

SIMULATION OF ADAPTIVE APPLICATIONS IN HETEROGENEOUS COMPUTING ENVIRONMENTS *

Bo Hong and Viktor K. Prasanna

Department of Electrical Engineering

University of Southern California

Los Angeles, CA 90089-2562

{bohong, prasanna}@usc.edu

Abstract

Designing adaptive applications for dynamic heterogeneous computing environments has received a lot of attention recently. In this paper, we propose a modular and extensible simulator that can be used to evaluate the performance of applications in dynamic heterogeneous environments. This simulator has built-in support for evaluating the performance of adaptive applications in such environments. The dynamic behavior of these resources can be described through random variables or trace files. The

*A preliminary version of this paper appeared in the *Proceedings of 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2002)*. Supported in part by the National Science Foundation under award No. ACI-0204046 and an equipment grant from Intel Corporation.

interactions between the applications and the resources are described through a closed adaptation loop. The simulator provides the user with flexibility in configuring the applications and resources and evaluating various adaptation algorithms. We illustrate the features of the simulator by simulating algorithms for distributed matrix multiplication in various computing environments.

Keywords: simulation, adaptive, heterogeneous computation

1. Introduction

High performance computing platforms have evolved from the Cray style vector supercomputers to clusters of massively parallel computers, and to Beowulf clusters consisting of commercial off-the-shelf hardware and software [5]. Recently, the Computational Grid [11] has become an attractive platform for high performance computing. With this evolving trend, heterogeneity and run-time performance variations in the resources have become two important characteristics of high performance computing platforms.

Application design for distributed and heterogeneous computing environments [6, 10, 14, 16] has been an active research area. With the rapid development of the Grid, designing adaptive applications for dynamic heterogeneous environments has received a lot of attention [11]. The singular feature of an *adaptive application* is that it re-arranges the executions and/or re-allocates the resources during run time, according to changes in the computing environment. Although the performance of applications can be evaluated by executing the applications on actual resources, this approach is often limited either because the required resources are not available, or the experiments are difficult to repeat. To evaluate the efficiency and understand the performance bottleneck of adaptive applications in a dynamic heterogeneous environment such as the Grid, we propose a modular and extensible simulator for evaluating

the performance of applications in such environments. Several features make this simulator particularly suitable for the simulation of adaptive applications on heterogeneous platforms:

1. Our simulator is designed to describe performance of the resources in dynamic heterogeneous environments. The performance of the resources can change during run time. In our simulator, the performance of the resources can be described through random variables or trace files. The resources may fail, or a new resource may become available during the execution.
2. The interactions between the applications and the resources are described through a closed adaptation loop in which (1) the performance variations of the resources are captured, (2) the computation plan (resource selection and task scheduling) is adapted at run time, and (3) tasks are assigned to the resources according to the computation plan to optimize certain predetermined objectives (e.g. the makespan). This description is an abstraction of various state-of-the-art design methods for adaptive applications in dynamic heterogeneous environments [13, 15]. With this model of the interactions, the simulator can easily be configured to estimate the performance of various applications and adaptive scheduling algorithms.
3. The design of the simulator is *modular* and *extensible*. With the modular structure of the simulator, new resources may be introduced into the system without extensive programming effort. The extensible design allows the user to plug in various applications and evaluate their performance.
4. The control and coordination overheads have a significant impact on the performance of an application. Our simulator estimates not only the computation and communication costs for executing an application, but also the control and coordination overheads incurred in the computation, thereby providing a useful tool to study the impact of these overheads.

The use of the simulator is illustrated through a set of simulations for distributed matrix multiplication in dynamic heterogeneous environments. Note that static as well as homogeneous computing environments are special cases of the environment considered by our simulator; simulations for these environments can also be performed using our simulator.

The rest of the paper is organized as follows: Related work is discussed in Section 2. Section 3 describes the models used in our simulator. Section 4 discusses the architecture and implementation of the simulator. Section 5 illustrates the use of the simulator. Section 6 concludes the paper.

2. Related Work

Several tools are currently available to simulate applications in a heterogeneous computing environment.

Bricks [17] is built to simulate a global computing system with client-server structure, a system in which multiple servers running scientific libraries and packages provide remote access to the clients. Clients send independent computation requests to the servers. Bricks focuses on the overall performance of the system.

Simgrid [9] is a C language based toolkit for the simulation of task scheduling in a distributed environment. In Simgrid, resources are assumed to be time-shared with other applications/users. The performance of a resource can be a constant number or specified through a trace file containing real run time load variations. Tasks are described (by the user) using a Directed Acyclic Graph (DAG). The task scheduling algorithm is defined by the Simgrid user. Thus, Simgrid offers the flexibility to simulate various applications and scheduling algorithms.

GridSim [8] is a Java-based discrete-event Grid simulation toolkit. It supports modelling and simula-

tion of heterogeneous Grid resources, Grid users and applications. It provides primitives for the creation of tasks and the mapping of tasks to resources. Designed for a market-like Grid computing environment, GridSim can simulate multiple competing Grid users, applications, and schedulers, each with their own objectives and policies.

The proposed simulator differs from the above simulators in the following ways:

We consider environments different from those simulated in Bricks and GridSim. Bricks simulates a multi-client multi-server structure where each client request is fulfilled by exactly one server. Compared to Bricks, our simulator focuses on the execution of a single application that needs to coordinate various resources in the environments. GridSim is based on a market-driven economic model of the Grid environment. It simulates a market-like environment where multiple Grid users compete for the available resources. However, in a dynamic and heterogeneous environment such as the Grid, it is difficult, if not impossible, to know the objective, policy, work load and resource requirement of each individual user. Compared with GridSim, our simulator focuses on the performance of a single application in such an environment. The impact of other users are abstracted through run time performance variation of the resources, which can be described through trace files or using random variables with pre-defined distributions.

Although Simgrid also simulates single structured applications in heterogeneous environments, our simulator differs from Simgrid as our simulator has built-in support for simulating adaptive applications in such environments. Additional programming effort is required in order to simulate run-time adaptive applications using Simgrid. Our simulator models the control and coordination overheads incurred in the computations, which is not considered in Simgrid.

3. Application and Resource Models

This section discusses the application and resource models used by our simulator. These models are used to describe applications in a dynamic heterogeneous computing environment, where the status of the application and the resources change during run-time. Note that the simulator itself does not adapt to dynamic environments. Instead, it facilitates the simulation of adaptive scheduling algorithms in dynamic environments.

3.1 Application Model

Unlike instruction level simulators, we assume that a problem consists of a set of tasks and the simulations are performed at the task level. In our simulator, the static view of an application consists of two components: the *problem* and the *scheduling algorithm* used to assign the tasks to the resources. In this paper, a *computation plan* is the result generated by the scheduling algorithm. This plan includes resource selection and task scheduling.

The application model describes the interactions among the problem, the scheduling algorithm and the resources when executing an application. The dynamic behavior of an adaptive application is modeled as follows:

Given a problem, an initial computation plan is determined. The tasks are then assigned to the resources according to this plan. While the resources execute these tasks, the status of the tasks and the resources are being tracked by the scheduling algorithm. By using this information, the scheduling algorithm adapts the computation plan, and, if necessary, re-partitions the problem into a different set of tasks. This updated computation plan is again assigned to the resources during run time. The above process is repeated till all the tasks in the problem are completed. This adaptive application model is one

of the basic features that makes our simulator different from other simulators.

Besides adaptive applications, many studies focus on static scheduling problems, where the task scheduling is determined at compile time with *a priori* knowledge of the performance of the resources. These problems are simplified versions of the adaptive applications that our simulator considers. For these static scenarios, we can configure our simulator to bypass run-time adaptations and perform task scheduling only at the beginning of the computation.

A problem in our simulator consists of a set of tasks. DAGs are used as internal representations of the problem. These DAGs are generated by a script tool, which is designed for the user to describe the problem. The script tool is discussed in Section 5.1. In the architecture of the simulator (discussed in Section 4), a programming interface is provided for the user to input the scheduling algorithm.

3.2 Resource Model

A typical heterogeneous computing platform consists of various compute nodes connected by a network. In our simulator, we model both the compute nodes and the network links as resources. In the current implementation of the simulator, we assume that the performance of a compute node is characterized by its computing power in MFLOPS and the performance of a network link is characterized by its latency and bandwidth.

Our simulator describes these performance characteristics through (1) constants, (2) random variables with pre-defined distributions, (3) trace files, or (4) at a system level of abstraction which is based on random variables. The distribution of the random variables and the parameters of the distributions are configured by the user. Trace files contain a series of time stamped values, indicating the performance of the resource at each time instance. Trace files can either be synthesized or recorded from real systems

(several tools are available for CPU and network performance monitoring, e.g. [7]). Besides run time performance variations, resources may fail and new resources may become available during the course of a computation. This function can be used to simulate hardware related events such as power shut down.

At the system level of resource abstraction, we model the computing power of compute node N_i ($1 \leq i \leq M_c$ where M_c is the number of compute nodes) as a normally distributed random variable with mean value $P(i)$ and standard deviation $\sigma_p(i)$. The network bandwidth of network link L_j ($1 \leq j \leq M_n$ where M_n is the number of network links) is also modeled as a normally distributed random variable with mean value $B(j)$ and standard deviation $\sigma_b(j)$. Let $\bar{P} = \frac{1}{M_c} \sum_{i=1}^{M_c} P(i)$; $\bar{B} = \frac{1}{M_n} \sum_{i=1}^{M_n} B(i)$. Further, we define $\sigma P = \sqrt{\frac{1}{M_c-1} \sum_{i=1}^{M_c} (P(i) - \bar{P})^2}$ and $\sigma B = \sqrt{\frac{1}{M_n-1} \sum_{j=1}^{M_n} (B(j) - \bar{B})^2}$. σP and σB represent the system heterogeneities in computing power and network bandwidth. The systems are classified into four categories based on the values of σP and σB : high σP and high σB , high σP and low σB , low σP and high σB , low σP and low σB . These are abbreviated as ‘HH’, ‘HL’, ‘LH’ and ‘LL’ respectively. This abstraction is similar to the approach used in [2] that describes the heterogeneities of compute nodes and tasks. Given the specifications for M_c , M_n , \bar{P} , \bar{B} , σP , σB , $\sigma_p(i)$, and $\sigma_b(j)$, the simulator automatically models the resources that satisfy these specifications. This system level abstraction is particularly useful when evaluating an application for a large set of scenarios, which is not provided by other simulators.

Another important aspect of the resources is the topology of the interconnection network. Assuming a fully connected network would minimize the difficulty of implementing the simulator. However, the major drawback of a fully connected model is that it may not accurately describe the situation in actual heterogeneous computing platforms. In an actual heterogeneous computing platform, a network link may be shared by several compute nodes. Our simulator does not impose any topology on the network.

The user configures the network topology by designating the set of network links to which each node can connect. This flexibility makes it possible for our simulator to simulate a variety of computing systems.

Each compute node has its own local scheduling policy. When configuring the resources, the user specifies the local scheduling policy for each node. At this time, possible choices provided by our simulator are Round Robin, First In First Out, and Shortest Job First. When multiple tasks are ready to run on a single compute node, the designated local scheduling algorithm is used to determine the order of their execution.

4. Architecture and Implementation of the Simulator

4.1 Architecture of the Simulator

The simulator consists of three *entities*: Scheduler, Global information directory, and Computing and network resources. The concept of entity is discussed in Sec. 4.2. Information about the progress of the computation and the dynamic performance of the resources are maintained in the global information directory. The scheduler is responsible for adapting the resource selection and task scheduling. The function of the scheduler is executed by one of the compute nodes. We call this node the scheduler node.

The architecture of the simulator is shown in fig. 1. The simulation of an application is performed as follows: First, the scheduler receives the application (internally represented by a DAG) and determines an initial computation plan. The tasks are then assigned to the computing resources. Each resource maintains a local task queue. When multiple tasks are ready to run, the execution of these tasks is simulated according to the local scheduling policy (set by the user). Upon completion of the execution of one task, the task status is updated in the global information directory. During the simulation, new resources may become available, some resources may fail, and the performance of the resources may

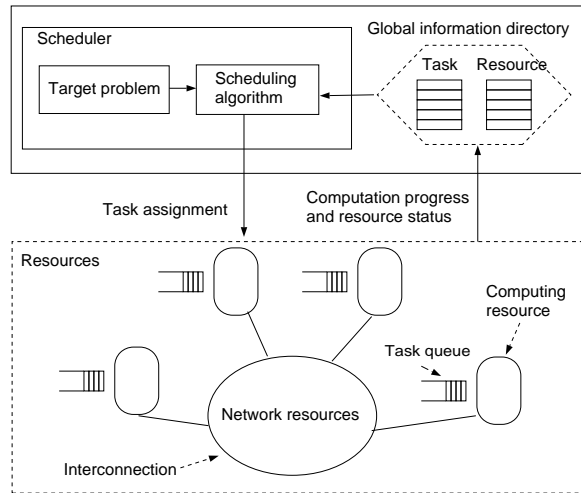


Figure 1. Architecture of the simulator

vary. These performance variations are also updated in the global information directory, either at fixed intervals of time or driven by events (e.g. when a task completes). Both the scheduling algorithm and the time instances of adaptation are configured by the user and the scheduler is invoked accordingly. By querying the global information directory, the scheduler adapts the computation plan, and adapts the DAG representation of the problem or re-selects the resources if requested by the adaptation algorithm. This updated computation plan is again assigned to the computing and network resources. The above process is repeated till all the tasks complete execution.

This architecture describes the interactions among the application and the resources. Various components of the simulator are configurable by the user. For example, the scheduling algorithm, the dynamic behavior of the resources, the time interval between generation of tasks, the method for partitioning the problem, etc. This flexibility in configuration makes it possible to simulate various applications and dynamic heterogeneous computing environments.

When executing distributed applications in actual environments, two types of overheads are incurred:

(1) *Control overheads*: Extra computing cycles to discover and select the resources and determine the

computation plan. (2) *Coordination overheads*: Extra cycles spent to transfer control messages among the compute nodes (e.g. data transfer request that specifies what data to transfer). These overheads are addressed by our simulator.

The cost of determining the computation plan is simulated by considering the complexity of the scheduling algorithm in terms of the number of floating/fixed point operations. The cost of each execution of the scheduling algorithm is determined by its complexity and the computing power of the scheduler node. The user also has the option to specify the execution time directly. If the computation plan is configured to be adapted when a compute node completes a task, multiple nodes may complete their tasks and send the adaptation requests concurrently. These requests will be processed in a FIFO manner. Therefore, the actual cost of determining the computation plan is not fixed during the simulation. Instead, it is determined by the cost per execution of the scheduling algorithm and the status of the system, i.e. the number of concurrent requests.

Our simulator has explicit entities for the scheduler, the global information directory and the resources. Hence the simulator is able to trace each individual message transfer among these entities, thereby simulating the related coordination overheads. For example, the cost of resource discovery is simulated through the query/answer messages transferred between the scheduler and the global information directory. Each occurrence of these messages is treated in the same way as a data transfer is simulated. The user may specify the size of each message. The cost of transferring a message is then determined by the bandwidth and the latency of the network link at the time of transfer. The user may also directly specify the transfer time required for a message.

4.2 Implementation of the Simulator

The simulator has been implemented using the PARSEC (PARAllel Simulation Environment for Complex systems) language [3]. PARSEC is a C-based discrete-event simulation language. PARSEC programs consist of entities, which represent real world objects. These entities interact with one another by exchanging messages. Unlike typical graphical user interface (GUI) based simulation tools in which the objects often have a pre-defined and fixed behavior, PARSEC utilizes the C programming language to describe the complex behavior of the components and is therefore capable of simulating systems with complex behavior.

Our simulator implements entities for the scheduler, global information directory, compute nodes and network links. The global information directory receives messages from the resources, maintains information about the performance of the resources and the progress of the computation, and answers queries from the scheduler; the resources receives tasks from the scheduler, executes the tasks according to its local scheduling policy, and reports performance data updates to the global information directory; and the whole computation is control and adapted by the scheduler. A ‘driver’ procedure initializes these entities, sets up the occurrence of events such as resource failure and creation, and starts the simulation.

Our simulator provides a set of commands to collect the key performance statistics (e.g. the start and finish time of each task and the overall execution time of an application) of a simulation from the global information directory. Our simulator is also able to analyze the activities of entities for a simulation and determine possible performance bottlenecks. For example, when studying the impact of control and coordination overheads on the performance, our simulator can detect whether multiple resources send requests to the scheduler simultaneously and cause the scheduler to respond at a slower rate. Our simulator can also trace the idle time of all the compute nodes and find the bottleneck node(s) that causes

the idling of other nodes. In addition to this information, the user may write his own procedure to obtain a desired set of statistics at a particular level of details for the simulations performed.

In order to validate the proposed simulator, we have conducted simulations for various applications and scheduling algorithms. The performance of these applications was obtained analytically and then compared against the results from the simulations. By ‘analytically’, we mean tracing the execution of the application through hand calculation. These experimental results showed that the results generated by the simulator are close to the expected results.

Table 1 shows one such example. We studied a first-come-first-serve algorithm that schedules a set of independent tasks onto a set of heterogeneous compute nodes. In this algorithm, a task is assigned to a compute node whenever the compute node becomes free. The simulations were configured such that the tasks were uniformly distributed between 40 and 160 giga-floating operations; and the compute power of the nodes was uniformly distributed between 0.3 and 1.2 GFLOPS. In table 1, the overall computation time generated by the simulator is compared against the analytically calculated time, and the error is reported. We can see that the difference between the simulated and the expected results is negligible. Ideally, these two results should match. The slight difference is due to the rounding errors: time is represented as integer values in PARSEC. For example, when the execution time of a task is 200.8, our simulator considers it as 201. When a higher precision in the results is desired, the simulation error can be reduced by increasing the time precision of the simulator, at the cost of slower simulation speed.

5. Using the Simulator

In this section, we first describe how to configure our simulator to simulate adaptive applications in a dynamic heterogeneous environment. Then simulations for an adaptive matrix multiplication algorithm

Table 1. Validation of the simulator

	Number of tasks = 1000	Number of tasks = 2000	Number of tasks = 5000
10 nodes	0.17%	0.10%	0.08%
20 nodes	0.61%	0.29%	0.14%
40 nodes	1.1%	0.43%	0.11%

are provided for illustration purpose.

5.1 Configuring the Simulator

The simulator can be configured for a specific simulation by performing the following:

1. *Definition of the resources.* Resources in the system should be created and registered in the global information directory. For each compute node, the user needs to specify its local scheduling policy, whether the computation and the communication can be overlapped and its computing power. For a network link, the user needs to specify its latency and bandwidth. The user also needs to specify which compute nodes this network link connects. Instead of specifying the parameters for each individual resource as in the above example, the user may also utilize the system level abstraction to define the resources as stated in Sec. 3.2.
2. *Problem description.* Our simulator provides a script tool for the user to describe the problem and automatically generate the DAG representation for the same. This script tool is written in Perl. It requires two user defined input files: the problem definition file and the parameter file. In the problem definition file, the user describes the problem in Perl language. The structure of the problem is indicated by using the data transfer primitive `GetData(data size, preceding`

Table 2. Configuration options available to the user

Parameter	Options
Problem	Specified by the user (using the script tool)
Scheduling algorithm	Provided by the user
Description of resource performance	Constants, random variables, trace files, system level abstraction
Local scheduling policy of the compute nodes	FIFO, Round Robin, Shortest Job First
I/O of compute nodes	Blocking or non-blocking
Network topology	Specified by the user
Time instances of resource performance data updates	Periodic (period defined by user) or event driven
Execution instances of the scheduling algorithm	Periodic (period specified by user) or event driven
Cost of executing the scheduling algorithm	Function of cost per execution and number of concurrent executions
Cost of transferring each control message	User specifies the message size or the transfer time

task) and the computation primitive `Compute(work load)`. Values of the parameters used in the problem definition file are provided by the parameter file. The simulator calls the script tool and obtains the DAG representation of the problem. The abstraction of applications in to Perl script is similar to that used in [4].

3. *Scheduling algorithm description.* Finally we need to plug in the scheduling algorithms. This is achieved by the programming interface provided in the scheduler entity. The scheduling algorithm obtains the status of the resources and tasks through function calls `getResourceList()` and `getTaskList()`. Given this information, the user has the flexibility to plug in his own scheduling algorithm and assign the tasks by using the function call `AssignTask(...)`.

These three steps configure the simulator to describe the behavior of the resources, the problem and the scheduling algorithm. Besides these configurations, the user can also control the following aspects of the operation of the simulator: (1) the frequency at which the scheduler invokes the scheduling algorithm; (2) the cost of executing the scheduling algorithm (either the execution time or the complexity in the number of floating/fixed point operations), and (3) the cost of transferring a control message (either

the transfer time or the message size). Table 2 lists the configuration options available to the user. Because the simulator is configured using pre-defined function calls, the configuration can be performed efficiently without elaborate programming effort.

5.2 Example Uses of the Simulator

To demonstrate that our simulator can be used to evaluate the performance of adaptive algorithms in dynamic environments, we have used the simulator to study an adaptive algorithm for distributed matrix multiplication [12]. In this algorithm, the nodes in the system are either data servers that store one or more of the matrices, or compute nodes that can be assigned particular sub-tasks in the overall computation. The basic task unit of matrix multiplication is a $b \times b$ block of data, where $1 < b < N$ and input matrices are of size $N \times N$. Each task unit will be executed by a chosen compute node, and the required source data will be provided by some data server. Tasks are adaptively assigned to the compute nodes during run time, based on the current status of the system.

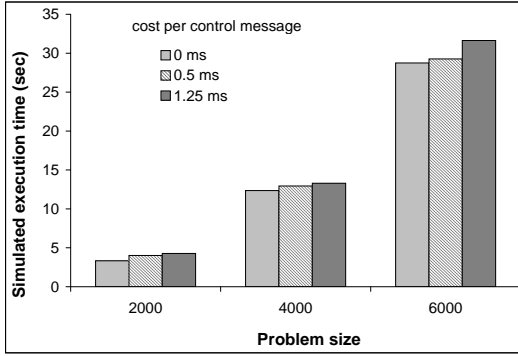
In the simulated system, the performance of the compute nodes were specified through a set of normally distributed random variables. The compute nodes were configured to update their performance information to the global information directory after the completion of each task. The local scheduling policy for all the compute nodes is FIFO. The execution of the scheduling algorithm is event driven. The MCT heuristic in [6] is used to schedule the tasks. Block size b is also dynamically adapted during run time. The adaptation of b results in the modification of the DAG representation of matrix multiplication during run time.

5.2.1 Impact of Control and Coordination Overheads

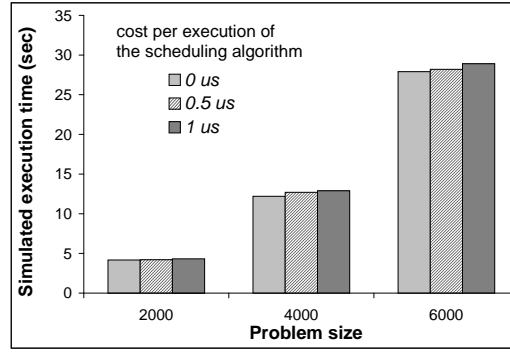
We first used our simulator to study the impact of control and coordination overheads on the overall execution time. A system with 64 heterogeneous nodes was simulated. In the simulations, the average sustained network bandwidth is 100 Mb/s and the average sustained computing power of the compute nodes is 500 MFLOPS . The results shown in fig. 2 (a) were obtained by varying the cost for transferring control messages. These results indicate that only a small portion of the overall execution time was spent transferring the control messages. (12 KB data can be transferred over a 100 Mb/s in 1ms . Considering the typical network latency, 1.25ms is a reasonable estimate for transferring a control message.) fig. 2 (b) illustrates the impact of executing the scheduling algorithm on the overall execution time. The results show that executing the scheduling algorithm has little impact on the overall performance. (Results in [6] showed that the execution of the MCT is less than $1\mu\text{s}$ on a Pentium II 400MHz computer, for a typical problem with 512 tasks and 16 compute nodes.) Combining the simulation results in figs. 2 (a) and (b), we can conclude that the performance bottleneck of this approach is not in the control and coordination overheads. Hence, in order to improve the performance of this approach, we need to focus on the efficiency of the scheduling algorithm and the resource allocation.

5.2.2 Performance of the Data Server

For this matrix multiplication algorithm, we also simulated the performance of the data server. The four 16-node systems listed in table 3 is simulated. The problem size is 6000×6000 . We collected the response time of the data server, which is the time interval between (1) a compute node sends the request and (2) the data transfer completes. The average response time of the data server was reported in table 4. As can be seen from the results, although the network bandwidth in System 1 is 1/10 of that in System 2, the average response time of the data server in System 1 was 33.7 times larger than that



(a) Impact of coordinationoverheads



(b) Impact of control overheads

Figure 2. Impact of control and coordination overheads on the execution time of matrix multiplication

in System 2. This is because the data server in System 1 processes the requests at a lower rate, which not only affects the transfer time of every single data transfer, but also introduces congestions into the system. Similar results can be observed between System 3 and 4. When comparing System 2 and 4, we can see that the compute nodes in System 4 have higher compute power. Because the computes nodes in System 4 generates data transfer requests at a higher rate than those in System 2, the average response time of the data server increased from 1.31 sec to 1.93 sec. These simulation results indicate that the data server may cause performance bottleneck. Hence it is necessary to study techniques that can improve the performance of the data server.

Table 3. Heterogeneities of the simulated systems. \bar{P} and σP are in MFLOPS. \bar{B} and σB are in Mb/s

Systems	\bar{P}	σP	\bar{B}	σB
16-node HL(Sys 1)	300	50	100	12
16-node HL(Sys 2)	300	50	1000	100
16-node HL(Sys 3)	600	100	100	12
16-node HL(Sys 4)	600	100	1000	100

Table 4. Performance of the data server and its impact on the overall execution time

System	Sys 1	Sys 2	Sys 3	Sys 4
Average response time of the data server (sec)	44.2	1.31	43.6	1.93
Execution time (sec)	574.5	107.0	567.3	67.3

5.2.3 Impact of Different System Heterogeneities

The next set of simulations were conducted on four dynamic heterogeneous systems. Each system has 64 compute nodes. Heterogeneities of these systems are configured as HH, HL, LH, and LL, respectively. Resources in these systems were specified through the system level abstraction as stated in Sec. 3.2. Specifications for the HH system are as follows: $M_c = 64$, $M_n = 150$, $\bar{P} = 440$ MFLOPS, $\bar{B} = 100$ Mb/s, $\sigma P = 84$ MFLOPS, $\sigma B = 29$ MFLOPS, $\sigma_p(i) = 0.15P(i)$, and $\sigma_b(j) = 0.12B(i)$. Specifications for the other systems are similarly defined. Although these four systems have different heterogeneities, they were configured to have the same overall computing power (the sum of the computing power of all nodes in a system). The results in fig. 3 show small performance variations for these systems. This verifies that this algorithm is robust to run-time performance variations and system heterogeneities.

6. Conclusion

In this paper, we have proposed a simulator for evaluating the performance of adaptive applications in dynamic heterogeneous environments. A potential application of the simulator is to evaluate computation schemes in the Grid environment, since the Grid environment is dynamic. The modular design of the simulator facilitates the evaluation of various adaptive algorithms for many scenarios of interest to the community. The application model in the simulator is abstracted from various state-of-the-art design methods for adaptive applications. Our preliminary experiments with the simulator show results as ex-

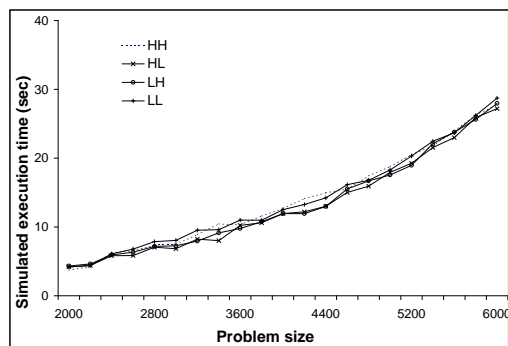


Figure 3. Simulation of matrix multiplication in four dynamic heterogeneous systems (HH, HL, LH, and LL)

pected. However, we are further validating the simulator using actual experiments in Grid environments.

Although the simulator is useful for evaluating adaptive applications and algorithms in a heterogeneous environment, several limitations need to be addressed. Currently, the simulator assumes the existence of a centralized controller that is enforced by the Global Information Directory and the scheduler. However, as pointed out in [11], a centralized controller may not be available in many distributed systems. We are investigating the applicability of our simulator for distributed self-scheduling applications. In our simulator, the performance of a compute node is modeled by its computing power only. We are expanding this model to other resources such as the available memory of the nodes. The simulator currently models the network links by using its latency and bandwidth and uses a simple model to capture data movement overheads. Further work is needed to model the behavior of these resources at a finer level. Our simulator uses DAG as the internal representation of the applications, which can be restrictive. Currently, our simulator does not support the scenario where multiple applications arrive at the system during run time. These resource and application models need to be expanded to overcome these limitations.

References

- [1] ADVISOR Project. <http://advisor.usc.edu>.
- [2] S. Ali, H.J. Siegel, M. Maheswaran, and D. Hensgen. Task Execution Time Modeling for Heterogeneous Computing Systems. *HCW 2000*, pages 185–199, 2000.
- [3] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H. Song. PARSEC: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, 31(10):77–85, 1998.
- [4] Z. K. Baker and V. K. Prasanna. Performance Modeling and Interpretive Simulation of PIM Architectures and Applications. *Proc. of Euro-Par 2002*, August 2002.
- [5] G. Bell and J. Gray. What’s Next in High-Performance Computing? *Communications of the ACM*, 45(2), 2002.
- [6] T. D. Braun, H. J. Siegel, and N. Beck. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.
- [7] D. G. Brian. NetLogger: A Toolkit for Distributed System Performance Analysis. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.
- [8] R. Buyya and M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, Wiley Press, 14(13-15), November - December 2002.

- [9] H. Casanova. SimGrid: A Toolkit for the Simulation of Application Scheduling. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, May 2001.
- [10] G. Chen, F. C. M. Lau, and Cho-Li Wang. Building a Scalable Web Server with Global Object Space Support on Heterogeneous Clusters. *Cluster Computing 2001*, pages 313–320, 2001.
- [11] I. Foster and C. Kesselman (editors). The Grid: Blueprint for a New Computing Infrastructure. *Morgan Kaufmann*, 1999.
- [12] B. Hong and V. K. Prasanna. Adaptive Matrix Multiplication in Heterogeneous Environments. *2002 International Conference on Parallel and Distributed Systems*, December 2002.
- [13] K. Kennedy, M. Mazina, J. M. Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kesselman, R. Aydt, D. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski. Toward a Framework for Preparing and Executing Adaptive Grid Programs. *IPDPS 2002*, April 2002.
- [14] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.
- [15] L. Rauchwerger, N. Amato, and J. Torrellas. SmartApps: An Application Centric Approach to High Performance Computing. *Proc. of the 13th Annual Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2000.

- [16] José D. P. Rolim, editor. *Parallel and Distributed Processing, 15 IPDPS 2000 Workshops, Cancun, Mexico, May 1-5, 2000, Proceedings*, volume 1800 of *Lecture Notes in Computer Science*. Springer, 2000.
- [17] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. *HPDC '99*, 1999.