Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

FAST SIMULATION MODEL FOR GRID SCHEDULING USING HYPERSIM

Sugree Phatanapherom
Putchong Uthayopas

High Performance Computing and Networking Center
Faculty of Engineering
Kasetsart University
Bangkok, 10900, THAILAND

Voratas Kachitvichyanukul

Industrial Engineering & Management
School of Advanced Technologies
Asian Institute of Technology
Pathumthani, 12120, THAILAND

ABSTRACT

To develop grid scheduling algorithms, a high performance simulator is necessary since grid is an uncontrollable and unrepeatable environment. In this paper, a discrete event simulation library called HyperSim is used as extensible building blocks for grid scheduling simulator. The use of event graph model for the grid simulation are proposed. This model is well supported by HyperSim which yields a very high performance simulation. The experiments are conducted to compare HyperSim with other several simulators in terms of speed and scalability. The result shows a significant simulation speed improvement over many widely used simulators. Furthermore, sample simulation results of basic job scheduling problem are shown to compare to well-known heuristics.

1 INTRODUCTION

Grid computing system (Foster and Kesselman 1998b) is a promising infrastructure and technology for harnessing geographically distributed resources across many organizations. Grid has a great potential to help scientists solve problems faster than ever using much larger resource pool. However, resource discovery and allocation processes are still mostly done manually by job owner. These processes may lead to unsatisfactory results in terms of utilization, turnaround time, and load balancing of the system. Many scheduling heuristics have been developed over the past few years to efficiently allocate resources for a given batch of jobs with particular purpose. The main challenge of scheduling heuristics development is the unrepeatable nature of the grid system since no one has the capability to fully control all available resources and tasks. The second problem is the small size of current experimental grid. Although, there are many small, inter-organization, grid test-beds available, none of these test-beds are large enough to reflect the true behavior of the real production quality grid.

Therefore, a simulator is the most important tool for the evaluation of grid scheduling heuristics. A good simulator allows researchers to explore more alternatives and give an accurate, statistically valid results. Furthermore, the simulator can be used to study many heuristics which have their own different system model and application model.

In this paper, the modeling of grid environment using a newly developed, high speed, simulation library called HyperSim is presented. HyperSim is developed to be a general purpose, extensible, configurable, and high-speed simulation library. This simulator is based on event graph model to maximize speed and scalability. The main advantage of HyperSim is its superior simulation speed compared to other simulators available. This allows researchers to model a much larger Grid system than before. The design, architecture and example of HyperSim are discussed. In addition, HyperSim is compared to other well-known grid scheduling simulators in terms of simulation speed. The experimental evaluation shows that HyperSim can be used to simulate the same grid environment with a much faster simulation speed.

This paper is organized as follows. Section 2 discusses related work followed by Section 3 which introduce the HyperSim simulator. Section 4 proposes the approach of how to model grid scheduling using event graph model and shows briefly how to implement the model into HyperSim. Section 5 presents the experimental results and discussions. Finally, Section 6 concludes the paper and discusses future work.

2 RELATED WORKS

For the Grid system, one approach used to study the system characteristics is to emulate a grid system on a real computing system. This approach is used by MicroGrid (Song et al. 2000) which emulates multiple computing resources on a real resource to increase grid size using limited resources. MicroGrid is usually suitable for testing real application on real, controllable environment. However, runtime is still not reduced so MicroGrid is not suitable for

scheduling simulation due to turnaround time of each workload set which may take a substantially long time. Hence, in order to minimize the turnaround time for the extensive study of grid environment, a simulator is needed.

Simulator has been used for modeling and evaluating real world systems in many areas e.g., industrial, computing, mechanical, and more. Many general-purpose simulation modeling tools are available. Some of them are in form of a language e.g., Simscript (CACI 2003), an extension of existing language e.g., Parsec (Bagrodia et al. 1998), or library for specific language e.g., NS (LBNL et al. 2003), NS-2 (USC/ISI and ACIRI 2003), OMNeT++ (Varga 2001), SimJava (Howell and McNab 1998). Application specific simulators are also available with some customizable parameters of each run. This kind of simulator is usually available in industrial and related fields.

There are a few grid simulators available. For example, GridSim (Buyya and Murshed 2002), which is a Java-based discrete event grid scheduling simulator built on top of SimJava. GridSim provides high extensibility and portability through Java and thread technologies. Every components in the system model are initiated as a thread with a unique name. Each component runs individually with separate event queue. An event is transmitted to the target component's event queue directly. For system behavior, GridSim estimates the status of each component based on pre-defined condition. Although very flexible, GridSim is not scalable since it depends on the number of threads which is rather limited. In addition, the threads management in Java create a very high overhead which results in a very slow runtime.

Bricks (Aida et al. 2000) is another Java-based discrete event grid simulator built from the ground up. Bricks is designed to maximize modularity of restructuring system model based on client-server architecture. One may run Bricks to evaluate scheduling heuristics or to evaluate data movement algorithms on grid. Status of each component are estimation of real world system trace. Unfortunately, Bricks is not publicly available at this time.

SimGrid (Casanova 2001) is a C-based discrete event job scheduling simulation library developed by San Diego Super Computing Center (SDSC). SimGrid provides highly accurate network model for TCP and non-TCP transport. One may construct network topology (connection of hosts and routers) to represent real world system for data-intensive application simulation. SimGrid is much faster than most Java-based simulator. However, the approach of using user level thread to model resources also make SimGrid being limited by the thread switching capability and overhead of the system.

ChicSim (Ranganathan and Foster 2002) is a Parsec-based simulator for concurrent job and data scheduling. System model is fixed. The user just needs to specify resources, networks, and workload characteristics to the simulator by a list of files.

3 INTRODUCTION TO HYPERSIM

HyperSim is a general-purpose discrete event simulation library developed on C++. It provides comprehensive classes for constructing a simulator such as the distribution generator, statistical analyzer, event manipulator, automatic traceable simulation class, and much more.

HyperSim follows the event graph model (Schruben 1983). To construct a simulation model of any system, developer must design an event graph model of that system first. An event graph consists of at least one event represented by a node, denoted by a circle. Two events are linked by a directed line or edge denoted by an arrow. Figure 1 shows the representation of a basic event graph model with 2 events, *A* and *B*, and a transition from event *A* to event *B*. In this figure, the current event is *A*. If condition *i* is true, event *B* will be scheduled to occur after *t* time units. The scheduled event has its own attributes set by generator so that the event keeps track of the simulation state individually.

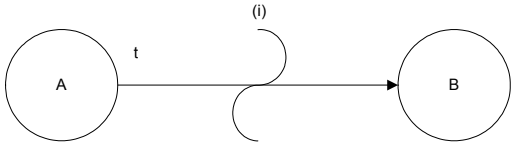


Figure 1: Basic Event Graph Model

At the start of the simulation, the simulator will be initialized by the schedule of at least one event. This scheduling will start the simulation process that dequeues the event. In HyperSim, complex events can be modeled using C++ code. So, developer can use C++ code to easily trigger the state change and implement a complex condition. This technique eliminates the need to define the complex interactions into the form of an event graph first. The result is the substantial reduction of the complexity and development time.

The simulation will stop if there is no event left in the queue or user-defined method, called isFinish(), returns true. This function enables a developer to stop the simulation at anytime before the processing of each event. In addition, developer can put some user-defined condition that stops the simulation when the required results are obtained.

Statistical analysis is done by overriding updateStat() method. This method is called prior to event handler method. Furthermore, every event is traced and logged. Two file formats are used, namely, the standard format and NetLogger format. Standard format displays simulation time, event name, the number of occurrences, and user-defined data. NetLogger format comes from NetLogger tool (http://www-didc.lbl.gov/NetLogger/) which is a set of online data logging APIs and offline visualization tools. In fact, NetLogger format conforms to IETF Internet-Draft for Universal Log Message (ULM). Thus, the trace output

of the simulation can be used as input to NetLogger visualization tool. The simulator developer can add more trace information by overriding trace() method.

4 MODELING THE GRID SCHEDULING

For the problem of grid scheduling, it is convenient to classify the grid system into 2 categories, namely, the one-level and two-level grid. In one-level grid (illustrated in Figure 2), the scheduler can directly access each resource. In contrast, in the two-level grid (as shown in Figure 3), the scheduler has no control over local resources but has to interact with a local resource manager. The example of this case is the use of grid to link multiple clusters together. Traditionally, two-level grid is more preferable since it allows each organizations participating in the grid to have better control over its local resources.

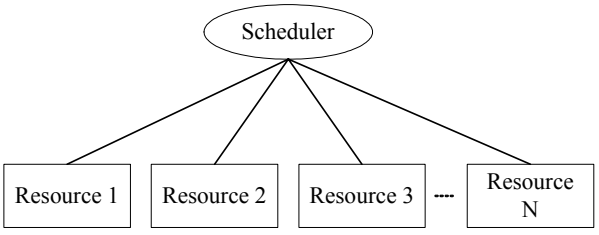


Figure 2: One-Level Grid

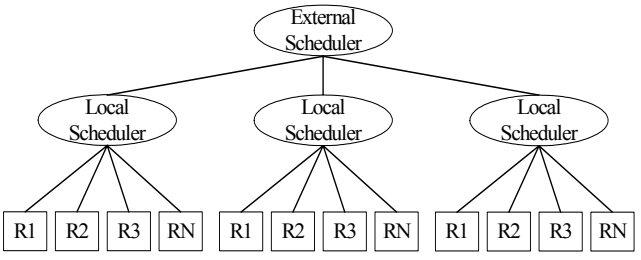


Figure 3: Two-Level Grid

The event graph model can be effectively used to model the behavior of both one-level and two-level grid. The event graph model of one-level grid is as shown in Figure 4.

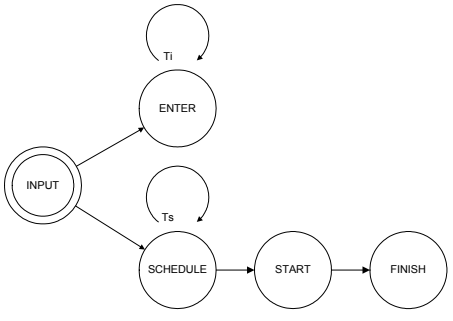


Figure 4: Event Graph Model for One-Level Grid

In Figure 4, the start event is INPUT which initializes the system. This INPUT event generates the ENTER event and SCHEDULE event. ENTER event represents the job submission into scheduler queue. For online scheduler, SCHEDULE event will be scheduled immediately. For batch scheduler, the ENTER event is then delayed by the random inter-arrival time (denoted by *Ti*). Then, SCHEDULE event is rescheduled for every *Ts* and all new scheduled jobs are then dispatched. Dispatch process is done by scheduling START event at the specified time. The FINISH event for that job will take place after the delay equal to the execution time of that job on the machine selected.

The event graph model for two-level grid is shown in Figure 5. There are 2 kinds of scheduler: External Scheduler (ES) used as the grid-level scheduler and Local Scheduler (LS) used as the cluster-level scheduler. EENTER, ESCHEDULE, ESTART, and EFINISH are grid-level events. They work very similar to what was described above in one-level grid. except for ESTART. ESTART, will submit the job to a cluster-level scheduler instead of starting the job. STAGE_IN, STATE_EXE, EXECUTE, and STAGE_OUT are added to study effect of input staging, job execution, and outputs staging over wide-area network that link grid-level scheduler and cluster-level scheduler together.

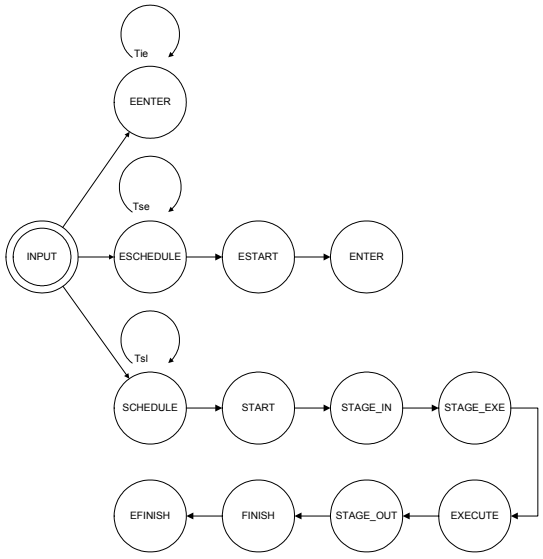


Figure 5: Event Graph Model for Two-Level Grid

After obtaining the event graph model. The simulator can be easily developed. In HyperSim, Host, Cluster, and Grid classes are implemented to represent each kind of resources. Host may be a single or multiple processors system. Cluster is represented by a scheduler and a set of hosts connected by high-speed interconnection and grid is represented by a scheduler and a set of clusters connected by wide-area network.

The job execution time is calculated based on workload, execution rate, and current load average of the computing resource used. Let W denotes the amount of workload, ε denotes the execution rate of the computer used, and λ denotes the current load average of that system. Then, the execution time, *Te*, of a job is as given in Eq. 1.

$$T_e = \frac{W}{\varepsilon}(1+\lambda) \quad (1)$$

Workload set can be generated online by giving the inter-arrival distribution, workload distribution, input size distribution, executable size distribution, and output size distribution. In addition, workload can be read from a file. This features enable user to use real workload trace to drive the simulator. In addition, a cluster configuration can be automatically generated or given to the simulator using a file. The automatic cluster configuration generation requires the users to specify some parameters such as the number of hosts, execution rate distribution, and load average parameter distribution. For grid configuration, it is necessary to explicitly specify each cluster configuration and its network characteristics to reduce the complexity of multi-level generation. Scheduling heuristics are separated from the simulator. Each heuristic is implemented in a dynamic linked library (shared object). This helps researchers to easily implement various new heuristics without making changes to the simulator.

Figure 6 shows the interface used to for the implementation of scheduling heuristic. One must implement schedule() method to schedule tasks in queue, *tasks*, by assigning each task to a host in host vector, *hosts*. Host vector may contain host, cluster, or grid so this heuristic may use on both cluster and grid, one-level and two-level grid. Additional parameters are passed to the heuristic by specifying a file containing the parameters.

```
class Scheduler {
public:
    Scheduler(char const *parameterPath);
    virtual ~Scheduler();
    virtual reset();
    virtual schedule(TaskQueue &tasks,
                     HostVector &hosts,
                     Clock simTime);
protected:
    string paramFile;
};
```

Figure 6: Scheduler Interface

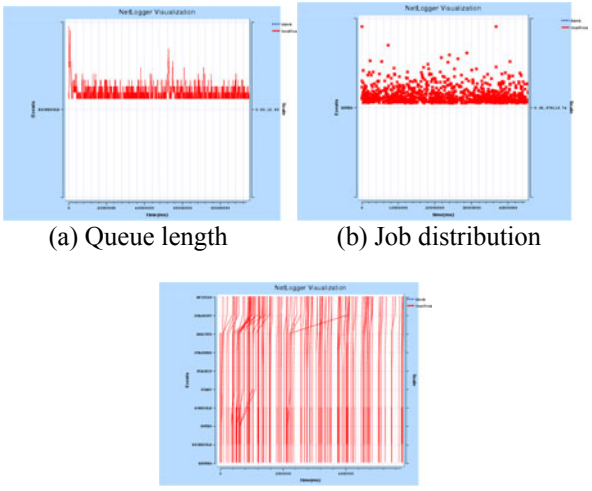
5 EXPERIMENTS AND DISCUSSION

In this section, the performance of several job scheduling algorithms for grid has been compared using several simulators available. The purpose is to compare the simulation speed and results of these simulator with HyperSim.

All experiments are conducted on PC/Linux using Athlon 1 GHz processors with 1 GB RAM on light load condition. GridSim 2.0 and SimGrid 2.09 are used for the comparison due to their availability. To evaluate performance of these simulators, MET (Minimum Execution Time) heuristic (Maheswaran et al. 1999) is implemented on GridSim, SimGrid, and HyperSim to compare the performance in term of run time used to finish the simulation.

MET heuristic is an online scheduling algorithm in which each job is scheduled whenever it arrives. Generally, MET assigns a job to the machine that supposes to complete it fastest. In detail, MET estimates execution time of the job on all machines and assigns the job to the machine with the least estimated execution time.

Since GridSim and SimGrid follow one-level grid structure, the experimental code uses the same structure. The simulation assumes that there is a grid scheduler in the system. In addition, network characteristics are not taken into consideration to reduce complexity of the simulation. All jobs are ready in the queue prior to the beginning of the first scheduling. Figure 7 shows the NetLogger visualization of results from HyperSim. The job distribution, queue length, and grid scheduling event are illustrated.



(a) Queue length (b) Job distribution

(c) Scheduling Event Transition

Figure 7: Netlogger Visualization of HyperSim Log

First, the test is conducted to measure the performance and scalability of the simulator when the number of resources increases while the number of tasks are fixed to 16384. The number of resources varies from 1, 2, 4, 8, up to 16384. The results are as depicted in Figure 8. The experiments show that GridSim can not scale to more than 512 resources. This is due to the thread creation error when OS and runtime resources are used up. Although SimGrid and HyperSim can scale up very well, HyperSim is approximately 10 times faster than SimGrid for this test.

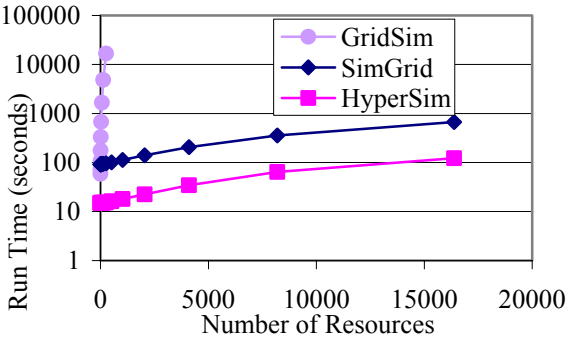


Figure 8: Comparison for Fixed the Number of Tasks

Second, the test is performed to measure the performance of simulator when the number of tasks are increased and the number of resources (hosts) are fixed at 256. The reason that resources are fixed at 256 hosts is because GridSim can only run successfully with 512 hosts.

Figure 9 shows the results. In term of the speed, HyperSim still gives the best result since HyperSim is about 1000 times faster than GridSim and nearly 10 times faster than SimGrid. From the experimental results, it is clear that GridSim suffers from the high overhead of Java thread management. In addition, GridSim always allocates a fix the number of resources for the whole simulation although the number of job decreases as the simulation proceeds. As a result, the memory used is likely to be inefficient due to the existence of large, and lowly utilize in-memory objects.

As for SimGrid, the results are very similar to HyperSim. The major difference is that in SimGrid, the event structure is blended into the code. Moreover, SimGrid only allows developers to simulate with pre-defined stop condition such as when one task or all tasks finish the execution. Although the intension is to ease the programming task, the results is that code are much more complex due to the need for conditional checking that occurred frequently. This can potentially degrade the speed of the execution. In contrast, HyperSim decouples events from each other so the com plexity will be at the modeling level. Hence, the result code is much simpler and faster.

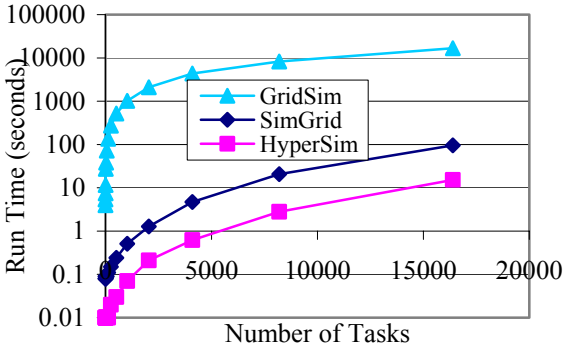


Figure 9: Comparison for Fixed the Number of Resources

In other point of view, grid can be looked as a set of parallel machines in manufacturing system and job is a material to be processed by the machine to produce some products. It is necessary to have a machine for dispatching new unbalance material to appropriate parallel machine. The dispatcher must be optimized to minimize makespan of the factory. Heuristics described above can apply on this problem also. Some factory may feed material to the dispatcher one at a time or batch at a time to reduce transportation cost.

At this point, next experimental is to compare two heuristics using HyperSim. The heuristics are MET and MCT (Minimum Completion Time). MET is described above. In contrast, MCT assigns each job to the machine with the minimum completion time. That means some jobs may not be assigned to the fastest machine but system makespan should be balanced.

Figure 10 shows makespan of the system when the number of incoming jobs are varied from 100 to 3200 stepped up by multiples of 2. Machines in simulation environment is heterogeneous machines randomly generated in exponential distribution with 500 mean. The generated value represents execution rate of each machine. Workload is also generated in exponential distribution with 500,000 mean. The number of machine is fixed to 32. At the starting state, all machine has no load. The result shows that MET gave better solution in system with low (100-400) jobs. After that point MCT gave the better solution. MET shows better result when cumulative wait time of job in fast machine are not too many comparing to execution time. Average wait time of MCT and MET are shown in Figure 11. It turns out that average wait time of MET is significantly increased by the number of jobs submitted to the system. The cause is MET tried to assign jobs to the fastest machine for that jobs. When average wait time reaches the point that the fastest machine cannot serve jobs on time, the overall performance will rapidly go down.

6 CONCLUSIONS

To efficiently utilize the grid resources, special scheduling heuristics are necessary. There are only a few large test

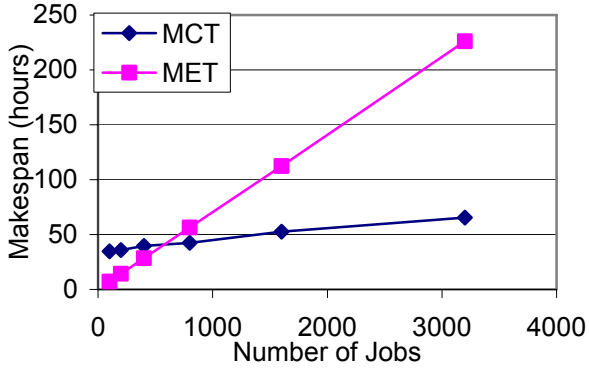


Figure 10: Makespan of MET and MCT

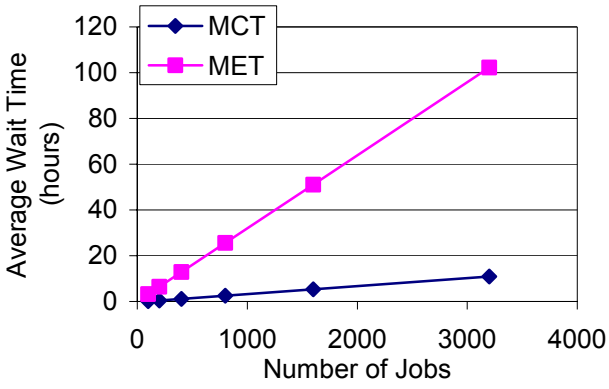


Figure 11: Average Wait Time of MET and MCT

beds among countries, e.g., ApGrid test-bed (Tanaka 2003), and small production quality in some big organizations. Thus, it is difficult to test the algorithm extensively on the real test-bed. Therefore, simulator is crucial in order to validate and evaluate scheduling heuristics. on grid infrastructure. In this paper, the simulator called HyperSim has been presented. HyperSim is developed as a general, portable, and extensible discrete event simulation library conforming to event graph modeling.

This paper also proposes the approach of how to model the grid scheduling using event graph model and run it efficiently under HyperSim simulator. The experiments shows that HyperSim is much faster than several well known simulator available for grid simulation.

There are many works that can be done in the future. For example, developing more library that allows HyperSim to be used as a core for other type of simulation such as industrial simulation. The comprehensive GUI tool can be helpful in speeding up the modeling process and visualize the results.

The latest version of HyperSim, including the source code, can be freely downloaded from the www at <http://hpcnc.cpe.ku.ac.th/moin/HyperSim>.

REFERENCES

Aida, K., A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima, 2000. Performance Evaluation Model for Scheduling in a Global Computing System. *The International Journal of High Performance Computing Applications*, 14 (3).

Bagrodia, R. R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. 1998. Parsec: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, 31 (10).

Buyya, R. and M. Murshed. 2002. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience, 14.

CACI. 2003. Simscript: a simulation language for building large-scale, complex simulation models [online]. Available via <http://www.caciasl.com/products/simscript.cfm> [accessed May 5, 2003].

Casanova, H. 2001. Simgrid: A toolkit for the Simulation of Application Scheduling. *The 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*. Brisbane, Australia.

Foster, I. and C. Kesselman. 1998a. Globus: A Toolkit-Based Grid Architecture, pp. 259-278. In I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann.

Foster, I. and C. Kesselman. 1998b. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann.

Howell, F. and R. McNab. 1998. SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling. *The 1st International Conference on Web-based Modelling and Simulation*. San Diego, CA, USA.

LBNL, Xerox PARC, UCB and USC/ISI. 2003. Network Simulator version 1. VINT Project, Lawrence Berkeley National Laboratory, http://www-nrg.ee.lbl.gov/ns/

Maheswaran, M., S. Ali, H.J. Siegel, D. Hensgen and R.F. Freund. 1999. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. *The 8th Heterogeneous Computing Workshop*. San Juan, Puerto Rico.

Ranganathan, K. and I. Foster. 2002. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. The 11th IEEE International Symposium on High Performance Distributed Computing (HPDC 2002). Edinburgh, Scotland.

Schruben, L. 1983. Simulation Modeling with Event Graphs, Communications of the ACM, 26, 957-963.

Song, H., X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. 2000. The MicroGrid: A Scientific Tool for Modeling Computational Grids. *IEEE Supercomputing (SC 2000)*. Dallas, USA.

Tanaka, Y. 2003. ApGrid Testbed [online]. Available via <http://www.apgrid.org/> [accessed July 11, 2003].

USC/ISI and ACIRI. 2003. Network Simulator ns-2 [online]. Available via <http://www.isi.edu/nsnam/ns/> [accessed May 5, 2003].

Varga, A. 2001. The OMNeT++ Discrete Event Simulation System. *The European Simulation Multiconference (ESM 2001)*. Prague, Czech Republic.

AUTHORS BIOGRAPHIES

SUGREE PHATANAPHEROM is a research assistant in High Performance Computing and Networking Center, Faculty of Engineering, Kasetsart University, Thailand. He

received a B.Eng in Computing Engineering from Kasetsart University and his M.Eng in Computer Engineering from Kasetsart University. His recent work has involved Grid resource scheduler, simulator, and algorithms. His email address is <sugree@hpcnc.cpe.ku.ac.th>.

PUTCHONG UTHAYOPAS is an Assistant Professor in Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Thailand. He received his PhD in Computer Engineering from University of Louisiana at Lafayette. His major research interests are in parallel/distributed computing, cluster and grid computing, and parallel software tools. He is serving as the regional committee of IEEE Task force on cluster computing. He is also being the core member of ApGrid organization and participating actively in the construction of Asia Pacific grid testbed. His email address is <pu@ku.ac.th>.

VORATAS KACHITVICHYANUKUL is an Associate Professor in Industrial Engineering & Management, School of Advanced Technologies, Asian Institute of Technology, Thailand. He received a Ph. D. from the School of Industrial Engineering at Purdue University in 1982. He has extensive experiences in simulation modeling of manufacturing systems. He had worked for FORTUNE 500 Companies such as Compaq Computer Corporation and Motorola Incorporated. He had also worked for SEMATECH as technical coordinator of the future factory program. His teaching and research interests include planning and scheduling, high performance computing and applied operations research with special emphasis on industrial systems. His email address is <voratas@ait.ac.th>.