Future Generation Computer Systems 26 (2010) 217–227

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Grids with multiple batch systems for performance enhancement of multi-component and parameter sweep parallel applications☆

Sivagama Sundari M.[a], Sathish S. Vadhiyar[a,*], Ravi S. Nanjundiah[b]

[a] Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India
[b] Centre for Atmospheric & Oceanic Sciences, Indian Institute of Science, Bangalore, India

ARTICLE INFO

Article history:
Received 15 August 2008
Received in revised form
6 August 2009
Accepted 7 August 2009
Available online 15 August 2009

Keywords:
Grids
Batch systems
Multi-component applications
Parameter sweep applications
Queue waiting times

ABSTRACT

In this work, we evaluate the benefits of using Grids with multiple batch systems to improve the performance of multi-component and parameter sweep parallel applications by reduction in queue waiting times. Using different job traces of different loads, job distributions and queue waiting times corresponding to three different queuing policies (FCFS, conservative and EASY backfilling), we conducted a large number of experiments using simulators of two important classes of applications. The first simulator models Community Climate System Model (CCSM), a prominent multi-component application and the second simulator models parameter sweep applications. We compare the performance of the applications when executed on multiple batch systems and on a single batch system for different system and application configurations. We show that there are a large number of configurations for which application execution using multiple batch systems can give improved performance over execution on a single system.

1. Introduction

Computational grids have been increasingly used for executing large scale scientific applications [1–11]. Most of the benefits of using grids are primarily due to the increase in the number of processors used for execution. In this work, we focus on yet another potential use of grids where the total processor space used for application execution is not increased. Specifically, we deal with grids with multiple batch systems (for brevity, we will refer to such grids as *batch grids*) and show that employing multiple batch systems can improve the response times of parallel applications than when using a single batch system.

Parallel batch systems provide space sharing of available processors among multiple parallel applications or jobs. These batch systems employ queues in which the incoming parallel applications are queued before allocation by a batch scheduler to a set of processors for execution. Thus a batch system is associated with a set of queues and a scheduling policy that selects a job from the queue and maps it to a set of processors. An application submitted to a batch system incurs additional time for waiting in a queue before actual execution. The overall response time of an application is the sum of its queue waiting time and execution time. Application with small processor requirements can be backfilled to the available processors and hence incur smaller queue waiting times than applications with large processor requirements. Thus the queue waiting times for applications are proportional to their processor requirements as illustrated in Fig. 1. The figure shows the average queue wait times for jobs with different processor requirements on an IBM SP2 system in SDSC (San Diego Supercomputer Center). The job traces were obtained from the logs maintained by Feitelson[12].

Consider a parallel application J needing P processors that consists of or can be decomposed into multiple sub-parallel applications or components, $J_1, J_2, \ldots, J_n$ with processor requirements, $P_1, P_2, \ldots, P_n$, respectively, such that $P = P_1 + P_2 + \cdots + P_n$. In this case, simultaneous submission of the multiple sub-parallel applications, $J_1, J_2, \ldots, J_n$, with small processor requirements to multiple batch systems of a batch grid can result in improved response times of the application over submitting the entire parallel application, J, with a large processor requirement to a single batch system. This is because the maximum of the queue waiting times in the former case can be lesser than the single queue waiting time in the latter case.

While this advantage of simultaneous submissions of the sub-applications to multiple batch systems is generally well understood [13], the actual improvements in response times of the applications depend on various factors, including decomposability

☆ 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.

* Corresponding author.
E-mail addresses: sundari@rishi.serc.iisc.ernet.in (Sivagama Sundari M.), vss@serc.iisc.ernet.in (S.S. Vadhiyar), ravi@caos.iisc.ernet.in (R.S. Nanjundiah).

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.08.009

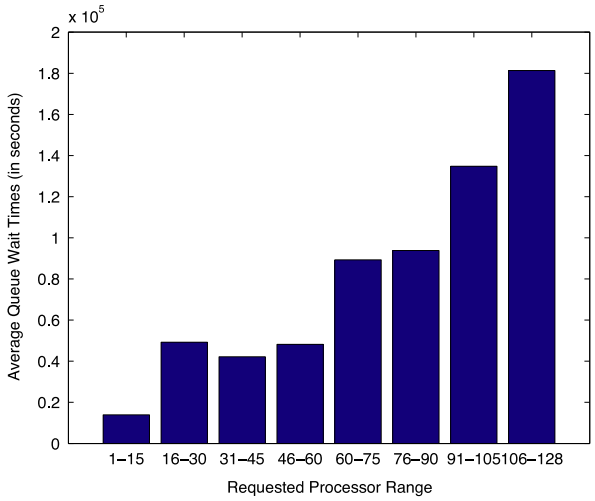

Fig. 1. Average queue wait times for jobs on a SDSC system.

of parallel applications, communication characteristics of the applications, speed of interconnections between the batch systems and batch queue characteristics. Multi-component MPMD applications [14,15] consists of component applications which are parallel application themselves. In these applications, the components are loosely synchronized and communications between components are lighter and less periodic than within components. These multi-component applications and parameter sweep applications, where the parallel tasks are independent, are amenable to decomposition into multiple parallel sub-applications and can potentially benefit due to submissions of the sub-applications to multiple batch systems. Batch queue characteristics including queuing and scheduling policies followed in the batch systems, the loads in the queues, the job distribution in terms of their processor requirements and maximum execution times associated with the queues impact the queue waiting times of the jobs in the queues.

In this paper, we study the effects of these different factors on the potential improvement in performance of a parallel application due to simultaneous submissions of the parallel sub-applications to multiple batch systems of a grid over submission of the entire parallel application to a single batch system. At the application level, we used two simulators, one that models the most prominent multi-component application, CCSM [14,15], and the other that models parameter sweep applications. At the network level, we simulated different interconnection speeds between the batch systems. At the batch level, we used different job traces produced from Feitelson's job models [16] and containing different distributions of jobs with different processor, execution time requirements and execution time limits. We then generated the queue waiting times of the jobs by using three different queuing policies, namely, FCFS, conservative and easy backfilling. We performed large number of simulations with different distributions of processors to components and system configurations with 24 different queues. We show that there are a large number of configurations for which performance improvements are obtained for the applications on batch grids. We further performed real experiments with CCSM by executing the components of CCSM across two AMD Opteron clusters and show similar benefits.

We assume that sub-components executed on different batch systems can communicate with each other. For simplicity, we evaluate the advantages of batch grids using two queues. Thus, our results show the comparison between executing the entire parallel application with P processor requirements on a single batch system and executing two sets of sub-components of the parallel application with processor requirements, $P1$ and $P2$ ($P = P1 + P2$), respectively, on two batch systems of a batch grid.

Section 2 motivates the use of batch grids. In Section 3, we explain in detail our simulation framework and calculations of probabilities of benefits of multiple batch executions for our two applications. Section 4 gives the simulation setup we used for our experiments. In Section 5, we present the results corresponding to CCSM simulations and real executions and simulations of parameter sweep applications. Section 6 describes related work on queue wait times of batch systems and executing applications on multiple batch grids. Conclusions are presented in Section 7 and future work is outlined in Section 8.

2. Batch grids — motivations and contributions

Application jobs submitted to batch systems incur overheads associated with the times spent in the batch queues waiting for resources to become available for execution. These queue waiting times tend to be higher when greater number of processors are requested. The central idea of this paper is to split the application into components and execute them on different batch systems. The smaller requests for the components on the different batch systems are expected to have lower wait times than the single complete request, leading to potential gains in the overall application execution rates. The focus of this paper is to study the incidence of this gain and its variation with different application and system factors. We have developed a simulator framework consisting of multiple components for studying the potential gains due to multiple batch executions for different application and system configurations.

The primary contributions of our work are:

(1) investigation of the benefits of execution of applications on batch grids over execution on single batch systems,
(2) development of a simulation framework, including development of an event-based batch system simulator, inter-site application execution model and a trace-based statistics calculator,
(3) definition of new probability metrics for comparison of batch grid vs. single batch system executions of long-running applications, and
(4) large scale analysis of incidence of benefits due to execution on batch grids for various system and application configurations.

The following section describes in detail our simulation framework.

3. Simulation methodology

We have developed a simulator framework consisting of multiple components for studying the potential gains due to multiple batch executions for different application and system configurations. Our simulation framework, shown in Fig. 2, consists of four components: (i) workload generator, (ii) application simulator, (iii) batch system simulator and (iv) statistics calculator. The workload generator is used to generate job traces with job arrival times, execution times and processor request sizes. The application simulator estimates the application execution rates for various intra-site and inter-site distributions of components. This is used in the calculation of our comparison metrics. The batch system simulator is our event-based simulator that processes the job traces produced by the workload generator and outputs the queue waiting time for each job. The new traces with the queue waiting times are used by the statistics calculator along with the application simulator, to estimate the probabilities of multiple-site executions outperforming single-site executions.

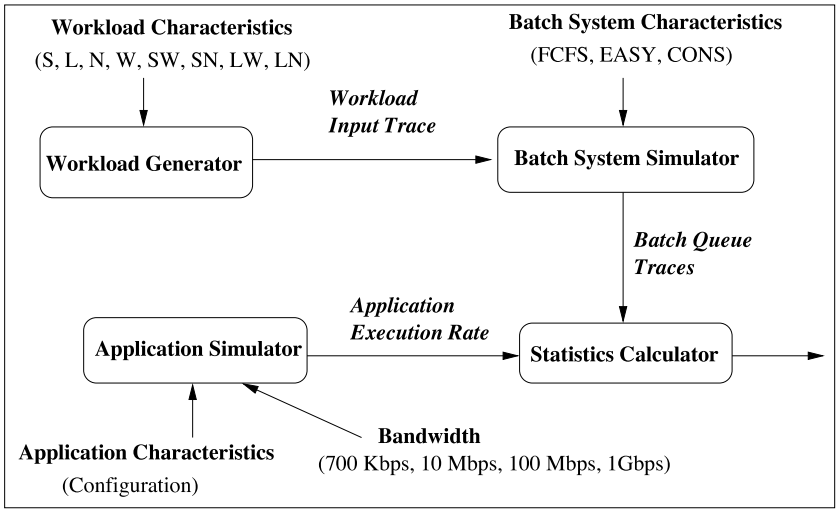

Fig. 2. Simulation framework.

The following subsections explain the various components of the simulation architecture.

3.1. Workload generator

For our simulation studies, we used the workload model developed by Lublin and Feitelson [16]. 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. The workload model generates a job trace consisting of arrival times, processor requirements and execution times of the jobs. Job processor requirements, runtimes and arrivals are modeled based on a two-stage uniform distribution, a hyper-Gamma distribution and a Gamma distribution, respectively. The model parameters are preset to values representative of real logs of supercomputers. The workload generator can be used to generate job traces with a range of processor requirements, execution times, and specific mean of inter-arrival times.

3.2. Batch system simulator

The job traces of the workload generator are input to the batch system simulator component of our simulator architecture. The batch system simulator uses the job traces along with a batch queue scheduling policy to generate *batch queue traces*. The batch queue traces contain information regarding queue waiting times of the jobs, namely, the times of submissions of the jobs to the queues and the start times of executions. We simulated three queue scheduling policies in the batch system simulator, namely, FCFS, conservative (CONS) and EASY backfilling [17], for scheduling jobs in the job traces and to generate the batch queue traces containing queue waiting times of the jobs. The queue waiting times in these traces are used by the statistics calculator described later.

3.3. Application simulator

In order to calculate statistics regarding the amount of work performed by an application in single and multiple batch systems, our simulator architecture performs simulations of application executions to estimate the amount of work for a given execution time. We have developed application simulators for a multi-component parallel application and parameter sweep applications.

3.3.1. Multi-component applications

Coupled multi-component applications [18,15,19,20] are an important class of scientific applications and have gained importance in recent times due to rapid advancements in computational science and multi-disciplinary simulations. One classic and foremost example of this application category is coupled climate models. In particular, we consider a coupled multi-component climate simulation application, the Community Climate System Model (CCSM) [14], an MPMD application developed by NCAR (National Center for Atmospheric Research). CCSM involves multiple components, four components corresponding to the four climate sub-systems: atmosphere, land, ocean and sea-ice, and a fifth component called coupler for coordinating the periodic communications between the other components.

For developing a simulator for CCSM, we executed CCSM on a cluster where the links of the cluster have a certain bandwidth denoted as *base intra-cluster bandwidth*. For the distribution of processors to the different components of CCSM for an execution on a given number of processors, we followed the general guidelines [21,22] and processor restrictions for the components. These restrictions include even number of processors for atmosphere and more than one processor for ice component. We observed the component computation times between their coupling period (period between communications with the coupler) and the component communication times with the coupler for different executions of CCSM on different number of processors and different distributions of processors to components.

We simulated the execution time of CCSM across two parallel systems or clusters where the intra-cluster communication networks have bandwidths of *base intra-cluster bandwidth* and the two clusters are connected by a single link of a specified bandwidth, denoted as *inter-cluster bandwidth*. For modeling communications between two components located at different clusters, we scaled the component-coupler communication times, that were earlier observed from real executions on a cluster with *base intra-cluster bandwidth*, based on the specified bandwidth of the inter-cluster link. For example, for simulating CCSM execution on a 1 Mbps inter-cluster link based on observations of real execution on a cluster with Gigabit Ethernet network, we scaled down the component communication times with the coupler observed on the Gigabit network by a factor of 1000.

3.3.2. Parameter sweep applications

Parameter sweep applications (PSA) constitute yet another important class of scientific applications. Since the individual tasks

of these applications are independent and do not involve communications, these applications have been widely demonstrated and deployed on grid systems.

The simulator for parameter sweep applications considers the execution of an application consisting of P tasks. For simulation of the application when executed on two queues, we divided P into different combinations of $P1$ and $P2$ tasks such that $P1 + P2 = P$. Unlike in CCSM, the execution times of the parameter sweep application are same when executed on a single batch system and on multiple batch systems since the tasks in the different batch systems do not communicate. The bandwidth of the link connecting the two systems does not affect the execution times of the applications. Hence we do not model the execution times of the applications in our simulator. The difference between single batch and multiple batch executions of the parameter sweep applications is only in terms of the queue waiting times.

3.4. Statistics calculator

The statistics calculator uses the *batch queue traces* along with the estimates from the application simulator to calculate statistics for the applications. The calculations of the statistics depend on the kind of the application. In the following subsections, we explain the calculations for multi-component and parameter sweep applications.

3.4.1. Statistics for multi-component applications

CCSM is typically executed for long periods to simulate climate systems for multiple centuries. The execution times for such runs can be several weeks. However, batch queue systems have execution time limits of few days for job executions. CCSM provides restart facilities where simulations for an execution can be continued from the previous executions. Hence, for multi-century simulations, CCSM is submitted to a batch system and resubmitted to the system after the previous batch execution. In order to compare the execution of such long-running CCSM when executing all components in a single batch system and when executing the components in multiple batch systems, response time is not a suitable metric for evaluation since it is used for jobs that can complete execution within the maximum execution limit of a batch system. We use simulated time per wall clock time, *SPW*, as a metric for comparison. For CCSM, *SPW* is the number of simulated days for a particular batch submission and is obtained by dividing the number of days, *simulatedDays*, that can be simulated by CCSM within the maximum execution time limit for a CCSM job in a queuing system, by the sum of queue waiting time for the job, *qwait*, and the maximum execution time limit for the jobs in the queue, *execTimeLimit*.

$$SPW = \frac{simulatedDays}{(qwait + execTimeLimit)}. \quad (1)$$

Thus, the *SPW* metric accounts for the amount of effective work performed in CCSM by considering the queue waiting time and the communication costs for multiple batch executions.

We used different configurations of CCSM and different queue traces for comparing single and multiple batch executions of CCSM. Each CCSM configuration corresponds to a total number of processors, P, for CCSM execution and distribution of the P processors to the components. For each queue trace, we used a single queue with the queue trace characteristics for a single batch execution and two queues with the same queue trace characteristics for multiple batch executions of different components of CCSM. For each CCSM configuration with P total number of processors and for each queue trace, we determine the best decomposition of the application with P processor requirements into two sub-applications with processor requirements, $P1_{best}$ and $P2_{best}$, such that $P1_{best} + P2_{best} = P$. We then calculate the probabilities that multiple batch executions for the best decomposition with $P1_{best}$ and $P2_{best}$ processor requirements, will provide benefits over single batch executions with P processor requirements. For determining the best decomposition for multiple batch execution with P total number of processors, we use *SPW* values as follows.

We first calculate *SPW* for the single batch execution, *SSPW*, using Eq. (1). The maximum execution time limit for the queue, *execTimeLimit*, used in the equation is the input to the workload generator of our simulator framework. The number of days, *simulatedDays*, that can be simulated by CCSM within the maximum execution time limit in the equation is estimated for P processors using the CCSM application simulator described in Section 3.3.1. For obtaining the queue waiting time, *qwait*, in Eq. (1) for a submission with P processor requirements, we used the average of queue waiting times in the queue trace for jobs using P processors.

For multiple batch executions with a given CCSM configuration of P total number of processors, and a given queue trace, we evaluate different combinations of locations of the components in the two queues. A particular combination of locations of CCSM components in the two queues corresponds to a particular decomposition of total number of processors, P, into processor requirements, $P1$ and $P2$, for the two queues, based on the number of processors allocated to each component in the CCSM configuration. For this particular decomposition with $P1$ and $P2$ processor requirements, we calculate *SPW* for the multiple batch execution, *MSPW*, using Eq. (1). Similar to the calculation of *SSPW*, the *execTimeLimit*, used in the equation is the input to our workload generator and *simulatedDays* in the equation is estimated for multiple batch execution with $P1$ and $P2$ processor requirements on the two queues using the CCSM application simulator described in Section 3.3.1. For obtaining the queue waiting time, *qwait*, in Eq. (1) for multiple batch submissions on the two queues, with processor requirements, $P1$ and $P2$ ($P = P1 + P2$), we obtained the average of queue waiting times for the jobs using $P1$ processors and the average of queue waiting times for the jobs using $P2$ processors in the queue trace and used the maximum of the two averages. The maximum of the two average queue waiting times is considered because the application can start execution only after both the sub-applications are dequeued from their respective queues and allocated processors for execution by their respective batch schedulers. We then obtained the gain, *MGAIN*, due to multiple batch submissions for the particular decomposition, ($P = P1 + P2$), as the difference between *MSPW* and *SSPW* values. For a given CCSM configuration with P processors, we experimented with different decompositions, and chose the decomposition, $P = P1_{best} + P2_{best}$, for which *MGAIN* is maximum.

For a given CCSM configuration requiring P total number of processors, a given queue trace and the best decomposition with $P1_{best}$ and $P2_{best}$ processor requirements on the two queues for multiple batch executions, we calculate the probability for multiple batch executions to provide improvements over single batch executions. We obtain the probability instead of an average measure because queue waiting times do not follow a strict non-decreasing relationship with the number of processor requirements. In some cases, jobs with smaller processor requirements can incur higher queue waiting times than jobs with larger processor requirements. The probability is calculated as follows. We divided the queue waiting times of the jobs with P processor requirements in the queue trace into 10 intervals, $[L_1, U_1], [L_2, U_2], \ldots, [L_{10}, U_{10}]$. For each interval, i, we calculate two probabilities, $p_1(i)$ and $p_2(i)$, corresponding to single and multiple batch executions, respectively. $p_1(i)$ is the probability that the queue waiting time of a job with P processor requirements is in the interval $[L_i, U_i]$. $p_2(i)$ is the probability that the queue waiting time of the jobs with requirements of $P1_{best}$ or $P2_{best}$ processors is less than L'_i. L'_i is the queue waiting time for the multiple batch execution that will result in *MSPW*

value equal to the *SSPW* value calculated using L_i as the queue waiting time for single batch execution. We calculate the joint probability, $p(i) = p_1(i) \times p_2(i)$, for multiple batch submissions to provide benefits over single batch submissions if the queue waiting time of the single batch submission with P processor requirements is in the interval $[L_i, U_i]$. We then obtain the total probability for a multiple batch submission to provide benefits over single batch submission by adding the probabilities $p(i)$ for all intervals. In these calculations, to obtain the probability that a queue waiting time for a job with a given processor requirement, N, is in an interval or less than the lower bound of the interval, we used the queue trace and divided the number of jobs with N processor requirements and whose queue waiting times satisfy the condition by the total number of jobs with N processor requirements.

3.4.2. Statistics for parameter sweep applications (PSA)

To obtain probabilities for the parameter sweep applications (PSA), we experimented with different configurations of the application corresponding to different number of tasks. For each configuration with P number of tasks and a given queue trace, we decomposed the P tasks into the best decomposition with $P1_{best}$ and $P2_{best}$ ($P1_{best} + P2_{best} = P$) tasks for multiple batch executions on two queues, such that the difference between the queue waiting times for the single batch and multiple batch submissions is minimum for the best decomposition. We then calculate the probability that decomposing the P tasks into $P1_{best}$ and $P2_{best}$ tasks using the best decomposition and submitting to multiple batch systems will give benefit over submitting the entire application with P tasks to a single batch system using similar probability calculations used for CCSM application. We divided the queue waiting times with P processor requirements into 10 intervals, and calculated probabilities $p_1(i)$ and $p_2(i)$, corresponding to single and multiple batch executions, respectively, for an interval i. For PSA, $p_2(i)$ is the probability that the queue waiting time of the jobs with requirements of $P1$ or $P2$ processors is less than the lower bound, L_i, of the queue waiting time for the interval. We then calculate the joint probability, $p(i) = p_1(i) \times p_2(i)$, for interval, i, and obtain the total probability, across all intervals, for a multiple batch submission to provide benefits over single batch submission.

For CCSM application, as described in the previous section, the probability of multiple batch executions providing benefits depends on both the difference in queue waiting times between multiple and single batch submissions, and the cost of communications between the two queues. However, for PSA, the probability depends only on the difference in queue waiting times between multiple and single batch submissions since PSA does not involve communications between the tasks. Thus a reduction in queue waiting times will lead to reduced response time in multiple batch executions for any parameter sweep application.

4. Simulation setup

In this section, we describe the specific parameters we used for our simulations using the simulator architecture.

In our workload generator, we specified the maximum processor requirement of the jobs as 128 processors and maximum execution time of 2 days. 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 8 job categories, namely, S, L, N, W, SN, SW, LN, and LW. The mean inter-arrival times of the jobs in our queue traces were 3000 s. We thus generated 24 different queue traces corresponding to 8 job characteristics and 3 scheduling policies. For all the queue traces, we fixed the maximum execution time limit of the jobs as 2 days.

For CCSM application simulator, we executed CCSM with 2X2.5_gx1v3 resolution and finite-volume (FV) dynamical core for atmosphere component on a 48-core AMD Opteron cluster consisting of 12 2-way dual-core AMD Opteron 2218 based 2.64 GHz Sun Fire servers with CentOS 4.3 operating system, 4 GB RAM, 250 GB Hard Drive and connected by Gigabit Ethernet. For our simulations of CCSM using two batch submissions, we used bandwidths of 700 Kbps, 10 Mbps, 100 Mbps and 1 Gbps on the links connecting the two submissions. The first three 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. Simulating this scenario is important since we claim that submitting multiple sub-applications of a single application to the same batch queue can also lead to benefits.

For obtaining statistics for CCSM, we used 50 configurations of CCSM corresponding to different total number of processors and different distribution of processors to components. For obtaining statistics for parameter sweep applications, we experimented with different configurations of the application corresponding to different number of tasks (2 to 128).

5. Results

In this section, we show results corresponding to simulations of CCSM and parameter sweep applications for different queue traces and bandwidths between two batch systems. We also show results for a real inter-cluster configuration where we obtained actual execution times of CCSM on a real configuration and used these times with our queue traces. All the results show probabilities for benefits with executing on two batch systems over executing on a single system. The probabilities were obtained using the calculations described in Section 3.

5.1. Simulations of multi-component applications

Fig. 3 shows the fraction of configurations corresponding to different probabilities of benefits with multiple batch executions for the four bandwidths. The total number of configurations were 1200 corresponding to 50 CCSM configurations and 24 queue traces for the four different bandwidths. The inter-cluster bandwidths are 1 Gbps, 100 Mbps, 10 Mbps and 700 Kbps, and are typical of links connecting two submissions to a single batch queue, two clusters in a single site, two different sites in a country, and two different sites located in two different continents, respectively.

We find that there are large percentage of configurations with significant probabilities of benefits with multiple batch executions. 50% of the total configurations have about 58% probability of obtaining benefits when executed across two batch queues for inter-cluster connection bandwidth of 100 Mbps. About 48% of the total configurations have about 50% probability for inter-cluster connection bandwidth of 10 Mbps. These two bandwidths correspond to queues located in different locations of a country. We also find that even for batch grids with 700 Kbps interconnection bandwidth corresponding to the two batch queues located in different continents, 20% of the configurations have up to 38% probability of obtaining benefits with multiple batch executions. For 10 Mbps, 100 Mbps, and 1 Gbps interconnection bandwidths, significant

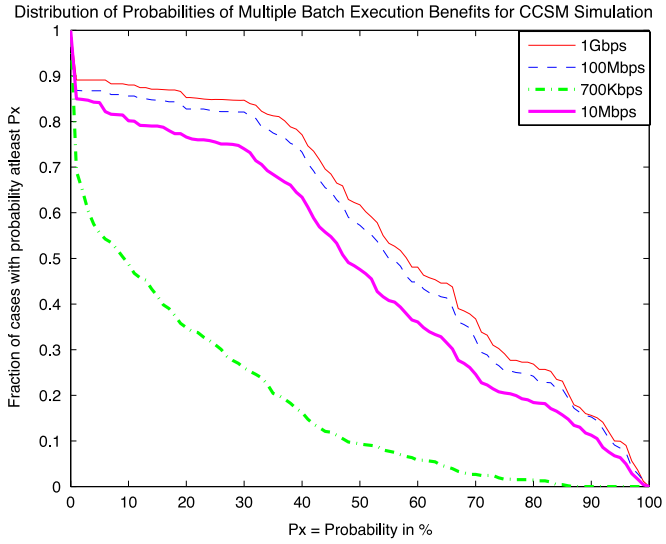

Fig. 3. Fraction of configurations with probabilities of benefits for CCSM simulations.

percentage of configurations (about 15%) have close to 100% probability and hence have definite chance of obtaining benefits with executions on batch grids.

In order to understand the effects of the system parameters including queue scheduling policies and job types, we show the probabilities for all 1200 application and system configurations for 100 Mbps inter-cluster bandwidth in Fig. 4. This figure gives an indication of the most frequent probabilities for each of the two system parameters. Fig. 5 shows the average probabilities obtained for the 50 application configurations for each of the 24 system configurations.

In general, we find that the probability of benefits increases when the queues have predominantly short and narrow (SN) jobs. This is because in such queues, at any given time, most of the available processor space will be occupied by the narrow jobs. The remaining processor space will not be sufficient for accommodating the larger processor requirements for single batch executions leading to large queue waiting times. However multiple batch executions with smaller processor requirements incur small queue waiting times in these cases since jobs with smaller processor requirements have better chances of finding the required processors on completion of other short and narrow jobs than the jobs with larger processor requirements. However, the probabilities of benefits with multiple batch executions on queues with predominantly long wide jobs (LW) are small since these wide jobs delay the executions of CCSM for long durations equally on both single and multiple batch queues in spite of decompositions in multiple batch executions. The extra cost of inter-cluster communications in multiple batch executions result in less benefits for the application. We also find that for queues with backfilling policies, the probabilities of benefits with multiple batch executions are higher. This is because the sub-applications in multiple batch executions with small processor requirements have better chances of backfilling in these queues than the single application with total processor requirements.

5.2. Real executions of multi-component applications

We also executed CCSM on real experiment testbed with 28 configurations and obtained real execution times. We then used these 28 real application execution times with the queue waiting times of the 24 queue traces to calculate probabilities. Our real

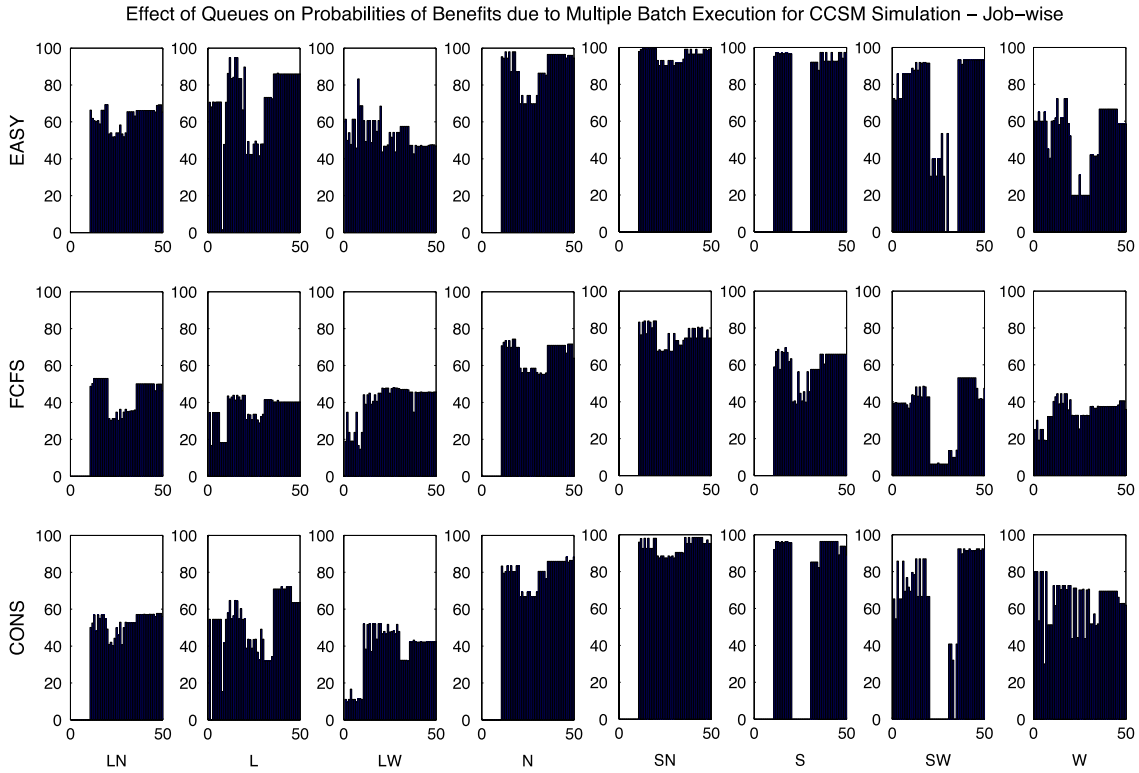

Fig. 4. Probability of benefits with multiple batch submissions for all application and system configurations for CCSM Simulations. The *x*-axis shows different application configurations and different job characteristics.

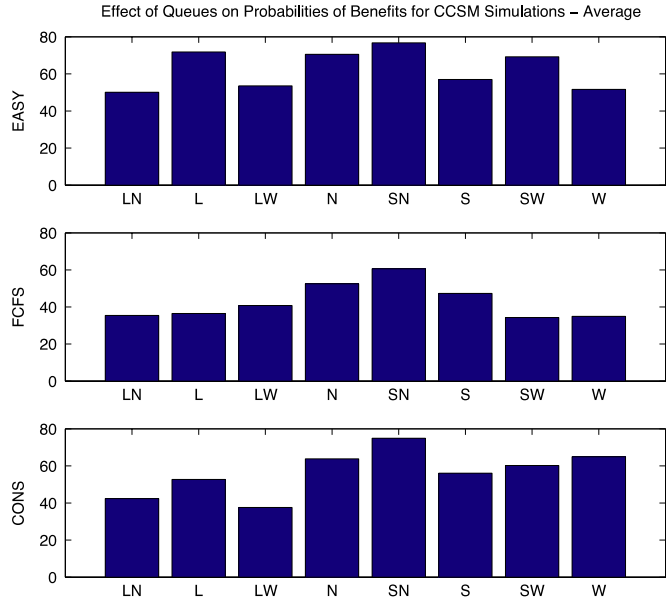

Fig. 5. Average probability of benefits with multiple batch submissions for different system configurations for CCSM simulations.

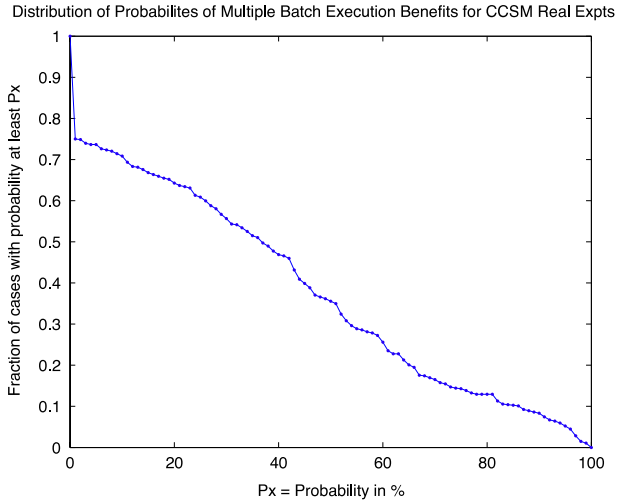

Fig. 6. Percentage of configurations with probabilities of benefits for CCSM real executions.

experiment testbed consists of a 16-core cluster called fire-16 and a 48-core cluster called fire-48. The fire-16 cluster consists of 8 dual-core AMD Opteron 1214 based 2.21 GHz Sun Fire servers, CentOS release 4.3, 2 GB RAM, 250 GB Hard Drive and connected by Gigabit Ethernet.The fire-48 cluster consists of 12x2 dual-core AMD Opteron 2218 based 2.64 GHz Sun Fire servers, CentOS release 4.3, 4 GB RAM, 250 GB Hard Drive and connected by Gigabit Ethernet. The Gigabit clusters are connected to each other by a 100 Mbps Ethernet connection through a 100 Mbps switch. For the single batch execution, we executed the 28 configurations on the fire-48 cluster. For multiple batch executions, we randomly distributed the different components of CCSM to the two clusters.

Figs. 6–8 show the results obtained for the real executions of CCSM. The results obtained with real executions of CCSM are similar to the simulation results shown in Figs. 3–5. The real results also show large number of application and system configurations with significant probabilities of benefits with multiple batch submissions. The relationship between the fraction of the configurations and the gain probabilities follows the same trend as that of the simulated experiments with the same bandwidth. We also find in real experiments that the probabilities improve when backfilling scheduling policies are used. Similar to the simulation results, the probabilities are generally high for short narrow (SN) jobs and generally low for short wide (LW) jobs for FCFS and CONS scheduling policies.

5.3. Simulations of parameter sweep applications

Figs. 9–11 show the results obtained for the simulations of different configurations of parameter sweep application corresponding to different total number of tasks of the application. The total number of tasks were varied from 2 to 128.

Similar to earlier results, we find large number of configurations where significant benefits were obtained with multiple batch submissions. We also find that backfilling scheduling policies lead to increase in probabilities of benefits since the decomposed tasks with small processor requirements in multiple batch executions are able to backfill more than the application with large processor requirements in single batch executions. Unlike in CCSM, multiple batch executions gave high probabilities of benefits for queues containing predominantly wide jobs (LW, SW and W) for PSA. In most of the PSA configurations, the total number of tasks were divided equally across the two batch systems. Hence the processor requirements in a single batch system for multiple batch submissions are larger in PSA than in CCSM. The larger processor requirements in PSA can be frequently met by the completion of wide jobs than by the completion of narrow jobs. Hence higher benefits with multiple batch submissions were obtained in queues with large number of wide jobs than in the queues with large number of narrow jobs where the processor space is mostly occupied by the narrow jobs.

Our primary focus in our experiments is on the incidence of benefits due to multiple batch executions rather than on the measure of benefits. This is because the benefits are spread over a wide range of both positive and negative values due to the randomness involved in the queue waiting times. Thus, metrics like mean or median of the reductions in total response times are non-representative and over-approximate the overall benefits due to multiple batch executions. The magnitude of gains due to multiple batch executions is of the order of a few hours. For many CCSM and PSA configurations, the average reduction in queue waiting times due to multiple batch execution is around 20 h while the maximum gain is around 40 h. These gains are significant when compared with the maximum execution time limit of 48 h for a job submission. These results show the huge benefits that can be obtained due to reduced queue waiting times without increase in the number of processors used for execution in multiple batch executions.

5.4. Summary of results

The use of large number of processors available in grids with distributed sites to provide performance benefits, especially for loosely-coupled applications, is well studied and understood. We have shown in our results that grids can also provide benefits when the number of processors used for application execution is not increased from non-grid, single-site systems and for both loosely-coupled and coupled multi-component applications with periodic communications between the sites. The benefits are due to decompositions of the applications into multiple sub-jobs with smaller processor requirements for executions on multiple batch systems of a grid and the corresponding reduction in queue waiting times incurred by the sub-jobs. Our results show that there are large number of application and system configurations for which grid executions can provide benefits even when the distributed

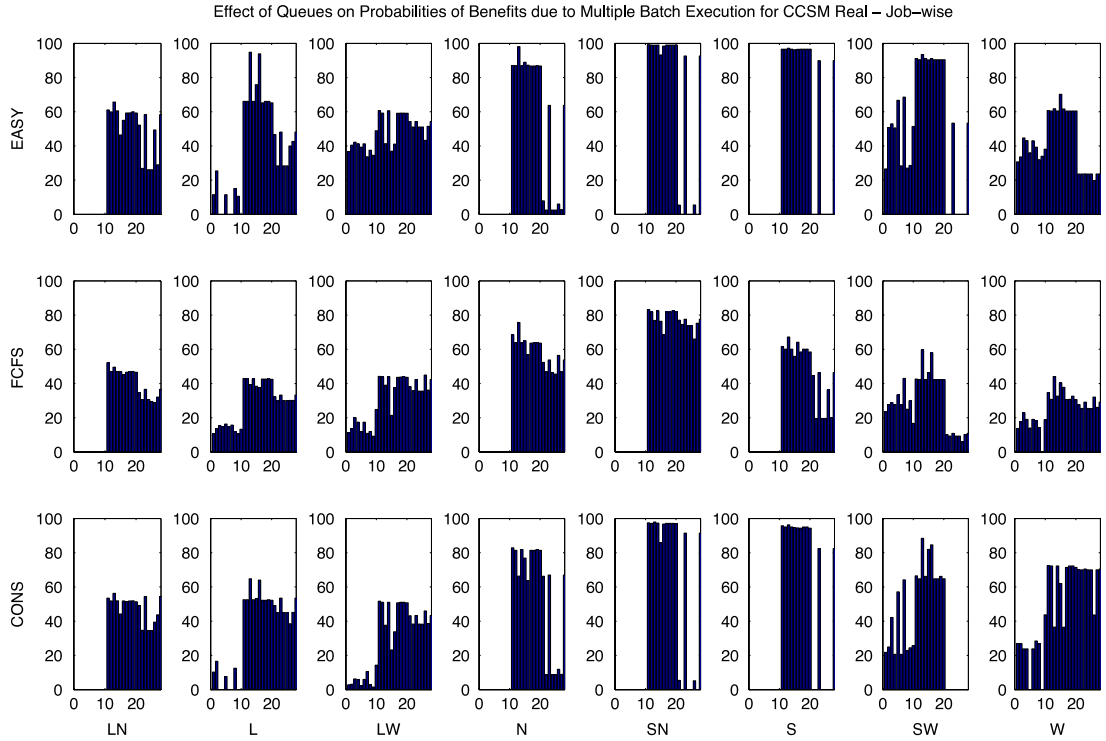

Fig. 7. Probability of benefits with multiple batch submissions for all application and system configurations for CCSM real executions.

sites are located in different continents. While the probability of benefits due to multiple batch executions are about 50% for about 50% of the configurations in general, there is a significant fraction of configurations for which the probabilities are close to 100%. The presence of large number of short narrow jobs (SN) for CCSM leads to higher probabilities of benefits due to multiple batch executions. This is because the occupation of the processor space by SN jobs will lead to inadequate number of processors for execution of a large job in single batch execution while the remaining processor space will be sufficient for accommodating sub-jobs with small processor requirements in multiple batch executions. In general, our simulation results for CCSM correspond with the real executions. We found that backfilling scheduling policies, that are commonly used in many batch systems, will lead to large probabilities of benefits for both CCSM and PSA applications. This is because the sub-jobs with small processor requirements are able to backfill more in multiple batch executions than the jobs in single batch executions.

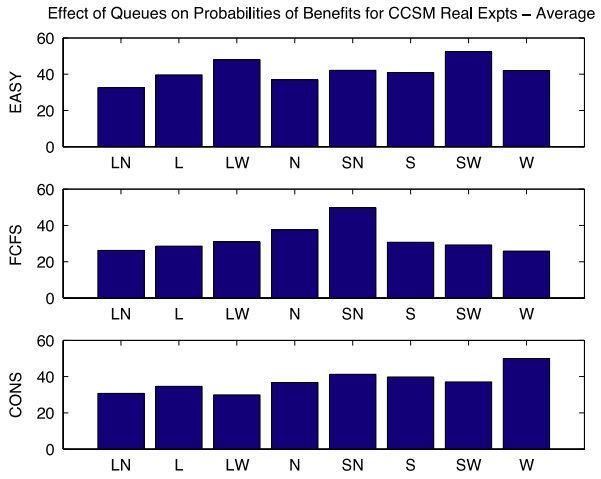

Fig. 8. Average probability of benefits with multiple batch submissions for different system configurations for CCSM real executions.

6. Related work

Multiple batch queues have been used for improving the response times of the jobs submitted to a system in the work by Subramani et al. [23]. In this effort, a job submitted to a system is redundantly submitted to other batch systems. When the job starts execution in one of the systems, the redundant jobs submitted to the other systems are aborted. Casanova analyzed the impact of such redundant submissions on the other jobs in the system [13]. The work concluded that such redundant tasks cause heavy load in the systems and unfairness to the users who do not use such redundant jobs. In our work, we do not replicate jobs on multiple batch queues. We decompose a single job into multiple sub-jobs and submit these sub-jobs to many batch queues.

Bucur and Epema have extensively studied the benefits of co-allocation of processors from different clusters in a grid for job executions [24–26]. They analyzed the benefits of such co-allocation for various workload logs, application characteristics, interconnections between multiple clusters and different scheduling policies. Using large number of simulations, they show that in spite of the high cost of WAN communications, execution of multi-component jobs across multiple clusters can reduce mean response times of jobs and improve processor utilization and that scheduling policies with only local queues perform better than those that consider global queues. They also conclude that restrictions on the number of components and component sizes help improve the performance of co-allocation. Our work is complementary to their efforts since we study the benefits obtained for two specific classes of applications. While they develop scheduling policies for co-allocation, our work deals with the queue waiting times corresponding to the existing scheduling policies on the clusters.

The work by Nurmi et al. [27] deals with execution of workflow applications on different batch systems of a grid. In their work,

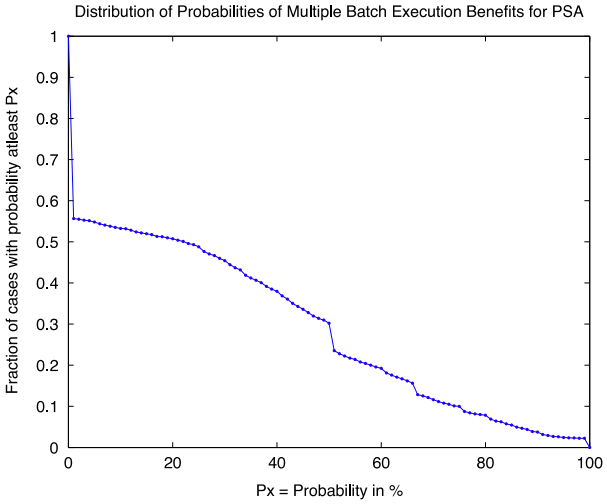

Fig. 9. Percentage of configurations with probabilities of benefits for PSA simulations.

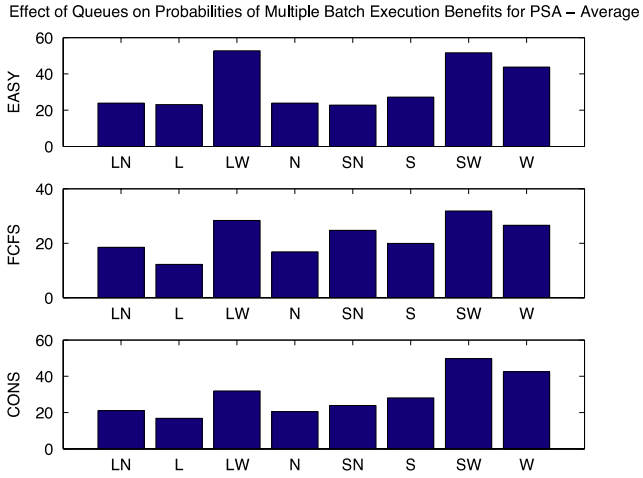

Fig. 11. Average probability of benefits with multiple batch submissions for different system configurations for PSA simulations.

they schedule different tasks of a workflow application to different batch systems of a grid based on predictions of execution times of the tasks on the systems and the queue waiting times in the systems [28]. They show that considering predictions of queue waiting times leads to efficient schedules. In our work, we consider multi-component applications that contain periodic communications between different batch systems unlike the workflow applications.

For distributed applications to make use of resources at different sites with heterogeneous characteristics and site autonomy, several middleware tools and services have been developed [29–34]. While most of them are limited to specific projects, resource types and middleware environments, or do not support co-allocation, the MetaScheduling Service (MSS) developed by Waldrich et al. [35] overcomes these limitations using the metacomputing-enabled MPI-implementation MetaMPICH [36]. In some recent works like the Grid Interoperability Project [37] and Gridbus Grid Service Broker [38], solutions have been developed for accessing resources from different grids. Kertesz et al. [39] have developed a meta-brokering architecture that enables the interoperability of various grids through their own resource brokers.

The work by Grimme and Papaspyrou [40] has built a service-oriented grid infrastructure to support workflow-based scientific applications for climate science. The framework performs scheduling and data management of loosely-coupled workflow tasks executed on distributed sites. The primary aim of the work is to provide coherent access to distributed data. Our work focuses on execution of multi-component applications on grids where the communications between the components are more frequent and intensive than the workflow applications. Elmroth and Tordsson [41] have developed a grid resource manager for performing resource brokering and job scheduling. The objective of job scheduling is to use the job and resource characteristics to minimize the response time that includes the times for file staging,

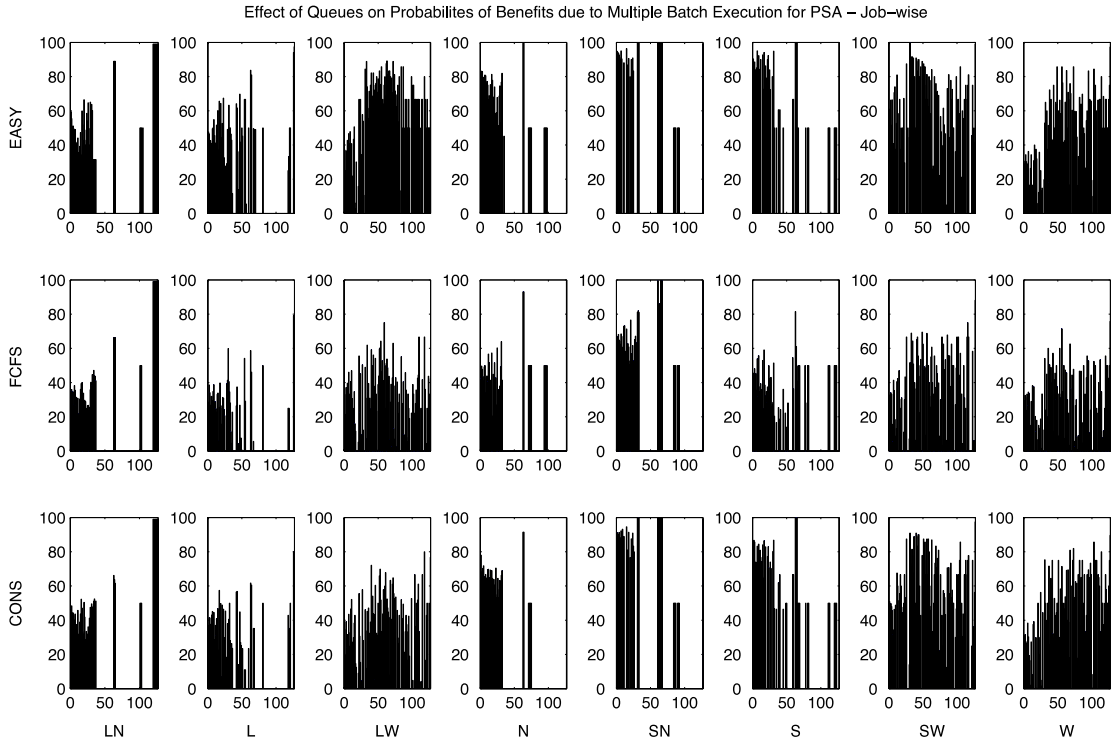

Fig. 10. Probability of benefits with multiple batch submissions for all application and system configurations for PSA simulations.

batch queuing and job execution. The work uses advanced reservations for co-allocation of resources. In our execution model, the components are co-allocated dynamically without reservations. Moltó et al. [42] have developed generic multi-user resource brokering and metascheduling techniques for remote execution of scientific applications. The metascheduling in our work is targeted for the specific application domain of climate modeling.

To our knowledge, our work is the first effort in quantitatively showing the benefits of executing a parallel application across multiple batch systems over executing the application on a single batch system without increasing the number of processors.

7. Conclusions

In this work, we have developed a simulator framework to analyze the potential benefits due to multiple batch executions on batch grids for multi-component and parameter sweep applications. The potential benefits are due to reduced queue waiting times on multiple batch executions. By performing simulated and real executions with large number of application and system configurations, we have shown that even without an increase in the number of processors, applications can gain from execution on multiple batch grids due to lower queue waiting times corresponding to the lower processor requirements on individual sites of the batch grid. Specifically, there are large percentages of configurations with significant probabilities of benefits with multiple batch executions for both multi-component CCSM simulations and parameter sweep applications. While the CCSM simulations showed higher benefits on queues with predominantly narrow jobs, the PSA simulations showed higher benefits on queues with predominantly wide jobs. We also found that for queues with backfilling policies, the probabilities of benefits with multiple batch executions are higher.

8. Future work

As part of our future work, we plan to develop robust models for queue waiting times. We also intend to extend our studies to include larger number of batches instead of two batches and consider various possibilities of heterogeneity of queues across the resultant batch grid. While MPI (Message Passing Interface) communication libraries including PACX-MPI [43] and MPICH-GX [44,45] 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, coordinated execution of an application across two batch systems with different startup times would require additional middleware infrastructure. We plan to build such middleware infrastructure for real executions of CCSM across two batch systems.

References

[1] W. Chrabakh, R. Wolski, GridSAT: A chaff-based distributed SAT solver for the grid, in: SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, 2003.

[2] C. Mueller, M. Dalkilic, A. Lumsdaine, High-performance direct pairwise comparison of large genomic sequences, IEEE Transactions on Parallel and Distributed Systems 17 (8) (2006) 764–772.

[3] S. Dong, N. Karonis, G. Karniadakis, Grid solutions for biological and physical cross-site simulations on the teragrid, in: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, 2006.

[4] X. Espinal, D. Barberis, K. Bos, S. Campana, L. Goossens, J. Kennedy, G. Negri, S. Padhi, L. Perini, G. Poulard, D. Rebatto, S. Resconi, A. de Salvo, R. Walker, Large-scale ATLAS simulated production on EGEE, in: E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007.

[5] L. Pearlman, C. Kesselman, S. Gullapalli, B.S. Jr., J. Futrelle, K. Ricker, I. Foster, P. Hubbard, C. Severance, Distributed hybrid earthquake engineering experiments: Experiences with a ground-shaking grid application, in: 13th IEEE International Symposium on High performance Distributed Computing, 2004.

[6] H. Takemiya, Y. Tanaka, S. Sekiguchi, S. Ogata, R. Kalia, A. Nakano, P. Vashishta, Sustainable adaptive grid supercomputing: Multiscale simulation of semiconductor processing across the Pacific, in: Proceedings of the ACM/IEEE Supercomputing Conference, SC 2006, 2006.

[7] C. An, M. Taufer, A. Kerstens, C.B. III, Predictor@Home: A p̈rotein structure prediction supercomputer' based on global computing, IEEE Transactions on Parallel and Distributed Systems 17 (8) (2006) 786–796.

[8] M. Gardner, W. chun Feng, J. Archuleta, H. Lin, X. Mal, Parallel genomic sequence-searching on an ad-hoc grid: Experiences, lessons learned, and implications, in: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006.

[9] M. Jayawardena, S. Holmgren, Grid-enabling an efficient algorithm for demanding global optimization problems in genetic analysis, in: E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007.

[10] C. Stewart, R. Keller, R. Repasky, M. Hess, D. Hart, M. Muller, R. Sheppard, U. Wossner, M. Aumuller, H. Li, D. Berry, J. Colbourne, A global grid for analysis of arthropod evolution, in: GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004.

[11] L. Han, A. Asenov, D. Berry, C. Millar, G. Roy, S. Roy, R. Sinnott, G. Stewart, Towards a grid-enabled simulation framework for nano-CMOS electronics, in: E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007.

[12] Logs of real parallel workloads from production systems. http://http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[13] H. Casanova, On the harmfulness of redundant batch requests, in: 15th IEEE International Symposium on High performance Distributed Computing, 2006.

[14] Community Climate System Model (CCSM). http://www.ccsm.ucar.edu.

[15] W. Collins, C. Bitz, M. Blackmon, G. Bonan, C. Bretherton, J. Carton, P. Chang, S. Doney, J. Hack, T. Henderson, J. Kiehl, W. Large, D. McKenna, B. Santer, R. Smith, The community climate system model: Ccsm3.

[16] U. Lublin, D. Feitelson, The workload on parallel supercomputers: Modeling the characteristics of rigid jobs, Journal of Parallel and Distributed Computing 63 (11) (2003) 1105–1122.

[17] D. Feitelson, A. Weil, Utilization and predictability in scheduling the IBM SP2 with backfilling, in: 12th Intl. Parallel Processing Symposium, IPPS, 1998.

[18] R. Delgado-Buscalioni, P. Coveney, G. Riley, R. Ford, Hybrid molecular-continuum fluid models: Implementation within a general coupling framework, Philosophical Transactions of the Royal Society London, Series A 363 (2005).

[19] G. Toth, O. Volberg, A. Ridley, T. Gombosi, D. DeZeeuw, K. Hanson, D. Chesney, Q. Stout, K. Powell, K. Kane, R. Oehmke, A physics-based software framework for sun–earth connection modeling, in: Multiscale Coupling of Sun–Earth Processes, Proceedings of the Conference on the Sun–Earth Connection.

[20] S. Lefantzi, J. Ray, A Component-based scientific toolkit for reacting flows, in: In Proc. Second MIT Conference on Computational Fluid and Solid Mechanics, 2003.

[21] G. Carr, An introduction to load balancing CCSM3 components, in: Proceedings of Software Engineering Working Group (SEWG) Meeting, CCSM Workshop, NCAR, 2005.

[22] G. Carr, M. Cordery, J. Drake, M. Ham, F. Hoffman, P. Worley, Porting and performance of the community climate system model (CCSM3) on the cray X1, in: Proceedings of the 2005 Cray Users Group (CUG) Meeting, Albuquerque, New Mexico, May.

[23] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, 2002.

[24] A. Bucur, D. Epema, Scheduling policies for processor coallocation in multicluster systems, IEEE Transactions on Parallel and Distributed Systems 18 (7) (2007) 958–972.

[25] A. Bucur, D. Epema, Trace-based simulations of processor co-allocation policies in multiclusters, in: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003, p. 70.

[26] A. Bucur, D. Epema, The maximal utilization of processor co-allocation in multicluster systems, in: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, 2003; 60.1.

[27] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, K. Kennedy, Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction, in: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006.

[28] J. Brevik, D. Nurmi, R. Wolski, Predicting bounds on queuing delay for batch-scheduled parallel machines, in: PPoPP '06: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2006.

[29] Load Sharing Facility, Resource Management and Job Scheduling System. Web site, http://www.platform.com/Products/Platform.LSF.Family/.

[30] A. Bose, B. Wickman, C. Wood, MARS: A metascheduler for distributed resources in campus grids, in: 5th International Workshop on Grid Computing, GRID 2004, IEEE Computer Society, 2004.

[31] J. Weinberg, A. Jagatheesan, A. Ding, M. Faerman, Y. Hu, Gridflow: Description, query, and execution at SCEC using the SDSC matrix, in: 13th IEEE International Symposium on High Performance Distributed Computing, HPDC'04, IEEE Computer Society, 2004.

[32] D. Thain, M. Livny, Building reliable clients and servers, in: I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2003.

[33] D. Katramatos, M. Humphrey, A.S. Grimshaw, S.J. Chapin, JobQueue: A computational grid-wide queueing system, in: Grid Computing — GRID 2001, Second International Workshop, in: Lecture Notes in Computer Science, vol. 2242, Springer, 2001.
[34] I. Foster, A. Roy, V. Sander, A quality of service architecture that combines resource reservation and application adaptation, in: 8th International Workshop on Quality of Service, IWQOS 2000, June 2000, pp. 181–188.
[35] O. Waldrich, Ph. Wider, W. Ziegler, A meta-scheduling service for co-allocating arbitrary types of resources, in: Proc. of the Second Resource Management Workshop (GRMWS'05) in conjunction with the Sixth International Conference on Parallel Processing and Applied Mathematics, PPAM 2005, Poznan, Poland, September 11–14, in: Lecture Notes in Computer Science, vol. 3911, Springer, 2006, pp. 782–791.
[36] M. Poppe, S. Schuch, T. Bemmerl, A message passing interface library for inhomogeneous coupled clusters, In Proc. of CAC Workshop at IPDPS'03, 2003.
[37] J. Brooke, D. Fellow, K. Garwood, C. Goble, Semantic Matching of Grid Resource Descriptions, in: Lecture Notes in Computer Science, vol. 3165, January 2004, pp. 240–249.
[38] S. Venugopal, R. Buyya, L. Winton, A grid service broker for scheduling e-science applications on global data grids, Journal of Concurrency and Computation: Practice and Experience 18 (2006) 599–685.
[39] A. Kertesz, P. Kacsuk, Grid meta-broker architecture: Towards an interoperable grid resource brokering service, in: CoreGRID Workshop on Grid Middleware in Conjunction with EuroPar'06, Dresden, 2006.
[40] C. Grimme, A. Papaspyrou, Cooperative negotiation and scheduling of scientific workflows in the collaborative climate community data and processing grid, Future Generation Computer Systems 25 (3) (2009) 301–307.
[41] E. Elmroth, J. Tordsson, Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions, Future Generation Computer Systems 24 (6) (2008) 585–593.
[42] G. Moltó, V. Hernández, J. Alonso, A service-oriented WSRF-based architecture for metascheduling on computational grids, Future Generation Computer Systems 24 (4) (2008) 317–328.
[43] E. Gabriel, M. Resch, T. Beisel, R. Keller, Distributed computing in a heterogenous computing environment, in: EuroPVMMPI'98, 1998.
[44] K. Park, S. Park, O. Kwon, H. Park, MPICH-GP: A private-IP-enabled MPI over grid environments, in: in Proc. of Second International Symposium on Parallel and Distributed Processing and Applications, ISPA04, Hong Kong, China, Dec. 2004.
[45] S. Choi, K. Park, S. Han, S. Park, O.-Y. Kwon, Y. Kim, H.-W. Park, An NAT-based communication relay scheme for private-IP-enabled MPI over grid environments, in: in Proc. of International Conference on Computational Science, 2004.

Sivagama Sundari M. received her BE(Hons) in Electronics and Instrumentation Engineering from the Birla Institute of Technology and Science, Pilani in 2005. She is now pursuing a Ph.D. at the Supercomputer Education and Research Centre at the Indian Institute of Science, Bangalore. Her current research interests include software architectures and performance of scientific applications, grid computing and high performance computing.

Sathish S. Vadhiyar is an Assistant Professor in Supercomputer Education and Research Centre, Indian Institute of Science. He obtained his B.E. degree from the Department of Computer Science and Engineering at Thiagarajar College of Engineering, India in 1997 and received his Masters degree from Computer Science at Clemson University, USA in 1999. He graduated with a Ph.D. from the Computer Science Department at University of Tennessee, USA in 2003. His research areas are in parallel and grid computing with primary focus on performance modeling of parallel applications, scheduling and rescheduling methodologies for grid systems, and grid applications. Dr. Vadhiyar is a member of IEEE and has published papers in peer-reviewed journals and conferences. He was a tutorial chair and session chair of escience 2007 and served on the program committees of conferences related to parallel and grid computing including IPDPS, CCGrid, eScience and HiPC.

Ravi S. Nanjundiah is a Professor at Centre for Atmospheric & Oceanic Sciences (CAOS), Indian Institute of Science (IISc). He obtained his B.E. degree in Mechanical Engineering from Rani Durgavati University (Jabalpur, India) in 1984, M.E. in Mechanical Engineering from IISc in 1986 and Ph.D. in Atmospheric Science from IISc in 1992. His research areas are study of study of monsoons – its variability and change using climate system models, and application of HPC/grid computing to climate system modelling. He has published papers in peer-reviewed journals and conferences. He is an associate editor of Journal of Earth System Science.