RC23573 (W0503-140) March 30, 2005
Computer Science

IBM Research Report

Simulation of Grid Computing Infrastructure: Challenges and Solutions

Sugato Bagchi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

SIMULATION OF GRID COMPUTING INFRASTRUCTURE: CHALLENGES AND SOLUTIONS

Sugato Bagchi

Mathematical Sciences Department
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, U.S.A.

ABSTRACT

Recent advances in middleware technologies such as grid computing have provided IT architects with the ability to design infrastructures that are more flexible and less dedicated to specific application workloads. However, the capabilities of design and analysis tools that IT architects use have not kept pace. In this paper, we describe our progress in developing an IT infrastructure modeling environment that supports an extensible set of analysis tools. We focus in particular on a discrete-event simulator for analyzing the performance of computational workloads on a grid. The unique modeling requirements and challenges presented by the grid computing infrastructure domain are discussed. Efficient event queue management and other simulation techniques to address these challenges are developed. Finally, we position the role of simulation analysis in the larger context of estimating the business and financial returns from grid computing investments.

1 INTRODUCTION

Grid computing technology has made it technically feasible to "virtualize" IT resources by removing the dedicated linkage between application software workloads and the hardware assets, such as servers, data storage and networks, required for processing them. At the same time, businesses are becoming more interested in understanding how their IT infrastructure can support various other aspects of an adaptive, on-demand operating environment, such as being responsive towards unanticipated surges in workload volumes and being resilient to unexpected outages of individual servers, network links, or entire locations. These technical objectives are coupled with the business need to shift from reliance on fixed capacity (and therefore cost) assets to a more variable cost structure for processing IT workloads.

These technology advances and business requirements have outpaced the traditional IT modeling, design, and evaluation tools that are currently in use. IT consultants, architects and their clients are therefore unable to clearly visualize, estimate, and quantify the technical and financial impact of these grid and other on-demand initiatives. For example, as IT resources become more flexible and less dedicated to specific applications, we need the capability to design schedulers to route workloads to resources and incorporate their behavior into planning for the optimal resource capacity required over time. To estimate the level of responsiveness and resiliency in an infrastructure design, we need the capability to simulate random variability and shocks in workload demands and resource availabilities. To estimate the financial impact of these on-demand characteristics, including variable cost structures that are pegged to the usage of infrastructure rather than their ownership, we also need to embed financial analysis capabilities in these design and modeling tools.

We have developed an infrastructure modeling environment named IBM Grid Value at Work to provide these analysis capabilities. Figure 1 shows its architecture. A set of analysis tools are plugged in to a common infrastructure model specified in XML. These tools can be operated through a common user interface.

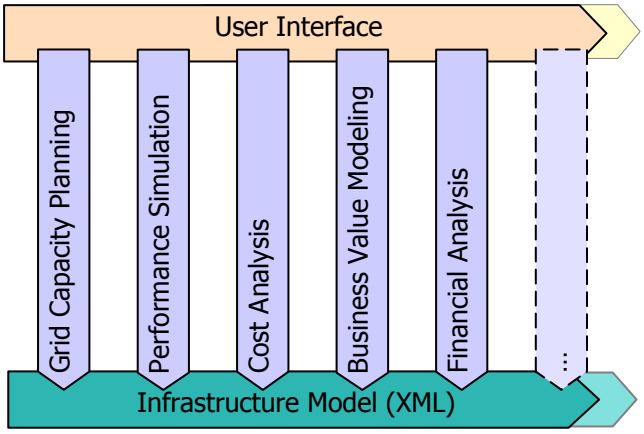

Figure 1: IBM Grid Value at Work

This architecture allows for the independent development of new analysis capabilities, to be used in conjunction with the existing analyses or in a standalone manner.

In this paper, we focus on the performance simulation capability in IBM Grid Value at Work. In the following section, we describe the grid computing environment to be simulated and identify the simulation requirements. Section 3 assesses the ability of existing simulation tools to address these requirements. Section 4 describes our simulator design, which addresses the gaps in existing simulation tools with regard to the grid simulation requirements. In Section 5, we analyze the performance of our design with respect to its scalability to large simulation models. We conclude in Section 6, by describing the role of the simulator in the larger context of estimating the business and financial returns from grid computing investments.

2 THE GRID COMPUTING ENVIRONMENT

Grid computing (Foster et al. 2002) enables the aggregation of distributed computing and data resources, that are heterogeneous and owned by multiple administrative units, into a large, integrated computing system that provides seamless access to multiple, redundant, and disparate resources. Applications that benefit from grid are those that have inherent parallelism, require high throughput volumes, or need to integrate data across multiple sources.

Figure 2 shows an operational perspective of a generalized grid environment.

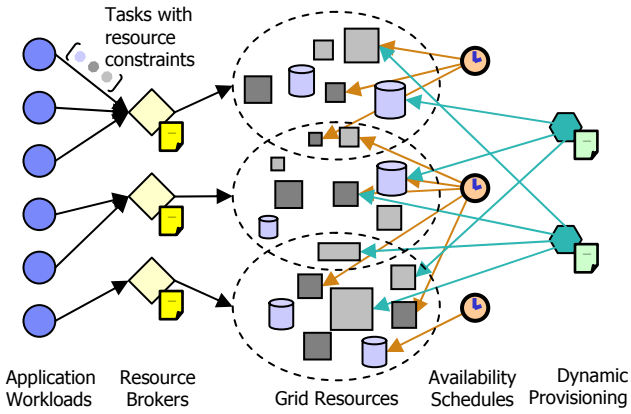

Figure 2: Operational Flows in a Grid Environment

Application workloads that contain a set of tasks to be processed by the resources on the grid are submitted to resource brokers (or managers) that serve as "gateways" to the grid. In general, there may be multiple such brokers in the grid environment. Each broker has a scheduling policy that dictates how to assign the tasks in the workload to the available resources on the grid. As the figure shows, a broker may be able to submit tasks to only a subset of all resources on the grid (shown by the dotted ovals). All the resources need not be of the same type (unlike clusters) and could be of heterogeneous hardware platforms and operating systems. The availability of a resource on the grid may be dictated by a calendar based schedule (shown by the clock icons). Resource availability may also be influenced by dynamic provisioning. Provisioning is the automated capability of configuring a resource to be able to process tasks of a certain type. For example, a server may be set up with application server software to process a certain type of online transactions. Dynamic provisioning (also known as orchestration) uses policies to determine when a resource should be switched over from being able to process tasks of one type to another, e.g., from a web server to an application server (Appleby et al. 2004).

The simulation of a grid environment requires the modeling and implementation of several features:

- **Multi-tasking IT resources**: Processors, network links, and data storage and servers are all preemptive multi-tasking resources. Tasks submitted to such a resource goes immediately to the processing queue containing all the other tasks that are being processed by the resource. Each of these tasks get processed by the resource for a predefined time-slice before being put back to the processing queue. As a result of multi-tasking, the submission of a new task can change the completion time of the existing tasks being processed by the resource. The multi-tasking behavior is also influenced by the fact that IT resources are often multi-processing (e.g., n-CPU servers). A detailed simulation of the task management within each resource would be too time consuming, when considering grid environments with hundreds of resources simulated for the duration of weeks. Instead, when a new task is submitted to a resource, the simulator should perform a good approximation of the multi-tasking behavior by re-estimating the completion time of all tasks being processed by that resource.
- **Job decomposition**: Each job from a workload may be composed of multiple resource requirements. We use the terminology of a "task" to define a single resource requirement within a job. Each resource requirement may be for a specific configuration of a server or a database. For example, a job may be composed of three tasks: get data from a customer database, perform data-mining on a compute server, and add the results to a sales database. Job decomposition is important to model because grid designs typically focus on providing a certain type of resource (e.g., compute servers) on a grid and overlook the impact of other resources (e.g., databases, network bandwidth) needed by the job.
- **Task parallelization**: Each task in a job (decomposed as described above) may be parallelizable. This is often the case with grid workloads. The parallelization could be of two types: "Embarrass-

ingly parallel" tasks can simply be split up into as many chunks as available resources. The second type of parallel tasks are more constrained and consist of a specific set of parallel paths, regardless of the number of available resources.

- **Heterogeneous resources**: Grid resources can be of various vendor platforms, models, and operating systems. Therefore, the processing time of a task on a resource is subject to its performance benchmarks. A resource may be associated with multiple performance benchmarks that are relevant for different types of computing tasks. For example, SPEC-FP and SPEC-INT ratings are relevant for compute-intensive tasks while TPC-C ratings should be used for transactional tasks that have substantial data input-output.
- **Resource scheduling**: The grid simulator must be able to model the scheduling policies used by resource brokers to determine which resource should process an arriving task. Typical policies are rules such as:

  – *load leveling*: the task is sent to the least utilized resource so as to balance utilization across all resources compatible to the task,
  – *greedy*: the task is sent to the resource that can complete it the fastest – which is a function of its processing power and current level of utilization,
  – *round robin*: the task is sent to the next compatible resource in a sequence,
  – *threshold-based*: tasks are sent to a preferred resource until a certain performance parameter threshold (e.g., server utilization) is breached.

  Grid designers can use simulation to evaluate these choices compare the respective grid performances to determine the best policy.
- **Resource provisioning**: The ability to provision resources for processing particular types of tasks is another feature to be simulated. These provisioning policies could be either calendar based (e.g., a department needs a server to be an email server during 9AM to 5PM every workday and can release it to be a data-mining processor at other times) or based on a more dynamic policy that monitors workload arrivals and resource usage and reacts accordingly. Simulation of these provisioning policies in conjunction with the resource scheduling policies would allow grid designers to determine whether the respective policies are aligned and consistent with each other and providing the desired grid performance in terms of throughput or processing times.
- **Non-programmer user interface**: The expected users of the simulation are grid designers and consultants, who are unfamiliar with simulation languages. Therefore, the simulator must provide a graphical modeling environment and user interface. It should also allow the grid designer to write new resource scheduling and provisioning policies and incorporate them into the simulation.

3 ASSESSMENT OF EXISTING SIMULATION TOOLS

The requirements for the simulation of grid computing infrastructure have been addressed to varying degrees by a few simulators designed for grid computing.

GridSim (Buyya and Morshed 2002) is a Java-based simulation toolkit based on the SimJava library (Howell and McNab 1998). It was designed specifically to analyze and compare the performance of resource scheduling algorithms for the grid. The salient capabilities of this toolkit are the modeling of heterogeneous, multi-tasking grid resources, calendar based resource provisioning, parallel application models, and resource scheduling. It has been used to evaluate the performance of various resource scheduling algorithms based on deadline and budget based constraints. The key drawback of this tool is its implementation of each grid task as a separate thread in the underlying Java Virtual Machine. As demonstrated by Phatanapherom and Kachitvichyanukul (2003), this limits the scalability of the GridSim in tackling enterprise level simulation requirements where thousands to millions of grid tasks may have to be simulated. The toolkit also assumes a programmer-level interaction with it. For example, it relies on the programmer to implement the application job decomposition model. This makes it not suitable for field use by grid designers and consultants.

SimGrid (Legrand et al. 2003) is another simulation framework for the purpose of evaluating grid scheduling algorithms. It has a heterogeneous and multi-tasking resource model. Multi-tasking is implemented as a "shared mode" usage of the resource where all ready tasks are executed concurrently. A key strength and focus of SimGrid is to model the grid network topology and simulate the data flow over the available network bandwidth. However, it does not incorporate a realistic view of grid application workloads by not modeling job decomposition and task parallelization characteristics. Resource availability (calendar based or dynamic provisioning) is also not modeled.

In contrast, OptorSim (Cameron et al. 2003) explicitly accounts for both dynamic provisioning and resource scheduling policies in its model. It focuses on analyzing the interaction of these policies and estimating the resulting performance. Dynamic provisioning is performed in the context of replication of data to resources in a data grid. Various data replication strategies can be evaluated from

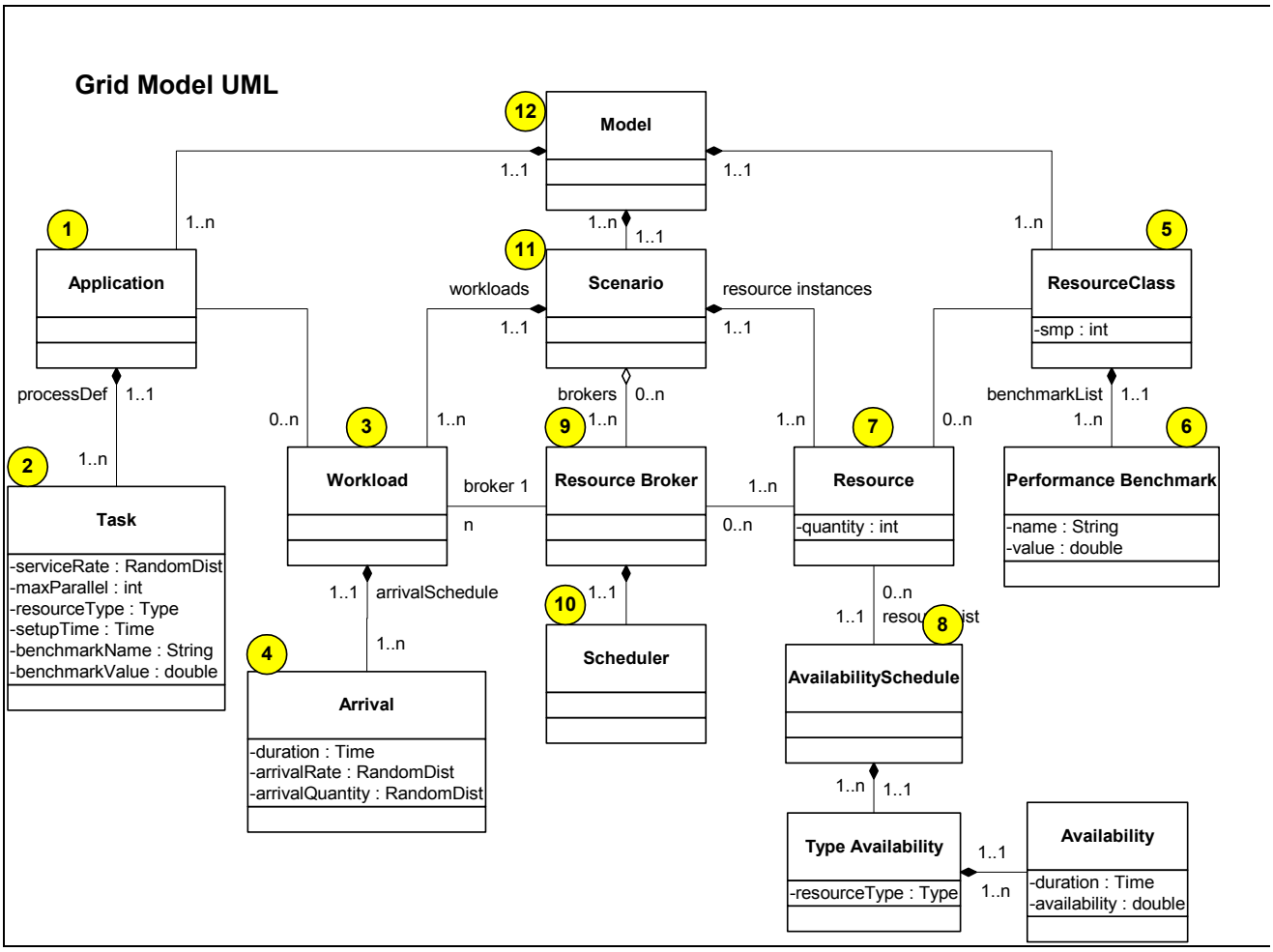

Figure 3: Grid Model for Simulation

the perspective of the maximizing job throughput. However, its focus on the data aspect is at the expense of the other computational requirements that are also usually provided from a grid. Applications are considered only from the perspective of their data requirements. The simulation model also does not consider heterogeneity among resources both in the sense of resource types and resource performance.

Common off-the-shelf simulators which provide good visualization and user interfaces are often used by IT designers and consultants. However, these packages are unable to model several of the key requirements for simulating grids, most notably, resource multi-tasking, task parallelization, resource scheduling and resource provisioning based on dynamic policies.

4 GRID SIMULATOR DESIGN

The gap between grid simulation requirements and the existing simulators led us to develop a simulator that can be used by grid designers and consultants to analyze their clients' grid infrastructure environments. It addresses all the requirements identified in Section 2.

Figure 3 shows the UML diagram of the portion of the Grid Value at Work data model relevant for simulation. The demand side of the grid is represented on the left, in terms of applications(1), their task requirements(2), application workloads(3) and their arrival statistics(4). The supply or capacity side is on the right, represented by resource classes(5) rated by multiple performance benchmarks(6), and instances of these resources(7) and their availability(8). The two sides are linked by one or more resource brokers(9) that manage the execution of tasks on the grid, using schedulers(10) to decide where to route an arriving task from a workload for processing.

The simulation is performed separately for each grid design scenario(11) as specified in terms of its workloads, resource instances, and resource brokers. Multiple scenarios can co-exist in a model(12) with common application and resource class definitions.

With this model, a discrete-event simulation of the grid can be performed to test various design elements of a scenario, such as:

1. Type of IT resources (e.g., computational servers, database servers, storage and network capacities) available on the grid: their quantities and availability profile over time.
2. Grid scheduling and dispatch policies that determine the resources to which each arriving job should be routed for processing.
3. Grid resource provisioning policies that determine how many grid resources to allocate for workload task types over any time interval.
4. Workload job arrival rates and patterns over time.

Alternate design scenarios, created by varying any combination of these elements, may be evaluated and compared on the basis of the outputs from the simulation analysis.

The simulation outputs provide various performance metrics on the components of the grid. For grid resources, these metrics are: the utilization of the available resources on the grid, the number of jobs processed by each resource, and the average processing time. The metrics for application workloads are: the average and peak response time for processing the jobs on the grid and the number of jobs processed or aborted due to lack of available resources.

4.1 Efficient Event Processing Design for Simulating Multi-tasking Resources

The implementation of multi-tasking resources requires the design of an efficient simulation event handling mechanism. As described in Section 2, the design alternatives for simulating multi-tasking resources are tradeoffs between accuracy and efficiency. Pre-emptive multi-tasking could be simulated by modeling the scheduling of tasks within each resource. However this will be too fine grained for the purposes of simulating a large grid of resources over a multi-day time frame. Therefore we choose an abstract model of multi-tasking. Consider a task being processed by a resource. Its remaining time to completion is a function of the following factors:

1. The service rate of the task on the resource. This rate assumes that the resource is processing no other tasks and is adjusted for the processing speed of the resource by applying the appropriate benchmark rating.
2. The number of symmetric multi-processors in the resource.
3. The number of other tasks in the processing queue of the resource.
4. The percentage availability of the resource. This parameter defines the fraction of the resource that may be use for processing tasks submitted through the grid. For a single resource, this is usually either 0 or 1. However, we can also model a cluster of homogeneous resources and specify that a fraction of these are available to the grid.

Figure 4 shows an example of multi-tasking behavior as tasks are submitted and processed by a resource. Each chart in the figure shows the tasks being processed (y-axis) and the processing time duration (x-axis).

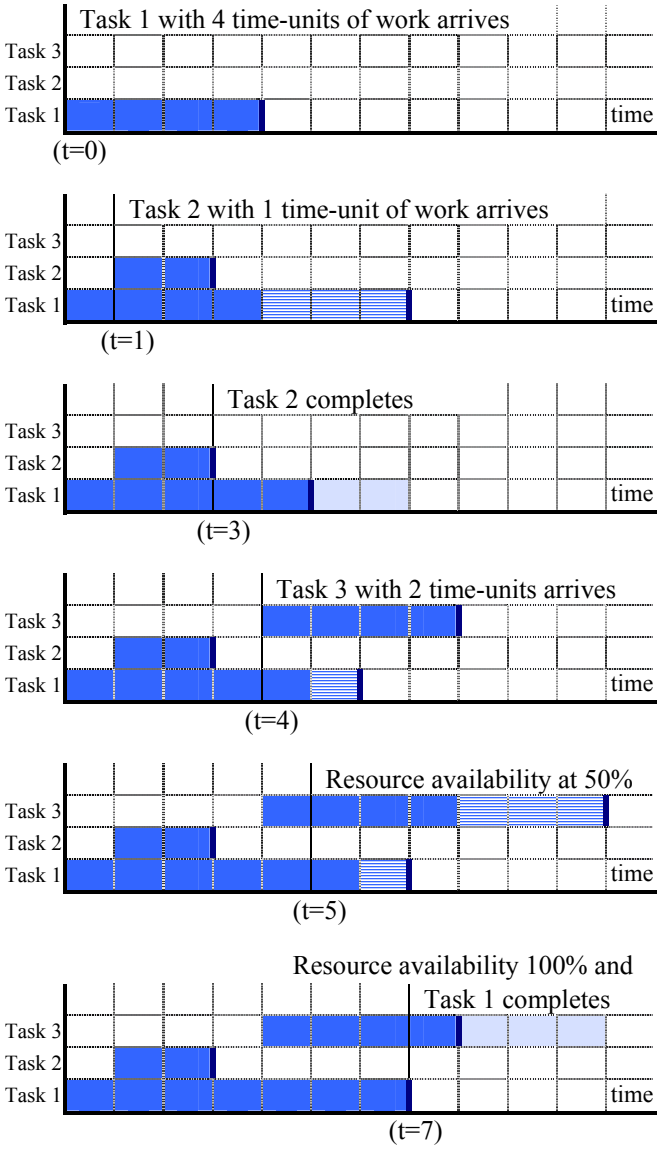

Figure 4: Multi-Tasking Event Processing

The sequence of events and the resulting multi-task processing behavior of the resource is as follows (the notation (t=*i*) is short-hand for denoting time-period *i*):

- Initially (t=0), Task 1 with 4 time-units of work (service rate of 0.25) is submitted to the idle resource. With no other tasks to process, the expected processing time is 4 time-units with a completion time at (t=4).
- At (t=1), Task 2 with 1 time-unit of work arrives. Now, with two tasks in the resource, each will take twice the raw time. The processing time for the Task 2 is therefore doubled to 2 time-units. For Task 1, the remaining 3 time-units of work is doubled to 6 time-units and the expected completion time is stretched from (t=4) to (t=7) (the horizontal lines in the bar for Task 1 shows the stretching). Note that this adjustment does not (yet) consider the completion time for Task 2 which is much earlier than that of Task 1.
- At (t=3), Task 2 completes. Now Task 1 has the entire resource to itself again and therefore, the remaining 4 time-units until its completion can be halved to 2 time-units. Its completion time shrinks to (t=5) (the lightly shaded portion of the bar for Task 1 shows the shrinkage in completion time).
- At (t=4), Task 3 with 2 time-units of work arrives. Again, with two tasks in the resource, each task will take twice as long. So Task 3 has a processing time of 4 time units, expecting to complete at (t=8). For Task 1, the remaining one time-unit of processing is stretched to two, with a new completion time at (t=6).
- At (t=5), the resource availability is reduced to 50% of its original. Now, the two grid tasks will take twice as long to process. So the remaining 1 time-unit of work for Task 1 will be stretched to 2 time units, completing at (t=7). Similarly, the remaining 3 time-units of work for Task 3 will be stretched to 6 time units, completing at (t=11).
- Finally, at (t=7), two events occur. The resource availability goes back to 100% and Task 1 also completes. As a result, the processing power that can be provided by the resource to Task 3 quadruples, shrinking the 4 time-units of remaining processing to 1 with completion expected at (t=8).

As shown by this example, the simulation of multi-tasking resources requires recalculation of the completion times of all in-process tasks in a resource whenever a new task arrives or the resource availability changes. This results in adjustments in the positions of the task completion events in the simulation engine's event queue (Law and Kelton 1991) or future events list (Schriber and Brunner 2004), which maintains all future events to be processed by the simulator in chronological order.

Adjusting the position of events in a simulator's event queue is a potentially expensive activity which can compromise the scalability of the simulator's performance. In the following section, we evaluate the time complexity of a simulation model with multi-tasking resources for two event queue designs: the traditional global event queue found in most simulation packages and the distributed event queue design that we have developed.

4.2 Simulation Time Complexity for Global and Distributed Event Queue Designs

Discrete event simulators use an event queue to manage the flow of a simulation. Each event in the queue has a time-stamp that indicates the simulation time at which it becomes ready to be processed. The events are sorted in the queue according to their time-stamp data. The primary role of the simulation engine is to step through the event queue and at each step, extract the most current event from it and process the event (as specified by the event type), which may insert new simulation events into the appropriate locations in the event queue so as to maintain its chronological sorting.

Figure 5 shows an abstract representation of the global event queue with chronologically sorted events in it. Each event is a task being processed by one of the *m* multi-tasking resources shown above the queue. The time-stamp on each event is the task completion time on the resource processing the task.

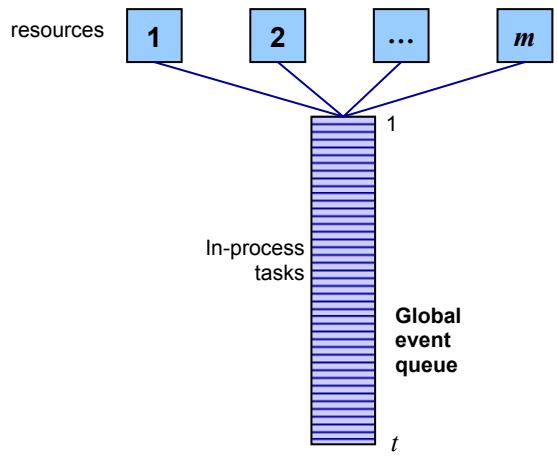

Figure 5: Global Event Queue Design

Let *t* be the number of tasks that arrive for processing during the course of a simulation. Each of these *t* tasks have to be inserted into the event queue once it has been assigned to a resource and a completion time for it has been calculated. During the insertion, the completion times of the other tasks being processed by the same resource will have to be adjusted. Assuming even distribution of the tasks among resources, the number of in-process tasks at a resource is in the order of *t/m*, which is denoted as O(*t/m*). Therefore, the order complexity of the number of inserts

and re-inserts in the event queue during the course of the t tasks is given by $O(t^2/m)$.

The time complexity of all event inserts into the queue is therefore given by:

$$O\left(\frac{t^2}{m}\right).O(t) = O\left(\frac{t^3}{m}\right),$$

where $O(t)$ is the worst-case time complexity for performing a single insertion into the event queue.

The time complexity of extracting the next event by the simulator is O(1) because the event queue is already sorted by time. Therefore the total time complexity of simulating t tasks on m multi-tasking resources is given by $O(t^3/m)$.

Assuming that $t >> m$, the simulation time complexity for a global event queue approximates to $O(t^3)$.

Now lets look at a distributed event queue design, where each multi-tasking resource has its own local event queue. This is shown in Figure 6.

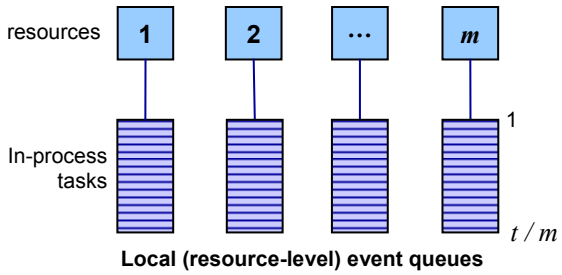

Figure 6: Distributed Event Queue Design

In this design, tasks that arrive for processing at one of the m resources, are inserted into an event queue that is local to that resource, containing events corresponding to the completion times of the in-process tasks in the resource. Upon the arrival of a new task, the completion time of the existing in-process tasks will have to be adjusted by a constant scaling factor. As a result, the chronological sort order of the existing events in the queue will not change. Therefore, no re-inserts have to take place unlike the global event queue design. Therefore, the number of inserts into the event queues is $O(t)$. Given that each insert is into a queue of $O(t/m)$ events, the time complexity of all event inserts into their respective local event queues is given by:

$$O(t).O\left(\frac{t}{m}\right) = O\left(\frac{t^2}{m}\right).$$

For the distributed event queue design, the time complexity of extracting the next event by the simulator is no longer trivial. The topmost event in each of the m local event queues will have to be examined with a time complexity of $O(m)$. Therefore the total time complexity of simulating t tasks on m multi-tasking resources with local event queues is given by:

$$O\left(\frac{t^2}{m}\right) + O(m) = O\left(\frac{t^2}{m} + m\right).$$

Assuming that $t >> m$, the simulation time complexity for distributed event queues approximates to $O(t^2)$.

This order of magnitude improvement due to the distributed event queue design translates to significant time efficiency improvements in enterprise-level large simulation analyses with hundreds of thousands to millions of tasks. This is achieved without any additional memory requirements because it can be seen that the space complexity of the event queues remains $O(t)$ in both designs. The next section provides empirical evidence of this speedup.

5 SIMULATOR PERFORMANCE

We ran multiple simulations of a grid with 10 multi-tasking resources processing 1000 to 10,000 tasks and measured the time taken for each simulation. This was done for both the global and local event queue designs. Figure 7 shows the results, plotting the simulation execution time (in log scale) against the number of tasks processed in that simulation run.

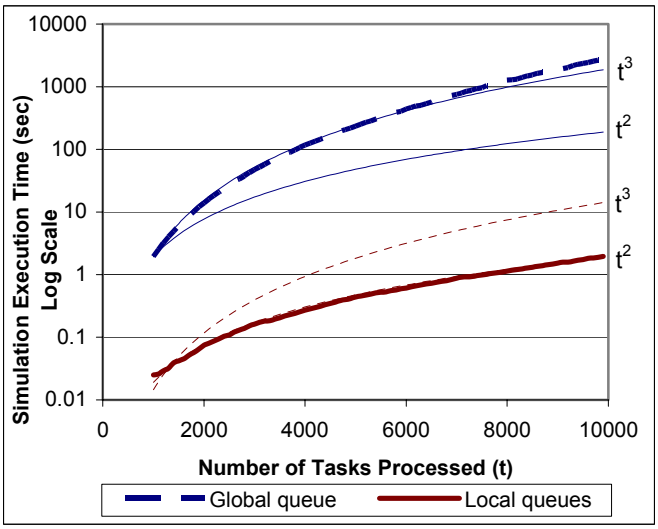

Figure 7: Simulation Execution Time Comparison for Global and Local Event Queue Designs

The figure verifies our expectation in Section 4.2 that the execution time for the global queue design (thick dashed line plot) is in the $O(t^3)$ time complexity, as shown by the thin guide lines for the plot. The time complexity guide lines show how the execution time for the initial simulation run with 1000 tasks would scale up as the square or the cube of the number of tasks processed. The local event queue design is verified to be in the $O(t^2)$ time complexity, as shown by its execution time plot (thick solid line) which closely tracks t^2 the guide line. In practi-

cal terms, a simulation of 10,000 tasks can be performed in a few seconds using the local event queue design, instead of an hour with the global event queue design.

6 CONCLUSIONS

The grid simulator described in this paper has been implemented with the objective to provide the means by which IT consultants and designers can quantify the value of grid designs in financial terms and compare with other non-grid alternatives. The role of the simulator is to assess whether the expected application workload response times and throughputs are being met by the grid design and the grid resources are operating at the desired level of utilization. Once this assessment is successful, grid infrastructure details such as the quantity and type of hardware is sent to the cost analysis module (see Figure 1) for a cost of ownership analysis. This analysis also accounts for the software, implementation, and maintenance costs for the grid.

Simulation is also required to capture the dynamic cost drivers that are becoming more prevalent in IT environments. For example, businesses can rent compute capacity by the CPU-hour instead of having to own it. Alternatively, application service providers can process workloads and charge on the basis of the number of jobs or other related units of work. The grid infrastructure can be designed to incorporate both owned assets as well as these pay-per-use or on-demand services on the grid in a manner transparent to the user. For example, the business may own enough compute capacity to handle the usual or average volumes of workload and access on-demand services during peak hours or unexpected surges in volume. In such a scenario, simulation would be necessary to estimate the overall cost of processing the workloads and to compare it with the cost of ownership of conventional IT infrastructure.

In addition to costs, IT managers are also interested in quantifying the business value realized from processing application workloads. As part of Grid Value at Work, we have developed industry and application specific business value models that translate IT metrics such as application throughput or response time to financial value such as reduction in business expenses or increase in revenue. In these instances, simulation of application performance on a grid provides the necessary inputs for the business value analysis. Using these analyses, alternate grid and non-grid infrastructures can be evaluated with the objective of maximizing the value to the business after accounting for infrastructure costs.

As the above scenarios show, simulation of IT infrastructure has become an essential tool for both the technical and financial assessment of grid computing infrastructure. IBM Grid Value at Work provides the necessary simulation capabilities to match the grid requirements and integrates simulation with financial analysis tools.

REFERENCES

Appleby, K., S. B. Calo, J. R. Giles, and K.-W. Lee. 2004. *IBM Systems Journal* 43: 121–135.

Buyya, R. and M. Murshed. 2002. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience* 14: 1175–1220.

Cameron, D. G., R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini. 2003. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. *Proceedings of the Fourth International Workshop on Grid Computing (GRID'03).*

Foster, I., C. Kesselman, J. M. Nick, and S. Tuecke. 2002. Grid Services for Distributed System Integration. *IEEE Computer* June 2002: 37–46.

Howell, F. and R. McNab. 1998. SimJava: A discrete event simulation library for Java. In *Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation*, 51-56. The Society for Computer Simulation International, San Diego, CA.

Law, A. M., and W. D. Kelton. 1991. *Simulation Modeling and Analysis*. McGraw-Hill.

Legrand, A., L. Marchal, and H. Casanova. 2003. Scheduling Distributed Applications: The SimGrid Simulation Framework. *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID.03).*

Phatanapherom, S. and V. Kachitvichyanukul. 2003. Fast Simulation Model For Grid Scheduling Using HyperSim. *Proceedings of the 2003 Winter Simulation Conference*. 1494–1500.

Schriber, T. J., and D. T. Brunner. 2004. Inside Discrete-Event Simulation Software: How it Works and Why it Matters. *Proceedings of the 2004 WinterSim Conference*. 142–152.

AUTHOR BIOGRAPHY

SUGATO BAGCHI is a Research Staff Member in the Mathematical Sciences Department at IBM Thomas J. Watson Research Center in Yorktown Heights, NY. His research interest is in financial modeling and valuation of information technology applications and infrastructure. He has developed business value models as well as their underlying tools and techniques to help potential adopters of emerging technologies quantify the predicted benefits in terms of business and financial impact. In this role, he has assisted several IBM Global Services engagements with the task of developing business cases for their clients. Dr. Bagchi has a Ph.D. degree in Electrical Engineering from Vanderbilt University, where his thesis was on task planning under uncertainty. He is a member of INFORMS.