GridWay Internal Report: Scalability

Tino Vázquez, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid, Spain
2nd September, 2007
{tinova, ehuedo}@fdi.ucm.es, {rubensm, llorente}@dacya.ucm.es

1 Introduction

The GridWay Metascheduler enables large-scale, reliable and efficient sharing of computing resources (clusters, computing farms, servers, supercomputers...), managed by different LRM (Local Resource Management) systems, such as PBS, SGE, LSF or Condor, within a single organization (enterprise grid) or scattered across several administrative domains (partner or supply-chain grid). GridWay is now a Globus project and part of the Globus Toolkit, and question have arose ever since on how robust or scalable GridWay is. This report is an attempt to answer those questions.

2 Testbed

The resources used for the testbed are located in the UCM local grid, so issues like network saturation and the likes can be more controlled. These resources are Intel Pentium 4 3.2GHz with 2GB of RAM that communicate with each other over a 100Mb/s LAN network. The testbed for the scalability test uses four computing elements with the aforementioned specifications, served by GRAM preWS. The GridWay client is also a machine with the same characteristics.

GridWay is not really a monolithic application. It is composed of several modules:

- GridWay daemon - this is the core that coordinates the whole process by means of a state machine.
- GridWay MADs (Middleware Access Drivers) - these are independent processes that talk with the daemon using the standard I/O streams. They are used to perform execution, transfer and information tasks.
- GridWay scheduler - this too lives in a separate process, talking with the daemon.

GridWay was configured using:

- PreWS GRAM Execution MAD.
- GridFTP Transfer MAD.

– Static Information MAD.

The Static Information MAD allowed to configure the four hosts for GridWay as if having 200 hundred Fork slots free each. To achieve this, the CPUFREE attribute of each host was set to 20,000%. See the GridWay Install & Configuration Guide for details. The performance benefit of using the Static Information MAD as opposed to a dynamic one is negligible due to the small number of hosts. A future work will measure GridWay's scalability discovering and monitoring resources.

3 Experiment Explained

To measure the scalability of GridWay, we set up an experiment that tries to saturate all GridWay components by submitting 10,000 jobs. The submission was done using the GridWay command *gwsubmit*, and a REQUIREMENT was set in their job templates so they will fairly distribute themselves between the four hosts.

The GridWay instance was configured so it submits jobs at the following rate:

– DISPATCH_CHUNK = 60 - in each iteration, the scheduler submits a maximum of 60 jobs.
– SCHEDULING_INTERVAL = 180 - each iteration of the scheduler happens with a 3 minutes interval.
– MAX_RUNNING_USER = 0 - there is no limit on how many jobs can be running at the same time.
– MAX_RUNNING_RESOURCE = 0 - there is no limit on the maximum number of jobs that the scheduler submits to a given resource, so ideally all the 500 slots would be used per host.

This GridWay instance is running on a separate host, submitting jobs to the other four. In order to extract information about this GridWay instance, one script was ran in the GridWay host each 2 minutes to measure memory, CPU and disk consumption, as well as the number of pending, running and done jobs. As seen in the previous section, this task not only involves the monitoring of a single process, but rather of the daemon, the MADs (except the Information Manager, that doesn't do much as GridWay is using static information) and the scheduler.

About the nature of the jobs being sent, it is as follows. They consist basically on running the /bin/sleep command with different arguments, ranging from 1800 (half an hour) to 5400 (hour and a half). This was chosen because it doesn't make use of the CPU, and this is exactly what we want because we are not interested in measuring a computer saturation but rather GridWay's. The file staging carried by each job involves the creation of a directory in the prolog state and the removal of that directory in the epilog state, and also two set of files. Starting with staging in the prolog state, here are the file sizes (approx):

– job.env - 500 Bytes
– wrapper.sh - 12 Kilobytes

and for the epilog state, we have:

– stdout.wrapper - 500 Bytes
– stderr.wrapper - 16 Kilobytes

4 Results

The first interesting result of the test is the time that GridWay needs to accept, sequentially, 10,000 jobs. The average time in seconds for this is 226 seconds, i.e., a bit less than four minutes.

We can see in Figure 1 the disk consumption of GridWay. For ten thousand *completed* jobs the disk consumption gets to 1,320,168 Kilobytes, and that gives an average of 132 KB per job. Is worth noting that this number includes all the logging that GridWay is capable of producing, including information for debugging. A good estimate on how this number would be on a GridWay instance without debug enabled (the default option) is to take it's 10%.

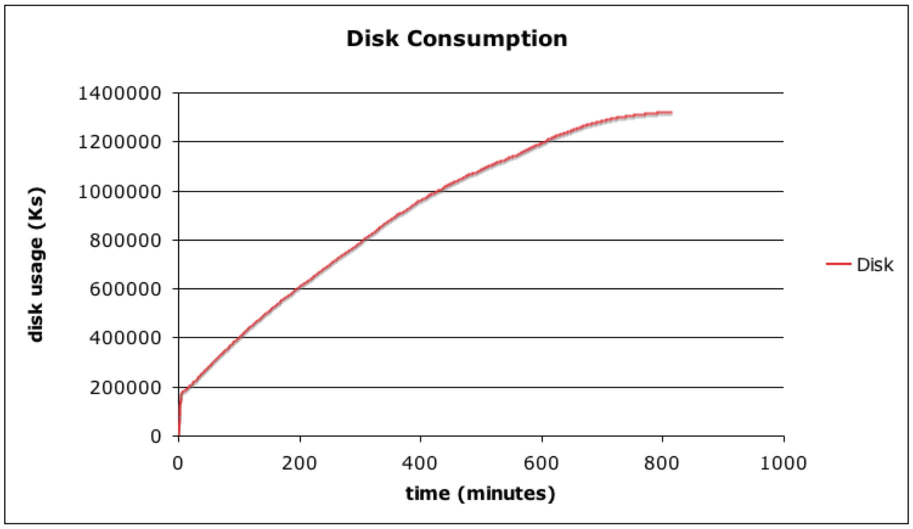

Fig. 1. Disk consumption.

A more interesting figure is Figure 2. It shows the percentage of memory consumption for all GridWay components and the total add between all of them. We can see how the Transfer Manager gets most of the memory, but it is interesting to note how the GridWay daemon and the scheduler are much more discreet in memory consumption.

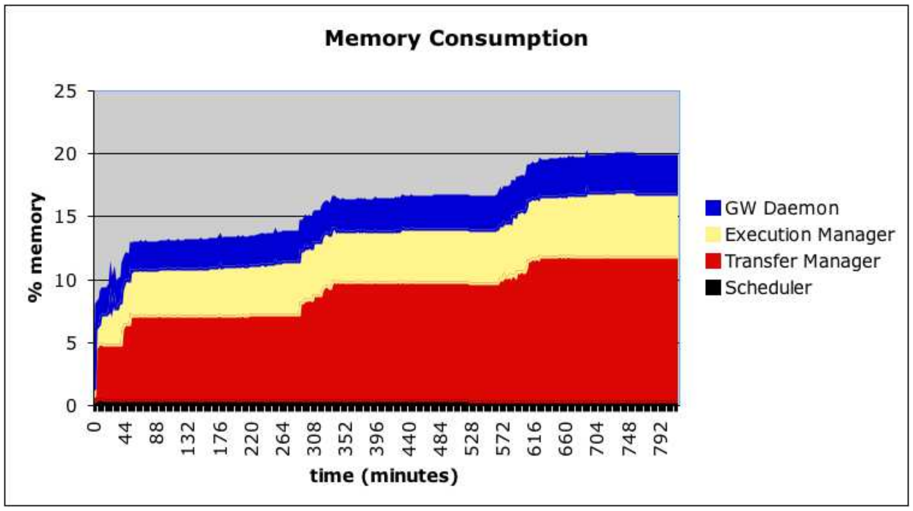

Fig. 2. Memory consumption.

Figure 3 shows the CPU consumption. We can see here a similar trend, where the MADs take the most CPU time and the GridWay daemon and the scheduler don't take that much.

We can see in Figure 4 how the jobs get themselves from the pending to the done state, maintaining an acceptable average rate of 800 jobs running.

5 Conclusions

GridWay showed great stability, robustness and scalability during this test. Scheduling 10,000 jobs and keeping track of them is not an easy task, but GridWay completed it nevertheless. This by no means implies that GridWay is perfect, there are a number of issues that were discovered during these tests. Notwithstanding, GridWay core showed a remarkable robustness and quickness, better showed in the fact that it can get 10,000 jobs in pending state in less than four minutes.

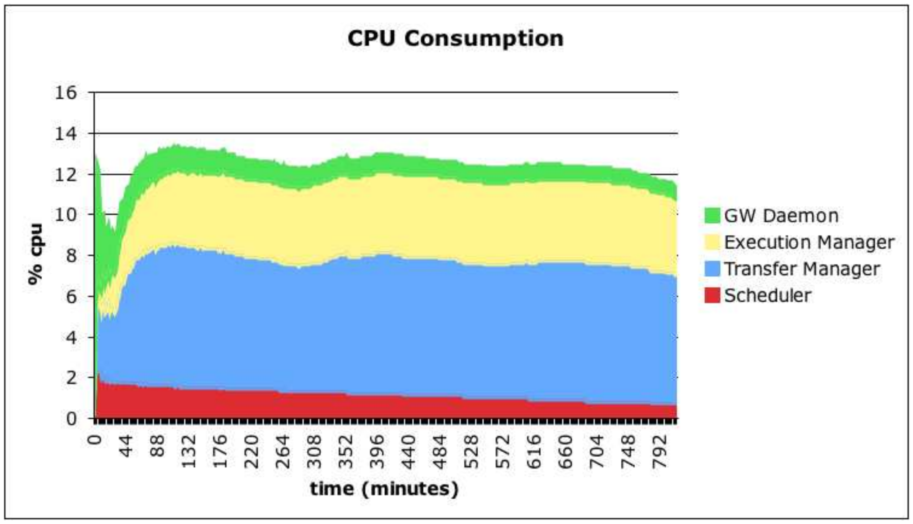

Fig. 3. CPU consumption.

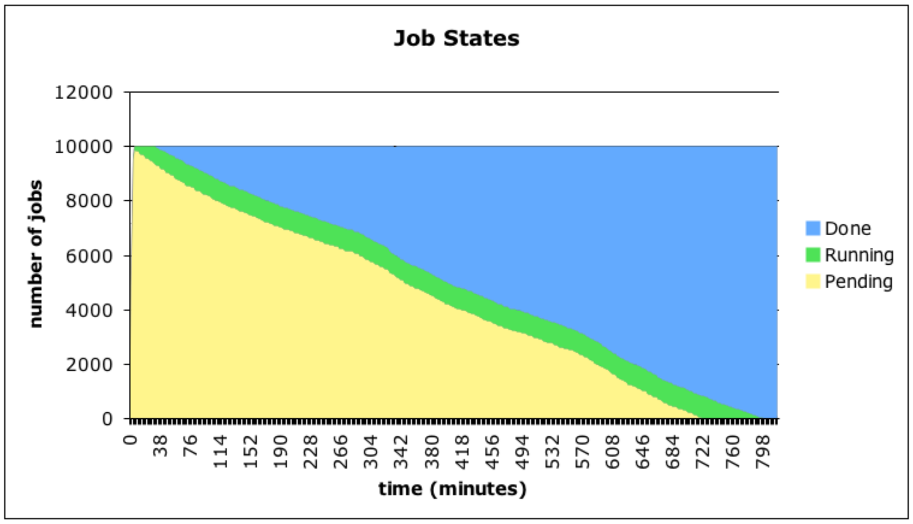

Fig. 4. Job states.