Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms

Atsuko Takefusa
Ochanomizu University
Tokyo 112-8610, Japan
takefusa@hn.is.ocha.ac.jp

Satoshi Matsuoka
Tokyo Institute of Technology/JST
Tokyo 152-8552, Japan
matsu@is.titech.ac.jp

Hidemoto Nakada
Electrotechnical Laboratory
Ibaraki 305-8568, Japan
nakada@etl.go.jp

Kento Aida
Tokyo Institute of Technology
Kanagawa 226-8502, Japan
aida@noc.titech.ac.jp

Umpei Nagashima
National Institute for Advanced
Interdisciplinary Research
Ibaraki 305-8562, Japan
umpei@nair.go.jp

Abstract

While there have been several proposals of high performance global computing systems, scheduling schemes for the systems have not been well investigated. The reason is difficulties of evaluation by large-scale benchmarks with reproducible results. Our Bricks performance evaluation system would allow analysis and comparison of various scheduling schemes on a typical high-performance global computing setting. Bricks can simulate various behaviors of global computing systems, especially the behavior of networks and resource scheduling algorithms. Moreover, Bricks is componentalized such that not only its constituents could be replaced to simulate various different system algorithms, but also allows incorporation of existing global computing components via its foreign interface. To test the validity of the latter characteristics, we incorporated the NWS system, which monitors and forecasts global computing systems behavior. Experiments were conducted by running NWS under a real environment versus the simulated environment given the observed parameters of the real environment. We observed that Bricks behaved in the same manner as the real environment, and NWS also behaved similarly, making quite comparative forecasts under both environments.

1. Introduction

High performance global computing systems fueled by the rapid progress of high-speed networks and computing resources are now regarded as the computing platform of the future[9]. In order to effectively employ computing resources therein, most proposed global computing systems embody a resource scheduling framework whose components monitor the global computing environment and predict availability of the resources. For effective investigation and objective comparison of scheduling algorithms and the implementation of the scheduling frameworks, large-scale benchmarks with reproducible results under various environments parameterized by the following constituents over time are required:

- networks — topology, bandwidth, congestion, variance, and
- servers — architecture, performance, load and variance.

However, reproducibility over a wide-area network is extremely costly to achieve, if not impossible. Thus, currently it is unrealistic to compare the different scheduling algorithms proposed by other researchers, let alone compare the systems themselves. Cost and scale of possible benchmarks are also extremely limited. The resulting lack of impartial comparative approaches is a great hindrance to global computing research and deployment.

In order to resolve this situation, we are building a performance evaluation system that would allow analysis and comparison of various global computing systems under reproducible, controlled environments, called *Bricks*[1]. The current version of Bricks mainly focuses on the evaluation of different scheduling algorithms and schemes based on a canonical model of high-performance global computing system we proposed in [4, 5], simulating the behaviors of networks and resource scheduling algorithms. Moreover, as Bricks is constructed in a componentalized fashion, such that not only its constituents could be replaced to simulate various different system algorithms, but also allows incorporation of existing global computing components via its foreign interface.

To test the validity of the latter characteristics, we incorporated the NWS (Network Weather Service) system[12, 13], which physically monitors and forecasts the behavior of global computing systems in an actual environment. Experiments were conducted by running NWS under a real environment versus the Bricks simulated environment given the observed parameters of the real environment, without essential changes to the NWS itself, and we observed the following results:

- Simulated Bricks global computing environment behaved in the same manner as the real environment.
- Under both environments, NWS behaved similarly, making quite comparative forecasts.

2. Overview of the Bricks System

Bricks is a performance evaluation system for scheduling algorithms and frameworks of high performance global computing systems. It is written in Java, and embodies the following characteristics:

- Bricks consists of the simulated Global Computing Environment and the Scheduling Unit (Figure 1), allowing simulation of various behaviors of
  - resource scheduling algorithms,
  - programming modules for scheduling,
  - network topology of clients and servers in global computing systems, and
  - processing schemes for networks and servers.

  The configuration and parameters of the Global Computing Environment can be easily described with the *Bricks script*. Users can construct and alter the script in a composible way, using the building 'bricks' within the script, to test and evaluate a variety of simulations in a reproducible manner[1].

- To systematically obtain information on global computing resources for resource scheduling algorithms, Bricks embodies a framework and constituent components which monitors and predicts the resources in the global computing environment. Bricks provides several default components for monitoring, predicting, and scheduling the jobs in the simulated network. Because the components are designed to be orthogonal with carefully-defined component APIs, they could easily be replaced by other Java-written components; for example, one could describe a new scheduling algorithm in Java according to the Bricks Scheduling Unit SPI (Service Provider Interface), and test it under a variety of situations using Bricks. Moreover, the components could be external, in particular real global computing scheduling components. Bricks can supply simulated time as well as various monitored simulated information to the external resource-related systems, and receive the results of scheduling decisions made, which is fed back into the simulation. Although it is still too early to claim that Bricks could easily integrate every possible global computing components, we have been successful in integrating the NWS system, which had been developed earlier at UCSD, by defining a Java API for the NWS.

[1]As one might expect, this is how the simulator had been named so.

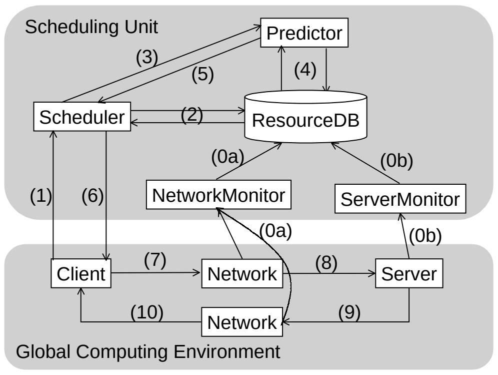


Figure 1. The Bricks Architecture.

Underneath, Bricks employs a queuing network model in which servers and networks are modeled as queuing systems in the Global Computing Environment. In Figure 2, the network from the client to the server, the network from the server to the client, and the server are represented by queues, Q_{ns}, Q_{nr} and Q_s, respectively. Service rates on Q_{ns}, Q_{nr} and Q_s indicate the bandwidth of each of the networks, the processing power of the server, respectively (A and A' denote the same client, but distinguished for notational convenience). Details of the model could be found in [4, 5].

3. The Bricks Architecture

We now give more detailed descriptions of Bricks. In Bricks, the Global Computing Environment and the Scheduling Unit coordinate to simulate the behavior of

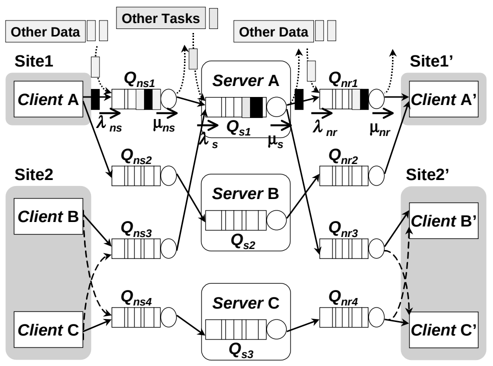


Figure 2. An example of the Global Computing Environment Model.

global computing systems. Overall, Bricks operates as a discrete event simulator of a queuing system in virtual time. An overview of the steps of the Bricks simulation is illustrated in Figure 1.

3.1. The Global Computing Environment Simulation Part

The Global Computing Environment represents the global computing simulation environment, and consists of the following modules:

Client: represents the user machine, upon which global computing tasks are initiated by the user program.

Network: represents the (wide-area) network interconnecting the Client and Server, and is parameterized by e.g., bandwidth, congestion, and their variance over time.

Server: denotes the computational resources of the given global computing system, and is parameterized by e.g., performance, load, and their variance over time.

Both Network and Server are modeled as queues, whose processing schemes can be replaced. The model of a task invoked by a client machine (Client), communication models of Network and server models of Server are given next.

3.1.1. Task Model

It is important for the simulator to manage and discover the time duration of communication and computation for a given task. In the current Bricks implementation, a task is represented by:

- the amount of data transmitted to/from a server with the task as an input/output of the computation and
- the number of executed instructions (operations) in the task.

3.1.2. Communication Models

With Bricks, one can flexibly simulate various communication models of the network with simple specifications of the Bricks script. Currently, there are two major model families supported by Bricks: The first family assumes that the congestion of a network is represented by adjusting the amount of arrival data from extraneous traffic generated by other nodes in the system (Figure 2)[4, 5]. Here, one needs to specify ideal bandwidth, the average of actual bandwidth, the average size of extraneous data from other nodes, and their variance. Specifying smaller packets will result in greater accuracy at the expense of larger simulation cost.

In the second family, the variation of the Network bandwidth at each time-step is determined by the observed parameters of the real networking environment. Although one needs to accumulate the measurements prior to the simulation, the Network behaves as if it were real network. Furthermore, the cost of simulation is much smaller than that of the first. Because the actual measurements are discrete, we specify an interpolation method, including linear or curve fitting methods.

The two families already serve to generate a rich set of models for network behavior of global computing systems, due to the various parameters that can be specified (such as various probablistic functions of the arrival rate). Furthermore, we are working to extend Bricks to accommodate more families, for increased accuracy, better execution speed, user convenience, etc.

3.1.3. Server Models

The current Bricks models the computing servers in the following way. A server machine processes tasks in a FCFS manner, and is modeled as a queue as is with the networks. Its load can be specified and simulated not only by the arrival rate of tasks from other users (i.e., extraneous tasks), but also could be specified by the observed parameters of the real environment.

3.2. Scheduling Unit

The other major portion of Bricks is the Scheduling Unit that models a canonical scheduling framework for global computing systems. The constituent modules of the Scheduling Unit represent common features found in global computing systems as follows:

NetworkMonitor: measures network bandwidth and latency in global computing environments. The measured values are stored into the ResourceDB module.

ServerMonitor: measures performance, load, and availability of server machines. The measured values are also stored into ResourceDB.

ResourceDB: serves as a scheduling-specific database, storing the values of various measurements. The measured values are accessed by the Predictor and the Scheduler in order to make forecasts and scheduling decisions.

Predictor: reads the measured resource information of certain time duration from the ResourceDB, and predicts the availability of resources. The predicted information is typically used for scheduling of a new global computing task.

Scheduler: allocates a new task invoked by a client on a suitable server machine(s), making scheduling decisions based on the resource information provided from ResourceDB and Predictor.

As is with the Global Computing Environment, the components of the Scheduling Unit are written in Java, which facilitates APIs called SPIs. The SPIs allow replacement of the components with alternative, user-supplied modules. For example it would be possible to replace the Scheduler to accommodate new scheduling algorithms, or replace the Predictor to incorporate external predictors such as the NWS.

4. Incorporating Existing Global Computing Components

As mentioned above, Bricks allows incorporation of external components including existing global computing components allowing their validation and benchmarking under simulated and reproducible environments. This is mainly achieved by replacing the modules of the Scheduling Unit, and exploiting the foreign module SPIs to pass on and receive various information on scheduling, such as those measured by the monitors, etc.

4.1. Overview of the Scheduling Unit SPI

Each component of the Scheduling Unit is replaceable by any Java-written component implementing the SPIs in Figure 3. NetworkInfo represents information of Network status such as bandwidth, latency, etc. and ServerInfo represents Server information such as load average, CPU utilization, etc. Initialization routines for user defined

```
interface ResourceDB {
  // stores networkInfo
  void putNetworkInfo(
    NetworkInfo networkInfo
  );
  // stores serverInfo
  void putServerInfo(
    ServerInfo serverInfo
  );
  // provides NetworkInfo between
  // sourceNode and destinationNode
  NetworkInfo getNetworkInfo(
    Node sourceNode,
    Node destinationNode
  );
  // provides ServerInfo of serverNode
  ServerInfo getServerInfo(
    ServerNode serverNode
  );
  // implements process when a simulation
  // finishes
  void finish();
}

interface NetworkPredictor {
  // returns Prediction of the Network
  // between sourceNode and destinationNode
  NetworkInfo getNetworkInfo(
    double currentTime,
    Node sourceNode,
    Node destinationNode,
    NetworkInfo networkInfo
  );
}

interface ServerPredictor {
  // returns Prediction of serverNode
  ServerInfo getServerInfo(
    double currentTime,
    ServerNode serverNode,
    ServerInfo serverInfo
  );
}

interface Scheduler {
  // returns serverNodes for the request
  ServerAggregate selectServers(
    double currentTime,
    ClientNode clientNode,
    RequestedData data
  );
}
```

Figure 3. Overview of the Scheduling Unit SPI.

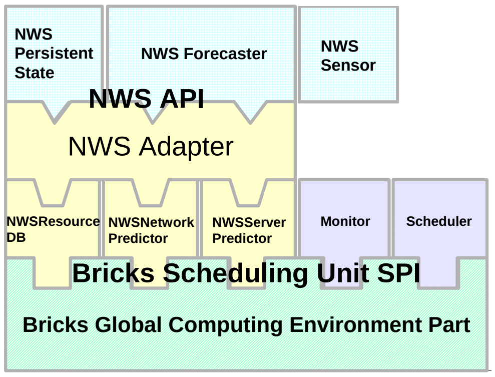
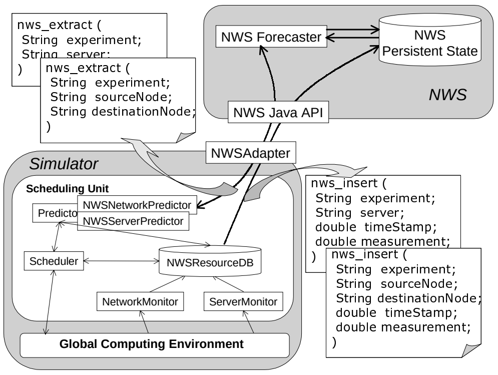


Figure 4. The Interrelationship between the Bricks SPI and the NWS API.



```
nws_extract (
 String  experiment;
 String  server;
)
```

```
nws_extract (
 String  experiment;
 String  sourceNode;
 String destinationNode;
)
```

```
nws_insert (
 String  experiment;
 String  server;
 double  timeStamp;
 double measurement;
)
```

```
nws_insert (
 String  experiment;
 String  sourceNode;
 String destinationNode;
 double  timeStamp;
 double measurement;
)
```

Figure 5. Incorporating the NWS modules into Bricks.

components are automatically invoked by the Bricks via Java reflective API.

The Interrelationship between the Bricks SPI and the NWS API is shown in Figure 4.

4.2. Incorporating the NWS system

Although we are still at an experimental stage, as a first step we have chosen to incorporate the NWS system, which monitors and forecasts the behavior of global computing systems based on past observations. NWS was developed at UCSD prior to Bricks, so there were no special provisions to run NWS under a simulated environment.

Although there have been several works in integrating NWS into existing global computing systems by the use of its C-based API, such as AppLeS[6], Legion[10], Globus[8], and our Ninf system[11], all the systems executed under a *real* environment, and as such NWS required little or no change despite that parts of its modules had been written with assumptions about its underlying execution environment. This is because the systems were orthogonal to NWS, and assumed an identical or similar execution environment.

For incorporation of NWS into Bricks, the situation is somewhat different. In this case, NWS must be made to work in simulated virtual time, and that the observed measurements will be fed from Bricks.

In order to incorporate NWS, we developed the NWS Java API[2], which largely offers the same feature as the C-based NWS API. Most of the work in incorporation had been done using the Java API, removing some of the underlying dependencies when necessary, so that NWS could be managed to work under virtual time.

NWS consists of the following modules:

Persistent State: is storage for measurements. It is similar to the Bricks ResourceDB.

Name Server: manages the correspondence between the IP address and the domain address for each independently-running modules of NWS.

Sensor: monitors the states of networks and server machines in global computing systems. *Sensor* works in a similar manner to the NetworkMonitor and the ServerMonitor in Bricks.

Forecaster: predicts availability of the resources. Again, this is similar in behavior to the Predictor in Bricks.

We substituted the default Bricks ResourceDB and the Predictor with the NWS *Persistent State* and the *Forecaster* in Bricks, respectively. The Monitors store their measurements into the *Persistent State*, and the Scheduler allocates a task using resource availability predicted by the *Forecaster*. The NWSResourceDB, the NWSNetworkPredictor and the NWSServerPredictor implement the SPIs; finally the NWSAdapter, which converts the data formats between Bricks and the NWS Java API, mainly serves to interface NWS and Bricks.

Figure 4, 5 illustrates the incorporation of the NWS modules into Bricks. Measurements made by the NetworkMonitor and the ServerMonitor are handed off to the NWSResourceDB with request for storing the measurements; The NWSResourceDB in turn converts and stores the measurements into the *Persistent State* via the NWSAdapter and the Java API. The NWSNetworkPredictor and the NWSServerPredictor also retrieve the predicted values from the *Forecaster* via the adapter and the API in a similar manner.

Parameter	Value
the interval of server monitoring	10 [sec]
the interval of network monitoring	60 [sec]
probed data size	300 [kbytes]

Table 1. The parameters of the Sensors.

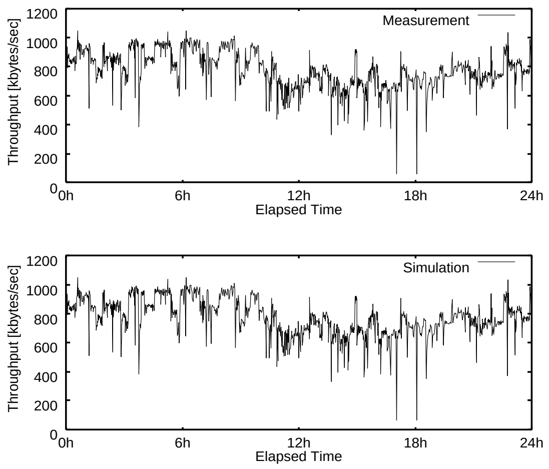


Figure 6. One day's worth of bandwidth measured between TITECH and ETL under the real environment in the figure above versus Bricks in the figure below.

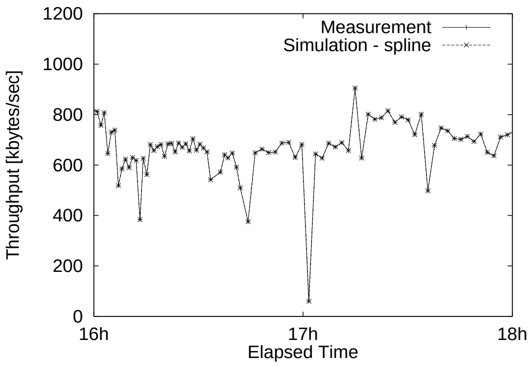


Figure 7. The comparison of bandwidth measured under the real environment versus Bricks for two hours.

5. Bricks Experiments

We now describe the experiments conducted by running NWS under a real environment versus the Bricks simulated Global Computing Environment. The experiments show that NWS behaved similarly under both environments, serving as strong supportive evidence that Bricks can provide a simulation environment for global computing with reproducible results.

5.1. Experiment Procedure

The overall experiment procedure is as follows. Initially, we set up the NWS modules in a real wide-area environment to measure real-life parameters such as network bandwidth. Then, we have the NWS *Forecaster* predict the parameters. At the same time, we drive Bricks under the observed measurements, and have the *Forecaster* make a prediction under the simulated environment. Finally we compare the results of predictions for the real environment versus Bricks.

First, we prepare the NWS *Sensor*s on two different nodes located at different sites, namely, Tokyo Institute of technology (TITECH) and Electrotechnical Laboratory (ETL) in Tsukuba, situated about 80kms away. The *Sensor*s measure the network bandwidth and latency between the nodes, and CPU availability on each. The NWS *Forecaster* predicts the availability of resources for each time-step of the measurements. Table 1 shows the parameters of the *Sensor*s.

Next, we define a Bricks simulation under the second family of models mentioned earlier, employing the observed parameters of the real environment measured by *Sensor*s, with cubic spline parameter interpolation, chosen because the interpolated value only depends on the local past (three time-steps). We incorporate the NWS *Persistent State* and *Forecaster* into Bricks and set the parameters identically as shown in Table 1.

5.2. Experimental Results

Figure 6 shows one day's worth of bandwidth measured between TITECH and ETL under the real environment on Feb. 1, 1999, versus that simulated with Bricks. The horizontal axis indicates real elapsed time or virtual elapsed time in the Bricks simulation in hours, while the vertical axis indicates the bandwidth in kbytes per second. These graphs show the bandwidth measured under Bricks is quite similar to that for the real environment. Figure 7 magnifies the time axis to two hours for direct comparison of the real versus simulated environments. Here, we can confirm that the bandwidth measured under the real environment and Bricks coincide quite well. Although there have been proposals of communication models for TCP/IP transmissions and simulations using the model, such models have been limited

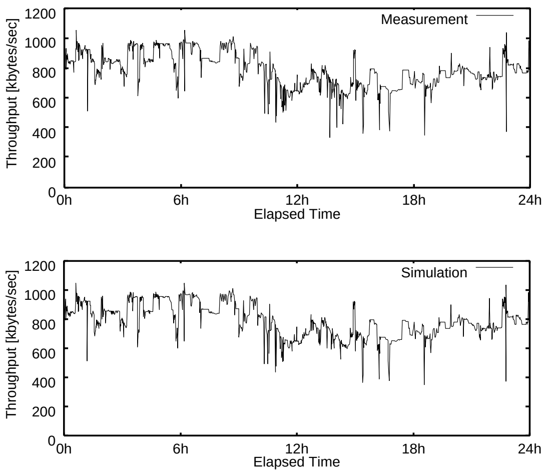


Figure 8. One day's worth of bandwidth predicted by the NWS Forecaster under the real environment in the graph above versus Bricks in the graph below.

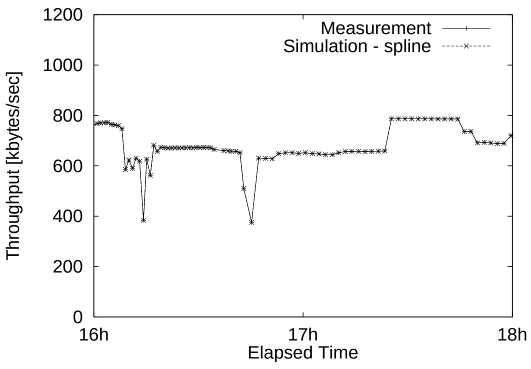


Figure 9. The comparison of the behavior of the NWS Forecaster under the real environment versus Bricks for two hours.

to describing the behavior of particular packets types, such as WWW, FTP or Telnet, due to their complexity. Bricks can adopt such models, as well as real-world measurements in cases where analytical modeling of network characteristics is difficult.

Figure 8 shows one day's worth of bandwidth predicted by the NWS *Forecaster* under the real environment versus Bricks. Figure 9 magnifies the graph and shows the comparison of the prediction for two hours. Here, we again confirm that both predictions are very similar, serving as supporting evidence that the NWS *Forecaster* functions and behaves normally under the Bricks simulation [2].

6. Related Work

While there have been abundant research on scheduling algorithms, many of them have not been implemented or well investigated. In fact there have been very little study of application of resource scheduling algorithms for global computing. The primary reason is that, for realistic environments, conducting controlled experiments for objective comparisons against other proposed algorithms and their implementations is quite difficult. The approach we have taken in Bricks is to simulate a global computing environment, and allow integration of various algorithms as well as modules from real global computing systems. In this regard, there are a couple of projects that are following a similar approach.

Osculant[3] from University of Florida is a bottom-up task scheduler for heterogeneous computing environment. To evaluate their scheduling algorithms, there is an Osculant Simulator which can also represent various network topologies and node configurations. Compared to Bricks, Osculant Simulator was not designed to be a performance evaluation environment that can integrate various components.

WARMstones being proposed by the Legion group at University of Virginia is conceptually similar to Bricks, although it seems to not have been implemented yet. WARMstones is based on the MESSIAHS[7] system, which consists of the system description vector to represent the capabilities of a server machine, the task description vector which denotes the requirement for the task, and MIL (MESSIAHS Interface Language) and Libraries to represent different scheduling algorithms in an easy and flexible manner. Instead, we choose to provide an object-oriented framework, namely the Scheduling Unit SPI as Java interfaces for implementing various scheduling algorithms, as well as foreign components. Although there are several ambitious technical aspects of WARMstones, it remains to be seen whether WARMstones, when implemented, will offer easy extensibility or allow integration of modules from existing global computing systems.

[2]To be more precise, we did experience a small discrepancy between Bricks and the real measurements. Currently, we are conjecturing that this is due to missing measurements due to lost packets in the real environment (i.e., for Bricks-driven simulation, NetworkMonitor is always successful at making a measurement, whereas in the real environment a measurement might not be made due to packet being lost in the network). When we compensated for the dropped packets, the measurements matched quite well. We are still conducting research to investigate this phenomenon.

7. Conclusions and Future Work

We proposed the Bricks performance evaluation system that allows objective and reproducible evaluation of high-performance global computing systems with queuing theory-based simulation, especially the behavior of network and resource scheduling algorithms. The users of the Bricks system can specify network topologies, server machine architectures, communication models and scheduling framework components using the Bricks script, allowing easy construction of a particular global computing system configuration. Moreover, Bricks is componentalized such that not only its constituents could be replaced to simulate various different system algorithms, but also allows incorporation of existing global computing components via its foreign interface. Experiments conducted with NWS serve as a supportive evidence that Bricks is effective in this regard.

As a future work, we plan to extend Bricks in several ways. First, the current representations of task, communication and server models need to become more sophisticated, requiring extensions to represent wider class of global computing system configurations. Task models have to be extended to allow representation of parallel application tasks, and server models should represent various server machine architectures, such as SMPs and MPPs, as well as scheduling schemes of realistic machine-specific resource schedulers such as LSF (Load Sharing Facility). Moreover, aggregate constraints on resource scheduling, such as co-scheduling requirements, should be representable. As a system configuration language, we plan to substitute XML with the Bricks script for wider usage. Finally, we plan to investigate suitable scheduling algorithms themselves for global computing systems, in particular our Ninf system, using Bricks. We plan on running Bricks on a dedicated parallel NT cluster of 33 nodes to conduct various parameter studies.

Acknowledgments

We would like to thank Rich Wolski at UTK and Jim Hayes at UCSD of the NWS team for their precious advice. We also thank the contributions of the Ninf project members, in particular, Satoshi Sekiguchi, Hiromitsu Takagi and Osamu Tatebe at ETL, Mitsuhisa Sato at RWCP, Hirotaka Ogawa at TITECH and Haruo Hosoya at Ochanomizu University. This research is being conducted with grants from the JST PRESTO program and the Software Engineering Foundation.

References

[1] Bricks. `http://ninf.is.titech.ac.jp/bricks/`.

[2] NWS Java API. `http://ninf.etl.go.jp/~nakada/nwsjava/`.

[3] Osculant. `http://beta.ee.ufl.edu/Projects/Osculant/`.

[4] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, and U. Nagashima. Performance evaluation model for job scheduling in a global computing system. In *Proc. 7 th IEEE International Symposium on High Performance Distributed Computing*, pages 352–353, 1998.

[5] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima. Performance evaluation model for scheduling in a global computing system. *International Journal of High-Performance Computing (submitted)*.

[6] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proc. the 1996 ACM/IEEE Supercomputing Conference*, 1996.

[7] S. J. Chapin and E. H. Spafford. Support for implementing scheduling algorithms using messiahs. *Scientific Programming*, 3:325–340, 1994.

[8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[10] A. Grimshaw, W. A. Wulf, and the Legion team. The legion vision of a worldwide virtual computer. *Comm. ACM*, 40(1):39–45, 1997.

[11] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A network based information library for a global world-wide computing infrastructure. In *Proc. HPCN'97 (LNCS-1225)*, pages 491–502, 1997.

[12] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The network weather service. In *Proc. the 1997 ACM/IEEE Supercomputing Conference*, 1997.

[13] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. Technical Report TR-CS98-599, UCSD, 1998.