Application-Level Simulation Modelling of Large Grids

Serafeim Zanikolas and Rizos Sakellariou
School of Computer Science, The University of Manchester
Oxford Road, Manchester M13 9PL, U.K.
{zanikolas,rizos}@cs.man.ac.uk

Abstract

The simulation of large grids requires the generation of grid instances and an approximation of grid components' behaviour. To generate grid instances, this paper outlines a set of high-level properties of grids and considers ways to assign values to those properties. The paper also brings together existing application-level network and host models and discusses how they are used for the simulation of large grids. A grid instantiation is described in detail as part of a master-slave case study, and a large grid is simulated for the evaluation of a variety of scheduling strategies. The case study also motivates a performance prediction method, which is assessed against simulation results.

1 Introduction

Discrete-event simulation (hereafter referred as simulation) is a powerful means for comparative evaluation and analysis of systems behaviour, performance prediction, and identification and analysis of performance problems. As such, simulation is often used in the context of grid computing to assess hypotheses about a system's behaviour. Despite the number of simulation studies of a variety of problems in the grid, and the availability of several simulation toolkits, community effort has been mainly focused on relatively small scale simulations. Modelling small grids can be facilitated by relevant data from existing testbeds and planned deployments. However, when the subject of modelling is a grid of thousands of administrative domains with hundreds of thousands of hosts then questions arise, such as the latter's distribution and characteristics, as well as how these evolve over time.

This work is motivated by the need for a discrete-event simulation testbed for the evaluation of a scalable grid monitoring infrastructure [23], which is meant to form the basis of large-scale grid information services (such as grid search engines [8]). This paper contributes towards the simulation of large grids, particularly on how to generate grid instances and approximate grid components' behaviour. On the generation of grid instances, the paper outlines a set of high-level properties of grids and considers ways to assign values to those properties. The paper also brings together application-level network and host models and discusses how they are used effectively for the simulation of large grids. The above may be particularly useful to developers of custom grid simulators. These ideas are illustrated in a master-slave case study, in which several scheduling strategies are evaluated. A byproduct of the case study is a method to predict the best strategy for any given setting.

The remainder of the paper is organised as follows. The next section briefly introduces basic simulation concepts, elaborates on the parts of the grid that are actually modelled, and briefly considers related work. Section 3 outlines high-level grid properties and suggests ways to characterise those properties to generate grid instances. Section 4 considers existing work for modelling the dynamics and evolution of such grid instances, with a focus on hosts and networks. Using our simulator, which follows the outlined modelling approach, Section 5 presents evaluation results of several scheduling strategies for a master-slave problem in the context of a large grid. Section 6 concludes the paper.

2 Background and Related Work

2.1 Simulation Concepts

Simulation modelling refers to the abstraction of an otherwise complex system for the purpose of studying its behaviour with respect to particular aspects of interest. For instance, a host performance model has to account for relevant properties (e.g., hardware configuration) and ignore those that do not affect performance significantly. Application-level models in particular capture a system's specific aspect (e.g., host availability) without explicitly modelling all the reasons that may affect it (such as software or hardware failures, or network partitions.)

A discrete-event simulator implements a collection of models, such that simulated entities have a specified state;

0-7803-9074-1/05/$20.00

entities' state can change by the occurrence of events; and events are defined with respect to an internal clock. In the above example, a host's state is defined by a set of properties, including compute capacity and current system load, is affected by events such as an incoming request, and the execution of the request consumes some of the host's resources (and thus affects its state) for a particular set of time units.

An important tradeoff in simulation modelling is between complexity and accuracy. A simulator that implements detailed models is likely to deliver more accurate results but also to have a higher computational complexity, which means longer execution times. The complexity/accuracy tradeoff is even more significant in problems with large solution spaces, large model sizes (i.e., those involving many simulated entities), or both. It is in these cases that the use of application-level models is of major importance.

2.2 The Grid

The core components of grids are: (i) on-line general- or special-purpose computer devices and (ii) the network infrastructure that interconnects them. Thus, grid simulation models need to capture the characteristics and behaviour of those core components. Based on the needs of a specific case study, more models may need to be added on top of the core components to support data, software, services or any other resource types or abstractions of interest.

A grid's hosts and networks are structured in *grid sites*, in the same fashion that the Internet is organised in Autonomous Systems. An Autonomous System is one or more networks under the same administrative entity and thus with a single routing policy. Similarly, a grid site is one or more networks and resources therein with a common sharing and security policy.

Even given these assumptions, many modelling parameters depend on the type of the grid that is modelled. For instance, whether the subject of modelling is a traditional high-performance grid or a more diverse setting that, in addition to high-performance resources, includes commodity resources, such as PCs. In the remainder of this paper, these two types of grids will be referred as "traditional" and "diverse" respectively.

2.3 Related Work

Simulation is often used in the context of grids and peer-to-peer networks, to study a variety of problems. Common problem domains include job scheduling (e.g., [21], Bricks [22], SimGrid [13], HyperSim [17], GridSim [4]); and data replication algorithms to achieve better performance (ChicagoSim [18], OptorSim [5], GridNet [12]) and or high availability of data ([19], [24]). Regardless of the problem of interest, simulation studies have to model host and network behaviour, particularly the duration of data transfers and that of processing requests or jobs. For instance, the practices for the estimation of a data transfer's duration, vary from a constant end-to-end bandwidth to a consideration of the utilisation levels and bandwidth sharing mode in all the links that are actually used in the transfer.

Despite the variety of studied problems and modelling practices, the existing discrete-event simulation studies are typically concerned with only a small number of grid sites, and tenths or up to a few hundreds of hosts. In contrast, due to the large-scale nature of our motivating use case [23], this work is focused on application-level simulation of grids of thousands or up to millions of hosts.

A notable exception is the work in [17], which illustrates the scalability of HyperSim in a performance study that includes up to 16384 hosts. HyperSim uses an application-independent method to reduce the number of simulation events and hence computational complexity. The particular study, however, focuses mainly on hosts modelling and entirely ignores network behaviour. On the other hand, our work models both networks and hosts behaviour and attempts to achieve this by using high-level models and making simplifications that do not drastically affect accuracy.

In a broader context, packet-level network simulators, such as NS [3], have been traditionally used by the networking community to study low-level network behaviour to assess protocol enhancements. In contrast, the work in this paper focuses at the application level and specifically in grid settings.

3 Generating Grid Instances

3.1 Grid Configuration

From a high level perspective, a grid instance (i.e., a grid at a particular point in time) is described along the following parameters, collectively referred as a *grid configuration*: (1) the number of grid sites; (2) the number of hosts that are shared via the grid; (3) the distribution of Internet connection capacity of grid sites; (4) the distribution of hosts among grid sites; (5) the mapping of the set of Internet connection capacities to that of grid sites; (6) the distribution of host types, such as desktops, cluster hosts, parallel machines, on-line scientific and other special-purpose instruments; (7) the distribution and characteristics of resource types within hosts (e.g., the available storage within a host).

Depending on the problem considered, some of these parameters may be of minor or major importance, and thus may be ignored or specified in more detail. The assignment of values to these parameters and the generation of relevant data for a particular case study is called a *grid instantiation*.

3.2 Grid Instantiation

Grid instantiation depends heavily on the type of grid that is modelled. The number of grid sites and hosts should obviously take into account the grid type and the characteristics of the studied problem.

The distribution of Internet connection capacity of grid sites and that of hosts among sites is modelled with uniform and highly skewed, such as Zipf, distributions for traditional and diverse grids, respectively. In traditional grids, most grid sites are well-connected and have a considerable number of hosts; thus, a uniform distribution seems appropriate. On the other hand, we can assume that in diverse grids the norm will be low-end grid sites with a few hosts, and a considerably smaller number of high performance sites. This assumption is made on the basis that federations of self-governed nodes (grid sites in this case) tend to evolve into scale-free networks [1], which consist of a few nodes of significant importance (hubs) and numerous insignificant nodes. Examples of highly-skewed distributions in scale-free networks are various Internet properties, such as the connectivity degree in the Internet topology [10, 15] and in peer-to-peer networks [20], and the number of pages and visitors per web site [7]. On this basis, diverse grids may have a large number of poorly connected grid sites with only a few resources and a small number of highly connected grid sites with a large number of resources.

The mapping of the set of Internet connection capacities to that of grid sites, is performed considering the number of local hosts. In particular, the largest site is assigned the highest available bandwidth and so on, until all assignments are in place. To account for real-world cases where a site has less resources but more bandwidth than another site, a number of swaps is performed between randomly selected sites on the condition that the difference between the two bandwidth values does not exceed a threshold. Using this mapping method, a grid instance has reasonable grid site bandwidth assignments with a few exceptions that have smaller or larger bandwidth with respect to what one would expect given the number of their local hosts.

The distribution of host types, and the distribution and characteristics of resource types within hosts are defined using conditional probability rules [15]. Particularly in diverse grids, we consider the former distribution as highly skewed (e.g., Zipf) on the basis that commodity hosts are far more common than high performance hosts.

4 Modelling Grid Dynamics and Evolution

Once a grid instance is in place, a discrete-event simulator, that implements host and network models, can simulate a grid throughout time. This section briefly describes existing models for this purpose, and discusses how they are used in our simulator. The criterion for choosing these models is simplicity. Apart from the attributes that are defined as part of grid instances, these models require little state information that is specific to entities (such as grid sites, links or hosts).

4.1 Network Dynamics

Networks are immensely complex due to diversity at all levels: end hosts with various implementations of a TCP/IP protocol stack, communication via diverse network devices over different mediums of various characteristics. Also, features like asymmetric routing often result in counterintuitive behaviour, such as considerably different transfer rates between the inbound and outbound channels of the same software connection [11].

Because of the focus on application-level studies in large grids, we model key features of network behaviour instead of low-level activities such as packet switching. Our main concern is the duration of data transfers, typically estimated as the round trip time (RTT), plus the ratio of transfer size over available bandwidth. In addition to RTT, estimations of available bandwidth need to account for bandwidth sharing in bottleneck hosts and current utilisation levels of the links that are involved in a transfer.

RTT is the interval from a packet's transmission until the receipt of the corresponding TCP acknowledgement. RTT appears to be described well by a shifted gamma distribution [16] with unit scale. In particular, if Z is a gamma distributed random variable, $RTT = mZ + c$, where c is a constant relating to the smallest observed RTT and m is the gamma distribution's scale parameter. RTT can vary significantly during the course of a day (partly due to daily human activity) and depending on the physical distance between the two communicating ends. Fig. 16 in [16] shows indicative parameter values to capture these variations. Note that latency and RTT can be different for the two directions of a software connection (due to asymmetric routing), thus separate values must be sampled. Our simulator implements this RTT model because it introduces a small time overhead that is proportional to the number of active network connections, and it does not need any special entity attributes (beyond the basic network information in grid instances.)

Another aspect of network behaviour is the way bandwidth is shared in bottleneck nodes, for instance in a server that is overwhelmed (bandwidth-wise) by downloads. The intuitive, though naive, approach is to queue the requests, so that bandwidth is used on a first come first serve basis. Casanova and Marchal [6] propose a bandwidth sharing model for TCP where the bandwidth allocated to competing transfers is inverse proportional to their RTT. Thus, for every transfer t with latency RTT_t that goes through a bottleneck node with nominal bandwidth capacity C, the effective

bandwidth is

$$\frac{1/RTT_t}{\sum_{t' \in T} \frac{1}{RTT'_t}} \times C, \quad (1)$$

where T is the set of competing transfers. It follows from the equation that when all competing transfers have the same RTT, the bandwidth is allocated in equal fractions. This is the case for instance when hosts in a LAN compete for bandwidth in the outgoing link.

Returning to our primary concern, the end-to-end bandwidth of a data transfer is the minimum of all bandwidth allocations for that transfer in all the involved nodes. (Actually the reality is slightly more complicated due to multi-pathing, i.e., a transfer's packets may follow different routes.) The outlined model combined with simple network topologies is used in the SimGrid simulation framework [13] to estimate end-to-end bandwidth, and hence transfer durations. However, this model is too expensive for large settings, where intermediate bottleneck links may be shared among hundreds or thousands of network connections. Because our simulator is intended for significantly larger settings than SimGrid, we assume a network topology without intermediate bottlenecks and thus bandwidth sharing is applied only to end hosts.

Finally, despite the existence of generation methods for self-similar traffic [14], our simulator does not currently account for routine load due to their high computational complexity, which is further exaggerated by the large number of network links.

4.2 Host Dynamics

Host properties of interest may vary among different studies; this section is limited on host availability. Host availability refers to whether a host is operating properly and is reachable over the Internet (i.e., "up and running"). A host may be unavailable for a variety of reasons, including software, hardware, or network failures. Based on empirical evidence, Douceur [9] formulates a model, in which the logarithm of hosts availability (i.e., the duration of uptime periods) is uniformly distributed in the ranges $[0, b]$ and $[0, c]$, where $b < c$. The first class corresponds to hosts that are turned off on a regular basis (e.g., daily or weekly); the second class is about hosts that do not exhibit a routine availability behaviour. The distribution of hosts in the two classes is indistinct, which means that some hosts belong in any of the two classes throughout time. To the best of our knowledge, no corresponding generic model exists for the duration of unavailability intervals. Bolosky [2] constructs an empirical model for downtime intervals of hosts in a corporate environment. The model is a mixture of two uniform distributions for 14- and 64-hour downtime intervals, and a gamma distribution for the hosts that do not have a cyclical availability behaviour. Only the latter distribution should be used in the simulation of traditional grids (in which hosts are not meant to be turned off routinely.) This model, currently implemented in our simulator, introduces a time and space overhead that is proportional to the total number of hosts in a grid instance. The model needs every host to have a few extra attributes (such as the current availability status and the timestamp this will change) that need to be periodically updated. Despite the introduced complexity, a hosts availability model has general use, and eliminates the need to account for network failures (at least those related to access networks.)

4.3 Evolution Patterns

In addition to dynamics, the evolution of grids may also be of interest in studies that involve large simulated intervals. This evolution can be quantified at a high level by means of the rate at which new (resp. existing) sites and resources are being added to (resp. removed from) the grid. Despite the lack of any grid-specific work in this area, we would expect traditional and diverse grids to evolve as random and scale-free networks, respectively. Barabási and Bonabeua [1] have observed that new nodes in scale-free networks tend to associate with well-connected nodes; a "rich get richer" process called preferential attachment. In the context of diverse grids, this suggests that the probability of new resources being added to a site is proportional to the number of resources already at that site.

5 A Master-Slave Case Study

The simulator was partially validated by reproducing the task scheduling simulation experiment in [13]. Using our simulator, an evaluation of the scheduling strategies considered in the above reference in a similar setting, yields identical performance trends. In this section, we report simulation results on a master-slave case study where a master dispatches a given task, from a set of identical tasks, to slaves, one at a time. The master receives requests from slaves, at exponentially distributed intervals with a mean inter-arrival time of two seconds. The master selects a slave from the set of heterogeneous slaves that have sent requests and dispatches a task to it; once the task transfer is complete, the master proceeds with scheduling the next task, until all tasks are assigned. The aim of this scheduling process is to minimise the overall execution time.

Using a baseline measurement, each task is assumed to demand a certain amount of computation (referred to as *computation cost* and expressed in mega floating point operations), and a certain amount of data that needs to be transferred from the master to the slave before its execution starts (referred to as *communication cost* and expressed

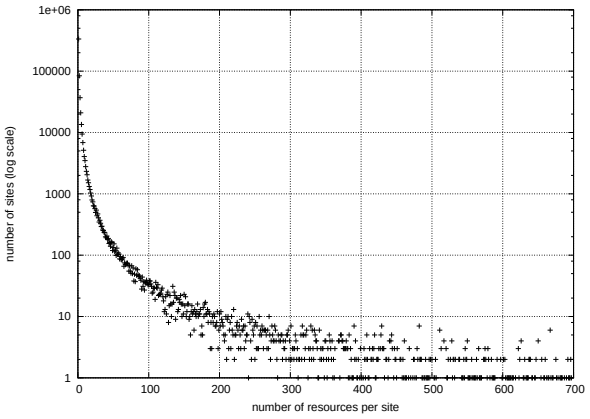


Figure 1. The Zipf distribution of hosts among sites (with a maximum of 700 hosts per site.)

in MB). The combination of (i) computation and communication costs of tasks; and (ii) compute and network capabilities of slaves (that is, the hosts that choose to send a request to the master, from all the hosts in a grid instance), constitute a *setting*. The aims of this case study are (i) to understand the performance behaviour of a number of different scheduling strategies across different settings; and (ii) to define and assess a method to predict which strategy would be the best for a particular setting.

5.1 Simulation Settings

Simulations were run using a large diverse grid instance (i.e., including unreliable low-end grid sites) of approximately 2.3 M hosts, Zipf-distributed among 500 K sites, with an upper bound of 700 hosts per site. Figure 1 shows the distribution of hosts among sites in the used grid instance. The nominal bandwidth of sites is between 256 Kbps and 64 Mbps, and was generated using 2^{17+n} where n is Zipf-distributed in the range $[1, 9]$ (one being the most common and nine the most rare.) Given the set of resource assignments to sites and the set of bandwidth values, bandwidth values were assigned to sites as described in Section 3.2. For the reasons described in the aforementioned section, a number of swaps were performed (10% of all assignments) between randomly selected sites on the condition that the difference between the two bandwidth values is at most three (in the range $[1, 9]$.) The threshold is used to minimise the possibility of swaps between sites with extremely low and high connections.

The host types are set to PCs, cluster nodes, supercomputers, and two more types corresponding to special-purpose hosts. Again, the overall frequency of resource types is Zipf-distributed (approximately $0.92, 0.05, 0.012$ for PCs, cluster nodes and supercomputers, respectively; the remainder being for special purpose hosts.) The assignment of types from the overall distribution to the particular hosts of every site is biased based on a site's bandwidth, to reflect for instance the intuition that supercomputers are not typically found on low-bandwidth sites. The computational capacity of hosts is uniformly distributed in the range $[40, 200]$ (MFlops) for PCs and cluster nodes, and in $[80, 300]$ for supercomputers.

The availability of hosts is determined as described in Section 4.2, with PCs most likely being in the set of hosts that have a cyclical availability behaviour (i.e., turned off routinely) and cluster nodes and supercomputers most likely having a non-cyclical behaviour. Unavailable hosts are not considered during scheduling. Available hosts that are assigned a task are assumed to remain available for the time needed to process the task.

5.2 Scheduling Strategies and Metrics

The following scheduling strategies have been evaluated. In these definitions w_i (resp. c_i) refers to the time that slave i needs to process (resp. receive) a task. (i) *Greedy (FCFS)*: select slaves on a first come first serve basis; (ii) *Compute-centric (CPU)*: select the slave with the highest compute capacity; (iii) *Bandwidth-centric (BW)*: select the slave with the highest bandwidth capacity; (vi) *Throughput (THR)*: select the slave with the highest throughput, defined as $\frac{1}{c_i+w_i}$ i.e., the number of completed tasks per time unit by slave i; (v) *Compute-biased throughput (CTHR)*: as in THR but favouring slaves with large compute capacity: $\frac{1}{(c_i+w_i)\times w_i}$; (vi) *Bandwidth-biased throughput (BTHR)*: as in THR but favouring well-connected slaves: $\frac{1}{(c_i+w_i)\times c_i}$; (vii) *Earliest finish time (EFT)*: select the slave that will yield the earliest finish time for the next task, i.e., the one with the smallest $max(x_i, c_i) + w_i$, where x_i is the remaining compute time of slave i if it already runs a task, or zero otherwise.

In a homogeneous environment, an intuitive approach to select a strategy would be based on whether the setting is compute- or communication-bounded. However, in a heterogeneous environment, such as the grid, this depends on the capabilities of the slaves selected. Since a strategy may select different slaves, we use the assignment of each strategy as a basis to express such properties. Thus, for a strategy r, we define the *Communication-Computation Ratio* as

$$CCR_r = \frac{\sum_i c_i}{\sum_i w_i}, \tag{2}$$

where i iterates over all tasks.

Another useful metric is the *Maximum Number of Slaves* (MNS) than can be used at the same time in a particular setting. This is related to the number of task transfers that can be completed during the computation of a single task and as a result it is inversely proportional to CCR_r.

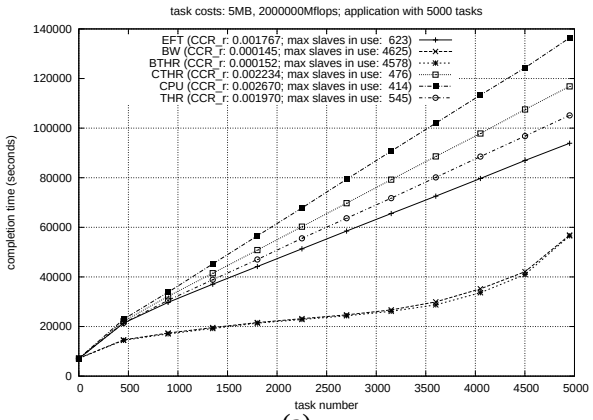
5.3 Results and Discussion

We have run simulation experiments in the described grid instance using 56 different combinations of communication cost, computation cost and number of tasks, in the intervals $[1, 200]$, $[1000, 4000000]$, and $[500, 10000]$, respectively. Results from three of those experiments are shown in Fig. 2. Each figure shows the completion time of the n-th task throughout an experiment, for each strategy except FCFS (FCFS is by far the worst in all experiments; being an outlier it clutters the figures, thus it is omitted). For each strategy r, the value of CCR_r and the maximum number of slaves used (MNS) are also shown.

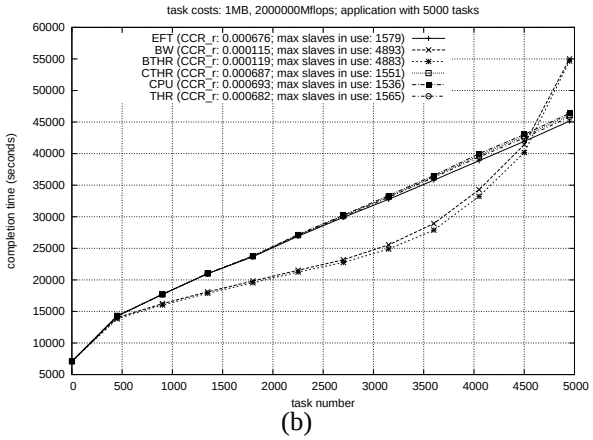
As seen from the figures, the behaviour of the strategies can be classified in two groups: (i) CPU, CTHR, THR and EFT; and (ii) BW and BTHR. Typically, either BTHR or EFT delivers the best performance. In all our experiments, there were a few only exceptions in which BW was better than BTHR, and CPU and CTHR were better than EFT but the difference in terms of time was negligible (no more than 0.027%). Another interesting observation, which can be noticed in Fig. 2(b) and 2(c), is that even though the same setting is used, EFT is best in the first case and BTHR in the second. A final observation is that CPU, CTHR, THR and EFT tend to complete tasks in even intervals, whereas BW and BTHR tend to complete earlier tasks much quicker than later tasks. An observation not obvious in the figures shown is that results among CTHR, THR, EFT and BTHR are almost identical when MNS is very small. This happens because when only a small number of slaves is needed, it is likely that the same or comparable slaves are chosen.

The key to understand the strategies' behaviour is that task communication is a serial process (one at a time) whereas task computation is taking place in parallel (since multiple tasks may be running by different slaves at the same time.) Thus strategies that ignore (e.g., CPU) or underestimate (e.g., CTHR) the communication costs perform poorly. THR and EFT weight communication and computation costs equally and thus perform better in environments where computation costs dominate. THR and EFT do not perform well in environments where communication dominates because they fail to acknowledge that computation costs are mitigated by parallelism and thus should be treated as less important. BTHR, being a bandwidth-biased variation of THR, is better in environments where communication dominates because it underestimates the importance of compute costs. On the other hand, BW completely ignores compute costs and thus is typically worse than BTHR.

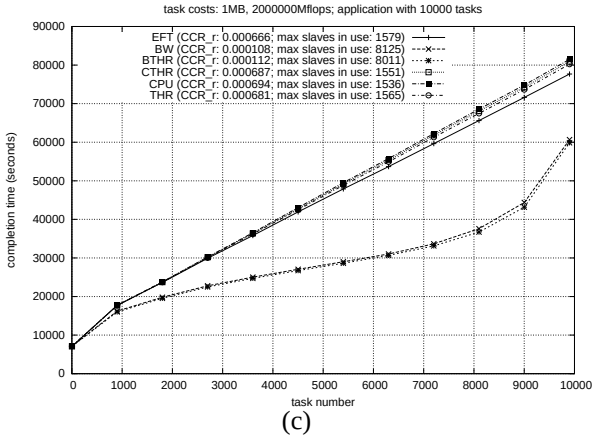
In addition to completion time, the efficiency of slave usage is of interest. That is, the compute capacity utilisation level of the selected slaves throughout an experiment. For instance, in Fig. 3, more than half of the slaves used by BTHR are idle for more than half of the overall execution



(a)



(b)



(c)

Figure 2. Completion time of the n^{th} task, for all strategies in various settings and number of tasks.

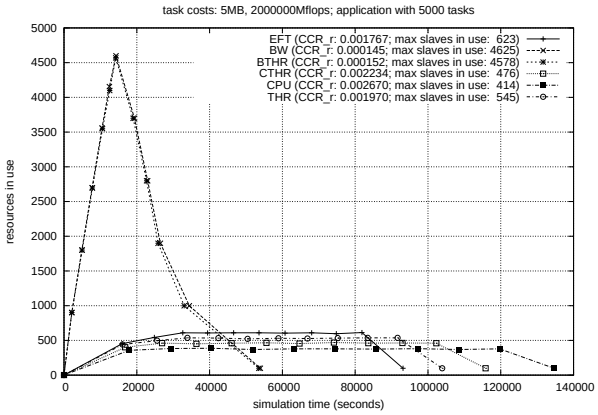


Figure 3. Slave usage throughout the experiment in Fig. 2(a)

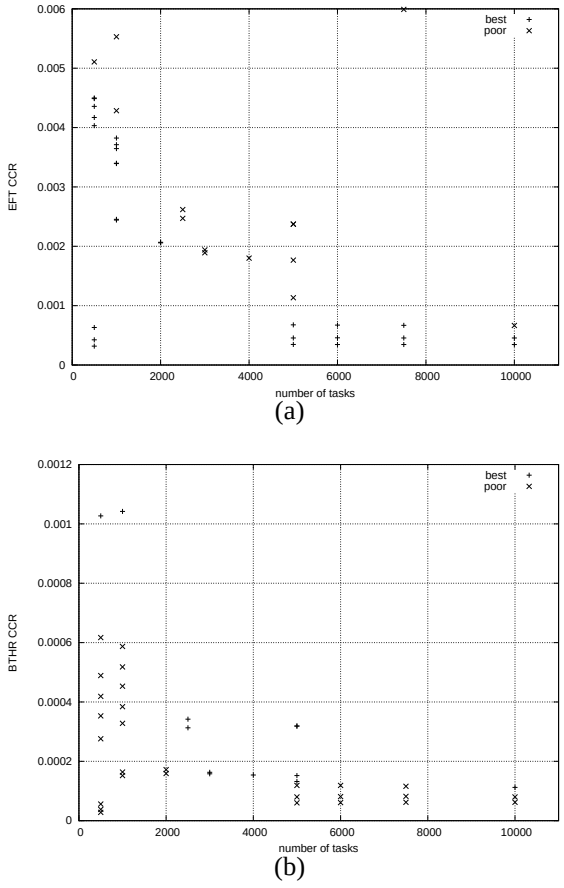


Figure 4. Performance of EFT (a) and BTHR (b) with respect to CCR_r and number of tasks.

time. In contrast, most of the slaves used by EFT are busy for the most part of the overall execution time. Although we do not formally define slave usage efficiency, these observations suggest that BTHR is less efficient than EFT.

In order to be able to predict for a given setting and number of tasks which strategy is best, we consider the two typically best strategies, EFT and BTHR, and how they perform in relation to their CCR value and the number of tasks. The results, shown in Fig. 4, indicate which of the two strategies is best in all the experiments performed.[1]

The results in Fig. 4(a) suggest that EFT is the best strategy when CCR_{EFT} is below a threshold, and that the threshold is inversely proportional to the number of tasks. Similarly, BTHR in Fig. 4(b) is the best strategy for settings with CCR_{BTHR} above a threshold, and the threshold follows a similar trend as in EFT. To formalise the situations where each of the two strategies is best, we have taken the product of CCR_r and number of tasks. For all experiments, the value of this product that distinguishes the performance behaviour of EFT and BTHR is about 5 when CCR_{EFT} is used in the product, and 0.5 when CCR_{BTHR} is used. In either case BTHR is best for product values above 5 or 0.5.

As a result, in order to predict which is the most appropriate strategy to use for a given setting and number of tasks, all that is needed is to come up with a good estimate for CCR_{BTHR} or CCR_{EFT}. We calculated an estimation for CCR_{BTHR} based on

$$CCR'_{BTHR} = \frac{commDemand/bandwidthCapacity}{compDemand/computeCapacity} \quad (3)$$

where $commDemand$, $compDemand$ refer to the communication and computation costs of a task, and $bandwidthCapacity$, $computeCapacity$ are estimations of the communication and compute capacity of the slaves that will be chosen by BTHR. In order to estimate $bandwidthCapacity$ and $computeCapacity$, we experimented with combinations of average and best values of bandwidth and computer capacity of all available slaves. We found that using best bandwidth and average compute capacity yielded the best results. In that case, choosing a strategy as follows (N denotes the number of tasks)

$$strategy = \begin{cases} \text{EFT, if } N \times CCR'_{BTHR} < 0.5 \\ \text{BTHR, otherwise} \end{cases} \quad (4)$$

we were able to select the right strategy in 82% of the cases. If MNS is small (e.g., < 5), either strategy can be chosen.

[1] We show two figures because CCR_r varies between BTHR and EFT since this calculation depends on the capabilities of the slaves used. EFT selects more powerful compute-wise slaves, thus the compute times are smaller (the denominator in Eqn 2) and hence CCR_{EFT} is larger compared to CCR_{BTHR}. Nevertheless the overall trends are similar.

6 Conclusions and Future Work

This paper outlined a set of high-level grid properties that form a grid configuration, and considered ways to specify their values, for the generation of (data that describe) grid instances. The paper also discussed how existing application-level models can capture the behaviour of networks and hosts, with affordable computational complexity for the simulation of large grids. Grid instantiation was illustrated in a master-slave case study, and a large grid was simulated for the evaluation of several scheduling strategies. Our simulator allowed us to explore a significantly larger search space than other studies [13], improving the understanding of how performance behaviour varies in that space. Based on simulation results, a method was defined and assessed to predict the most appropriate strategy for a given setting. Finally, as part of our motivating use case, we are interested in testing the described modelling approach in studies that involve a considerably larger number of concurrent network connections, compared to those in the case study in this paper.

References

[1] A.-L. Barabási and E. Bonabeau. Scale-Free Networks. *Scientifi c American*, 288:50–59, May 2003.

[2] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 34–43, 2000. ACM Press.

[3] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in Network Simulation. *IEEE Computer*, 33(5):59–67, 2000.

[4] R. Buyya and M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):53–62, November-December 2002.

[5] D. G. Cameron, R. Carvajal-Schiaffi no, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini. Analysis of Scheduling and Replica Optimisation Strategies for Data Grids Using OptorSim. *Journal of Grid Computing*. (to appear).

[6] H. Casanova and L. Marchal. A Network Model for Simulation of Grid Application. Technical Report 2002-40, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, Oct. 2002.

[7] M. E. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffi c: Evidence and Possible Causes. In *1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 160–169, 1996. ACM Press.

[8] M. Dikaiakos, Y. Ioannidis, and R. Sakellariou. Search Engines for the Grid: A Research Agenda. In *1st European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 49–58, 2004. Springer-Verlag.

[9] J. R. Douceur. Is Remote Host Availability Governed by a Universal Law? *ACM SIGMETRICS Perform. Eval. Rev.*, 31(3):25–29, 2003.

[10] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, 1999.

[11] S. Floyd and V. Paxson. Diffi culties in Simulating the Internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403, 2001.

[12] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of Dynamic Data Replication Strategies in Data Grids. In *International Parallel and Distributed Processing Symposium (IPDPS'03)*, page 100, 2003.

[13] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 138–145, 2003. IEEE CS Press.

[14] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the Self-Similar Nature of Ethernet Traffi c (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15, 1994.

[15] D. Lu and P. Dinda. Synthesizing Realistic Computational Grids. In *Supercomputing 2003 (SC 2003)*, 2003.

[16] A. Mukherjee. On the Dynamics and Signifi cance of Low-Frequency Components of Internet Load. *Internetworking: Research and Experience*, 5(4):163–205, Dec. 1994.

[17] S. Phatanapherom, P. Uthayopas, and V. Kachitvichyanukul. Fast Simulation Model for Grid Scheduling Using HyperSim. In *Winter Simulation Conference*, pages 1494–1500, 2003.

[18] K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[19] K. Ranganathan, A. Iamnitchi, and I. Foster. Improving Data Availability Through Dynamic Model-Driven Replication in Large Peer-To-Peer Communities. In *Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, pages 346 – 351. IEEE CS Press, 2002.

[20] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *IEEE Internet Computing*, 6(1), 2002.

[21] G. Shao, F. Berman, and R. Wolski. Master/Slave Computing on the Grid. In *9th Heterogeneous Computing Workshop*, pages 3–16. IEEE CS Press, 2000.

[22] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 97–104. IEEE CS Press, 1999.

[23] S. Zanikolas and R. Sakellariou. Towards a Monitoring Framework for Worldwide Grid Information Services. In *10th International Euro-Par Conference*, volume 3149 of *Lecture Notes in Computer Science*, pages 417–422, 2004. Springer-Verlag.

[24] K. G. Zerfi ridis and H. D. Karatza. Large Scale Dissemination Using a Peer-to-Peer Network. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 421–427. IEEE CS Press, 2003.