ADAPTIVE GRID COMPUTING FOR MPI APPLICATIONS

Juemin Zhang and Waleed Meleis
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
{jzhang, meleis}@ece.neu.edu

ABSTRACT

Our objective is to provide location-, topology-, and administrative-transparent grid computing for MPI applications, while hiding the physical details of computing platforms and heterogeneous networks from the application developers and users. To achieve this objective, we introduced a new resource allocation model, workflow structures to specify MPI applications involving multiple tasks, and message relay to enable communication across different networks. We developed the SGR framework, which integrates workflow scheduling, task grouping, and message relay services, while hiding resource allocation, heterogeneous networks, and decentralized resource management systems from application developers and users. The SGR system has been implemented on a Globus-enabled computing grid.

We created a simulation environment to investigate our model and various schedulers. Using the findings from simulation, we implemented the SGR framework and tested the model's implementation on a two-cluster grid. We observed that duplication can improve performance by more than 15%, which matches our simulation results. Moreover, we evaluated our new message relay service for cross-site message passing. The test results indicate that although the SGR's message relay service has some communication overhead, the system is scalable with respect to the number of processes and the message size.

KEY WORDS

Grid computing, resource allocation and co-allocation, workflow, task scheduling.

1 Introduction

Computational applications rely on high-performance computing. Computers are becoming faster, closely following Moore's law [11]. Parallel and distributed computing has made it possible to aggregate computing power from numerous computing sources. However, these advances cannot match continuously increasing computational demands. Scientific computations frequently have non-polynomial complexity and process large quantities of data that can easily exceed the computing capacity of modern computers. Emerging applications, such as high resolution protein structure prediction [2, 14] used in protein folding, and molecular dynamic simulation [12] used in computational physics, continuously demand faster processing capability. However a single parallel system, such as a cluster, has limited resources and may not be capable of providing sufficient computing power for those computation-intensive applications. One solution to this problem is to collect and utilize distributed computing resources that are owned and maintained by different institutions, labs or computing centers.

Grid computing [7] is one way to transparently harness distributed computing power to meet the demand of computational intensive applications. When computing resources are distributed across multiple organizations, such a computing environment is expected to hide institutional and system-level differences from application users and developers as much as possible, so that an application's portability would not increase software complexity and implementation efforts. A computing grid is constructed on multiple parallel computing systems, each of which can be in a different administrative domain and separated geographically. Therefore, a major objective of grid computing is to provide location-, topology- and administration-transparent computational services for users and applications.

On a Globus-enabled computing grid, source code modifications or system-level alterations may be required to run legacy MPI programs, because heterogeneous networks and resource allocation across multiple resource administrative domains are not transparent to application developers and users. Our objective is to enable network topology-transparent execution for MPI applications, including legacy programs, to run on a computing grid, while hiding decentralized resource management and resource allocation process from applications. To achieve this objective, we extend the conventional message passing executions to workflows. We specify an adaptive grid computing framework, consisting of task scheduling, task grouping, and message relay services.

1.1 Preliminaries

In this paper, a computing resource is the computing hardware used by a single MPI process. A computing resource is a computer, a computing node of a cluster, a processor, or a CPU. A computing site is a collection of computing

resources which are connected by a private or local area network within a single administrative domain. A single administrative domain means that all computing resources within a computing site are managed by one resource management system, such as OpenPBS [9] or LSF [16]. A computing grid then consists of multiple computing sites interconnected by a public network, and software infrastructure enabling resource sharing among these computing sites.

In a resource-sharing environment, such as a cluster or a mainframe computer that has a large number of computing resources, a resource management system (RMS) is designated to manage how these resources are used. An RMS is also known as a job queuing system or a batch job system. To launch an application in such an environment, application users submit job requests (or resource requests) to the host RMS. A job request specifies the number of computing resources required by the application, the application execution code, input and output files, etc. In this study, we view the resource requirement (or resource specification) as the number of computing resources. [1]

1.2 Workflow execution

Since a computing grid can provide much more computing resources than a single computing site, it usually serves as a computing platform to solve more computation-intensive and complex applications. Such applications may need more computing resources that may exceed a single computing site's capacity, involve computing resources from different computing sites, or consist of multiple parallelized procedures with different degrees of parallelism. These applications could be different from conventional, single program applications. To address this challenge, we use a workflow to specify how a multi-task MPI application is executed.

A workflow, which contains a sequence of related activities or operations, is used to represent how these activities are scheduled within a simple flow chart. A workflow application contains a list of tasks that are executed in a specified sequence. In this study, we define the task is an execution of a specified program, which can be either a serial execution on one computing resource or a parallel execution using multiple computing resources within a computing site. Accordingly, a task is only allowed to use computing resources within one administrative domain. The workflow specifies the execution procedure of all the tasks, and interactions among them as well.

Three fundamental structures can be used to specify a workflow: dependent, parallel forking, and optional forking structures. The dependent structure contains dependent tasks that are executed sequentially and one-by-one in a flow. Tasks in a parallel forking structure are concurrent tasks. The parallel forking structure divides a flow into two or more flows, and each flow is allowed to have a task executed concurrently with other flows. The optional forking structure divides a flow into two or more candidate flows, and only one of the candidate flows that satisfy the user-specified condition will be executed at runtime. Tasks within the candidate flows are called optional tasks.

[1] Other resource requirements include the memory size, disk space, and network speed. However, they are not considered in this study.

1.3 Problem statement

Our objective is to provide location-, topology- and administrative-transparent grid computing for MPI workflow applications. To achieve these objectives for MPI applications, we must overcome three major challenges: application scheduling, resource co-allocation, and cross-site message passing.

The application scheduler performs site selection and application job submission in a computing grid environment. Besides providing services to interact with host resource management systems, the application scheduler is responsible for balancing workloads among computing sites.

The second challenge is how to allocate resources for a cross-site execution, or more specifically, resource co-allocation on a computing grid. Resource co-allocation is defined as the process of allocating and synchronizing equal amounts of computing resources from two computing sites so that the resources can be used to perform a cross-site execution. To conduct resource co-allocation, two questions must be addressed: which sites should be selected in a cross-site execution and how should resources from different sites be synchronized?

The third challenge is how to enable inter-site communication. In a grid environment, seamless communication across different computing sites for MPI applications may not be feasible at the network layer. Data communication across different networks requires network reconfiguration by either adding ports or changing routing tables. Many computing sites adopt a single access point which allows monitoring network traffic and filtering suspicious or virus-contaminated data packages. Therefore, we must provide network-transparent and adaptive cross-site message passing to enable conventional MPI applications to execute on multiple computing sites.

1.4 Related work

Many researchers have worked to develop software to enable and facilitate grid computing. These software tools provide a range of services, including interoperability among different systems, data transfer, authentication and authorization, data encryption and secured communication, and meta-scheduling. Nevertheless, none of them provides a complete solution to all those problems we discussed above.

MPICH-G2 [10] is an implementation of Globus-enabled MPI 1.1 based on the Globus Toolkit. MPICH-G2 requires that headnodes of each computing site participate in the cross-site execution. Any two computing resources in different computing sites cannot communicate

with one another directly unless they are assigned public IP addresses. MPICH-G2 uses the DUROC library in resource synchronization, which only supports static resource allocation. MPICH-GX [4] uses system proxies, which run on the headnodes, to forward inter-site messages to destination processes. However, each proxy must connect to all application processes, introducing large overhead during initialization. Furthermore, application users have to manage job submission and resource co-allocation when they start cross-site execution.

The Condor-G system [8] extends the resource management capabilities of Condor from a single administrative domain to multiple domains through integration with the Globus Toolkit. While Condor-G addresses the location- and administration- transparency issues, its execution environment cannot create a network-transparent environment for parallel applications running on multiple computing sites. Nimrod/G [3] relies on resource reservation and bidding techniques to achieve optimized results in terms of computing cost. However, in order to do so, the scheduler must know resource availabilities and the task execution time. Advance reservation requires job preemption which interrupts job queues and can slow down the system throughput [13]. The Pegasus framework [5] maps a complex and abstract workflow onto distributed computing resources. However, its scheduler is based on available resources and the task execution time must be known. Furthermore, the Pegasus system does not address the resource co-allocation issue for multiple workflow tasks that must be executed concurrently.

2 Resource allocation model

We create a new resource allocation model, called the dynamic resource allocation model, for the computing grid environment. This model allows an MPI application to request more resources than needed; however, the application uses only the needed resources and the remaining redundant resources must be released.

In general, it is difficult to accurately predict job queuing times. Given that it is unknown which computing site will allocate the requested resources first, our new model allows application users to submit a job request and its copies to those sites that are most likely to produce shorter queuing times. Therefore, our model relaxes the requirement that the site with the shortest queuing time be located.

This new model provides dynamic resource allocation by using duplicate job requests. Once the requested resources are allocated and the application's resource requirement is met, the application starts to run. Hence, all resources that are allocated later become redundant, and should be released.

A cross-site parallel execution can also use this model to collect computing resources from multiple computing sites dynamically. When job requests and job duplications of a cross-site execution are used to allocate resources, those earliest allocations that can jointly satisfy the application's resource requirement are used to perform the cross-site execution. The resources that are allocated later become redundant and will be released.

We introduce two boundary conditions to specify the application resource requirement. The two boundary conditions are: a lower-bound resource requirement and an upper-bound resource requirement. The lower-bound resource requirement is the minimum number of computing resources required for a parallel execution to start. In contrast, the upper-bound resource requirement is the maximum number of computing resources that are allowed to participate in the parallel execution, and any additional resource allocation above this value is considered to be redundant.

For a single site execution, the lower-bound and the upper-bound values are equal. However, for a cross-site execution, the lower-bound and the upper-bound can be different, providing a range of numbers of resources which can be collected from different resource allocations. The advantage of this approach is that it allows an application to adapt to changing load conditions by using a range of numbers in the resource specification. The application can use fewer resources when the environment becomes heavily loaded.

2.1 Function specification

Based on our resource allocation model, we developed a model specification that includes five function components. These function specifications can complement the existing MPI standard, when targeting computing grids.

Given an MPI application targeting a computing grid, site selection is the process of creating a list of computing sites to which the application's job request and its duplications are submitted. The site selection algorithm selects computing sites that are most likely to have shorter queuing times for the job request among all sites of the targeting grid. This function does not need to rely on accurate queuing time prediction, instead, it can use a ranking process. System workload information, such as the job queue length, the number of unused resources, and system throughput can be used as the ranking criteria. The higher ranking value, the more likely to have the shortest queuing time. Application users should avoid too many duplications for each job request. This is due to the fact that submitting a duplicate job request and releasing a redundant resource allocation inevitably consumes system resources and causes overhead. If every application has its job request duplicated on every computing site, this flooding may congest the network and decrease the throughput of all computing sites.

The job management component is responsible for managing applications' job requests and duplications. It provides three functions: job submission, job status query, and job cancellation. To achieve portability, adapters are needed, through which job management can interact with different types of host RMSs, such as OpenPBS, LSF, and

loadlever.

In our model a job request and its duplications are processed independently from one another, since they are submitted to different administrative domains. We do not assume that resource management systems can communicate with one another or have interoperability. Therefore, resource assessment is used to determine whether such a resource allocation is needed or redundant for the application. Bound resources are allocated computing resources that are used to satisfy the application resource requirement and participate in the application execution. Given newly acquired resources for a job request, the resource assessment function compares the two boundary conditions with the total allocated resources, which is the sum of the existing bound resources and the newly acquired resources. The state of the newly acquired resources is either *Insufficient*, *Adequate*, or *Redundant*.

Based on our model, a cross-site MPI execution on a computing grid consists of multiple MPI executions, and each execution is independently launched on one computing site. The execution topology, which specifies an MPI execution on one site, contains information about computing resources and process ranks. The global execution topology, which is used to specify a cross-site MPI execution, contains the execution topology of each individual MPI execution on a site and the identifier of that site. Resource binding is then the process of adding the execution topology of an individual MPI execution to the global execution topology.

Resource synchronization is used to coordinate resources allocated on different sites for cross-site MPI executions. This functions can be implemented within the MPI initialization routine, and therefore hidden from application developers and users. Three synchronization methods are specified, including stop-and-wait, hibernation, and resubmission.

In stop-and-wait synchronization, all application processes running on the allocated resources stop executing application code, and wait until the resource requirement is met. This method leaves allocated resources idling, and thus may reduce utilization of computing resources. Hibernation can put an MPI execution into the hibernation state and release its allocated resources. The hibernated execution is woken up when the application resource requirement is met. To reclaim those resources, this method relies on job preemption, which must be supported by the host RMS. In the resubmission method, an MPI execution withdraws from bound resources and is terminated. Then, the job request is resubmitted back to the host RMS.

3 Model simulation

In the previous section, we described the model specifications. Next, we will analyze the performance of our model in terms of job queuing time by using a simulation environment. One of our objectives is to quantify the performance improvement and the impact from using job duplications.

The simulation environment is developed based on CSim [1], which can simulate multiple concurrent events and running processes. In our simulation environment, each computing site of a grid is simulated as a single job queue configured with 64 computing resources. Two commonly used queuing policies are implemented: first-come-first-serve(FCFS) and backfill conservative policy. Each site is assigned a local workload, which contains a list of local job requests. A global workload consists of job requests that are assigned to specific computing sites by the global job scheduler.

Given a global job, the global job scheduler ranks all computing sites based on a specified ranking criterion. It submits the global job to the top-ranked site without considering other global job requests, scheduled or unscheduled. However, this greedy approach may not produce optimal scheduling results. We investigated five ranking criteria as follows:

1. Random value (Random).
2. Queuing length (Qlen), which is the number of job requests queued in a job queue.
3. System workload (Workload), which is the number of jobs being processed in the last time period.
4. Estimated queuing time (Est. QT), which is the waiting time calculated based on estimated run times of all queuing and running jobs.
5. Actual queuing time (Ideal Min), which is calculated based on the actual execution times of all queuing and running jobs. This method can produce the best scheduling result for each global job, and it is only used as the benchmark for comparison.

Figure 1 shows the results from the simulation of 8-site grids when using different numbers of duplications. Each site uses the FCFS queuing policy (shown in the left figure), or the conservative backfill queuing policy (shown in the right figure). Since a conservative backfill queue yields higher throughput than an FCFS queue, we assigned a local workload of 0.6 to each FCFS queue and a local workload of 0.7 to each conservative backfill queue. Three schedulers are compared using 0 to 7 duplications. The y-axis shows the normalized average queuing times for global jobs over the average queuing time of the Ideal Min scheduler. The global workload level is 0.4.

By using job duplication, the Random scheduler achieves the highest performance improvement, as it is compared to other schedulers using different ranking criteria. In addition, only marginal performance differences are found between the Est. QT scheduler without job duplication and the Qlen scheduler with 2-duplication. This result indicates that job duplication can improve a simple scheduler's performance, achieving performance equivalent to that obtained by more sophisticated schedulers.

Job duplication increases workloads on computing sites across a grid. Those computing sites could decrease

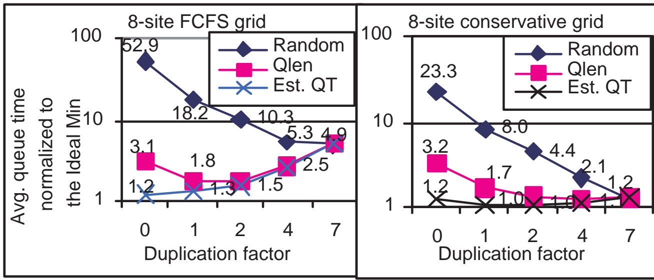

Figure 1. Normalized average queuing times of global jobs using a range of duplication factors

overall system performance when they process redundant job requests. In our simulations, it costs 1 simulation second to terminate one redundant resource allocation. Figure 2 shows the effects of job duplication on local jobs' queuing times during the simulation of an 8-site grid which uses the conservative backfill queuing policy. The queuing times for local jobs across 8 computing sites are represented by vertical lines, with the largest time on the top (Local Max), average in the middle (Local Avg), and the smallest time at the bottom (Local Min). On the left side of Figure 2, all global jobs are scheduled by the Random scheduler. On the right side, all global jobs are scheduled by the Qlen scheduler.

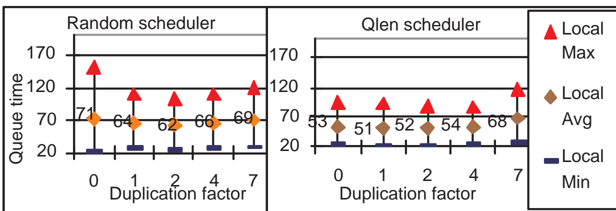

Figure 2. Job duplication's impact on local jobs' queuing times

With a duplication factor of 1 or 2, we can observe reduced queuing times for local jobs on all computing sites. This performance improvement is a result of more balanced workloads across computing sites, which can also be identified by the decrease in Local Max and Local Min. From Figure 2, we observe that flooding, the most greedy approach which has each global job duplicated on all computing sites, can cause overall system performance degradation.

Figure 3 shows the simulation results for stop-and-wait synchronization, described in Section 2.1. In our simulation, each site of the 8-site grid runs the conservative backfill queuing policy, and is given a local workload of 0.6. The global workload level is assumed to be 0.4. We tested the Random, Qlen and Est. QT schedulers for resource co-allocation, while applying different duplication factors for global jobs. The Ideal Min scheduler gives the shortest queuing and synchronization time in co-allocation. Figure 3 shows the co-allocation times for global jobs normalized to the Ideal Min scheduler results. Job duplication significantly improves the performance of all the global job schedulers tested. While the Est. QT scheduler performs better than the other schedulers, there is only a marginal difference for the Qlen scheduler when using 2 duplicate job requests.

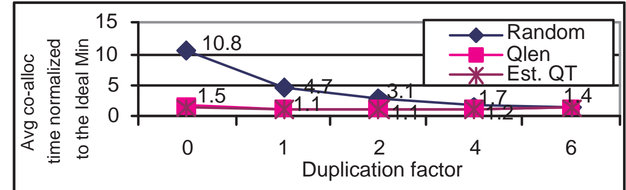

Figure 3. The average co-allocation time normalized to the Ideal Min using stop-and-wait synchronization

We also simulated hibernation, discussed in Section 2.1. Hibernation allows more jobs to be processed, and therefore, the average queuing time for all job requests is reduced. Figure 4 compares the reduction of the average queuing time in the hibernation-enabled environment with that in stop-and-wait synchronization, which has 2 duplications for each global job. In the tests, we simulated 8-site grids and each site uses the conservative queuing policy.

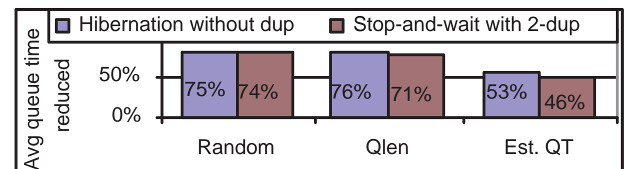

Figure 4. Reduction on average queuing time for all jobs when compared to stop-and-wait synchronization with no duplication

As shown in Figure 4, for each tested scheduler, 2-duplication can provide similar performance improvements as hibernation, when compared to stop-and-wait synchronization with no duplication. Unlike hibernation, our dynamic resource allocation model does not require job preemption for resource synchronization. This is a distinct advantage of our model.

Figure 5 presents the simulation results for heterogeneous grid environments. In these environments, half of computing sites of a grid are configured with 64 computing resources, and the remaining sites are configured with 32 computing resources.

Figure 5 shows the performance improvement of the Qlen and Workload schedulers when the duplication factor is 1. Since each global job has 1 duplicate job request,

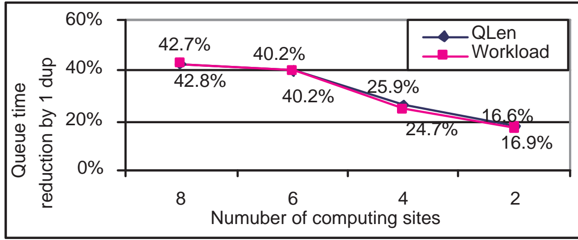

Figure 5. The reduction in average global job queuing time by using 1 duplication on heterogeneous grids

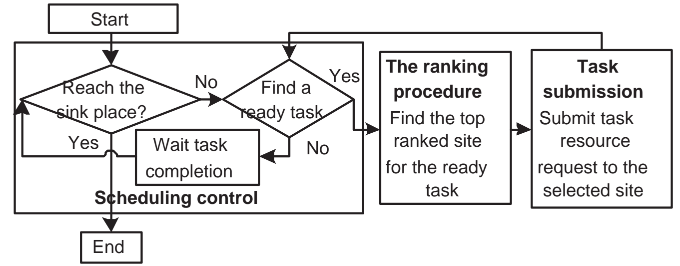

Figure 6. The function diagram for the workflow scheduler

both schedulers can reduce the average queuing time by approximately 42% on the 8-site heterogeneous grid, which is similar to that obtained in the 8-site homogeneous environment. Overall, the simulation results indicate that our dynamic resource allocation model is effective in improving the performance of global job schedulers in both heterogeneous and homogeneous environments.

4 SGR framework design

Based on our resource allocation model, we developed the Schedule-Group-Relay (SGR) framework, which can hide different administrative domains and heterogeneous networks, when launching MPI workflow applications on computing grids. Our SGR framework consists of three service components: a workflow scheduler, task grouping, and message relay. Given an MPI multi-task workflow application, the scheduler conducts task scheduling based on the application's workflow dependency, the task grouping component performs resource binding and synchronization for concurrent tasks, and the message relay component enables inter-task communication.

4.1 Workflow scheduler

Given a workflow specified by a Petri net, the SGR framework scheduler is designed to submit workflow tasks to selected computing sites. We partition the functions of the scheduler into three parts: scheduling control, the ranking procedure, and task submission, as shown in Figure 6. The scheduling control function schedules tasks that are ready. A task is ready only when all tasks on which it depends have finished their executions.

We introduce a parallel design for the workflow scheduler. The workflow scheduler has multiple processes, and each process runs on the headnode or the gateway node of a computing site. Each scheduler process performs local job queries and local job submissions within the same site, and inter-process communication is enabled. Therefore, the scheduler does not require interoperability between RMSs to implement remote job queries and job submission. Since we eliminate the remote operations, the scheduler's performance and scalability can be improved.

4.2 Task grouping

Task grouping is the process of acquiring computing resources from one or more tasks' resource allocations to satisfy the two resource boundary conditions. This process relies on the functions defined by the specifications of the resource assessment, and resource binding and synchronization. We introduced a parallel client-server approach, in which the task grouping server has multiple processes running in parallel and each server process is used to connect a client task running within the same computing site. Each server process must run on the headnode of a computing site, and can communicate with one another. The task grouping server maintains the global execution topology of all client tasks.

4.3 Message Relay

We developed 3-hop message relay to enable message passing between tasks. A parallel client-server approach is used, which is similar to the approach used in task grouping. The message relay server is a parallel program based on MPI, which is independent from the user application, and runs on each computing site's headnode with a public-IP address. Since all server processes run on headnodes with public-IP addresses, they can communicate with one another directly via the standard MPI communication routines.

During 3-hop message relay, each inter-site message travels three hops, using three consecutive point-to-point communication operations. The first hop is from the sender process to the server process running on the headnode within the message sender site. The second hop is from the headnode of the sender site to the headnode of the message receiver site. The final hop is from the headnode to the receiver process within the receiver site. The three hops of the inter-site message passing between two computing resources across different computing sites are shown in Figure 7.

4.4 SGR services integration

Figure 8 gives an overview of the layered structure of the integrated framework. The top layer is the application user

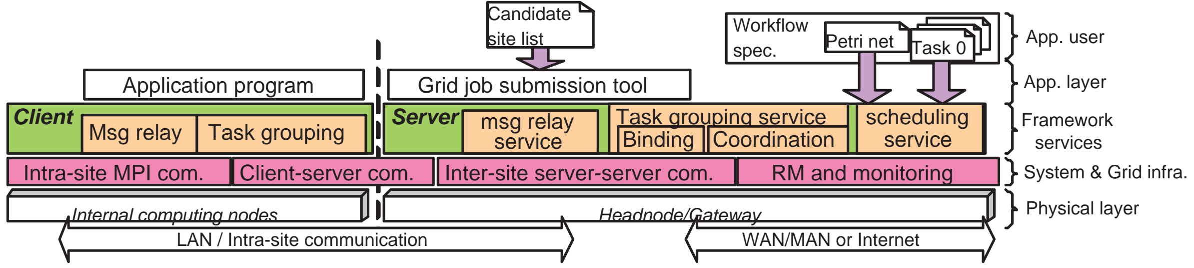

Figure 8. The structure of the integrated framework and its execution environment

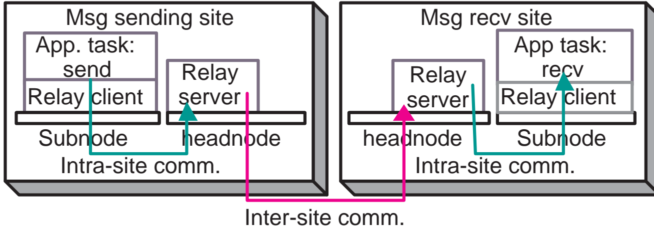

Figure 7. Illustration of 3-hop message relay

layer, which specifies the input information, including candidate computing sites for execution and a workflow application with all tasks' specifications. Below the SGR framework layer is the resource management module interacts with the scheduling service, providing resource management functions for task submission, status query, and job cancellation. Three communication modules are required including the intra-site communication, the client-server communication, and the inter-site server-server communication.

5 Experimental evaluation of the SGR system

We implemented our SGR framework based on Globus-enable computing grids. The SGR system is deployed on a grid, consisting of the Joulian and the Keys clusters. Joulian has 16 PII nodes connected to 100MB/s Ethernet and Keys has 8 PIII nodes connected to 1Gb/s Ethernet, while the two clusters are interconnected to the 10Mb/s campus network.

5.1 Cross-site communication test

We evaluated the performance of the SGR message relay service in a cross-site communication operation by comparing it to a theoretical lower bound on its latency. A lower bound on the latency of sending a message between two computers is:

$$Latency_{lower_bound} = T_{init_overhead} + N/P_{bandwidth} \tag{1}$$

where $T_{init_overhead}$ is the initial overhead for message passing, N is the size of the message in bytes, and $P_{bandwidth}$ is the physical bandwidth of the network connecting the two computers. The performance gap between the relay-based communication latency and its lower bound consists of the underlying system-introduced overhead and the SGR-introduced service overhead. The following section only presents the analysis of cross-site P2P, broadcast and alltoall communication.

Figure 9 shows the observed latency of cross-site point-to-point communication between two subnodes, when sending messages of different sizes. When the message size increases from 0 to 512 bytes, the initial overhead dominates the overall performance. After the message exceeds 512 bytes, the network bandwidth connecting the two clusters becomes the dominating factor. The performance gap varies between 29% and 45%, indicating that the relay-based communication is scalable with respect to the size of messages.

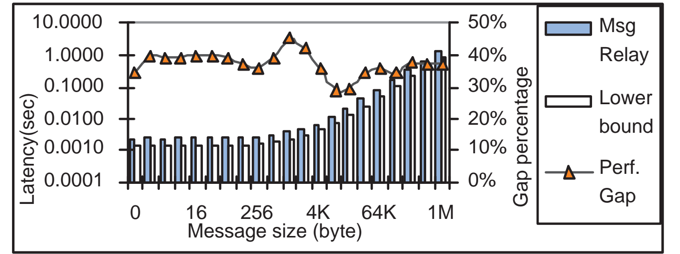

Figure 9. Evaluation of the communication latency of the cross-site point-to-point communication

In Figure 10, we show the latency of broadcasting messages of size $256KB$ to varied numbers of processes across the 2-cluster grid. The performance gap decreases from 37% to 24% when larger topology sizes are tested. This experiment demonstrates that the relay-based broadcast operation is scalable with respect to the size of the execution topology.

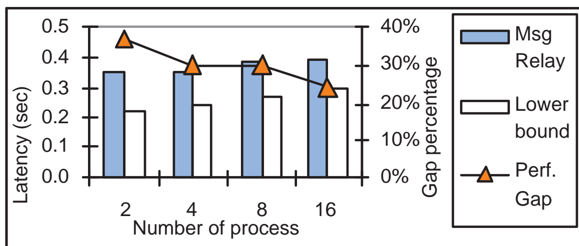

Figure 10. Latency of cross-site broadcasting 256 KB message

Figure 11 presents the performance of the relay-based alltoall communication. The performance gap decreases from 83% to 8%, when we increase the size of message blocks from 1 byte 32KB. This finding demonstrates that the SGR relay system allows the cross-site alltoall operation to be scalable with respect to the size of message.

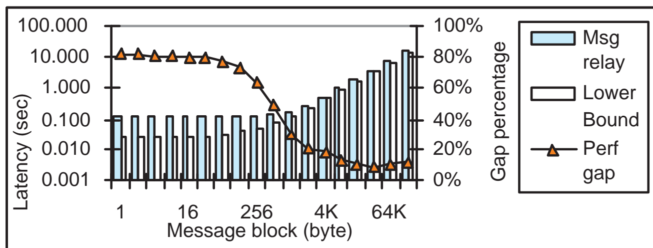

Figure 11. Latency of cross-site MPI alltoall on 16 nodes

5.2 Launching Tomosynthesis on the 2-cluster grid

We also used the SGR system to study the performance of parallelized Tomosynthesis Mammography [6, 15] on the Joulian-Keys grid. Tomosynthesis Mammography is a next generation breast imaging technique that is investigated at Massachusetts General Hospital. Given multiple x-ray projections of an object from different angles, Tomosynthesis reconstructs the 3D volume by using maximum-likelihood estimation. However, this image reconstruction process is very time consuming and requires a large amount of memory. We parallelized Tomosynthesis using data partitioning, and applied MPI to the implementation. Parallelized Tomosynthesis has been deployed on a 32-node cluster for the clinical trials.

Using our SGR system, we can investigate the performance enhancement of Tomosynthesis on a computing grid. The run time result of the image reconstruction is shown in Figure 12. In order to minimize the synchronization time among processors caused by the different computing speed, we fine-tuned the workload assigned to each process running on the two clusters.

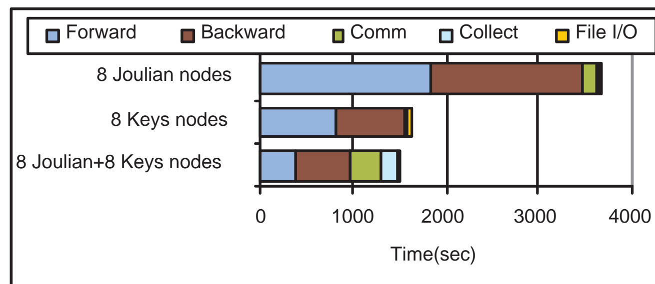

Figure 12. Profiling results of running Tomosynthesis on the Joulian-Keys grid

As shown in Figure 12, we can reduce the computational time (forward and backward) by using more computing resources from both clusters. However, this performance gain is offset by increased inter-cluster communication. Nevertheless, we can still improve performance by at least 8.2% using resources from the two cluster.

5.3 SGR scheduler test

We also evaluated the performance of the SGR scheduling and grouping services on the 2-cluster grid. Each cluster is assigned a synthetic local workload of 0.7 load level, emulating a resource utilization of 70%. A global synthetic workload consists of tasks, and each task is assigned by our SGR scheduler to the 2-cluster grid. We use a global synthetic workload of 0.5 for the Keys cluster, and each global task requires no more than 4 computing nodes. All local jobs and global tasks have an average execution time of 60 seconds.

Two ranking criteria are tested for the scheduler: *Qlen* (the queuing length) and *Workload* (the system workload). Figure 13 shows the average queuing times of the global tasks from using the two schedulers and the results using task duplication.

As shown in figure 13, task duplication outperforms the *Workload* and the *Qlen* schedulers by 29% and 15%, respectively. This performance improvement matches our earlier simulation result showing that the *Qlen* and *Workload* schedulers can achieve a 17% performance improvement in our simulation environment (see Figure 5). The marginal differences between the simulation and the real-world tests could be caused by the longer time used in actual job submission and redundancy invalidation. Overall, our experimental results based on live job submissions validate the new resource allocation model.

6 Conclusions

In this paper, we defined workflow structures to specify the execution process of an MPI application consisting of multiple tasks on a computing grid. We introduced a new re-

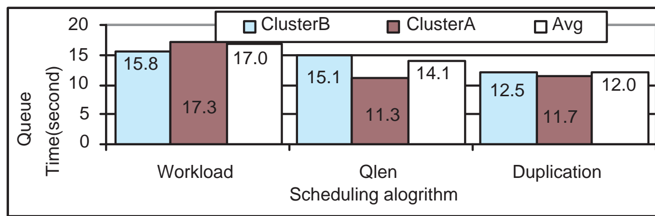

Figure 13. Average queuing time for global jobs submitted to the Joulian-Keys grid

source allocation model that allows applications to perform resource allocation and co-allocation efficiently without relying on sophisticated scheduling algorithms, accurate job execution time prediction, or privileged system support.

Based on our model and its specification, we implemented the SGR system for MPI multi-task workflow applications targeting Globus-enabled computing grids. Our SGR system provides a location-, topology- and administration-transparent computing environment for application developers and users. We used a simulation environment to evaluate our model and its task schedulers, and obtained clear performance improvement by using duplicated resource allocation.

We also validated the simulation results by evaluating resource duplication on a 2-cluster grid. In communication tests across the two clusters, the results indicate that message relay is scalable with respect to the number of processes and the size of messages. Using the SGR system, we demonstrated results of running Tomosynthesis in the 2-cluster grid environment.

References

[1] CSIM19 development toolkit for simulation and modeling. http://www.mesquite.com, 2005.

[2] P. Bradley, K. Misura, and D. Baker. Toward High-Resolution de Novo Structure Prediction for Small Proteins. *Science*, 309(5742):1868–1871, 2005.

[3] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid. In *Proceedings of the Fourth International Conference Exhibition on HPC in the Asia-Pacific Region*, volume 1, pages 283–289, 2000.

[4] S. Choi, K. Park, S. Han, S. Park, O. Kwon, Y. Kim, and H. Park. An NAT-based communication relay scheme for private-IP-enabled mpi over grid environments. In *International Conference on Computational Science*, pages 499–502, 2004.

[5] E. Deelman, G. Singh, M.H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob., and D.S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, 2005.

[6] D.Grant. Tomosynthesis: A three-dimensional radiographic imaging technique. *IEEE Trans. Biomed. Eng.*, 19, 1972.

[7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2), 1997.

[8] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[9] R. Henderson and D. Tweten. Portable batch system: External reference specification. Technical report, NASA, Ames Research Center, 1996.

[10] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *J. Parallel Distrib. Comput.*, 63(5):551–563, 2003.

[11] G. Moore. Cramming more components onto intergrated circuits. *Electronics Magazine*, 1965.

[12] D. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2004.

[13] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of the IPDPS Conference*, pages 127 – 132, 2000.

[14] M. Socolich, S. Lockless, W. Russ, H. Lee, H. Kevin K. Gardner, and R. Ranganathan. Evolutionary information for specifying a protein fold. *Nature*, 437, 2005, sept.

[15] J. Zhang, W. Meleis, D. Kaeli, and T. Wu. Acceleration of maximum likelihood estimation for tomosynthesis mammography. In *Proceedings of the ICPADS*, 2006.

[16] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proc. Workshop on Cluster Computing*, 1992.