Scalability of Grid Simulators: An Evaluation

Wim Depoorter, Nils De Moor, Kurt Vanmechelen, and Jan Broeckhove

University of Antwerp, BE-2020 Antwerp, Belgium
wim.depoorter@ua.ac.be

Abstract. Due to the distributed nature of resources in grids that cover multiple administrative domains, grid resource management cannot be optimally implemented using traditional approaches. In order to investigate new grid resource management systems, researchers utilize simulators which allows them to efficiently evaluate new algorithms on a large scale. We have developed the Grid Economics Simulator (GES) in support of research into grid resource management in general and economic grid resource management in particular. This paper compares GES to SimGrid and GridSim, two established grid simulation frameworks. We demonstrate that GES compares favourably to the other frameworks in terms of scalability, runtime performance and memory requirements. We explain how these differences are related to the simulation paradigm and the threading model used in each simulator.

Keywords: Simulation, Grids, Performance Analysis.

1 Introduction

Conducting research into resource management systems (RMS) on real grids is difficult because of two reasons. Firstly, the costs involved in setting up and maintaining such a system are high. Secondly, there is a need to test a new RMS under a variety of different load patterns and infrastructural arrangements, which is all but impossible to achieve with a real grid system. The large scale on which a grid RMS needs to be studied magnifies the impact of these problems. As a result, the only viable option for researchers is to resort to simulation.

The aim of a grid simulator is to allow easy *comparison* between different resource management approaches and to enable researchers to *focus on the design and implementation* of the chosen approach, while leveraging the strength of the existing framework in *setting up* the grid environment, *running* the simulation and *monitoring* the desired metrics. Because a grid is intrinsically a large scale system, a fundamental requirement for a grid simulator is scalability. An example of such a large scale grid system is the system built by the the European "Enabling Grids for E-sciencE" (EGEE) project. The EGEE infrastructure services over 10 000 users from 45 countries and offers a compute capacity of well over 40 000 CPUs, a figure which is expected to double over the course of the next year. In order to simulate such a vast infrastructure, a simulator has to be able to handle over 10 000 user entities and at least 100 000 computing nodes. In this contribution we evaluate the performance of SimGrid [1], GridSim [2] and GES [3] in light of such large scale simulations.

2 Simulator Overview

Many frameworks have been developed for simulating grid systems [1,2,4]. We have chosen to compare the performance of GES to SimGrid and GridSim because of their maturity, extensive user base and active development status. The main differences between the simulators are the chosen simulation paradigm and threading model. While GES uses single-threaded discrete-time based simulation, both SimGrid and GridSim use massively multi-threaded discrete-event based simulation. Whereas SimGrid offers the choice between the use of `ucontexts` (user space threads) and `pthreads` (native threads), GridSim only supports native threading, a consequence of the threading model used in current JVMs.

The differences in simulation paradigm can be easily explained by the focus and history of the simulators. SimGrid has a strong focus on accurate network simulation. This accuracy can be achieved best using discrete-event simulation. GridSim started out as a framework for testing resource management policies in grids and is built on top of the SimJava discrete-event engine. The addition of a simulated network infrastructure enables the incorporation of network effects in simulations [5]. GES was developed as a tool for studying economic resource management approaches for grids, with a main focus on testing the algorithms and protocols of various economic approaches. Under the assumption that network contention is low there is less need to simulate a network and it is more efficient to use a discrete-time engine.

2.1 SimGrid

SimGrid [6] is an extensive toolkit that provides core functionalities for the simulation of distributed applications. The codebase is written in C. Java bindings that call into the C core using JNI will be provided in future releases. The toolkit started out with a focus on centralized scheduling algorithms and was adapted later on to allow for decentralized scheduling [7]. The simulator takes into account the computational speed of nodes as well as latency and bandwidth of the network links connecting these nodes. SimGrid's network model allows for faster simulation times compared to approaches that use packet-level simulation.

The GRAS layer allows developers to implement distributed services and deploy them in a simulated setting using the Meta-SimGrid (MSG) layer, or in a real world setting using a socket-based communication layer. In the context of our survey on simulation scalability we will evaluate the performance of the MSG layer. This layer provides abstractions for *hosts*, *tasks* and *processes*. A host represents a physical resource with computing capabilities that is able to execute tasks. Hosts are linked to each other through a set of links. Currently, it is not possible to create multi-processor hosts[1]. A process is a piece of logic

[1] This might be included as a future extension as communicated to us by the SimGrid developers.

that runs on a specific host and corresponds to a thread. A process can execute tasks on its corresponding host or exchange them with other processes. A task is defined by a computation amount and size.

2.2 GridSim

GridSim is written in Java on top of the SimJava 2.0 basic discrete event infrastructure. The simulator allows for packet-level simulation of the network and provides an output statistics framework. It supports space and time shared allocation policies as well as advance reservation. Contrary to SimGrid it is possible to model clusters as a single entity. GridSim also contains a reusable *GridInformationService* (GIS) which is responsible for the registration, indexing and discovery of resource providers. It is possible to create a single GIS entity in the simulation, but also to organize multiple GIS entities in a hierarchy comparable to a DNS tree. Additionally, GridSim includes components oriented towards data grids, the most important one being the *ReplicaCatalogue* (RC). Like the GIS, a RC can be organized hierarchically. GridSim has also been used to study economic grid resource management [8,9].

Every simulated entity in GridSim extends the `GridSimCore` class which includes both an `Input` and `Output` object to send and receive events. All three of these classes extend the `Sim_entity` class of SimJava. Since every `Sim_entity` is actually a Java thread, this means that every user or resource provider entity requires at least three threads during simulation. GridSim entities use the `sim_pause()`, `sim_get_next()` and `sim_schedule()` primitives of SimJava to pause, receive the next `Sim_event` object or schedule such an event.

2.3 GES

The Grid Economics Simulator (GES) has been developed to study economic grid resource management systems [10,11,12]. The simulator offers a toolkit for analyzing and comparing different economic and traditional resource management algorithms. An overview of the GES core layer is given in figure 1.

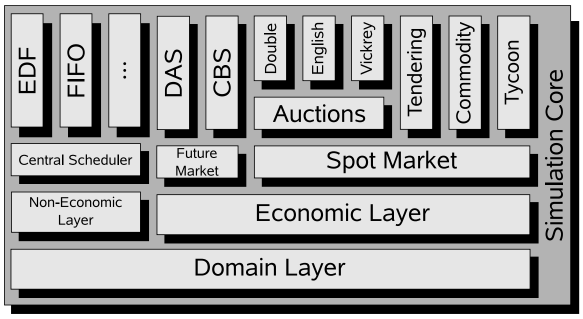

Fig. 1. Overview of the architecture of GES

Two of the key design goals of the architecture are extensibility and reusability. This “extend-and-refine” philosophy can be found throughout the whole `simulation core` and its components. The `domain` layer contains base classes for all domain entities such as `Consumer`, `Provider`, `Job`, `GridResource` and `GridEnvironment`. The `Bank` entity as well as other components supporting economic resource management are located in the `economic` layer. Support for traditional forms of resource management is provided through the `non-economic` layer. Existing components can be easily extended when new RMS algorithms are added to the framework. GES currently has support for a substantial number of spot- and future market mechanisms [3,12].

Simulations can be distributed over multiple processing nodes through the `distribution` layer. This layer interfaces with compute resources that host a Jini-enabled compute service, clusters fronted by a Sun Grid Engine head node, or clusters with a passwordless SSH setup. Currently, distribution is supported at the granularity of a simulated scenario. For a more in depth view of all the capabilities of GES, we refer to [3].

In order to investigate the communication complexity of resource management approaches in more detail, we are planning to extend the simulator with support for simulating the network infrastructure. This will also allow for the development of network-aware scheduling algorithms, market mechanisms for bandwidth pricing, as well as analysis of communication and data transfers.

3 Evaluation

We will test the general scalability of the different simulators and determine whether they are capable of simulating a system on the scale of the EGEE grid. In addition, we will investigate how the three simulators scale in terms of the number of jobs in the system. We use a synthetic scenario that is specifically designed to isolate the different variables which may affect the outcome of our tests. Note that in the following, we will refer to grid users as *consumers* while entities that contribute resources to the infrastructure are referred to as *providers*. All tests use variations of a base scenario with the following parameters:

– Number of consumers $N_\mathrm{c} = 1\,000$
– Number of providers $N_\mathrm{p} = 100$
– Total number of jobs $N_\mathrm{j} = 10\,000$
– Number of jobs per consumer $N_{\mathrm{j_c}} = N_\mathrm{j}/N_\mathrm{c}$
– Job length in processor time slots $L_\mathrm{j} \in [7, 13]$
– Total number of CPUs $N_\mathrm{cpu} = 1\,000$
– Number of cpus per provider $N_{\mathrm{cpu_p}} = N_\mathrm{cpu}/N_\mathrm{p}$

Since we want to focus on the performance properties of the simulators’ cores, we split up the consumers in groups of 10 and associated each group with a single provider which schedules incoming jobs in a round robin fashion. This simple setup allows us to evaluate the core performance of the different simulators while eliminating unwanted effects caused by complex network configurations or interactions between entities.

For GridSim we performed the tests without a simulated network. For SimGrid we were obliged to use a simple network topology because the network provides the only interaction channel for consumers and providers. Since it is currently not possible to aggregate multiple CPUs in one entity, we have modelled a provider node by combining one forwarding host with $N_{\mathrm{cpu}_\mathrm{p}}$ CPU hosts. The forwarding host is connected to its CPUs with one link. The consumers are also connected to their provider with one link. Every job is routed over these links from the consumer to the forwarder and then to a CPU node. All links were configured with maximal bandwidth and minimal latency properties. Because of their higher potential scalability and configurability we used `ucontexts` instead of `pthreads`. We limited the ucontext stack size to 64 KB.

All tests were performed on the CalcUA cluster at the university of Antwerp which hosts 256 Opteron 250 nodes running a 64-bit Linux distribution. All nodes used in the tests hosted 8 GB of RAM. During testing we measured the simulation time in milliseconds and the maximum memory usage, both real and virtual, by polling the simulation process every second. We used version 1.6.0 of Sun's JVM for the executing of all Java code.

3.1 Test I: General Scaling

This scenario is designed to evaluate the general scaling capabilities of each simulator. We scale N_c from 1 to 10 000 while changing N_p such that $N_\mathrm{c}/N_\mathrm{p} = 10$. The other parameters maintained their default values. In effect, this test scales up the entire base scenario.

Figure 2 shows the time it took to perform the simulation on a logarithmic scale as a function of the number of simulated consumers.

As shown in the graph, GES scales up better than both GridSim and SimGrid. The difference in simulation time is over two orders of magnitude compared to SimGrid and three orders of magnitude compared to GridSim when simulating

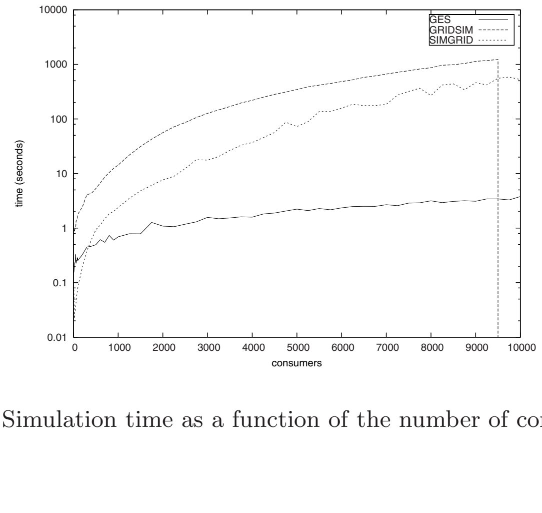

Fig. 2. Simulation time as a function of the number of consumers

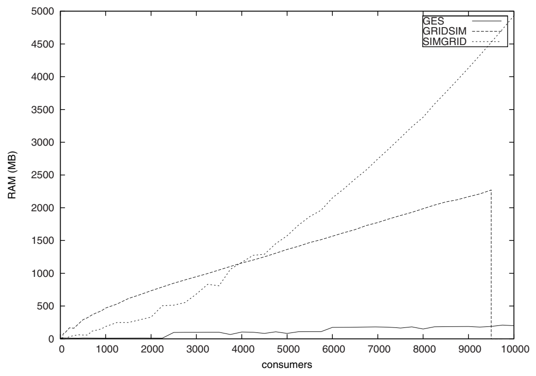

Fig. 3. Real memory requirements as a function of the number of consumers

a grid with 9 000 consumers. GridSim was unable to simulate an environment with 10 000 consumers because it would need to create well over 32 000 threads, more than a normally configured Linux kernel can handle. When scaling up even further we were able to determine that SimGrid could handle a maximum of 12 000 consumers while GES had no problem simulating 100 000 consumers.

Figure 3 shows the actual memory usage in function of the number of consumers. SimGrid does not scale linearly in this regard which can be explained by the fact that it uses a routing table and thus scales quadratically with the number of hosts. While GridSim does scale linearly, its actual memory usage is still substantial and likely related to its heavy use of threading.

3.2 Test II: Job Scaling

This scenario evaluates the influence of the number of jobs on the simulation time. While keeping the total workload per consumer at 100. We scaled $N_{\mathrm{j_c}}$ from 1 to 100.

The time it took to perform the tests is plotted in figure 4. We can see that SimGrid handles higher job loads better than GridSim and that GES is virtually unaffected by the amount of jobs. It is clear that when using the discrete event paradigm, the simulation time scales linearly with the number of jobs while this is not the case for the discrete time paradigm. A discrete time paradigm in contrast scales linearly with the size of a job while this has no effect on a discrete event system. The suitability of either paradigm is determined by the resolution at which we wish to model time. While it is interesting to model time in high resolution for the analysis of interaction protocols, it is not necessary to do so for the simulation of job execution.

The choice for discrete-event simulation often leads to a choice for a multi-threaded model as well, because it is logical to think of events being passed between independently running processes. It is not necessary however to use a multi-threaded model in combination with discrete-event simulation as demonstrated by a number of other projects [13,14]. It is clear from the previous test

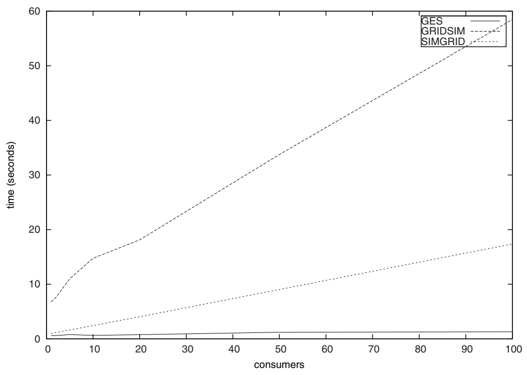

Fig. 4. Simulation time as a function of the number of jobs

that for the simulation of very large systems, a discrete-event simulator may scale up higher when used in combination with a single threaded approach.

4 Threading and Virtual Memory

Both GridSim and SimGrid use threads for each simulated entity. GridSim uses the Java threading model while SimGrid offers a choice between `pthreads` and `ucontexts` on Linux. Both Java threads and `pthreads` are native while `ucontexts` are user space threads. On a Linux machine with a normally configured kernel, the number of simultaneous native threads is limited to just over 32 000. Since user space threads are not visible to the kernel, they overcome this limitation. Moreover `ucontexts` offer more tweakable stack sizes than `pthreads`.

Because the choice of threading model is a key design issue, it is important to understand the consequences and limitations of using a multi-threaded model. One of the most obvious repercussions of using a native multi-threaded model is the overhead caused by context switching. Especially for thread-based discrete-event simulation this overhead can be substantial since a large number of threads will be created. Threads can also be suspended by the scheduler at any time, which may degrade performance unnecessarily. While threads are a good way to develop systems with *actual* concurrent behaviour, it is not necessary to use real threads to *simulate* this concurrency.

Each thread or context will also allocate a stack in virtual memory. This stack will be created in real memory only when it is needed on a per page basis. Therefore, threading does not necessarily have a direct impact on actual *real* memory usage. However there are limits on the amount of *virtual* memory a process can allocate. This limit is 4 GB on any 32-bit architecture of which in general only 3 GB can be used by the process itself on a Linux machine. On 64-bit Linux machines, the practical upper limit of virtual memory available is the sum of the physical memory and the swap size unless oversubscribing is allowed. To mitigate these limitations, it is possible to tweak the size of the

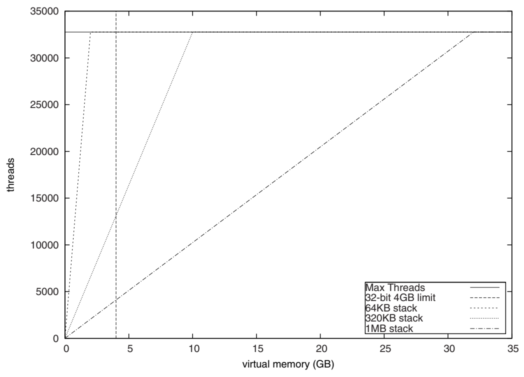

Fig. 5. Number of native threads as a function of available virtual memory

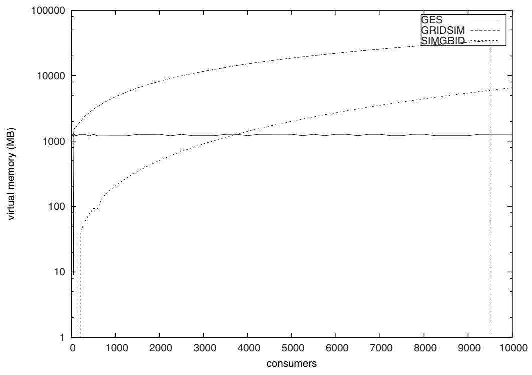

Fig. 6. Virtual memory requirements of the simulators under general scaling scenario

thread stack. The typical thread stack size for Java threads is 320 KB on a 32-bit and 1024 KB on a 64-bit Linux machine. It is possible to reduce this size with a JVM argument to a minimum of 64 KB. For `pthreads`, the minimum stack size is 56 KB while `ucontext` stacks can be even smaller. Another advantage of `pthreads` and `ucontexts` is their ability to adjust the stack size per thread while Java does not offer this flexibility.

The limitations on the scalability of a threaded approach are depicted in figures 5 and 6. The graph in figure 5 includes both the maximum native thread limit as well as the 32-bit memory wall. It also demontrates the effect of tweaking the stack size on the virtual memory usage. The graph in figure 6 shows the maximal virtual memory allocation of the three simulators as a function of the number of consumers in the general scaling scenario. It shows that the JVM by itself allocates 1 GB of virtual memory. This is used as a code cache which contains the interpreter and code generated by the compilers. Although the size of this cache is adjustable, and future JVMs will default to a more reasonable amount, this will not impact our results as virtual memory is basically free on

64-bit machines. From the graph we can also observe that GridSim is capable of claiming close to 35 GB of virtual memory. The vastness of this allocation is due to the standard thread stack size of 1 MB. SimGrid allocates a significantly smaller amount of virtual memory. When we contrast this with the actual memory usage in figure 3 we can observe that SimGrid uses almost all of its virtual memory while GridSim uses less than a tenth of its allocation.

5 Conclusion

In this contribution we have compared the scalability of GridSim, SimGrid and GES. While both GridSim and SimGrid use a *multi-threaded discrete-event* core, GES uses a *single-threaded discrete-time* core. As we have observed from the results of our tests, both SimGrid and GridSim are unable to scale to the level that would allow them to simulate very large scale grid infrastructures such as EGEE. For GridSim the problem is rather fundamental in that it requires an amount of threads during simulation that reaches the upper limit of threads manageable by a normally configured Linux kernel. While SimGrid can sidestep this issue using `ucontexts`, it still reaches an upper limit of 23 000 simulated entities. These results show that when trying to simulate very large scale grids, a massively multi-threaded simulator is not the best choice.

Depending on the resolution at which time is to be simulated, it is better to choose either the discrete-event or discrete-time simulation paradigm. Whereas a detailed analysis of the impact of communication delays requires a high resolution and thus leans towards discrete-event simulation, a lower resolution discrete-time engine is adequate for simulating the execution and scheduling of jobs in a grid system. Irrespective of the choice for a discrete-time or discrete-event model, one can decide to use a threaded or non-threaded simulation core. We have quantified the impact of these choices through a comparative analysis of GES, SimGrid and GridSim and have shown their effect on simulation performance and scalability.

References

1. Legrand, A., Marchal, L., Casanova, H.: Scheduling distributed applications: the simgrid simulation framework. In: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2003), pp. 138–145. IEEE Computer Society, Los Alamitos (2003)
2. Buyya, R., Murshed, M.: Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computat. Pract. Exper. 14(13-15), 1175–1220 (2002)
3. Vanmechelen, K., Depoorter, W., Broeckhove, J.: A simulation framework for studying economic resource management in grids. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 226–235. Springer, Heidelberg (2008)

4. Aida, K., Tekefusa, A., Nakada, H., Matsuoka, S., Sekiguchi, S., Nagashima, U.: Performance evaluation model for scheduling in global computing systems. The International Journal of High Performance Computing Applications 14(3), 268–279 (2000)
5. Sulistio, A., Poduval, G., Buyya, R., Tham, C.K.: On incorporating differentiated levels of network service into gridsim. Future Gener. Comput. Syst. 23(4), 606–615 (2007)
6. Casanova, H.: Simgrid: a toolkit for the simulation of application scheduling. In: Proceedings of CCGrid 2001, pp. 430–437. IEEE Computer Society, Los Alamitos (2001)
7. Legrand, A., Lerouge, J.: Metasimgrid: Towards realistic scheduling simulation of distributed applications. Technical Report 2002-28, LIP (2002)
8. Buyya, R.: Economic-based Distributed Resource Management and Scheduling for Grid Computing. PhD thesis, Monash University, Australia (2002)
9. Assuncao, M., Buyya, R.: An evaluation of communication demand of auction protocols in grid environments. In: Proceedings of GECON 2006, pp. 24–33. World Scientific, Singapore (2006)
10. Vanmechelen, K., Broeckhove, J.: A comparative analysis of single-unit vickrey auctions and commodity markets for realizing grid economies with dynamic pricing. In: Altmann, J., Veit, D. (eds.) GECON 2007. LNCS, vol. 4685, pp. 98–111. Springer, Heidelberg (2007)
11. Stuer, G., Vanmechelen, K., Broeckhove, J.: A commodity market algorithm for pricing substitutable grid resources. Future Generation Computer Systems 23(5), 688–701 (2007)
12. Vanmechelen, K., Depoorter, W., Broeckhove, J.: Economic grid resource management for CPU bound applications with hard deadlines. In: Proceedings of CCGrid 2008. IEEE Computer Society, Los Alamitos (in press, 2008)
13. Barr, R., Haas, Z.J., van Renesse, R.: Jist: An efficient approach to simulation using virtual machines. Software Practice and Experience 35, 539–576 (2005)
14. Jacobs, P.H.M., Verbraeck, A.: Single-threaded specification of process-interaction formalism in java. In: Proceedings of the 36th conference on Winter simulation, pp. 1548–1555 (2004)