An Experimental System for Grid Meta-Broker Evaluation

Yanbin Liu
Norman Bobroff
Liana Fong
Seetharami Seelam
IBM T. J. Watson Research, Hawthorne, NY, USA
{ygliu, bobroff, llfong, sseelam}@us.ibm.com

David Villegas
S. Masoud Sadjadi
Florida International University (FIU), Miami, Florida, USA
{dvill013, sadjadi}@cs.fiu.edu

Ivan Rodero
Technical University of Catalonia and Barcelona Supercomputing Center, Barcelona, Spain
irodero@bsc.es

ABSTRACT

Grid meta-broker is a key enabler in realizing the full potential of inter-operating grid computing systems. A challenge to properly evaluate the effectiveness of meta-brokers is the complexity of developing a realistic grid experimental environment. In this paper, this challenge is addressed by a unique combination of two approaches: using reduced workload traces to demonstrate the resource matching and scheduling functions of the meta-broker, and using emulation to provide a flexible and scalable modeling and management for local resources of a grid environment. Real workload traces are reduced while preserving their key workload characteristics to allow exploration of various dimensions of meta-broker functions in reasonable time. Evaluation of round-robin, queue-length, and utilization based meta-broker scheduling algorithms shows that they have different effects on various workloads.

Categories and Subject Descriptors

C.4. [Computer Systems Organization]: performance of Systems – measurement techniques

General Terms

Design, experimentation, measurement

Keywords

Grid computing, meta-brokering, meta-scheduling, trace, emulation, experimental systems

1. INTRODUCTION

The Latin American Grid (LA Grid) [1] is a multi-institute initiative established for the development of a computer grid that serves as a living laboratory for education and collaborative research in distributed systems and application areas such as bioinformatics, hurricane mitigation, and healthcare [2]. The promise of grids like LA Grid consists of making heterogeneous resources location transparent and accessible through common interfaces and protocols [3].

Meta-brokers (MB) [4] are key components in realizing this vision of the inter-operating compute grid. They enable communicating job and resource information between grids using command protocols, match and distribute workloads among the candidate resources, and assist global optimizations by job routing based on appropriate policies and scheduling algorithms. In addition, a meta-broker provides a layer of middleware on top of different local resource management systems (LRMSs). Figure 1 is an overview of the grid connecting compute clusters at IBM, Barcelona Supercomputing Center (BSC), and Florida International University (FIU).

A fundamental challenge in the development of meta-brokering function is to validate how the local behavior of the MB and its job forwarding algorithms affect job performance in a realistic grid environment. Given the large size and heterogeneous nature of grid environments, different strategies have been devised towards these validation goals. Examples of these strategies are small deployments in controlled environments, or the use of analytic or simulator models. However, the size of production grids such as the Open Science Grid makes small, controlled testbeds unrealistic and its heterogeneity, in terms of user requests, computing resources and implementations present significant challenge for simulators to capture the full behavior of all involved components.

In this paper, we introduce an approach to evaluating the effects of MB function based on using actual MBs, and using *emulation* of grid physical resources. We select emulation rather than simulation as typical event simulators don't allow the virtual resources to be plugged into the existing code. Additionally, emulation re-produces the behavior of a component in the system so that externally it appears to behave as the component itself. We can replace an emulated component with a real implementation without influencing other components in our grid system.

In particular, as the first contribution of this paper, we provide software emulation of the relevant features of the LRMS (Section 3.1). Our LRMS emulator provides job scheduling on clusters of compute nodes whose scale and power are parameters to the emulator. This enables experiments with multiple grid configurations by 'plugging' LRMS (compute clusters) into the MBs domain of control without modifying the MB implementation. The MB sees the virtual resources provided by the emulator as real ones. This greatly enhances the variety of tests that can be performed under different cluster configurations. Most importantly, by retaining the real MBs on their actual physical networks, it is possible to preserve the large scale structure of the grid that we use to study various interaction hahaviors.

The second contribution of our approach is to drive the MBs and emulated LRMS with realistic job demands in terms of arrival rate, execution time, and parallelism, yet be able to complete the experiments in reasonable times. To this extent, we use both cluster job traces (CTC [14]) and grid job traces (Grid5000 [5]). These traces normally execute over long time frame. So we introduce a heuristic scaling methodology in Section 3.2 and reduce the traces by about a factor of 60 in the arrival rate and execution dimensions, while preserving key properties (Section 4.2) of the original job traces.

Once our experimental grid platform is introduced in Section 4.1, we show how it is used to explore several interesting issues in MB-based grids in Section 4.3. We consider several aspects of job forwarding at the MB. In one experiment, we consider the relative effects of the job forwarding algorithms in the MB between round-robin, and the ones that using system metrics such as queue-length and node utilization.

2. BACKGROUND AND RELATED WORK

Figure 1 shows our LA Grid system with peer-to-peer meta-brokers connecting the different resource domains. In each domain, local resources are managed by their respective management entities (e.g., BSC by eNanos [6], IBM by TDWB [7], and FIU by GridWay [8]), while the MBs provide for inter-operating grid functions so that user workloads submitted by any domain may utilize resource from any of the domains dictated by policies.

The peer-to-peer communication protocols supported by the meta-brokering are: **Connection management,** which is responsible for negotiations among peers and keeping track of active neighbors via heartbeats; **Resource management**, which governs the information exchange of resources among peers; **Job management**, which handles job submission from users as well as job routing to peers for workload balancing; and **Notification management,** which sends and receives job status notifications to registered listening parties. Our previous publication [6] describes our verification and experimentation with our protocol design and implementation.

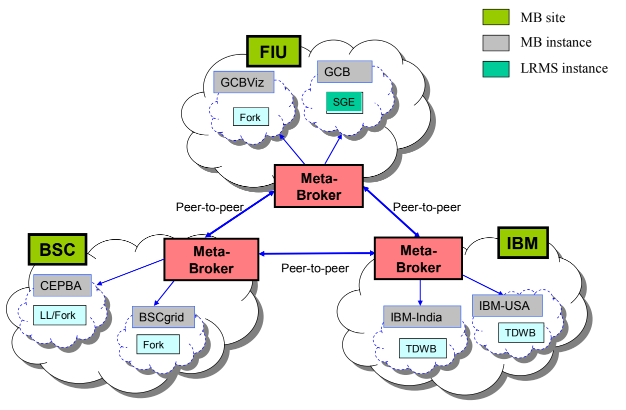

Figure 1: Meta-Broker Architecture

As argued by Iosup et. al. [9] on performance evaluation of grid systems, the large size of grids and their heterogeneity make realistic analytic modeling hardly tractable. Also, the non-deterministic and other dynamic behaviors of grid make simulation approach less suitable; therefore, performance studies using real systems would be an alternative. For example, real meta-brokering systems can provide realistic setup to take performance measurements such as the overhead in job routing, network latency, etc. While our meta-brokering system is built to be a real deployment in the LA Grid consortium to handle workload from our partner sites, it is also a living laboratory for experimentation of distributed system. Thus, we need not only to collect and analyze performance data to quantify the quality of services for our user workloads and the optimal usage of the resources contributed by partner institutes, but we desire to use our grid system to explore various aspects of performance evaluations of meta-brokering functions such as workload to resource matching and allocation algorithms.

However, performance evaluation using real systems has many challenges. The complexity of setting and controlling different characteristics of real systems is not trivial and often limit the flexibility to properly evaluate meta-brokering systems. For example, to vary the sizes of resource domains for testing brokering matching and allocation algorithms would likely require coordination of testing timing and participating administrative personnel. To alleviate such complexity and to provide the flexibility in varying sizes and types of resources in grids, in this paper we introduce an innovative approach in the use of emulated resources and their management schemes. The concept of using emulated resources and their allocations was also previously mentioned in [10] for cluster scheduler evaluation. We extend the emulation to multiple grid clusters for the meta-brokering evaluation of varying characteristics.

Argued that analytic modeling for grids is hardly tractable, Iosup et. al. [9] used simulation methods [11] to evaluate the performance of grid systems. Several other research groups also developed grid simulation frameworks and published several papers evaluating the various aspects of grid scheduling algorithms [11,12,13,22]. However, we argue that, due to the non-deterministic and other dynamic behaviors of grids, these simulation systems suffer from similar drawbacks as analytical models. As we will show in this paper, performance studies using real systems offer insights that are hard to learn from analytical or simulation methods. Our experimental system based on emulation is different from these former simulation systems. We have real MB (sometimes real LRMS too) instances and emulated LRMS deployed in real networks using real time. Our components use web services to communicate with each other such that we can replace an emulated component with real implementation or vice-versa in our system. This gives us a benefit to detect problems and evaluate performance in a close to reality environment.

Another challenge in meta-brokering evaluation is the availability and selection of realistic workloads. Iosup et. al. [5] have been leading the effort of establishing the availability of traces captured from real systems in the Grid Workload Archive (GWA) while Parallel Workload (PWA) [14] contains detailed workload models, which are based on workload logs collected from large scale parallel systems in production use. In [15], the authors discussed various challenges in selecting the appropriate workloads with particular characteristics for specific performance evaluations. Various aspects of real and synthetic workloads used in performance studies were also covered in many studies [16,17].

In our study here, we evaluate our experimental system using two job workloads that were constructed based on the characteristics of selected real workloads (CTC from PWS, Grid5000 from GWA). We would apply a set of trace reduction techniques to real workload traces to allow exploration of various dimensions of meta-broker functions in reasonable time. Trace reduction techniques have been used by computer architecture designers to shorten time for simulations for microprocessor design. Authors in [18] compared the approaches of sampling and reduced input sets by using different techniques, such as reduction in repetitiveness and input truncation, while maintaining statistical similarity to the original input traces. We detail our workload selection and trace reduction approach in Section 3.2.

3. META-BROKER EVALUATION PLATFORM

Figure 2 shows an architectural overview of the prototype meta-broker evaluation platform used for this paper. Instances of meta-brokers are deployed at the multiple resource domains. Each meta-broker implements the interoperability protocols described previously in Section 2. Based on brokering policies and resource information exchanged amongst peer brokers, jobs entered to a meta-broker are executed locally or routed to remote meta-brokers. Each meta-broker interacts with one or more LRMS to allocate resources for the jobs. As shown in Figure 2, LRMS can be either a real system (e.g., IBM LoadLeveler [19]) managing real physical resources and/or an emulated LRMS. Job traces are used to generate job workloads by a job submitter from any client site. The functional steps of the experiment are indicated in the figure by the labels: (1) Job forwarding to another MB, (2) Job submission to a LRMS, and (3) Job submission into the system by a user.

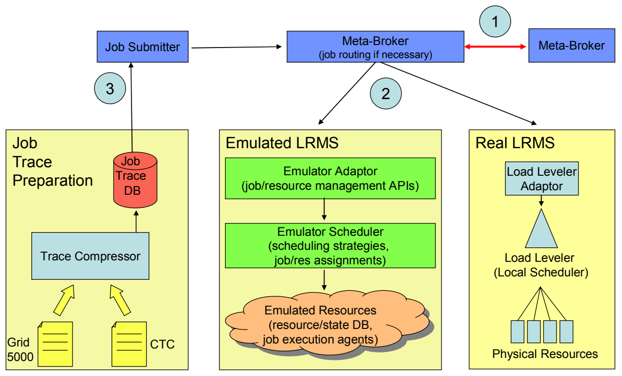
1
Job Submitter
Meta-Broker
(job routing if necessary)
Meta-Broker
3
2
Job Trace Preparation
Job Trace DB
Trace Compressor
Grid 5000
CTC
Emulated LRMS
Emulator Adaptor (job/resource management APIs)
Emulator Scheduler (scheduling strategies, job/res assignments)
Emulated Resources (resource/state DB, job execution agents)
Real LRMS
Load Leveler Adaptor
Load Leveler (Local Scheduler)
Physical Resources

Figure 2: Meta-broker evaluation platform

3.1 Emulated Local Resource Management

As shown in Figure 2, the LRMS emulator component consists of three main parts: the *Emulator Adaptor*, the *Emulator Scheduler*, and the *Emulated Resources*. Note the equivalency of the functions in the emulated LRMS to those of the real LRMS (IBM LoadLeveler or any other generic LRMS). The emulation adaptor interacts externally with meta-brokering for job management (e.g., submission, termination, and query) and resource information exchange. For example, the resource information may include types and numbers of resources available in LRMS, and various utilization information (e.g. statistics on job queue lengths and system utilization).

The second part of the emulated LRMS implements the behavior of a local scheduler (e.g., batch scheduler like Condor, PBS, SGE, or LoadLeveler) that performs assignments of jobs to resources managed by the LRMS. Our current implementation of the emulator scheduler is the first-in-first-fit algorithm. In this scheduling discipline the job queue is always sorted by arrival time. However, when the scheduler processes the queue, if there are insufficient resources for the first job to run, the next job is considered and will run if sufficient resources are available, and so on until a job can be run or the queue is exhausted. This scheduling policy keeps the resources utilized, but has the potential starvation of large jobs. More complex algorithms like backfill scheduling can mitigate the potential of resource starvation for jobs requiring large number of nodes. The experimental section 4.3 of the paper shows the occurrence of job starvation.

The third part of the emulated LRMS provides virtual resources from a configuration file, as in Listing 1.

It keeps the current state of each resource in terms of OS types, processor architecture, memory, and disk utilization. We assume that each job utilizes completely a processor, and that different jobs don't share the same set of processors. This may seem unrealistic in terms of emulating the performance of an application in a given machine (the impact of the instruction set, memory access speed, and I/O interrupts). However, the goal of the emulator is not to give an accurate representation of how fast an application runs on a given set of machines, but to understand the characteristics of resource utilization and job services.

```
<EmulatedResources xmlns="http://cs.fiu.edu/emulator/resources">
  <Resource>
    <Count>64</Count>
    <Architecture>x86</Architecture>
    <CPUCount>2</CPUCount>
    <OS>LINUX</OS>
    <PhysicalMemory>1024</PhysicalMemory>
  </Resource>
  <Resource>
    <Count>32</Count>
    <Architecture>powerpc</Architecture>
    <CPUCount>4</CPUCount>
    <OS>AIX</OS>
    <PhysicalMemory>4096</PhysicalMemory>
  </Resource>
</EmulatedReousces>
```

Listing 1: Emulated Resource File

3.2 Workload Trace Reduction

We have chosen to drive our experiments from real job traces representing two usage scenarios. One trace is selected from the Parallel Workloads Archive and contains 11 months of execution at the Cornell Theory Center (CTC) on a cluster of about 450 single processor IBM SP2 nodes with similar CPU, memory and disk. The second trace is available at the Grid Workloads Archive and corresponds to the Grid5000 experiment, which comprises 9 different locations and 15 computing clusters.

Real-time execution is used to allow the emulated resources to interact with the physical system components. In order to conduct experiments based on long traces in reasonable time the traces need to be compressed in the time domain. A heuristic approach in reduction is taken and shown to retain several relevant properties of the original trace. Because the timescale of MB scheduling is long (many seconds) it is unnecessary to preserve the fine scale details of the original trace. The initial work reported here targets an execution time of a few hours and is derived from samples of about one week's data from the original traces. The process has three stages:

1. Select a sample interval from the original trace. The archive traces contain many months of data. So the first step is to select an interval representative of the entire trace. This interval is compressed in subsequent steps. For example, the CTC trace, which corresponds to a cluster of homogeneous computers, contains well defined and repetitive periods of submissions with duration of a week so selection is straightforward. The Grid5000 trace contains more 'batched' submissions with periods of activity isolated by inactivity. Therefore we select a period of activity in which this behavior can be observed. We avoid using intervals that belong to the warm-up or wrap-up phases of the system. In this paper the sample interval from each trace is a week and is compressed by a factor of 60.
2. Trace sampling. After the trace interval is selected, a subset of jobs is chosen from this interval. This subset size is not fixed but depends on the sampling rate, which considers the target resource size appropriate in the emulated cluster experiments. Because the CTC trace is from a cluster at Cornell of about 450 machines, we decided to use this same size of resources in our test environment. We further reduce the number of jobs such that the total execution time is a few hours by job sampling from the weekly trace, for example taking only 2 out 3 jobs. The requirements of each remaining job are preserved in this phase. In particular, the number of requested processors, the job submission time, and the job execution time are left unchanged.
3. Time scaling. In this phase independent scaling factors are applied to the job inter-arrival and execution times. Although for the experiments reported here a common scale of 60 is applied to both.
Finally, the properties of the traces prior and subsequent to reduction are compared. Figure 3 shows the cumulative distribution of processors, which indicates that the ratio of requested CPUs stays the same after modifying the trace. Then we perform a hierarchical clustering analysis with average linkage to find the submission trends of the trace by grouping jobs that were submitted in similar time periods. The distance between clusters is defined by using the difference between submission times. We heuristically determine k, the number of clusters, by plotting the total within-cluster sum of squares (WCSS) for different values of k and then finding the value for which the WCSS has a smaller increase, or an "elbow", in the graph. This and other methods are discussed in [20]. Figure 4 shows that clusters are closely correlated between the original and sampled traces: this indicates that sampling retains the arrival time distribution of the original workload.

4. EXPERIMENTS AND RESULTS
In this section we demonstrate how our experimental system helps us evaluate the performance of meta-broker job forwarding algorithms in a grid environment. We evaluate MB forwarding algorithms: round robin, queue length, and node utilization. We observe that anticipating which scheduling algorithm works best for job forwarding is difficult. This may result from the complex interaction between the workload, cluster size, and the algorithm. After a brief description of the experimental setup and our trace characterization, results are presented in Section 4.3.

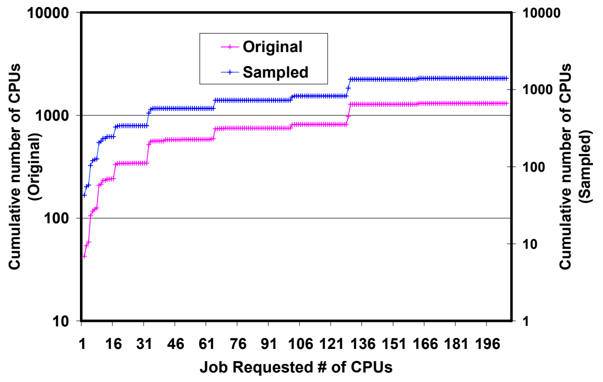

Figure 3: CTC original and reduced traces

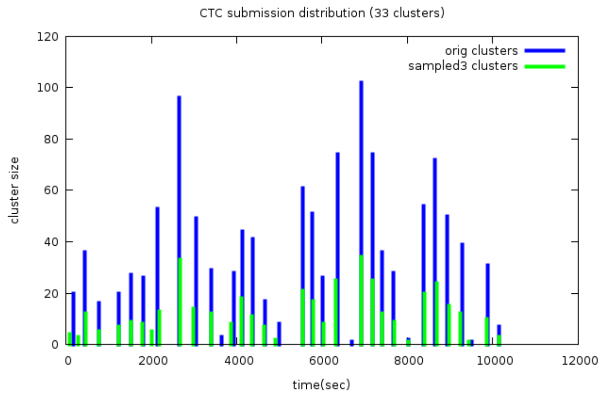

Figure 4: Cluster analysis of original and reduced traces

4.1 Experimental Configuration
The basic setup is shown in Figure 2. The implementation consists of one instance of the job submission component, and one or two instances of TDWB meta-brokers (TDWB-MB). Each MB has a single emulated LRMS. Jobs are submitted to one MB and may be forwarded to another MB.

The detailed implementation of TDWB-MB is described in [21]. The TDWB-MBs establish connections with each other and exchange resource information and forward job requests as dictated by the scheduling policies. The scheduling policies used in our experiments are based on job queue length and system utilization. The scheduling function of MB is invoked once every 5 seconds to assign jobs waiting on the MB input queue to its local LRMS or forward them to a remote MB.

The emulated resources managed by each LRMS are in a configuration file. The LRMS sends resource information, performance metrics (e.g. queue length, utilization), and job state (e.g. submitted, executing, complete) updates to the MBs. The utilization statistics include the instant, 30-second average, and 300-second average of both job queue length and node utilization. Utilization statistics are exchanged among MBs every 15 seconds. The scheduling algorithm emulated by the LRMS is first-in-first-fit as described in Section 3.1. The LRMS job scheduling interval is 5 milliseconds which is essentially instantaneous.

4.2 Trace Characterization

The properties of the reduced traces from CTC and Grid5000 are summarized in ***Figure 5***, ***Figure*** 6, Table 1, and Table 2. For each job, the figures show the execution time, number of nodes required, and the submission time. Both traces are primarily scientific workload. As shown in the tables, the Grid5000 workload has more variable execution time as well as parallelism than that of CTC. The execution time in the Grid5000 figure is in log-scale to highlight the wide range of execution time of the different jobs.

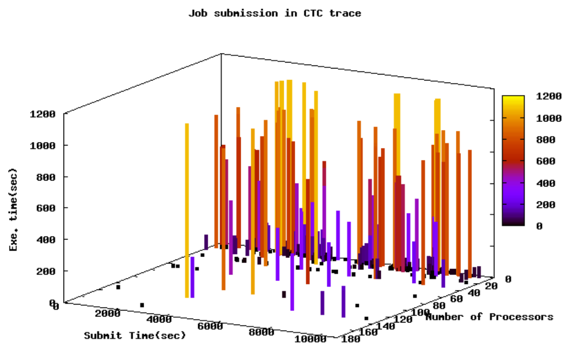

Figure 5: CTC job profile

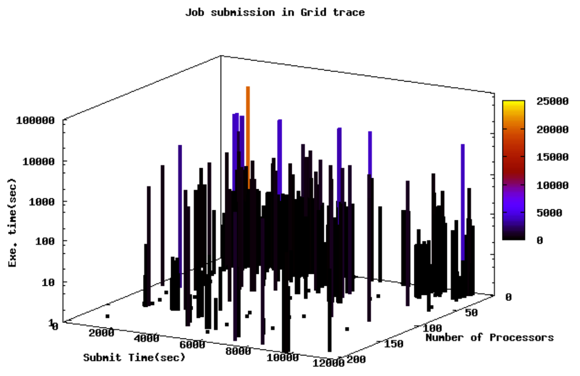

Figure 6: Grid5000 job profile

Table 1: Job execution characteristics

Execution Time	Max	Mean	Standard Deviation	Coefficient of Variation
CTC	1081	223.9	332.7	1.486
Grid5000	10000	476.8	1190.3	2.496

Table 2: Processor demand characteristics

Num of Processor	Max	Mean	Standard Deviation	Coefficient of Variation
CTC	162	14.0	29.4	2.095
Grid5000	204	18.7	39.4	2.106

Additional insight into the traces is provided by computing the resource 'demand'. A simple analytic evaluation is performed on each trace to compute the number of concurrent jobs and occupied processor nodes in the system as a function of time on the assumption of an environment with unlimited resources. In other words, each job is scheduled immediately as it is submitted, so there is no overhead and no contention waiting for free nodes. The results are shown in ***Figure 7*** and ***Figure 8***. For the CTC trace, there is a peak requirement of about 1400 processors at around 9000 seconds. Besides the peak, 400 processors can satisfy most requirements without resource competition. For the Grid5000 trace, the peak requirement is around 700, which is smaller than that of the CTC. However, Grid5000 has a much heavier workload than CTC because it has more concurrent jobs and they have consistently higher demand for the processors. This gives us hints on how to size our experimental environments.

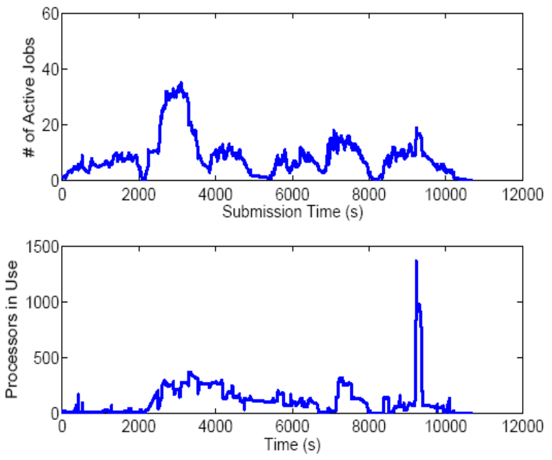

Figure 7: CTC execution under no contention

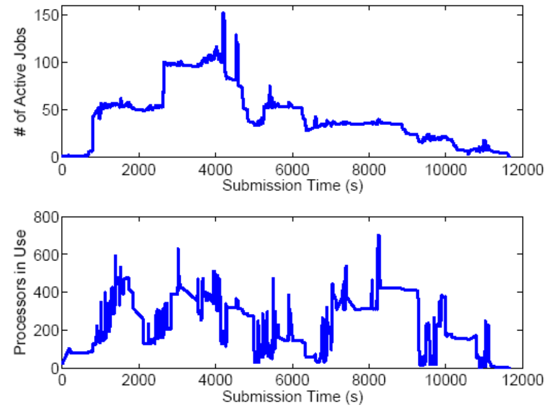

Figure 8: Grid5000 execution under no contention

4.3 Results

In this section, we present results of three sets of experiments: comparison of single grid and multiple grids, evaluation of job routing policies between MBs.

Since jobs were submitted to a single MB, we keep the record of the jobs' *arrival_time* and jobs' *finish_time* reported by LRMS in this particular MB. The job delay time is computed as:

Delay time = finish_time - arrival_time - job's execution time.

This delay time accounts for all the overhead caused by the system: the communication time between MB-to-MB, MB-to-LRMS, the scheduling algorithm execution time, and the waiting time on the job queues during the scheduling intervals of MB and LRMS and/or waiting for available resources in the LRMS.

We mainly demonstrate the delay in our results. We also calculate expansion ratio and show the results too.

4.3.1 Comparison of single cluster and multiple grid clusters

The first set of experiments provides baseline results of using one MB site under the load of the traces. We begin with CTC trace. We choose the number of machines (nodes), each of which has one processor, to be 450 for one MB site scenario. This number is close to the original environment, which has 451 processors, where the CTC traces were collected. Then, we halve the resource number to 225 for the MB site. We show the delay time of each job in Figure 9 indexed by its submission time.

With 450 machines, we have more machines than requested processors most of the time. When there is no resource competition, the delay time is around 2.5 seconds, which is half the scheduling interval of our MB. We have a peak of delay time after 9000 seconds on the x-axis that corresponds to a demand peak. It is caused by 9 jobs each of which requires 129 processors that arrive sequentially.

Then, we halve the number of machines to 225. Not surprisingly, we can see that the average delay time has increased by more than a factor of three as shown in Table 3.

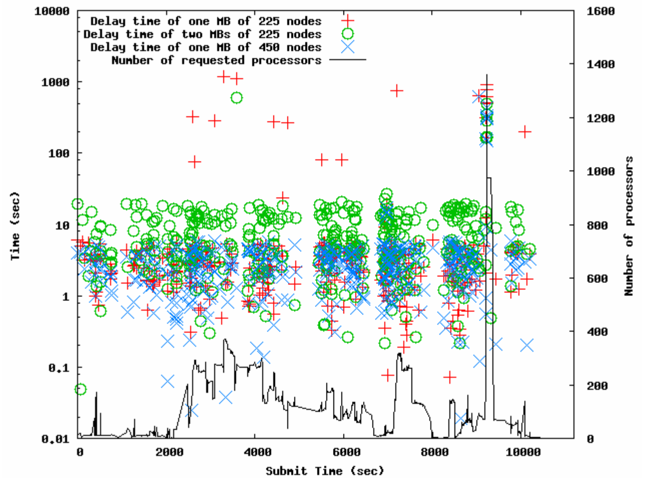

Figure 9: CTC execution under 450 and 225 resources

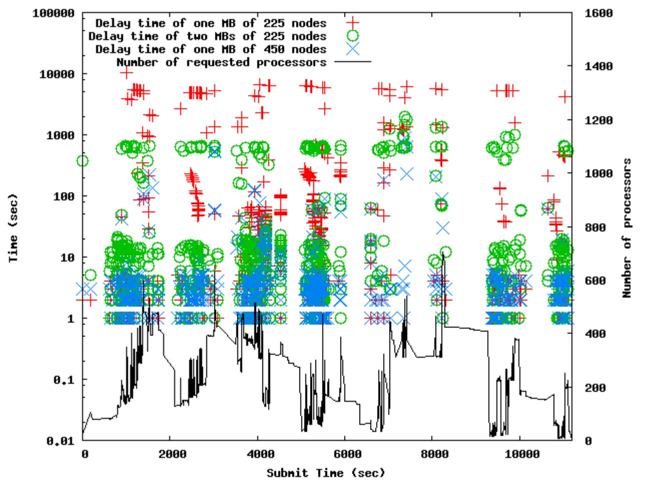

Figure 10: Grid5000 execution under 450 and 225 resources

To exercise two MBs and their functions, we inter-connect two MB sites each with 225 machines. We use Round-Robin algorithm to assign the jobs between the two sites. The result is shown in Figure 9. The delay time is larger than that of the 450-machine case. The delay time when there is no resource competition is increased to around 5 seconds. This is because now the job goes through two MBs before it is dispatched to a resource to execute**.**

Our second trace is from Grid5000. We also set the resource size to be 450 machines, which will satisfy the resource requirement most of time except for several of demand peaks. In Figure 10, the peaks of delay time match fairly well with the demand peak. We also collect results of one site of 225 machines and two sites of each with 225 machines using Round-Robin algorithm. Note that in the Grid5000 experiments, the delay time is calculated in seconds instead of milliseconds because of the long delay time such that we do not have delay time less than one second. As shown in Table 3 and 4, halving the resource size has dramatic impact on Grid5000.

Table 3: CTC/Grid5000: execution delay time

# machines	Max	Mean	Standard deviation	Coefficient of variation
CTC 450	628.6	7.09	58.6	8.27
CTC 225	1168.3	24.06	129.4	5.38
Grid 450	905.0	8.16	53.1	6.51
Grid 225	10362.0	420.21	1327.8	3.16

Table 4: CTC/Grid5000: execution expansion

# machines	Max	Mean	Standard deviation	Coefficient of variation
CTC 450	38.7	1.71	2.8	1.65
CTC 225	101.1	2.25	6.6	2.95
Grid 450	906.0	6.01	47.2	7.85
Grid 225	10363.0	189.14	864.8	4.57

4.3.2 Evaluation of job routing policies between meta-brokers

When we have more than one cluster in the system, we may want to route some of the received jobs to other clusters. The simplest algorithm is Round-Robin algorithm. In this set of experiment, we compare the performance of a couple of other algorithms to Round-Robin.

Our meta-brokers collect resource information from LRMSs and make scheduling decision based on the resource information. When LRMS can report more information, especially dynamic information, we may make better informed decisions. In our system, LRMS, which is implemented by the emulator, can report its queue length and the number of busy nodes to the meta-broker; meta-broker will exchange this information with each other. Then, the scheduling algorithm of MB can make scheduling decisions based on the data.

We experiment on three straight-forward algorithms: queue-length, free-node and utilization algorithms. MB will compare the queue length, the number of free nodes, or the utilization of different LRMSs and assign jobs to the one with the shortest queue length or the biggest number of free node or the smallest utilization. We use a 30-seconds-average queue-length number and instantaneous free node number for the results shown in the figure. The jobs assigned to remote LRMSs are routed to remote MB, which will then make its scheduling decision independently.

We set up two clusters, each of which has a MB and 225 machines. We show the results of CTC and Grid traces in Figure 11 and 12.

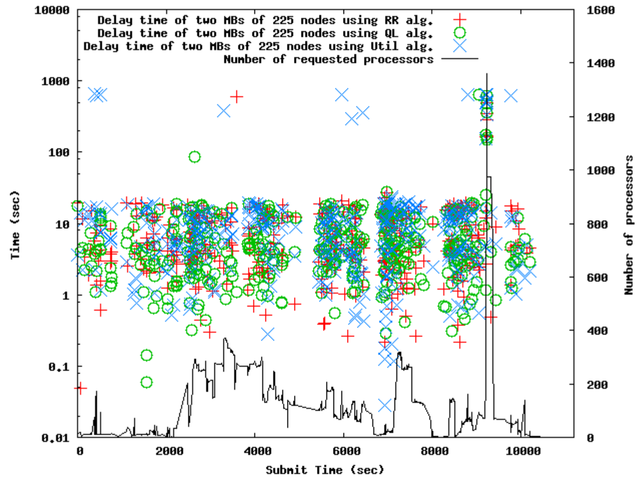

Figure 11: CTC execution with Queue Length & Utilization

To further demonstrate the performance improvement, we also list the mean and variance of the delay time in Tables 5, 6, 7, and 8. Since most requirement peaks consist of homogenous jobs arrived in short period of time, Round-Robin gives us a pretty good performance. While free-node algorithm helps marginally, utilization algorithm degrades the performance because it tends to allocate more jobs to the smaller utilization cluster during peak times.

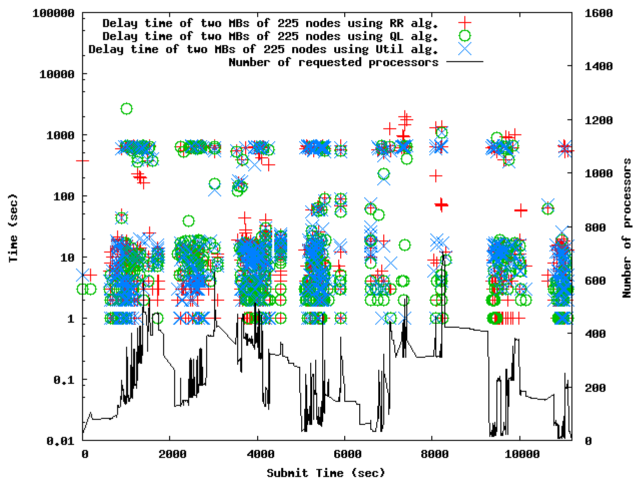

Figure 12: Grid5000 execution with Queue Length & Utilization

Table 5: CTC execution delay time for two sites

# machines	Max	Mean	Standard deviation	Coefficient of variation
225/225/RR	633.3	12.64	65.9	5.21
225/225/QL	633.0	11.37	65.0	5.71
225/225/UT	660.2	26.63	109.0	4.09

Table 6: CTC expansion ratio for two sites

# machines	Max	Mean	Standard deviation	Coefficient of variation
225/225/RR	38.9	2.98	4.7	1.57
225/225/QL	40.3	2.62	4.1	1.57
225/225/UT	639.6	8.87	55.38	6.25

Table 7: Grid5000 execution delay for two sites

# machines	Max	Mean	Standard deviation	Coefficient of variation
225/225/RR	1990	73.62	228.4	3.10
225/225/QL	2679	58.74	197.3	3.36
225/225/UT	1085	76.73	214.0	2.79

Table 8: Grid5000 expansion ratio for two sites

# machines	Max	Mean	Standard deviation	Coefficient of variation
225/225/RR	1991.0	37.19	148.4	3.99
225/225/QL	2680.0	34.07	147.9	4.34
225/225/UT	660.0	42.56	142.9	3.36

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced a new approach to evaluate the performance of meta-brokers in heterogeneous distributed grid computing environments. Our framework uses emulation to preserve the large scale structure of the grids with real deployments of meta-brokers, and uses reduced workload traces to speed-up the pace of the study. We have used the implementation of this framework to examine the impact of different job routing algorithms.

Finally, we conclude by using the discussed methods to exercise different aspects of our evaluation platform. We have shown the impact of adding more peers to the system versus adding more resources to a site, and how additional information may affect scheduling decisions at a meta-broker.

In the future, we plan to use our platform to investigate many different job routing algorithms between meta-brokers and with increase multiplicity of meta-brokers. We are also interested in measuring the network latency of job routing.

6. ACKNOWLEDGMENTS

This work was supported in part by IBM, and the National Science Foundation (grants OISE-0730065, OCI-0636031, and HRD-0833093) and in part by the Spanish Ministry of Science and Technology under contract TIN2007-60625.

7. REFERENCES

[1] LA Grid Initiative: http://latinamericangrid.org/

[2] R. Badia, G. Dasgupta, et al "*Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation*". High Performance Computing and Grids in Action, IOS Press - Amsterdam, Lucio Grandinetti, editor. Dec., 2007.

[3] Ian Foster, Carl Kesselman, Steven Tuecke. The Anatomy of the Grid. Lecture Notes in Computer Science, 2001.

[4] I. Rodero, F. Guim, et al. "Looking for an Evolution of Grid Scheduling: Meta-brokering". CoreGrid Workshop in Grid Middleware'07, Dresden, Germany, June 2007.

[5] A. Iosup, H. Li, M. Jan, et. al. "The Grid Workloads Archive". Future Generation Computer Systems, Vol, 24, Issue 7, July 2008, pg 672-686.

[6] N. Bobroff, L. Fong, et al. "Enabling Interoperability among Meta-Schedulers". IEEE 8th International Symposium on Cluster Computing and the Grid (ccGrid), May 2008.

[7] IBM Tivoli Dynamci Workload Broker: User's Guide; SG32-2281-01.

[8] GridWay: http://www.gridway.org/

[9] A. Iosup, D. Epema, et. al. "Synthetic Grid Workloads with IBIS, KOALA, and GrenchMark". In Proceedings of the CoreGRID Integrated Research in Grid Computing. 2005

[10] D. B. Jackson, B. D. Haymore, et al. "Improving Cluster Utilization through Set Based Allocation Policies". Proceedings of International Conference on Parallel Processing Workshops. 2001.

[11] A. Iosup, T. Tannenbaum, et al. Inter-operating grids through Delegated MatchMaking. Scientific Programming 16(2-3): 233-253 (2008)

[12] A. Takefusa, S. Matsuoka, et al., "Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms," hpdc,pp.11, Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8 '99), 1999

[13]I. Rodero, F. Guim, et. al. "Evaluation of Broker Selection Strategies". Computer Architecture Department, Technical University of Catalonia, UPC-DAC-RR-CAP-2008-41, 2008.

[14] http://www.cs.huji.ac.il/labs/parallel/workload/models.html

[15] Feitelson, D.G., Tsafrir, D. "Workload sanitation for performance evaluation". In IEEE International Symposium on Performance Analysis of Systems and Software, March 2006 .

[16] Virginia Lo. Jens Mache. Kurt Windisch. "A comparative study of real workload traces and synthetic workload models for parallel job scheduling". In Job Scheduling Strategies for Parallel Processing, 1998

[17] Iosup, A. and Epema, D. 2007. Build-and-Test Workloads for Grid Middleware: Problem, Analysis, and Applications. In Proceedings of the Seventh IEEE international Symposium on Cluster Computing and the Grid (May 14 - 17, 2007). CCGRID.

[18] L. Eeckhout, A. Georges, K. D. Bosschere. "Selecting a Reduced but Representative Workload". Middleware Benchmarking: Approaches, Results, Experiences. OOSPLA workshop, 2003.

[19] IBM Tivoli Workload Scheduler LoadLeveler, http://www-306.ibm.com/software/tivoli/products/scheduler-loadleveler/

[20] J. Hartigan. 1975. Clustering Algorithms, Wiley, New York

[21] N. Bobroff, G. Dasgupta, et al. "A Distributed Job Scheduling and Flow Management System". ACM Operating Systems Review, Vol. 42, Issue 1, Jan. 2008.

[22] H. J. Song, X. Liu, et al. "The MicroGrid: a Scientific Tool for Modeling Computational Grids". High Performance Networking and Computing Conference (SC), Dec. 2000.