A Modular and Extensible Simulator for Performance Evaluation of Adaptive Applications in Heterogeneous Computing Environments*

Bo Hong and Viktor K. Prasanna
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
{bohong, prasanna}@usc.edu

Abstract

In this paper, we propose a modular and extensible simulator that can be used to evaluate the performance of applications in a dynamic heterogeneous environment. Our simulator has built-in support for evaluating the performance of adaptive applications in dynamic heterogeneous environments. 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 and LU decomposition in various computing environments.

1 Introduction

Studying the performance of applications has been an active research area in distributed and, recently, heterogeneous computing environments [5, 10, 12]. With the rapid development of the Grid, designing adaptive applications for dynamic heterogeneous environments has received a lot of attention. 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. The resources may fail, or a new resource may become available during the execution. In our simulator, the performance of the resources can be described through random variables, facilitating statistical study of the performance of adaptive applications in dynamic heterogeneous environments.

2. The interactions between the applications and the resources is described through a closed adaptation loop in which (1) the performance variations of the resources is 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 [9, 11]. Using this description 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

*Supported by the US DARPA Data Intensive Systems Program under contract F33615-99-1-1483 monitored by Wright Patterson Air Force Base and in part by the National Science Foundation under award No. 99000613.

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 and LU decomposition 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 [14] 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 [8] 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 DAG (Directed Acyclic Graph). The task scheduling algorithm is defined by the Simgrid user. Thus, Simgrid offers the flexibility to simulate various applications and scheduling algorithms.

GridSim [7] is a Java-based discrete-event Grid simulation toolkit. It supports modeling and simulation 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. Each Grid user in GridSim has a private resource broker. This private resource broker performs discovery, selection, and utilization of distributed resources for an individual user and is targeted to optimize for the requirements and objectives of its owner. 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. Besides, Simgrid does not support run-time re-partitioning of the problem, which is often used by adaptive applications to adapt the computations during execution time. Further, 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.

3.1 Application Model

Different from 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 solve this problem. In this paper, a *computation plan* is defined as the result of the execution of 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.

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. The scheduling algorithm is not fixed in our simulator. Instead, 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 predefined 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. [6]).

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 $\overline{P} = \frac{1}{M_c}\sum_{i=1}^{M_c} P(i)$; $\overline{B} = \frac{1}{M_n}\sum_{i=1}^{M_n} B(i)$. Further, we define $\sigma P = \sqrt{\frac{1}{M_c-1}\sum_{i=1}^{M_c}(P(i)-\overline{P})^2}$ and $\sigma B = \sqrt{\frac{1}{M_n-1}\sum_{j=1}^{M_n}(B(j)-\overline{B})^2}$. 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, $\overline{P}$, $\overline{B}$, σP, σB, $\sigma_p(i)$, and $\sigma_b(j)$, the simulator automatically models the resources that satisfy these specifications. This system level abstraction is particular useful when evaluating an application for a large set of scenarios.

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. Many non-hardware-related events can also be described using this method. For example, data replication [13] is a very useful technique for distributed computation. The behavior of data replication can be simulated through the creation of a new resource, to which data transfer tasks can be assigned.

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.

While a data transfer is being conducted on the network, two compute nodes are actively involved (the sender and the receiver). The I/O processing of these compute nodes may work in one of the two modes: (1) blocking mode, the compute nodes are suspended while the network transfer is being conducted, and (2) non-blocking mode, the compute nodes can overlap the computation and communication. When working in the non-blocking mode, a compute node may be affected and lose a small fraction of its com-

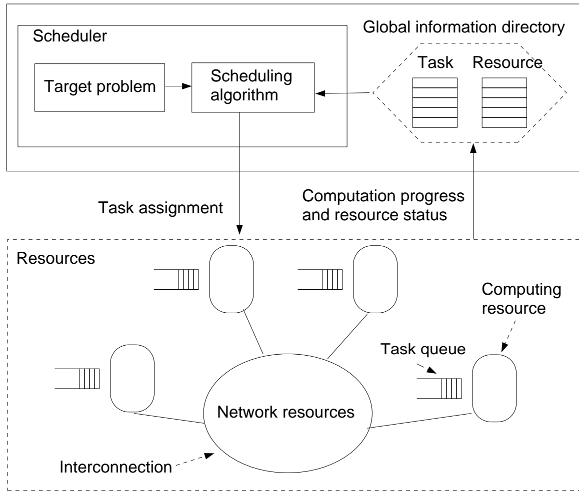

Figure 1. Architecture of the simulator

puting power. The value of this fraction can be specified by the user (e.g. 5%).

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 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. As described by the application model, 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 to be used, 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 several compute nodes finish their tasks and send requests to the scheduler simultaneously, 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 overheads. For example, the cost of resource discovery is simulated through the query/answer messages transferred be-

tween 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 contains entities for the scheduler, global information directory, compute nodes and network links. A ‘driver’ procedure initializes these entities, sets up the occurrence of events such as resource failure and creation, and starts the simulation. This ‘driver’ procedure in PARSEC language acts as the ‘main()’ function in C, and is used to initialize the simulator and trigger external events.

The global information directory is initialized by messages sent from the driver procedure. These messages provide information about the initial status of the resources. After receiving these messages, the global information directory stores the performance data for all the resources available in the system. The global information directory then starts receiving messages from the resources, maintaining information about the performance of the resources and the progress of the computation, and answering queries from the scheduler. This is repeated till the overall computation is terminated by the scheduler.

The resources are initialized by the driver procedure. After initialization, each compute node performs the following functions: it receives tasks from the scheduler, executes the tasks according to its local scheduling policy, and reports performance data updates to the global information directory. This loop is terminated by the simulator once all the tasks are completed. The execution time of a task on a compute node is determined by (1) the number of floating point operations in the task, (2) the local scheduling policy, (3) the computing power, and (4) the impact of I/O processing. The compute nodes may also receive messages from the scheduler to dequeue a task. This occurs when the partitioning of the problem is adapted at run-time. Network links are assumed to be time-shared if multiple data transfers are performed concurrently. The data transfer time over a network link is determined by the size of the data, the network bandwidth and the latency of the network link at the time of the transfer.

The scheduler is also initialized by the driver procedure. The kernel code in the scheduler is the scheduling algorithm. The scheduling algorithm queries the global information directory for the current status of the resources and tasks in the system. Using this information, the computation plan is adapted and the problem may be repartitioned into a different set of tasks if necessary. A programming interface is provided by the simulator so that the user can input various algorithms for partitioning the computation, selecting the resources, and scheduling the tasks.

A logical timer in the simulator broadcasts timing signals to all entities in the system. The period of the timing signals is a constant value that is set by the user. These signals may be used for various purposes. For example, resources may report their performance data to the global information directory at these instances of time. The scheduler can also use these signals to invoke the scheduling algorithm. The user can set these activities to be periodic, event driven, or a combination of both. If the user configures the activities of an entity to be event driven only, the timing signals will be ignored.

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, thereby increasing the response time of the scheduler. 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 detail for the simulations performed.

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 matrix multiplication and LU decomposition in heterogeneous environments are provided for the purpose of illustration.

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. The following sample procedure serves this purpose and is called by the driver procedure when initializing the simulator.

```
void Initialize_Resources() {
 Resource *r;

 // a compute node with normally distributed
 // computing power, mean value 300.0 MFLOPS
 // standard deviation 50.0 MFLOPS; local
 // scheduling policy is FIFO. Computation
 // can be overlapped with communication
 r=New_Resource("node1", 1, COMPUTING_NODE,
     FIFO, OVERLAP, NORMAL_DIST, NULL, 300.0,
     50.0, 0.0, 0.0);
 Register_Resource(r);

 r=New_Resource("node2", 2, COMPUTING_NODE,
     SJF, NO_OVERLAP, TRACE_FILE,
     "node2.trace", 0.0, 0.0, 0.0, 0.0);
 Register_Resource(r);

 r=New_Resource("link3", 3, NETWORK_LINK,
     ROUND_ROBIN, NO_OVERLAP, TRACE_FILE,
     "link1.trace", 0.0, 0.0, 0.0, 0.0);
 Register_Resource(r);

 // node 1 and 2 are connected by link 3
 AddLink(1,2,3);
 ...
}
```

   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 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].

   An example code segment of the problem definition file is shown below. This code describes blocked matrix multiplication. In this code segment, `size_n` is the parameter for the matrix size and `block_size` is the parameter for the block size. The two `GetData` primitives indicate that each task requires two data transfers. In these two primitives, `NULL` indicates that the data transfers do not depend on a previous task. The `Compute` primitive specifies that each task requires b^3 floating point operations, where b is the block size.

```
 $counter = 1; $b = $para{block_size};
 $size_nb = $para{size_n}/$b;
 for($i=1;$i<=$size_nb; $i++) {
    for($j=1;$j<=$size_nb; $j++) {
       for($k=1;$k<=$size_nb; $k++) {
          @task[$counter]=NewTask();
          @task[$counter]::GetData($b**2, NULL);
          @task[$counter]::GetData($b**2, NULL);
          @task[$counter++]::Compute($b**3);$
       }
    }
 }
```

3. *Scheduling algorithm description*. Finally we need to plug in the scheduling algorithms. This is achieved by the programming interface as shown in the following sample code. 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(...)`.

```
void Adaptation_Procedure() {
 Resource_List *r_list; Task_List *t_list;
 Resource *r; Task *t;

 r_list = getResourceList();
 t_list = getTaskList();

 // a simple 'minimum available time'
 // heuristic.
 for(t=t_list->first;t;t=t->next)
   if( t->status == TASK_NOT_SCHEDULED )
     if( t->type == COMPUTATION_TASK ) {
        r = GetMinAvblTimeResource(r_list);
        if(r) AssignTask(r, t);
     }
}
```

Table 1. 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

These three steps configure the simulator to describe the behavior of the resources, the problem and the scheduling algorithm. By using pre-defined commands in the driver procedure, the user can 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 1 lists the configuration options available to the user at the time of this writing.

5.2 Example Use of the Simulator

We first performed simulations of an adaptive algorithm for distributed matrix multiplication. 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. 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. I/O processing on the compute nodes is configured to be non-blocking unless otherwise specified. When performing overlapped computation and communication, a compute node loses 1% of its computing power. The execution of the scheduling algorithm is event driven. The MCT heuristic in [5] is used to schedule the tasks. Block

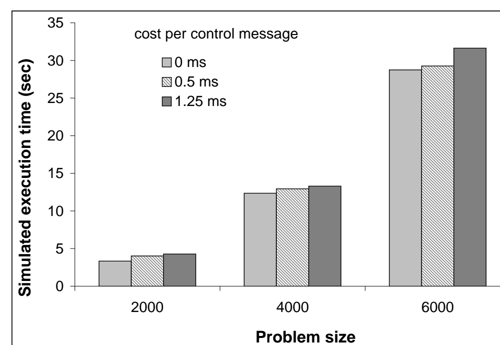

Figure 2. Impact of control messages on the execution time of matrix multiplication

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.

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 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. 3 illustrates the impact of executing the scheduling algorithm on the overall execution time. The results show that executing the scheduling algorithm

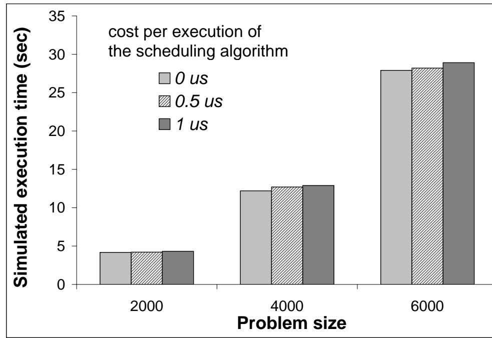

Figure 3. Impact of scheduling algorithm executing overhead on the execution time of matrix multiplication

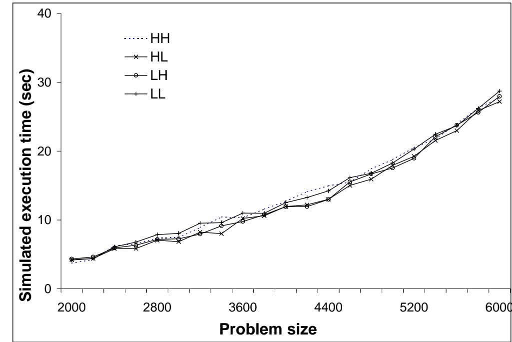

Figure 4. Simulation of matrix multiplication in four dynamic heterogeneous systems (HH, HL, LH, and LL)

has little impact on the overall performance. (Results in [5] showed that the execution of the MCT is less than 1μs on a Pentium II 400MHz computer, for a typical problem with 512 tasks and 16 compute nodes.) Combining the simulation results in Figures. 2 and 3, 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.

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: $M_c = 64$, $M_n = 150$, $\overline{P} = 440\ MFLOPS$, $\overline{B} = 100\ Mb/s$, $\sigma P = 84\ MFLOPS$, $\sigma B = 29\ MFLOPS$, $\sigma_p(i) = 0.15P(i)$, 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. 4 show small performance variations for these systems. This verifies that this algorithm is robust to runtime performance variations and system heterogeneities.

We also used our simulator to study the impact of I/O modes on the overall execution time. The simulations were performed on a 9-node system. Intuitively, non-blocking I/O should outperform blocking I/O. This is validated by the simulations and the results are shown in Fig. 5.

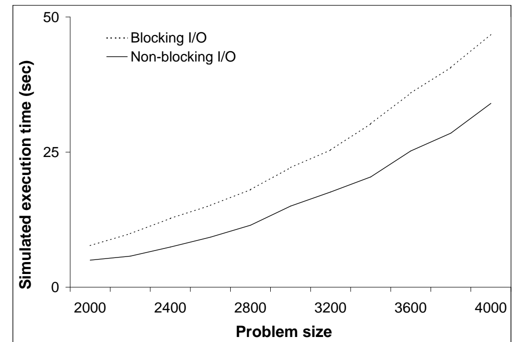

Figure 5. Impact of I/O mode on the execution time of matrix multiplication

The next experiment is for a simple LU decomposition algorithm, which is based on a blocked version of the classic right-looking algorithm for LU decomposition. By using the script tool, we obtained a DAG representation for this algorithm. Tasks in the DAG are scheduled by using a simple heuristic: Computing tasks are assigned in the increasing order of their ready time (A task is *ready* to run when all its parent tasks in the DAG are completed.). For each task assignment, the first available computing node is selected. (A compute node is *available* if it completes all the tasks assigned to it.). By configuring the resources and plugging in this scheduling algorithm, we studied the impact of the number of compute nodes on the overall execution time. The simulation results for a problem of size 2400×2400 are presented in Fig. 6, which verifies the well-known fact that the speedup and the number of compute nodes do not display a linear relationship.

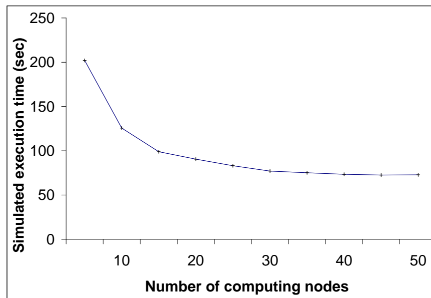

Figure 6. Simulation of LU decomposition

6 Conclusion

In this paper, we have proposed a simulator for evaluating the performance of adaptive applications in dynamic heterogeneous environments. 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 expected. However, we are further validating the simulator using actual experiments in Grid environments.

Although the simulator is a useful tool for evaluating adaptive applications and algorithms in a heterogeneous environment, several limitations need to be addressed. Currently, 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 current version of the simulator models the network links by using its latency and bandwidth and uses a simple model of Data Grid 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. We are investigating these issues and expanding the resource and application models to overcome these limitations.

This work is part of the Algorithms for Data IntensiVe Applications on Intelligent and Smart MemORies (ADVISOR) Project at USC [1].

7 Acknowledgment

We would like to thank Dhananjay Raghavan for careful reading of drafts of this work. We also would like to thank Zack Backer for his valuable inputs in defining the scope of this work.

References

[1] ADVISOR Project. http://advisor.usc.edu.

[2] S. Ali, H. 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. Baker and V. Prasanna. Performance Modeling and Interpretive Simulation of PIM Architectures and Applictions. *Proc. of Euro-Par 2002*, August 2002.

[5] 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.

[6] D. G. Brian. NetLogger: A Toolkit for Distributed System Performance Analysis. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[7] 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)*, pages 1–32, May 2002.

[8] 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.

[9] 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 Progr ams. *IPDPS 2002*, April 2002.

[10] Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[11] N. A. L. Rauchwerger 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.

[12] 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.

[13] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and Object Replication in Data Grids. *HPDC 2001*, 2001.

[14] 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.