be followed by the inserting of parameters related to that mode.

After preparing the messages, they can be sent to the servers using the *Servers* menu in the menu bar or the *Start Servers* button in the toolbar. In addition, the servers can be paused, pinged, or killed. All of these actions can be found under the *Servers* menu in the menu bar. After pausing a server it must be started again to function properly. Pinging can be used to make sure the server is turned on. After killing the server, the puller executable must be run on that server (using *java -jar Puller.jar*) to restart it, as the kill signal makes the puller.jar process exit.

LP

In this phase the LP allocation can be solved. To solve the LP allocation, the solve button should be clicked. The resulting matrix will be displayed.

After the completion of the definitions, you can save them to a file using the *Save definitions* button. Saved definitions can be restored using the *Load definitions* button. All the information in the definition is saved except for the Scheduling scheme chosen which should be determined before every test.

Monitoring

After the completion of the definition phase, the experiment can be started by clicking on the *Start* button in the tool bar or *Action* in the menu bar. Under the *Monitoring* tab, there are two items to monitor: the Jobs Table and the Available Servers. In the *Jobs Table* sub tab, jobs can be monitored. This table is updated whenever an event occurs. Under the *Available Servers* sub tab, the available servers can be monitored. To see the currently available servers, the *Update* button must be clicked to see the changes.

Statistics

To obtain statistics about the tests, the main tab Statistics is used. This main tab has three sub tabs:

- *General* sub tab which shows the general statistics such as: the start time of the test, the time units elapsed and the response time.
- *Job Classes* sub tab which shows statistics about each job class. Such statistics include the average response time, average waiting time, total number of jobs arrived, desired arrival rate and actual arrival rate
- *Processing Rates* sub tab which shows the mue matrix and the actual processing rates per machine per job class.

All these statistics can be saved into files. This can be done using the Tables menu in the menu bar. A save dialogue appears on the screen. The user can browse to the target folder and then type the name of the test (e.g. LPAS). As a result four files will be saved (e.g. LPAS_classesStats.txt, LPAS_jobs.txt, LPAS_mue.txt, LPAS_systemStats.txt). The four files can be opened with spread sheet applications.

To Add a New Mapping Scheme

If the tester wishes to add a new mapping scheme, the scheme must extend the *MappingScheme.java* class located in the mapping package. This scheme must define several different methods including the method which starts and stops the scheme, the actual mapping of a job to a server, and how the scheme handles servers being down or jobs failing. Complete details are included in the comments in the *MappingScheme.java* class.

Next, in order for the scheme to appear in the drop down menu the following must be added to the class *SystemLevelParametersJPanel.java*:

- A *private static int* for your scheme, this corresponds to its position in the combo box. Find the existing indices at the start of the class and add your scheme at the end
- In the constructor *public SystemLevelParametersJPanel(Mainframe frame)* add your scheme in the *mappingSchemeCB.addItem* section
- Finally, in the *applyInput()* method, add your scheme in the *//What mapping scheme to use*, section

There are also options to add new arrival generators, probability distributions and execution layers if it is desired. The abstract classes for these are located in their respective packages and each contains comments on which methods need to be created.

To change the arrival generator, simply import your generator in the *generating.JobsGenerator.java* class and then set it as the arrival generator in the *public JobsGenerator(JobClass jobClass, Mapper mapper)* class.

To change the mapper side executer, import your executer in the *mapping.Mapper.java* class and then set it as the executer in the *public Mapping()* class.

To change the puller side executer, import your executer in the *pulling.Puller.java* class and then set it as the executer in the *public static void main(String[] args)* class.