CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2012; **24**:1775–1791
Published online 20 December 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1878

RESEARCH ARTICLE

Large improvements in application throughput of long-running multi-component applications using batch grids

Sivagama Sundari M[1], Sathish S. Vadhiyar[1,*,†] and Ravi S. Nanjundiah[2]

[1]*Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India*
[2]*Centre for Atmospheric and Oceanic Sciences, Indian Institute of Science, Bangalore, India*

SUMMARY

Computational grids with multiple batch systems (batch grids) can be powerful infrastructures for executing long-running multi-component parallel applications. In this paper, we evaluate the potential improvements in throughput of long-running multi-component applications when the different components of the applications are executed on multiple batch systems of batch grids. We compare the multiple batch executions with executions of the components on a single batch system without increasing the number of processors used for executions. We perform our analysis with a foremost long-running multi-component application for climate modeling, the Community Climate System Model (CCSM). We have built a robust simulator that models the characteristics of both the multi-component application and the batch systems. By conducting large number of simulations with different workload characteristics and queuing policies of the systems, processor allocations to components of the application, distributions of the components to the batch systems and inter-cluster bandwidths, we show that multiple batch executions lead to 55% average increase in throughput over single batch executions for long-running CCSM. We also conducted real experiments with a practical middleware infrastructure and showed that multi-site executions lead to effective utilization of batch systems for executions of CCSM and give higher simulation throughput than single-site executions. Copyright © 2011 John Wiley & Sons, Ltd.

Received 23 June 2009; Revised 18 July 2011; Accepted 21 August 2011

KEY WORDS: grids; co-allocation; long-running applications; multi-component applications; climate modeling; CCSM; throughput; batch systems; queue waiting times

1. INTRODUCTION

Computational grids have been increasingly used for executing large scale scientific applications [1–5]. Most of the benefits from using grids are primarily due to increase in the number of processors available for execution. In this work, we focus on the use of grids when the processor space for application execution is not increased. Specifically, we deal with grids with multiple batch systems (*batch grids*, for brevity) and show that employing multiple batch systems can improve the *application throughput* of long running multi-component applications, where *application throughput* is defined as the amount of work performed by the application in a certain amount of wall-clock time.

Multi-component Multiple Program Multiple Data (MPMD) applications [6, 7] consist of component applications, which are parallel applications themselves. In these applications, the components are loosely synchronized and communications between components are lighter and less periodic

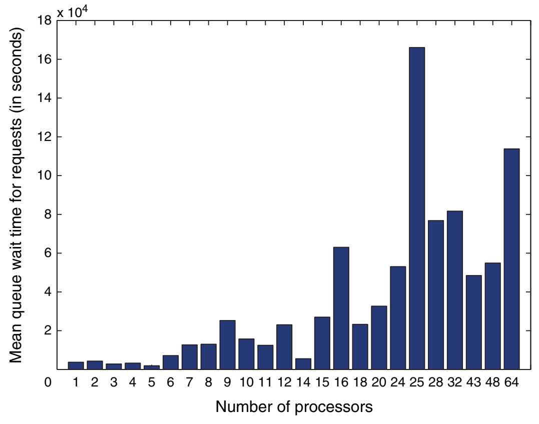
*Correspondence to: Sathish S. Vadhiyar, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India.

†E-mail: vss@serc.iisc.ernet.in

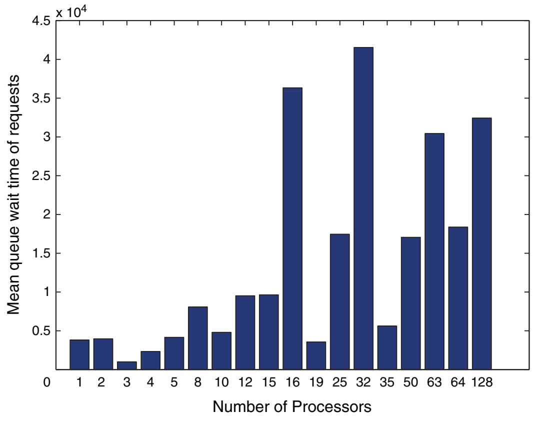
than within components. We consider multi-component applications with malleable components in which the number of processors used for a component execution can be changed. Although the components of the multi-component applications are malleable, they cannot cannot be *split* across multiple sites because of large amount of interactions within components. The emphasis on multi-component applications in our work is important because these components are *coupled*, that is, they are separate applications by themselves but communicate with each other, and they all need to be executed simultaneously *together* to make progress. Hence, co-scheduling of components and migration of *entire* components across the queues will have to be considered by scheduling and rescheduling mechanisms. We differentiate the multi-component applications from traditional workflow applications [8–10] that contain a sequence of levels of executions with well-defined control and data dependencies. In workflows, the components in a level are dependent on the earlier levels, whereas components within a level are independent and execute concurrently. The communications on an edge of a workflow application mostly happen once after the completion of an application component. In contrast, multi-component applications like Community Climate System Model (CCSM) [6, 7] involve coherent and continuous execution of the same set of components throughout application execution, with complex communication patterns and data exchanges between the components.

The primary reason for the improvement in application throughput on batch grids is due to the potential decrease in *queue waiting times* incurred by a multi-component application when its components are simultaneously submitted as individual jobs with small processor requirements to multiple batch systems than when the entire application is submitted as a single job with the total processor requirements of all the components to a single batch system. This is because the queue waiting times of the jobs submitted to a batch system generally increase with their processor requirements as illustrated in Figure 1. The figure shows the average queue waiting times for jobs with different processor requirements on two IBM systems in San Diego Supercomputer Center. The job traces were obtained from the logs maintained by Feitelson [11]. We find that except for some outliers, the general trends in the graphs show that the queue waiting times increase with request sizes.

However, batch systems or queues are associated with maximum execution time durations for the jobs. The number of *active batch systems* available for simultaneous execution of the components of a long-running multi-component application varies at different points of execution. Components will have to be checkpointed and continued from the previous executions on possibly different batch systems as the set of active batch systems changes because of the execution times of some component jobs exceeding the maximum execution time durations of the systems on which they are executing. Thus, whereas the reduced queue waiting times because of multiple batch submissions



(a)



(b)

Figure 1. Average queue wait times for jobs on two San Diego Supercomputer Center (SDSC) systems. (a) SDSC 64 processor system; (b) SDSC 128 processor system.

of application components can lead to improvement in the throughput of a long-running multi-component application, the overheads because of checkpointing and restarting of the components can adversely impact the throughput. The improvement in application throughput also depends on various factors including the frequency and amount of communications between the different components, the interconnection speeds on the links connecting the batch systems and characteristics of the batch queues, namely, queuing and scheduling policies, the loads, and the processor requirements of the jobs submitted to the queues. Hence, the benefits of multi-site execution are not clear and require further studies.

In this paper, we study the improvements in throughput of a foremost long-running multi-component parallel application, CCSM [6, 7], because of simultaneous submissions of the components to multiple batch systems of a grid over submission of the entire application to a single batch system. We developed a performance model for CCSM for predicting the execution times of CCSM for different number of clusters and processors. By using the performance model with a multi-cluster system simulator that simulates submissions of jobs with different processor and execution time requirements to batch queues of different queue scheduling policies and different interconnection speeds between the batch systems, we performed a large number of simulations with different application and system configurations. We show that multiple batch executions can lead to 60% average reduction in queue waiting times for CCSM jobs and 55% average increase in application throughput of CCSM over single batch executions. Our primary focus on simulations is, because our application case study, namely CCSM, takes one-to-several weeks of execution for a single system configuration (batch queue loads, scheduling policy, network, etc.), and it is impossible to perform large scale evaluations and comparisons of single and multi-site executions using a large number runs with different values of system parameters, using real executions. We also developed a practical grid middleware infrastructure for execution of CCSM across multiple batch systems. We conducted real experiments with the infrastructure and showed that multi-site executions lead to effective utilization of batch systems for executions of CCSM and give higher simulation throughput than single-site executions.

Although performance benefits because of co-allocation of multiple batch systems for application execution have been analyzed for applications with small execution times [12–14], our work is the first effort, to our knowledge, that studies the improvement in throughput of long-running multi-component applications in which the execution times of the components are significantly greater than the execution time limits associated with the batch systems. We employ a novel execution model in which the set of active batch systems available for execution is dynamically shrunk and expanded, and the components are checkpointed and continued or rescheduled on possibly different batch systems. Although various efforts exist on execution strategies for malleable single component applications [15–17] on interactive systems, starting from the early 2000s, our work differs from all these strategies in the aspects of malleability of multi-component multi-physics applications and rescheduling because of batch queue events of queue waiting and execution time limits. These are very important contribution of our work, since our analysis and our demonstration with real experiments will motivate and allow leveraging the power of multiple batch systems for solving large-scale scientific problems, both (batch systems and large applications) of which are becoming widely prevalent.

Thus, the primary novel aspects of work that distinguishes from other efforts are the following. Our work focuses on improving the simulation rate of a single long running multi-component application where the different components are submitted multiple times to different batch queues. Although submissions are made to the queues of all the clusters, an execution of the multi-component application can proceed when at least one of the clusters become available for execution. Our work also considers independently administered multiple batch sites with predefined and possibly different job execution policies. Thus, our work is practical and can be applied directly to a given batch grid, whereas their efforts require changing the scheduling policies. The primary contributions of our work are as follows: (i) a novel execution model for long-running multi-component applications in which the set of active batch systems changes during execution, and the components are coscheduled, submitted multiple times to different systems, checkpointed and continued for coherent execution; (ii) a simulation framework that models submissions of the components on different batch systems with different queue scheduling policies, job arrival dynamics, queue waiting times and inter-cluster network speeds; and (iii) large-scale analysis of decrease in queue waiting times and increase in resource availability and application throughput because of multiple batch submissions.

Section 2 describes the execution model used for executing different components of CCSM in multiple batch systems and explains the metrics used for comparison of single and multiple batch executions. Section 3 describes in detail the various components of our simulation architecture including the application simulator and scheduler. Section 4 describes our experiment setup and presents various results related to simulations of single and multiple batch simulations. Section 5 details our attempts to develop a practical middleware infrastructure for executing multi-component applications on batch grids. In Section 6, we briefly compare our work with other existing efforts for co-allocation on batch systems. Section 7 summarizes our work and lists our future plans.

2. EXECUTION MODEL AND EVALUATION METHODOLOGY

In this work, we consider a classic and foremost example of a multi-component application, CCSM [6]. CCSM is a global climate system model from National Center for Atmospheric Research [18]. It is a MPMD application consisting of five components, namely, atmosphere, ocean, land and ice and a coupler component, which transforms data and coordinates the exchange of information across the other model components. CCSM is typically executed for long periods to simulate climate systems for multiple centuries. The execution times for such runs can be several weeks. To support such long running simulations, CCSM contains *restart* facilities where the application can be made to store its execution state as *restart dumps* at periodic intervals or after some simulated days and simulations for an execution can be continued from the previous executions using the *restart dumps* of the previous executions. Batch queue systems are associated with limits for execution time for a job. If a job submitted to batch queue exceeds the execution limit of the queue, it is aborted by the system. The execution time limits are typically few days. Hence a long running application like CCSM involving multi-century simulations will not be able to complete execution within the execution time limit of a batch system. The CCSM application executing on a batch system will have to be made to create a *restart dump* before the execution time limit, stopped, resubmitted to the batch system and continued from the previous execution using the restart dump.

In our execution model, when CCSM is executed on B batch systems or queues, B job submissions are made to the queues with different processor requirements or request sizes. We refer to a job submission to a queue and the corresponding batch system as becoming *active* when the job completes waiting in the queue and is ready for execution. When some subset of submissions become active, the components of CCSM are executed on the corresponding subset of active batch systems, $B_A \subseteq B$. When a submission reaches the execution time limit of the corresponding queue, the batch system becomes unavailable for components that were executing on the system and is hence removed from B_A. In this case, the CCSM is made to create a restart dump and all the components are remapped or rescheduled to the new/updated active batch systems, B_A, and continued execution from the restart dump. Also, a job submission is made to the batch system that became unavailable for execution. At some point, when this batch system becomes active, the CCSM executing on the already active systems is made to create a restart dump, the set B_A is updated to include the system that became newly active, components are remapped to new/updated B_A and continued execution from the restart dump. Specifically, CCSM is rescheduled whenever the set B_A changes, either because of some batch systems becoming active or because of one of the active batch systems becoming unavailable upon the corresponding submission reaching its execution time limit on the system. These points of execution at which the application are rescheduled are denoted as *rescheduling points*. This process continues until CCSM completes simulation of a certain number of simulated years. The process is illustrated in Figure 2 with an example of a two-component application executing on two batch sites.

We evaluate and compare the single-site and multi-site executions of CCSM in terms of the queue waiting times of the submissions, the *resource availability rate* (RAR), and application throughput. RAR gives a measure of the amount of processor-hours the resources were available for CCSM

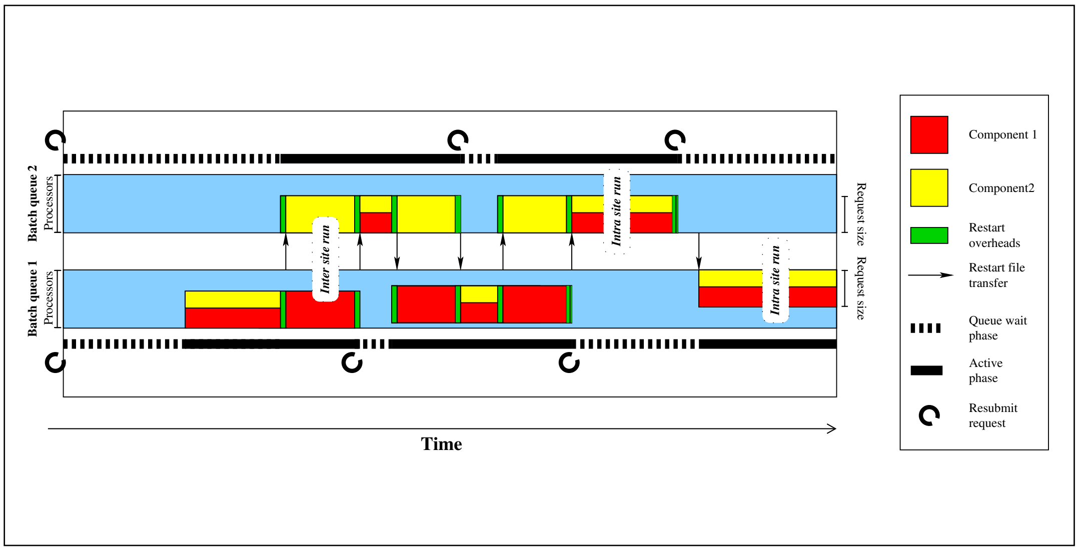


Figure 2. Execution of a long-running two-component application on two sites.

execution. Processor-hours for a submission is defined as the product of the number of processors used for the execution and the execution time corresponding to the submission. For a long-running execution consisting of multiple submissions, RAR is calculated for a system as the ratio of the sum of processor-hours for the submissions to the total execution time considered for evaluation. For multi-site executions, RAR is calculated as the ratio of the sum of the processor-hours between the *rescheduling points* and the total execution time considered for evaluation. We also compare the single and multi-site executions using *application throughput*. For CCSM, throughput is calculated as the number of climate days simulated per wall-clock day. For comparison of single and multiple system executions, we generated a set of four batch systems with different number of processors and considered the largest of the systems for single system executions. This is to enable fair comparison because single-system users would typically want to use the largest batch system available to them for better performance, larger problem sizes, and smaller queue waiting times.

3. SIMULATION METHODOLOGY

Our simulation setup comprises of five major components: (i) a workload simulator, (ii) a multiple batch system simulator, (iii) statistic calculator, (iv) CCSM (re)scheduler, and (v) CCSM simulator as shown in Figure 3.

The workload simulator produces a list of jobs with submission time, processor request size and expected execution time for each job. This produces the external load of non-CCSM jobs for our experiments. We used the workload model developed by Lublin and Feitelson [19]. This model was developed by applying rigorous statistical procedures to logs collected from real batch systems of three different locations and was shown to be the most representative model available in a general sense. For all our job traces, we specified the maximum execution time of 2 days for the jobs. In order to generate job traces of different job characteristics, we categorize the jobs in terms of their execution times and processor requirements. We call jobs with small execution times (mean execution time of 3-4 min) as short jobs (S) and those with large execution times (mean execution time of 6 h) as long jobs (L). Similarly, jobs with small processor requirements (< 10 processors) are called narrow jobs (N) and those with large processor requirements (> 10 processors) as wide jobs (W). We then tuned the input parameters of the workload model to generate job traces consisting of predominant number of jobs belonging to one of the four job categories, namely, S, L, N, and W. The mean inter-arrival times of the jobs in our queue traces were set to 3000 s.

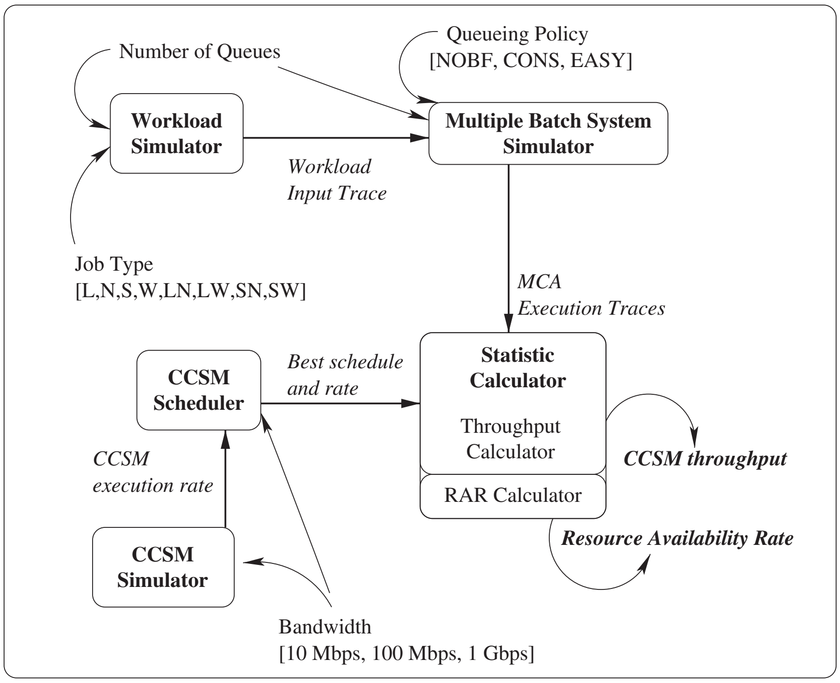


Figure 3. Simulator architecture.

The batch system simulator uses the job traces produced by the workload simulator and simulates the job executions on a given set of batch systems with given queue scheduling policies. Three queue scheduling policies, namely, FCFS (First Come First Serve), conservative (CONS) and EASY backfilling were used for scheduling or selecting jobs in the queues for allocation to processors. The simulator also simulates submissions corresponding to CCSM application by adding the submissions to the job traces of each batch queue. A CCSM submission is added to the job trace immediately after the completion of the execution time limit for the previous CCSM execution. The simulator using the specified queuing policy for each batch queue or system, produces *multi-component application execution traces* containing the queue waiting times in addition to the arrival and execution times and request sizes of the CCSM jobs. The batch simulator has been developed using C and Message Passing Interface (MPI). The simulations thus can be conducted in parallel.

The statistic calculator uses the multi-component application execution traces to calculate the application throughput and RAR. To calculate application throughput, the statistic calculator needs to determine the allocation of processors in the different batch systems to the different components of CCSM whenever the number of active batch systems for CCSM execution changes in the execution traces. The calculator invokes the CCSM scheduler, described in Section 3.2, component to determine the schedule for CCSM execution for a given number of active batch systems, namely, the systems. The CCSM scheduler evaluates different candidate schedules for CCSM execution in terms of predicted execution times and chooses the schedule with the minimum predicted execution time. To determine the predicted execution time for a particular candidate schedule, the CCSM scheduler invokes the CCSM simulator, described in Section 3.1, which simulates the execution flow and the computational and communication phases of the different components of CCSM on the processors of the different batch systems to predict the execution time.

3.1. Application simulator

We developed a discrete-event simulator for CCSM for modeling the execution flow of the components. Each component communicates the data processed by it to the coupler at periodic intervals. This interval of communication, *coupling period* (CP), can be different for different components.

Within each CP, a component performs some computations of its local data, receives data from the other components through the coupler, performs some computations of this received data and sends its processed data to the other components through the coupler. These phases are denoted as *send-to-recv computations (S-R)*, *recv communications (Re)*, *recv-to-send computations (R-S)*, and *send communications (Se)*, respectively.

The CCSM simulator takes as input, the total wall-clock time for CCSM execution, the CPs for the four components, the periodicity of anomalous phases, the number of clusters, the inter-cluster bandwidth, and the allocation of processors in the clusters for the components. The simulator then models the execution flow of CCSM until the execution pattern per simulated-day converges. The number of simulated days and the execution time for the simulation are obtained, and the ratio between the two values is scaled to obtain the total number of simulated days or application throughput for the total wall-clock time available for CCSM execution.

In order to model the execution flow and predict the number of simulated days for a given execution time (simulation rate), the simulator uses models for the different phases, namely, S-R, Re, R-S, Se, and An, for each component. These *phase models* predict the execution times for the phases for a given number of processors. To construct these *phase models*, we conducted many experiments by executing CCSM with T42_gx1v3 resolution, a medium resolution climate model with Eulerian dynamical core for atmosphere components, across two AMD Opteron clusters, *fire-16* and *fire-48*, with different application and system configurations. We used a simple equation, $computeTime = a + b/componentSize$, for modeling each of the computation phases of a component. $componentSize$ is the number of processors allocated for the component and $computeTime$ is the execution time corresponding to the computation phase. a and b denote the model coefficients and were obtained by linear regression using the observed execution times corresponding to the actual experiment. For modeling a communication phase for a given inter-cluster bandwidth, we used the average of the observed communication times for the phase corresponding to the actual experiments across the two clusters. Thus, the *phase models* for a given inter-cluster bandwidth can be used for predicting the simulation rate of CCSM for any number of processors allocated to the components for an inter-cluster bandwidth of 10 Mbps, 100 Mbps or 1 Gbps. We validated the accuracy of the application simulator by conducting different experiments with CCSM using *fire-16* and *fire-48* AMD clusters. We use our performance models primarily to rank the candidate schedules for determining the best CCSM configuration and best schedule for execution of the application. We found that in most cases, our performance models resulted in good correlations between the predicted and the actual execution times for different configurations involving different component sizes and distributions, with correlation coefficients of 0.98.

3.2. CCSM scheduler/rescheduler

In our execution model, the CCSM components are scheduled and rescheduled to the available batch systems at various points of execution. Scheduling involves determining the set of processors for each component to minimize the application execution time. Scheduling for single batch executions involves determination of *processor allocation* or the number of processors for each component. Scheduling for multiple batch executions involves processor allocation and *component mapping*. Component mapping determines the mapping of the different components to the different active batch systems. A processor allocation is *valid* if it adheres to the processor restrictions for the CCSM components. A component mapping is valid if the number of processors determined for each component is less than the number of processors available in the corresponding batch system on which it is allocated.

Because our primary focus is on showing the benefits with multiple batch executions without increasing the number of processors used for single batch executions, we determine for multiple batch executions, a processor allocation that is similar to the best processor allocation or schedule determined for single batch execution. Hence, we first determine the *single-system best schedule* for the largest batch system by evaluating all *valid processor allocations* and choosing the processor allocation or schedule for which the application simulator predicts the minimum execution time. To restrict the number of valid processor allocations, the scheduler chooses only power-of-two number of processors for some components. The *single-system best schedule* is used for execution of CCSM on a single batch system and as a *base processor allocation* for determining the schedule for multiple batch executions.

Because the *base processor allocation* may not give the best schedule for multiple batch executions because of communications on the slow inter-cluster links and may not yield a valid component mapping, it cannot be directly used for multiple batch executions. Hence, we consider valid processor allocations equivalent to the base processor allocation by considering for each component, few valid processor allocations immediately lower and higher than the number of processors for the component in the base processor allocation. For each of the valid processor allocations, the scheduler evaluates all valid mappings of the four components to the four batch systems in terms of the execution times predicted by the application simulator for the mappings and chooses the schedule with the minimum predicted execution time. During rescheduling, for each of the valid mappings, the predicted cost for the transfer of the restart dumps of the components corresponding to the previous mapping for the previous execution is added to the predicted execution time for evaluation of the mapping. We found that our scheduler typically spends less than 5 s to determine the best schedule for multiple batch executions. In real settings, the scheduler will be invoked only once in several hours and hence scheduling overheads of even few minutes are tolerable.

4. EXPERIMENTS AND RESULTS

4.1. Experiment setup

We conducted 1000 simulation experiments for single and multiple batch executions for each inter-cluster or inter-site bandwidth to analyze the gains in executing long-running CCSM applications on batch grids. For our experiments, we used three different inter-cluster bandwidth values, namely, 10 Mbps, 100 Mbps, and 1 Gbps. The first two bandwidths are commonly observed on the links connecting two clusters located at two different sites in many grid systems. The last bandwidth is seen on the links connecting two batch systems in a single site and on the links connecting two different submissions in a single batch system.

For each simulation experiment for a given inter-cluster bandwidth, we randomly chose four queues corresponding to execution of four components of CCSM. For each queue, the total number of processors in the queue is randomly chosen from $\{32, 64, 128, 256, 512\}$. The workload characteristics, namely, the number of long (L), short (S), wide (W), and narrow(N) jobs submitted to a queue, were randomly generated for each queue. The queue scheduling policy for each queue is also randomly chosen from one of FCFS, CONS and EASY policies. We used the four queues to simulate long running CCSM execution on multiple batch systems and used the largest of the four queues to simulate execution on a single batch system. We then compared the multiple and single batch CCSM executions in terms of queue waiting times, RARs and application throughput.

4.2. Results

The primary reason for the potential improvement in throughput of CCSM when executed on multiple batch systems is the reduction in queue waiting times incurred by the CCSM component jobs on the individual queues. For each simulation experiment, we compared the queue waiting times of the CCSM jobs in the largest queue when used for single and multiple batch executions. The largest queue, besides being common to single and multiple batch executions, also contributes most of the processor space for CCSM application in multiple batch executions. Figure 4(a) shows the average queue waiting times for single and multiple batch executions for different number of processors in the largest queue. We find that the average reduction in queue waiting times because of multiple system executions is 60%. Figure 4(b) compares the percentage difference in queue waiting times between single and multiple batch systems for different ratios of request size for CCSM and total size of the largest queue in multiple batch executions. We find that for small ratios or for large gaps between request sizes and total sizes, the percentage reductions in queue waiting times because of multiple batch executions are large.

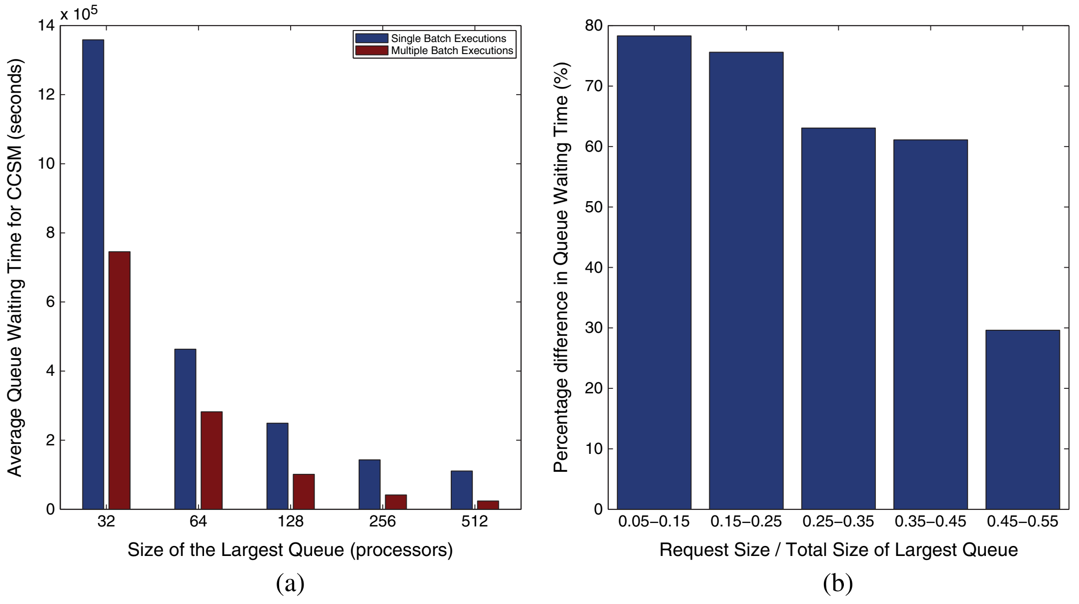


Figure 4. Queue waiting times for Community Climate System Model jobs. (a) Queue waiting times in single and multiple batch; (b) Percentage reduction in queue waiting times because of multiple batch executions for different (request size/total size) ratios of largest queue executions for different sizes of largest queues.

Figure 5(a) shows the average RARs for single and multiple batch executions for different sizes of the largest queue. The figure shows that the average increase in RAR because of multiple batch executions is 12%. Figure 5(b) shows the average RARs for different request sizes of CCSM on the largest queue for multiple batch executions. The figure shows that when the request sizes are less than 55 processors on the largest queue, the RAR increases with the request sizes because of more processors available for CCSM execution. When the request sizes are greater than 55, the RAR saturates because the benefits because of larger number of processors for execution are negated by the larger queue waiting times incurred for higher number of processors.

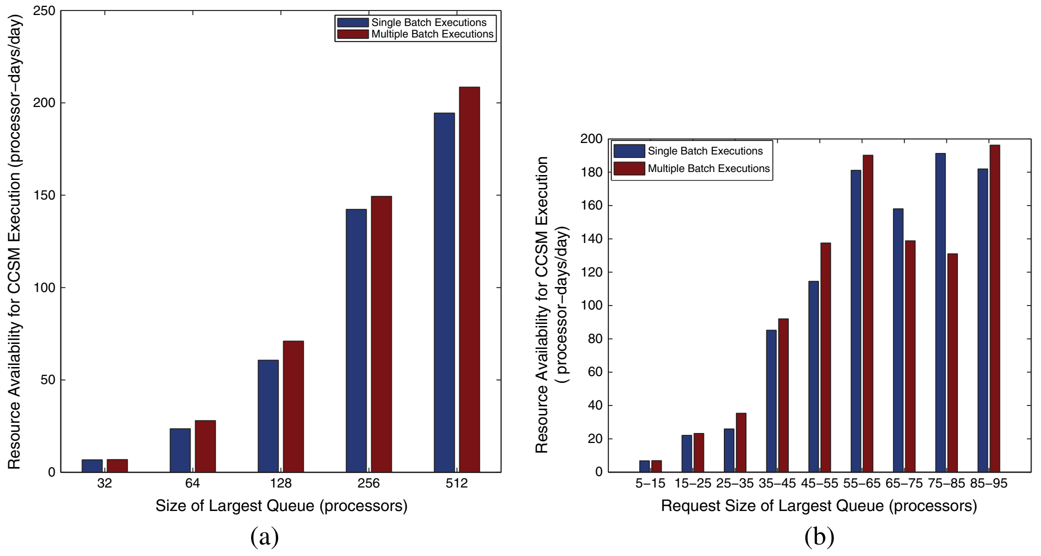
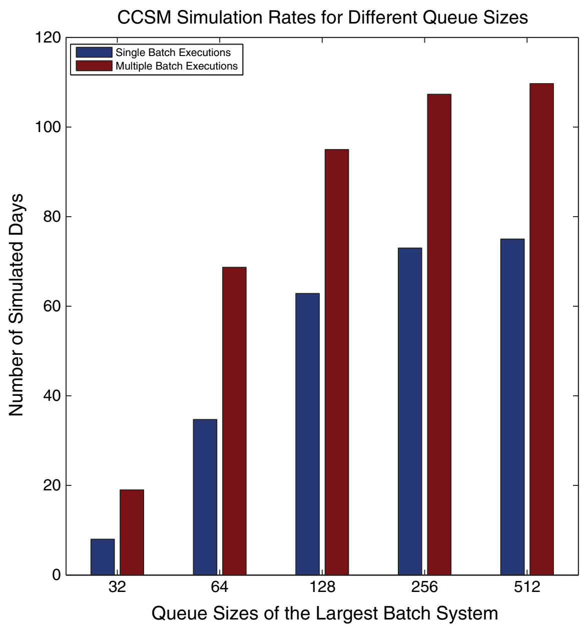
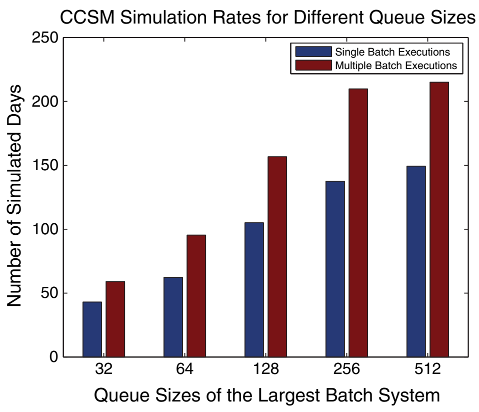


Figure 5. Resource availability rates (RARs) for Community Climate System Model jobs in single and multiple batch executions. (a) RARs for different sizes of largest queue; (b) RARs for different request sizes of Community Climate System Model on largest queue.

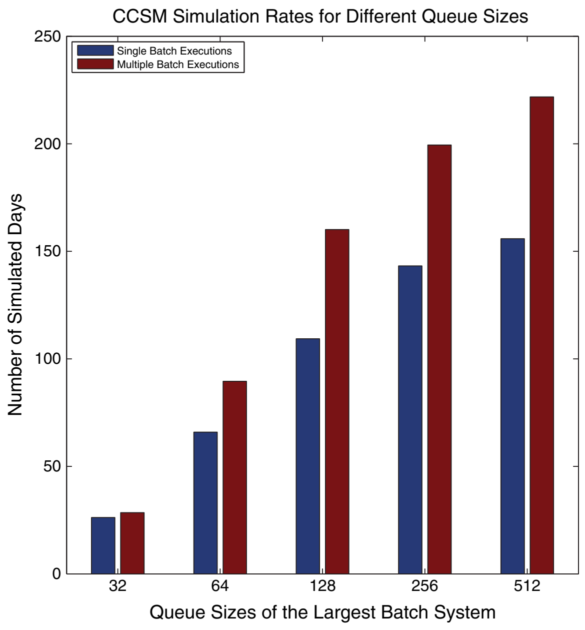
Figure 6 shows the comparison of average CCSM simulation rates with single and multiple batch executions for different queue sizes or number of processors of the largest batch system and for three different inter-cluster bandwidths. The figure shows that irrespective of the number of processors available in a single system, adding more processors from other system for execution of CCSM will yield performance benefits to CCSM. We find that the average number of simulated days with multiple batch executions for a given size of the largest queue is at least 8% greater than the average with single batch executions. These results show that CCSM, in spite of involving periodic communications between different components through a coupler, is highly suitable for execution across



(a)



(b)



(c)

Figure 6. CCSM simulation rates in single and multiple batch executions for different queue sizes of largest queue. (a) 10 Mbps; (b) 100 Mbps; (c) 1 Gbps.

different batch systems on batch grids. These results also illustrate that batch grids involving batch systems with complicated queuing and workload dynamics can be used for efficient executions of multi-component applications.

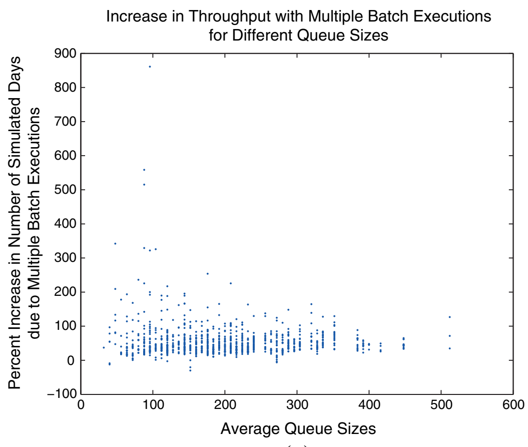
Comparing the results across the three inter-cluster bandwidths, we find that the number of simulated days increases when the inter-cluster bandwidth increases from 10 Mbps to 100 Mbps. However, the application throughput saturates at 100 Mbps and does not increase when the inter-cluster bandwidth is increased to 1 Gbps because the communications of small messages exchanged between the CCSM components reach the maximum speed at 100 Mbps. In general, we find that the bandwidth of the links connecting the different batch systems do not impact the percentage increase in application throughput because of multiple batch executions. This is due to high ratios of computations to inter-component communications in the individual components of CCSM.

Figure 7(a) shows the percentage increase in the number of CCSM simulated days for all the experiments with single and multiple batch executions for different averages of queue sizes of multiple batch systems and for inter-cluster bandwidth of 100 Mbps. Similar results were observed for inter-cluster bandwidths of 10 Mbps and 1 Gbps. We find that multiple batch executions provide 55% average improvement in throughput over single batch executions and provide huge gains for some small queue sizes. This is because when small queues are used for single system executions, the request sizes for CCSM are almost equal to the queue sizes. This leads to large queue waiting times for CCSM on single batch executions because of the presence of other jobs in the system. However, the request sizes of CCSM are split into small request sizes when executed on multiple batch executions leading to significant decrease in queue waiting times in the largest queue. This results in high gains in throughput in multiple batch executions.

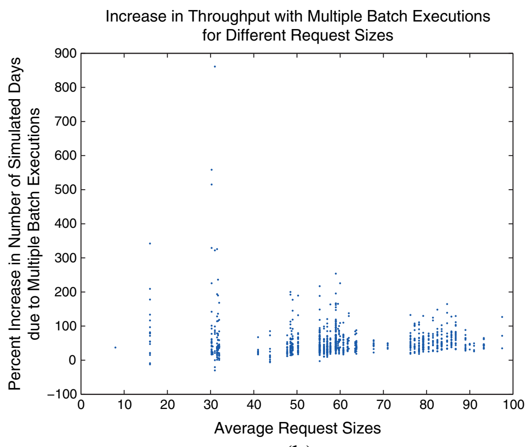
Figure 7(b) shows the percentage increase in the number of CCSM simulated days for all the experiments with single and multiple batch executions for different averages of CCSM request sizes on the multiple batch systems and for 100 Mbps inter-cluster bandwidth. The figure also shows that for very large average request sizes, the gains because of multiple batch executions are low. This is because very large request sizes lead to similar large queue waiting times on both single and multiple batch systems. This result regarding limiting request sizes to obtain high benefits because of co-allocation agrees with the conclusions in the work by Bucur and Epema [12].

5. A PRACTICAL IMPLEMENTATION

We have developed a middleware framework realizing our execution model for execution of multi-component applications like CCSM across multiple batch systems. Our framework consists of three



(a)



(b)

Figure 7. Percentage increase in CCSM simulation rates because of multiple batch executions for different averages of queue and request sizes in multiple batch (100 Mbps). (a) For different averages of queue sizes; (b) For different averages of request sizes.

primary components to synchronize the executions of the components on multiple batch systems: a *coordinator* that determines mapping of components, and schedules and reschedules the component executions on the systems, a *job monitor* on each front end node of the batch systems that interfaces with the coordinator, and a *job submitter* on the front end node that repeatedly submits a CCSM job upon completion of the previous CCSM job. Our framework also consists of a *CCSM job script* that executes and re-executes the CCSM MPI application on a system corresponding to specified mappings of components to processors at various points of time within a CCSM job submitted by the *job submitter*. The architecture is illustrated in Figure 8.

The *coordinator* daemon is the most significant daemon and is executed on a location that is accessible from the front-end nodes of all the systems. The coordinator contains a record of all the information pertaining to the queues and the CCSM jobs. Because the coordinator has knowledge of the state of the entire system, it can take actions and/or instruct other daemons to take actions. Some of the actions taken by the coordinator include determining the mapping of components to batch systems, scheduling and rescheduling component executions and transferring restart files. The *job monitor* daemons track the local behavior of the CCSM jobs on the batch systems and interfaces with the coordinator. It sends messages to the coordinator related to the corresponding batch queues becoming active and inactive. The job monitor also processes the configuration data, related to mapping of components to the processors in the queue, supplied by the coordinator at every reconfiguration event and writes the configurations to local files for reading by the CCSM jobs. *Job submitter* is another daemon that is started for each queue and runs on the front-end node of the respective batch system. Its main functionality is to iteratively submit CCSM jobs to the queue through a *CCSM job script*. The job submitter submits the *CCSM job script* to the batch queue using the local queue submission mechanisms. The MPI application with a set of CCSM components is executed on a set of processors in a queue by the execution of the *CCSM job script* on the queue. Within a CCSM job submission, corresponding to the execution of the *CCSM job script* by the job submitter, the job script executes multiple MPI applications corresponding to multiple rescheduling events with the component mappings specified by the coordinator.

We used our framework for performing real execution of CCSM across multiple batch queues. For our experiments, we used three batch queues in two clusters, namely, *fire-48*, a AMD Opteron cluster with 12 × 2 dual-core 2.64 GHz processors, and *varun*, an Intel Xeon cluster with 13 8-core 2.66 GHz processors. Three queues were configured on these systems with OpenPBS: one queue, *queue-48*, of size 48 on *fire-48*, two queues, *queue-32* and *queue-64*, of sizes 32 and 64, respectively,

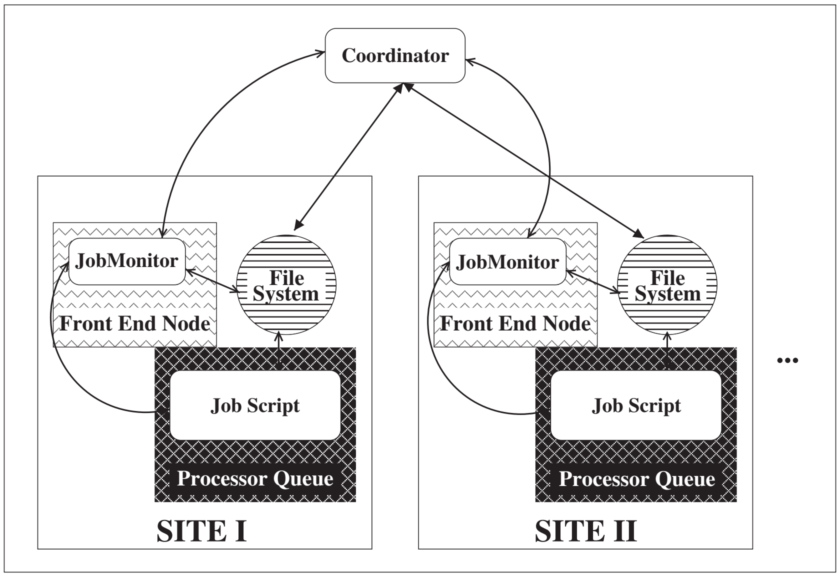


Figure 8. A practical infrastructure.

on *varun*. The AMD cluster is located at the Supercomputer Education and Research Centre, and the Intel Xeon cluster is located at the Centre for Atmospheric and Oceanic Sciences and are connected through a campus network with a bandwidth of around 500 Kbps. The connections within the two clusters are using switched Gigabit Ethernet. We performed our multi-site execution of CCSM for 2 days and 14 h and compared with a same-duration execution of single-site execution. For the single-site execution, we used *queue-64* of the *varun* cluster. For the multi-site execution, we used all the three batch queues of the two clusters for coordinated executions. External loads were simulated by submitting synthetic MPI jobs to the queuing systems based on the workload model developed by Lublin and Feitelson [19]. The maximum execution time limit for all jobs on all queues was set to 12 h. The coordinator was started on the front-end node on *fire-16*. A job monitor and a job submitter corresponding to each queue were started on the front-end of its cluster.

The comparison of simulation rates is shown in Figure 9. and 10. As shown in Figure 9, we find that the single-site run on *queue-64* performed climate simulations of 1090 days (2 years, 11 months and 25 days), whereas the multi-site run performed climate simulations of 1200 days (3 years, 3 months and 14 days). The multi-site grid execution gives better overall progress of executions than the single-site executions in spite of the various overheads related to multi-site executions including restart overheads, multiple rescheduling, inter-site communication, reconfiguration and rebuilding overheads. Figure 10 shows the percentage of wall-clock time of 2 days and 14 h spent by the single

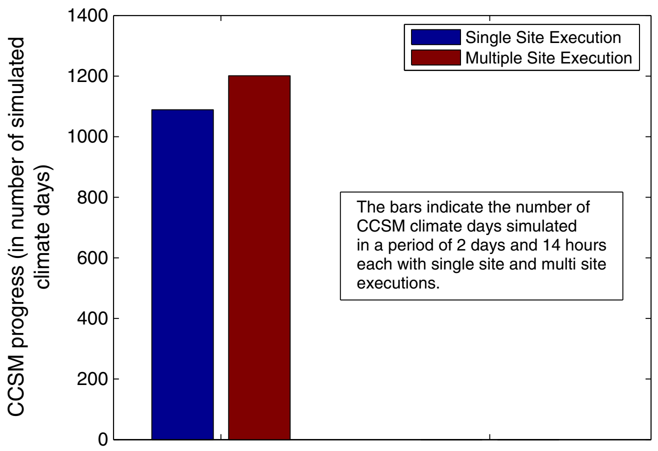


Figure 9. Comparison of simulation progress with single and multi-site runs of 2 days and 14 h.

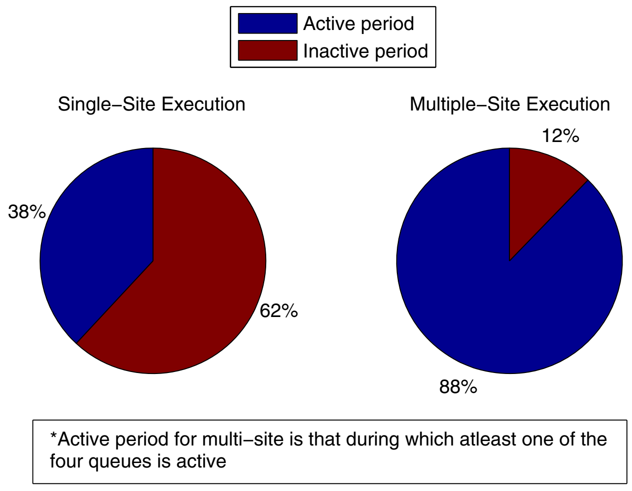


Figure 10. Comparison of active and inactive periods with single and multi-site runs.

and multi-site executions in *active* states corresponding to execution of CCSM in at least one of the batch queues and *inactive* states corresponding to waiting in the queues. Single-site executions incur large inactive periods, spending more than 1.5 days (62%) in inactive state, whereas multi-site executions result in only about 7.5 h of inactive periods. We also noted that the total overhead involved was about 5% of the total experimental duration: 3% because of overheads related to restart I/O and transfer, and 2% because of compilation at reconfigurations. About 2 days and 6.5 h were spent in useful executions for multi-site run. Thus, the multi-site executions lead to effective utilization of systems for CCSM executions.

6. RELATED WORK

There has been increasing interest in co-allocating parallel applications across multiple clusters [12–14]. Casanova analyzed the impact of redundant submissions on the other jobs in the system [20]. In our work, we do not replicate jobs on multiple batch queues but decompose a single job into multiple sub-jobs and submit these sub-jobs to many batch queues. The work by Nurmi *et. al.* [21] deals with execution of workflow applications on different batch systems of a grid. In our work, we consider multi-component applications that contain periodic communications between different batch systems unlike the workflow applications. The work by Platt *et al.* [22] and Bal *et al.* [13] analyzed the impact of inter-cluster speeds on the performance of parallel applications when executed across wide-area clusters and the benefits of wide-area computing because of the increase in the number of processors made available to the applications. In our work, we analyze the benefits of wide-area computing over computing in a local cluster for the same number of processors.

The work by Ernemann *et al.* [14] analyzes the mean response times of synthetic applications when executed across multiple clusters for different ratios of execution times of applications when executed on a single local cluster and on multiple clusters. Their results show that multi-cluster computing can yield improved response times because of decreased queue waiting times as long as the execution times because of multi-cluster computing does not increase more than 1.25 times the execution times on local clusters. Bucur and Epema have extensively studied the benefits of co-allocation of processors from different clusters in a grid for job executions [12, 23–25]. In their work, they analyze the impact of using different scheduling policies, component sizes, and number of components on co-allocation. Their work considers three different scheduling policies, namely, GS policy in which there is a single global scheduler with a single global queue for submitting both single and multi-component jobs from all clusters, LS policy that deals with only local queues of the clusters where both single and multi-component jobs from a cluster are submitted to the local queue of the cluster and LP policy that considers both a single global queue for multi-component jobs and local queues of the clusters for single-component jobs from the clusters. Using large number of simulations with various workload logs, application characteristics and inter-cluster speeds, they show that execution of multi-component jobs across multiple clusters can reduce mean response times of jobs and improve processor utilization.

The efforts by Bucur and Epema and by Ernemann *et al.* consider improving mean response times of short jobs where the different components of a job are submitted to the different batch queues only once and the components complete executions within the execution time limits associated with the batch queues. However, our work focuses on improving the simulation rate of a single long running multi-component application where the different components are submitted multiple times to different batch queues. The execution of a component is stopped within the execution time limit associated with its submission to a batch queue and the component is submitted again to the same or different batch queue and continued from its previous execution. Another important difference is that unlike their efforts that assume that all clusters become simultaneously available for execution of components, in our model, whereas submissions are made to the queues of all the clusters, an execution of the multi-component application can proceed when at least one of the clusters become available for execution. The third primary difference is that, whereas their efforts assume control over local queuing policies, our work considers independently administered multiple batch sites with predefined and possibly different job execution policies. Thus, our work is practical can be applied directly to a given batch grid, whereas their efforts require changing the scheduling policies.

In our previous work [26], we explored the possibility of improvements in response times because of submissions of multiple components to multiple batch systems. Although the concept of splitting a job and submitting as multiple subjobs on multiple batch systems is common to our current and previous work, our current work differs significantly from our previous work in terms of using a completely novel execution model which, unlike the model in our previous work, does not require the application to start execution only upon all the batch systems becoming active and considers cases for non-overlapping active periods of queues. Further, that work was statistical in nature and used single queue simulations to estimate average queue wait times, that were analytically combined to estimate multi-queue simulation measures. And because of the simplistic modeling, only the probabilities of benefits could be estimated. However, with the new execution model and a realistic simultaneous multi-queue simulation framework used in this work, we are able to simulate realistically the execution of CCSM across multiple queues along with other jobs, and thus obtain 'actual' benefits. Other factors new to this work include scheduling and rescheduling policies, a robust application simulator, use of queues equal to the number of CCSM components (only two queues in our previous model) and an analytical model for a new metric, RAR.

A recent work by Ko *et al.* [27] deals with multi-physics application execution on batch queues. However, this work deals with applications that finish execution within the execution time limits, and not long-running multi-physics applications that span multiple execution time limits and the associated rescheduling and migration dealt in our work.

Workflow applications also consist of multiple component applications [8, 10]. However, our application is a time-marching or time-evolving MPI MPMD application consisting of multiple 'components' executing concurrently throughout the execution duration. Our applications involve complex communication patterns and data exchanges between the components unlike the well-defined interfaces and data-flow patterns or dependencies in traditional workflow applications. In most of the practical workflow applications, the components perform similar tasks. Hence, our application, in its current form, is different from a workflow application and cannot be supported by any of the existing workflow-based technologies [28, 29].

Our work can also complement or use existing middleware infrastructures. Cactus [30] is a generic and modular middleware framework for enabling execution of scientific applications on high performance systems. Our work can be integrated with the Cactus framework by developing specific 'thorns' [31] for the different components in our framework, our rescheduling policies and component-level migrations, and using the common thorns and core components of Cactus for I/O of restart file, batch queue monitoring, resource discovery, prediction, resource selection, basic scheduling, and checkpointing functionalities. Many of our coordinator's functionality can be performed by the Cactus flesh. Such integration will help efficient executions of our multi-component applications in a generic and portable manner. In this work, we had assumed that sub-components executed on different batch systems can communicate with each other. This assumption is reasonable because some MPI communication libraries including PACX-MPI [32] and MPICH-GX [33] support communications between MPI applications executed on different batch systems by means of special communication processes or proxies executed on the front-end nodes of the batch systems. Another solution called Smartsockets [34] has been proposed to facilitate direct connections between multiple MPI processes executing on different grid sites.

7. CONCLUSIONS AND FUTURE WORK

In this work, we analyzed in detail the benefits of using multiple batch systems of a grid for improving the throughput of a long-running multi-component parallel application over using a single batch system for executions. We constructed an application-level simulator for modeling a foremost multi-component application, CCSM, and a comprehensive system simulator for modeling the characteristics of multiple batch systems. By means of large number of simulations with different application and system configurations, we showed that multiple batch executions can reduce the queue waiting times of CCSM jobs by an average of 60% and increase the throughput of CCSM application by an average of 55% over single batch executions. We also analyzed in detail, the impact of request sizes of CCSM jobs, inter-cluster bandwidths, characteristics of external workloads on the batch systems, and queue scheduling policies on the benefits because of multiple batch executions. We found that small request sizes of CCSM jobs and presence of large number of narrow and short jobs in the external workloads lead to larger benefits, whereas inter-cluster bandwidths and queue scheduling policies do not impact the benefits because of multiple batch executions. We also conducted real experiments with a practical middleware infrastructure and showed that multi-site executions lead to effective utilization of batch systems for executions of CCSM and give higher simulation throughput than single-site executions.

In this work, we had analyzed the benefits of multiple batch executions when the number of processors are not changed from single batch executions. We plan to analyze the additional benefits because of increase in the number of processors. We plan to develop more scheduling and processor allocation strategies for multiple batch executions and compare the benefits using real executions on different batch systems. We also plan to develop efficient rescheduling strategies for migration of the components to different systems in response to resource and application dynamics.

ACKNOWLEDGEMENTS

This work is supported partly by Department of Science and Technology, India, project ref no. SR/S3/EECE/59/2005/8.6.06 and partly by Ministry of Information Technology, India, project ref no. DIT/R&D/C-DAC/2(10)/2006 DT.30/04/07.

REFERENCES

1. Mueller C, Dalkilic M, Lumsdaine A. High-performance direct pairwise comparison of large genomic sequences. *IEEE Transactions on Parallel and Distributed Systems* 2006; **17**(8):764–772.
2. Espinal X, Barberis D, Bos K, Campana S, Goossens L, Kennedy J, Negri G, Padhi S, Perini L, Poulard G, *et al.* Large-Scale ATLAS Simulated Production on EGEE. *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, Bangalore, India, 2007; 3–10.
3. An C, Taufer M, Kerstens A, III CB. Predictor@Home: a p̈rotein structure prediction supercomputer' based on global computing. *IEEE Transactions on Parallel and Distributed Systems* 2006; **17**(8):786–796.
4. Gardner M, chun Feng W, Archuleta J, Lin H, Mal X. Parallel genomic sequence-searching on an ad-hoc grid: experiences, lessons learned, and implications. *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, Florida, USA, 2006; 104.
5. Jayawardena M, Holmgren S. Grid-enabling an efficient algorithm for demanding global optimization problems in genetic analysis. *E-SCIENCE '07: Proceedings of the Third IEEE International conference on e-Science and Grid Computing*, Bangalore, India, 2007; 205–212.
6. Community Climate System Model (CCSM). Available from: http://www.ccsm.ucar.edu.
7. Collins W, Bitz C, Blackmon M, Bonan G, Bretherton C, Carton J, Chang P, Doney S, Hack J, Henderson T, Kiehl J, Large W, McKenna D, Santer B, Smith R. The community climate system model: CCSM3, 1998.
8. Fox GC, Gannon D. Wokflow in grid systems. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1009–1019.
9. Yu J, Buyya R, Ramamohanarao K. Workflow scheduling algorithms for grid computing. In *Metaheuristics for Scheduling in Distributed Computing Environments*, Xhafa F, Abraham A (eds). Springer: Berlin, Germany, 2008. SBN: 978-3-540-69260-7.
10. Ramakrishnan L, Koelbel C, Kee Y-S, Wolski R, Nurmi D, Gannon D, Obertelli G, YarKhan A, Mandal A, Huang T, Thyagaraja K, Zagorodnov D. VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance. *Sc '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, Oregon, USA, 2009; 1–12.
11. Logs of real parallel workloads from production systems. Available from: http://http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.
12. Bucur A, Epema D. Scheduling policies for processor coallocation in multicluster systems. *IEEE Transactions on Parallel and Distributed Systems* 2007; **18**(7):958–972.
13. Bal H, Plaat A, Bakker M, Dozy P, Hofman R. Optimizing parallel applications for wide-area clusters. *Proceedings of the International Parallel Processing Symposium*, Orlando, Florida, USA, 1998; 784–790.
14. Ernemann C, Hamscher V, Schwiegelshohn U, Yahyapour R, Streit A. On advantages of grid computing for parallel job scheduling. *Ccgrid '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, 2002; 39.
15. Fernandes R, Pingali K, Stodghill P. Mobile MPI programs in computational grids. *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, New York, USA, 2006; 22–31.

16. Wrzesinska G, Maassen J, Bal H. Self-adaptive applications on the grid. *PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, San Jose, California, USA, 2007; 121–129.
17. Vadhiyar S, Dongarra J. A performance oriented migration framework for the grid. *CCGRID '03: Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, 2003; 130.
18. The National Center for Atmospheric Research (NCAR). Available from: http://www.ncar.ucar.edu.
19. Lublin U, Feitelson D. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing* 2003; **63**(11):1105–1122.
20. Casanova H. Benefits and drawbacks of redundant batch requests. *Journal of Grid Computing* 2007; **5**(2):235–250.
21. Nurmi D, Mandal A, Brevik J, Koelbel C, Wolski R, Kennedy K. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, Florida, USA, 2006; 119.
22. Plaat A, Bal HE, Hofman R, Kielmann T. Sensitivity of parallel applications to large differences in bandwidth and latency in two-layer interconnects. *Future Generation Computer Systems* 2001; **17**(6):769–782.
23. Bucur A, Epema D. Trace-based simulations of processor co-allocation policies in multiclusters. *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, Washington, USA, 2003; 70.
24. Bucur A, Epema D. The performance of processor co-allocation in multicluster systems. *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, 2003; 302.
25. Bucur A, Epema D. The maximal utilization of processor co-allocation in multicluster systems. *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, Nice, France, 2003; 60.1.
26. Sivagama Sundari M, Vadhiyar S, Nanjundiah R. Grids with multiple batch systems for performance enhancement of multi-component and parameter sweep parallel applications. *Future Generation Computer Systems* 2010; **26**(2):217–227.
27. Ko S, Jha S. Efficient runtime environment for coupled multi-physics simulations: dynamic resource allocation and load-balancing. *CCGRID 2010: The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Melbourne, Australia, 2010.
28. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K. Task scheduling strategies for workflow-based applications in grids. *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05) - Volume 2*, Cardiff, UK, 2005; 759–767.
29. Wieczorek M, Podlipnig S, Prodan R, Fahringer T. Bi-criteria scheduling of scientific workflows for the grid. *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, Lyon, France, 2008; 9–16.
30. Cactus Computational Toolkit Webpage. Available from: http://www.cactuscode.org.
31. Allen G, Angulo D, Foster I, Lanfermann G, Liu C, Radke T, Seidel E, Shalf J. The cactus worm: experiments with dynamic resource discovery and allocation in a grid environment. *International Journal of High Performance Computing Applications* 2001; **15**(4).
32. Gabriel E, Resch M, Beisel T, Keller R. Distributed computing in a heterogenous computing environment, EuroPVMMPI'98, Liverpool, UK, 1998.
33. Park K, Park S, Kwon O, Park H. MPICH-GP: a private-IP-enabled MPI over grid environments. *Proceedings of Second International Symposium on Parallel and Distributed Processing and Applications, ISPA04*, Hong Kong, China, Dec 2004; 469473.
34. Maassen J, Bal H. Smartsockets: solving the connectivity problems in grid computing, HPDC '07: Proceedings of the 16th International Symposium on High Performance Distributed Computing, Monterey, California, USA, 2007; 1–10.