J Supercomput (2013) 63:367–384
DOI 10.1007/s11227-012-0741-6

Multi-domain job coscheduling for leadership computing systems

Wei Tang · Narayan Desai · Venkatram Vishwanath · Daniel Buettner · Zhiling Lan

Published online: 18 January 2012
© Springer Science+Business Media, LLC 2012

Abstract Current supercomputing centers usually deploy a large-scale compute system together with an associated data analysis or visualization system. Multiple scenarios have driven the demand that some associated jobs co-execute on different machines. We propose a multi-domain coscheduling mechanism, providing the ability to coordinate execution between jobs on multiple resource management domains without manual intervention. We have evaluated our mechanism based on real job traces from Intrepid and Eureka, the production Blue Gene/P system and a cluster with the largest GPU installation, deployed at Argonne National Laboratory. The experimental results show that coscheduling can be achieved with limited impact on system performance under varying workloads.

Keywords Coscheduling · Coupled system · Heterogeneous computing · Resource management

W. Tang (✉) · Z. Lan
Illinois Institute of Technology, Chicago, IL, USA
e-mail: wtang6@iit.edu

Z. Lan
e-mail: lan@iit.edu

N. Desai · V. Vishwanath · D. Buettner
Argonne National Laboratory, Argonne, IL, USA

N. Desai
e-mail: desai@mcs.anl.gov

V. Vishwanath
e-mail: venkatv@mcs.anl.gov

D. Buettner
e-mail: buettner@alcf.anl.gov

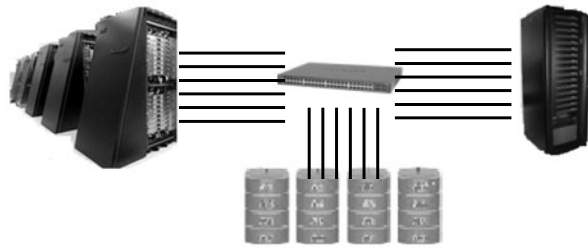


Fig. 1 Typical coupled HEC systems: a large-scale compute system (*left*) and a special data analysis/visualization system (*right*), which share a single storage system via network

1 Introduction

Leadership-class systems are providing unprecedented opportunities to advance science in numerous fields, such as climate sciences, biosciences, astrophysics, computational chemistry, materials sciences, high-energy physics, and nuclear physics. As the need for computation power grows, current high-end computing (HEC) systems are increasingly using heterogeneous processing elements in their designs. Meanwhile, the volume of data produced by high-end applications continues to grow, leading to an increasing demand for data analysis and visualization capabilities.

The trend has driven the deployment of so-called *coupled systems* [21], where a general-purpose compute system runs scientific computation or simulations, and is connected via a shared filesystem or over the network, to a special-purpose system used for data analysis and visualization. The former is generally a large-scale system; the latter is relatively small but usually equipped with special processing units such as GPUs. Thus, a common scenario on coupled systems is post-processing: a computational job runs on the compute system, generating output data written to the shared storage system; the data is then read by an associated job running on the analytics system conducting data analysis or visualization. Such systems are already common, with examples like Intrepid and Eureka at Argonne National Laboratory, Jaguar and Lens at Oak Ridge National Laboratory, Ranger and Longhorn at Texas Advanced Computing Center, and so on. Figure 1 depicts a typical configuration of coupled systems.

While post-processing is common, co-execution is increasingly demanded. One reason is that co-execution enables monitoring of simulations, debugging, and visual debugging of the simulation code at run time. Another reason is that co-execution can accelerate the I/O time [27]—one of the critical bottlenecks faced by simulations—by staging simulation I/O data to the memory of a coupled resource and avoiding writing data to persistent storage. This strategy requires that both compute and analysis applications be alive at the same time. Furthermore, computations in several scientific domains require access to heterogeneous resources to concurrently execute models wherein one or more models are tailored for GPU-based systems while others are optimized for CPU-based clusters. This kind of computation requires co-execution spanning heterogeneous coupled resources. Thus, many existing applications already can benefit from co-execution on coupled systems, and more are anticipated if job co-execution can be conveniently achieved.

The above scenarios motivate the need for coscheduling, which in this paper refers to the scheduling mechanism that guarantees that the associated jobs submitted to the

different machines in the coupled system can start at the same time. The challenge of coscheduling is mainly from the fact that each machine in the coupled system manages jobs with independent resource managers thus the scheduling domains are totally different. For example, when a job gets the highest priority to run on one system, the associated job on the other system may not be able to start because of low priority. Moreover, the jobs in pairs that need coscheduling are coexisting with regular jobs that need not. We want to minimize the impact of coscheduling on the regular jobs.

In this paper, we proposed a coscheduling mechanism that coordinates execution between jobs on different systems. Specifically, the coscheduling can guarantee that associated jobs (e.g., a compute job with the associated data analysis job) start simultaneously across the coupled systems. To this end, we have defined a light-weighted protocol to enable coscheduling of jobs across multiple resource management domains. We implemented our mechanism in an existing resource manager and evaluated it with event-driven simulations using real job traces from the production Blue Gene/P [4] and data analysis GPU cluster deployed at Argonne National Laboratory. Our experimental results demonstrate that coscheduling can be achieved with limited impact on system performance under varying workloads.

The remainder of this paper is organized as follows. Section 2 discusses some relevant work. Sections 3 and 4 describe our coscheduling design and implementation. Section 5 evaluates coscheduling performance via trace-based simulations. Section 6 summarizes the paper.

2 Related work

The term "coscheduling" is commonly used to denote a specific mechanism proposed for concurrent systems that schedule related processes to run on different processors at the same time [13]. It has a strict version named gang scheduling [28] and other versions or enhancements proposed in the literature, such as demand-based coscheduling [17], dynamic coscheduling [16], buffered coscheduling [14], and flexible coscheduling [8].

These mechanisms share a similar goal with our coscheduling: running related processes (in our case jobs) simultaneously. The difference is that the former are used for time-sharing systems belonging to a single scheduling domain while our coscheduling coordinates the execution of associated jobs across multiple scheduling domains (i.e., different resource managers with independent scheduling policies).

Advance resource co-reservation has been widely proposed to coordinate resource allocation across multiple systems. MacLaren [11] presented the Highly-Available Resource Co-allocator (HARC), a system for creating and managing resource reservations used for metacomputing applications [6] or workflow applications. Foster et al. [9] proposed GARA, a general-purpose architecture for reservation and allocation. Yoshimoto et al. [29] presented GUR, a system for coscheduling compute resources in a Grid computing environment using user-settable reservations similar to travel arrangements.

Co-reservation can be used to start related jobs on multiple systems at the same (reserved) time, however it is not suitable in coupled HEC systems. Firstly, since

the two coupled systems are administratively heterogeneous, co-reservations on both machines involve expensive manual efforts in policy negotiation. Secondly, excessive use of reservations will leave temporal fragmentations on the computing resources, thereby leading to worse response times for regular jobs [18]. Our work fits the coupled HEC environment well because it is free from manual intervention and will leave no temporal resource fragmentation.

Metascheduling [25] also involves scheduling jobs on multiple clusters. It aims at optimizing computational workloads by combining an organization's multiple distributed resource managers into a single, aggregated view, allowing batch jobs to be directed to the best location for execution. Existing work includes GridWay by Globus Alliance [10] and Moab by Adaptive Computing Inc. [12]. However, these schedulers require a global job submission portal. A unique feature of our work is that we remove the restriction of a global submission portal by enabling independent job submission on each resource.

The term coscheduling is also used in other problem domains with various objects, such as coscheduling of computation and data [15], coscheduling CPU and network capacity [2], and coscheduling CPUs and GPUs for heterogeneous computing [22]. The object of our coscheduling is associated jobs that need co-execution on different machines in coupled HEC systems.

A typical use case which can benefit from our coscheduling is the coupled applications in which computing and data analysis or visualization are conducted separately. For example, FLASH [23] is used to simulate buoyancy-driven turbulent nuclear burning; it uses vl3 [3] for visualization. If both applications are running simultaneously, they can exchange a large amount of data via the network instead of using a permanent storage system, thus accelerating the I/O time [26, 27]. Also, our coscheduling can work together with data staging optimizing work to facilitate the cooperation between simulation and analysis applications [1].

3 Coscheduling design

3.1 Problem statement

Suppose two systems (machines) A and B are running workloads of parallel jobs. Jobs on each system are managed by different resource managers R_1 and R_2, respectively. R_1 and R_2 use independent scheduling policies. Among all the jobs on the two systems, there exist some pairs of associated jobs. In such a job pair, one job is submitted to one system, and the other job to the other system. These two associated jobs need to be started at the same time even though they are scheduled separately on different systems.

We need a coscheduling mechanism to guarantee all the associated jobs in the same pair start at the same time without manual reservation. Meanwhile, the mechanism must limit the side effect on system utilization and the response times of both paired and nonpaired jobs.

3.2 Basic schemes

In job scheduling, we say that a job is "scheduled," or "ready," when a job is selected by the scheduler to start next. A scheduled job can have the highest priority or have been given the opportunity of backfilling [24]. The job should be assigned with a designated number of nodes. Normally, when a job is scheduled, it can start immediately. With coscheduling, however, a scheduled job may not be started because it may need to wait for its mate job (i.e., the associated job that needs to be started at the same time).

When a job is ready to run but its remote mate cannot, we have two basic schemes, "hold" and "yield," to choose from. With hold, when a job is ready but its mate job is not, it will hold the needed computing nodes, not allowing them to be used by others, until the remote mate gets ready in the future. With yield, it will give up the chance to run without occupying any nodes and allowing the scheduler to schedule another job.

3.3 Main algorithm

The core coscheduling algorithm extends the existing function in the traditional resource manager, which is invoked when a job is scheduled and ready to run. Normally, the function starts the scheduled job on the assigned nodes. But with coscheduling, additional logic will be executed before the job can actually start. Algorithm 1 describes this function, which we call *RunJob*, enhanced with coscheduling algorithm.

Resource managers on different machines run the same algorithm with local configuration (no centralized control). The algorithm can call either local functions or remote functions. As shown in Algorithm 1, "self." starts a local function and "remote." starts a remote function.

As shown in Algorithm 1, if coscheduling is not enabled or if the candidate job has no mate, the candidate job will start normally (lines 35 and 31, respectively), skipping the coscheduling logic. If coscheduling is enabled and a valid mate is found, we will first get the status of the mate, based on which the job will act. If the remote job is in hold status, both jobs will be started immediately (lines 7 and 8). If the mate job is waiting in the queue or unsubmitted, a remote function will be called to try to run the mate job. Function $remote.tryStartMate(k)$ (line 12) invokes an additional scheduling iteration on the remote machine and returns *True* only if the mate job k gets started. If the mate job is started, the local job is also started (line 14). If the mate job cannot run at this moment (including the unsubmitted case), the local job will either hold (line 17) or yield (line 20) according to the preconfigured local coscheduling scheme. Function $self.holdJob(j, N)$ sets job j to holding status and marks the nodes N busy, not allowing another job to use them. Function $self.yieldJob(j)$ invokes an additional local scheduling iteration to try to run other jobs. If the remote status is unknown, the algorithm continues to start the local job (line 26).

The algorithm is fault-tolerant: a job will not wait forever when the remote machine or its mate job is down or already gets started. If the remote system is down, line 2 will return nothing so that the ready job will start immediately. If the mate job fails alone or already gets started (by exception), the mate status will be returned as unknown (line 25). The ready job thus will start normally, too.

Algorithm 1: RunJob(j, N)

```
   Input: A scheduled job j with assigned nodes N
   Result: Job j either starts, or holds, or yields. Its remote mate job k, if existing,
           could be triggered to start under certain condition.
 1 if cosched_enabled then
 2     k = remote.getMateJobId(j)
 3     if k then
 4         mate_status = remote.getMateStatus(k)
 5         switch mate_status do
 6             case "holding"
 7                 self.startJob(j, N)
 8                 remote.startJob(k)
 9             end
10             case "queuing"
11             case "unsubmitted"
12                 mate_started = remote.tryStartMate(k)
13                 if mate_started then
14                     self.startJob(j, N)
15                 end
16                 else
17                     if self.scheme == "hold" then
18                         self.holdJob(j, N)
19                     end
20                     if self.scheme == "yield" then
21                         self.yieldJob(j)
22                     end
23                 end
24             end
25             case "unknown"
26                 self.startJob(j, N)
27             end
28         end
29     end
30     else
31         self.startJob(j, N)
32     end
33 end
34 else
35     self.startJob(j, N)
36 end
```

This algorithm only presents the abstract form of the communication protocol. The implementation may vary regarding how resource managers on different machines

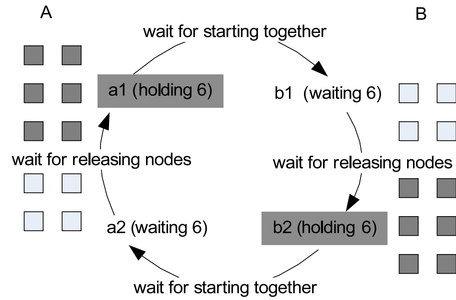


Fig. 2 Example of deadlock. Machine A has a job a_1 holding 6 nodes, waiting for its mate b_1 queuing on machine B and also requesting 6 nodes. But machine B has another job b_2 holding 6 nodes, waiting its mate a_2 queuing on machine A and requesting 6 nodes

communicate with each other to exchange job status information and invoke remote functions.

3.4 Scheme combinations

In order to apply coscheduling on the coupled systems, each machine must be preconfigured with a coscheduling scheme. To this end, we have identified four combinations of configuration: hold-hold, yield-yield, hold-yield, and yield-hold. We discuss how these combinations work to achieve coscheduling.

Hold-hold means using the hold scheme on both machines of the coupled system. In this setting, when one job of the paired job gets ready, it will enter hold status. Once the second job gets ready, both jobs can start immediately. Hold-hold is effective for synchronizing two jobs, but it may result in deadlock. Indeed, theoretically, hold-hold meets all of the four conditions causing deadlock: mutual exclusion (a node can be assigned only to a job), hold and wait (a job holds some nodes and waits until its mate is ready), no preemption (we do not have preemption), and circular wait (both machines use hold, so that circular wait is possible). Figure 2 shows a simple example of deadlock. Deadlock can be solved by forcing the holding jobs to release their resources periodically (e.g., every 20 minutes) so that other waiting jobs can use the previously held resources. This preemptive scheme can break circular wait.

Yield-yield means using the yield scheme on both machines of the coupled system. In this setting, the paired jobs can be started only when both get scheduled and assigned with sufficient nodes simultaneously. Before that, both jobs may alternately yield. But they can eventually reach the both-ready condition, because they will eventually get the highest priority on their respective machine if job priority increases by time. The most commonly used FCFS (first-come, first-served) policy [7] and some of its variations such as WFP [19] serve this purpose well.

Hold-yield and yield-hold use different schemes on different machines. In this setting, the goal of coscheduling can also be achieved. Suppose machine A uses hold and machine B uses yield. If job a on A gets ready first, it will hold; when its mate job b gets ready, they can both start immediately. If job b is ready first, it will yield; when job a gets ready, it checks whether b is ready. If it is, both start; if not, job a holds until job b gets ready for the next time.

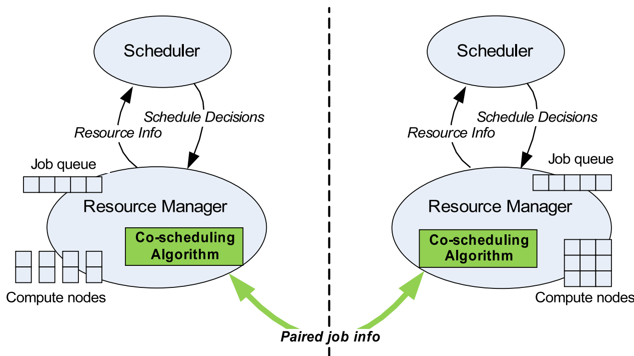


Fig. 3 Implementation of coscheduling in addition to existing resource management systems

4 Implementation

Because the newly added interfaces are simple and practical, our algorithms can be easily implemented in any resource manager. As an example, we have implemented the algorithm and strategies based on Cobalt [5], a production resource management system used for the Blue Gene system and common clusters. Cobalt is an open source project from Argonne National Laboratory.

Figure 3 presents the framework of the Cobalt resource management system with the coscheduling enhancements. In the figure, two Cobalt systems are illustrated, each for one resource management domain. The resource manager and scheduler are two legacy functionality components. The resource manager maintains job queues and the status of compute nodes; the scheduler makes scheduling decision at each scheduling iteration (e.g., every 10 seconds). The coscheduling algorithm described in Sect. 3 is implemented in the resource manager part, which is invoked after each job is scheduled by the scheduler. In order to exchange job status across multiple resource management domains, a remote manager component and corresponding interfaces need to be implemented. Moreover, the client for job submission needs to be extended to allow users to specify the mate job information in the submission script. Because of the lightweight prototype, this framework is easy to apply to any resource management system.

5 Evaluations

In this section, we evaluate coscheduling by simulations using the job traces collected from the production coupled HEC systems deployed at Argonne.

5.1 Experiment setup

To simulate coscheduling, we have extended Qsim [19], the event-driven simulator along with the Cobalt resource manager [5], to support multi-domain coscheduling simulation. For simulation, we need only to replace the real resource managers with

the event-driven simulators, which maintains jobs from job traces and compute nodes from the configuration files.

In the experiment, we use real system configurations; that is, 40,960 nodes on Intrepid and 100 nodes on Eureka are available for scheduling. The job traces were collected from the production Intrepid and Eureka systems within the year of 2010. On Intrepid, the job size ranges from 512 nodes to 32,768 nodes. On Eureka, the job size ranges from 1 node to 100 nodes.

The scheduling policy used on both schedulers is the same as that used on the real machines, namely, WFP [19] plus backfilling. The coscheduling algorithm can be enabled or disabled, so that we can compare the performance with and without coscheduling. To break the hold-hold deadlock, we set the held nodes' releasing period at 20 minutes.

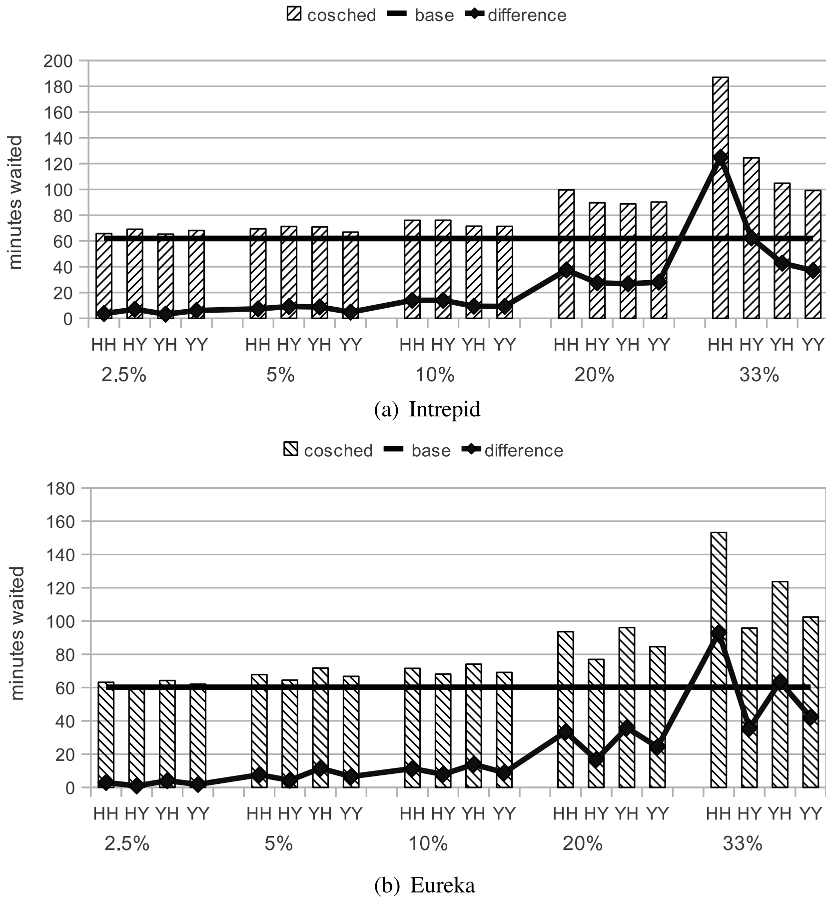
We ran a large number of simulation cases to verify the coscheduling mechanism. A simulation case is determined by a combination of the scheme configurations and certain proportion of paired jobs. All simulations cases ran successfully to completion, and the output logs show that all the paired jobs start at the same time with their own mate jobs no matter which one gets ready first. In the rest of this section we present a set of results to quantify the performance and cost of the coscheduling. Specifically, the following four metrics are used in performance evaluation:

- *Waiting time (wait)*: the time period between when the job is submitted and when it is started.
- *Slowdown (slowdown)*: a job's response time (waiting time plus running time) divided by the job's running time. It captures the fact that a long job can endure longer waiting than a short job can. The average *wait* and *slowdown* among all the jobs can measure the overall system performance of job scheduling.
- *Paired job synchronization time (sync time)*: the extra time a job has to wait for its mate in a coscheduling setting. By examining the average synchronization time among all the paired jobs, we can measure the performance cost to the paired jobs.
- *Service unit loss*: the wasted computing capabilities caused by holding. We measure this metric in node-hours and system utilization rate. This metric reflects the impact of coscheduling on system utilization.

5.2 Performance under various paired job proportions

We conducted several simulations to explore the impact of coscheduling under different proportions of paired jobs. This study can provide insight to system owners to control the coscheduling configuration under certain conditions.

We set the time span as one month. For Intrepid, we used the real job trace in a month containing 9219 jobs (with system load around 0.75). For Eureka, we used partially-synthetic job traces to simulate various workloads on Eureka. That is, by adjusting the job arrival intervals we can pack the workload in multiple months (with the same number of jobs) into one month. By doing so, we can conveniently tune the proportion of paired jobs on both traces. The system utilization rate of this special Eureka workload is around 0.5. For the various simulations, we set the paired job proportions to 2.5, 5, 10, 20, and 33%.



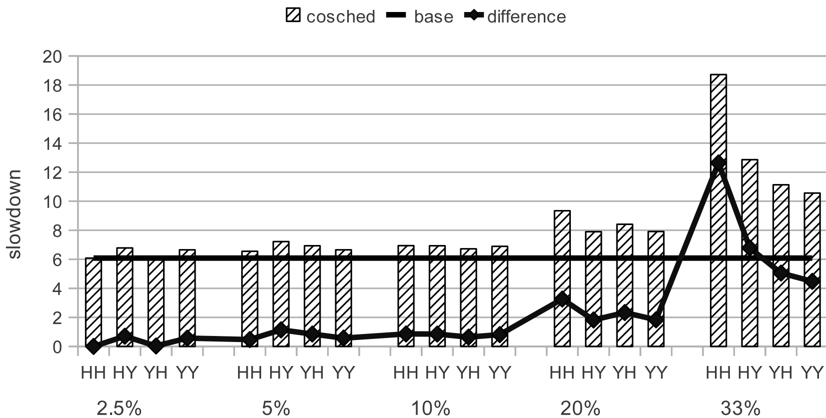
(a) Intrepid

(b) Eureka

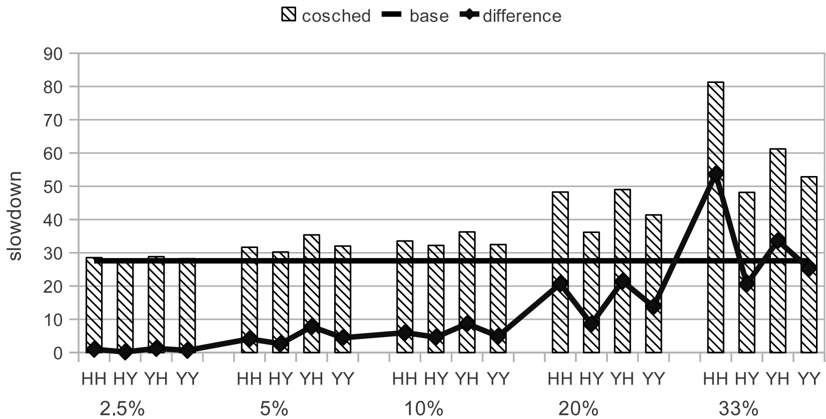
Fig. 4 Average wait time by paired job proportion. H and Y means hold and yield, respectively. Their combination determines a scheme; the first letter represents the local scheme and the second represents the remote scheme. For example, in (**a**) HY means using hold on Intrepid and yield on Eureka

As shown in Fig. 4a, the higher the paired job ratio is, the more the relative waiting time increase is on Intrepid. When the paired job ratio is 10% or less, the average waiting times are 3–14 minutes more than the base, relatively increased by 5–22%. When the paired job ratio is 20%, the average waiting times are 26–38 minutes more than the base. When one third of the total jobs (33%) are paired, the average waiting time gets considerably worse. However, if the yield scheme (YH or YY) is used, the average waiting in this case is comparable to the case with 20% paired jobs (around 100 minutes).

For Eureka (Fig. 4b), the trend is similar to that of Intrepid. The overall performance is not significantly impacted when the proportion of paired jobs is under 20%, regardless of which coscheduling scheme is used. With 33% paired job proportion, using hold (HH and YH) can cause a noticeable performance degradation, and using yield (HY and YY) can achieve performance similar to the 20% proportion case.



(a) Intrepid

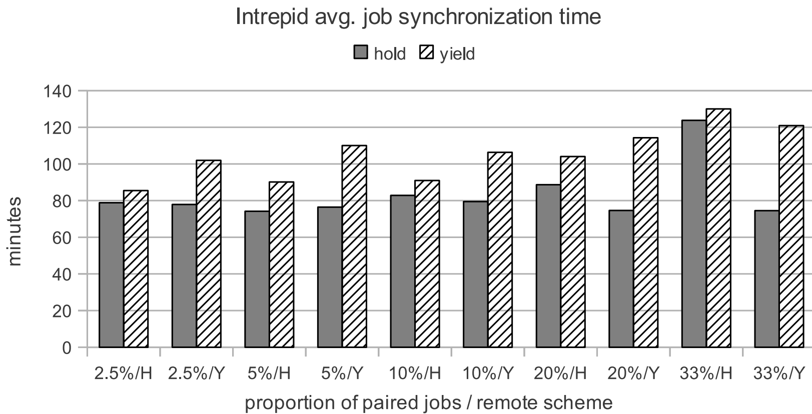


(b) Eureka

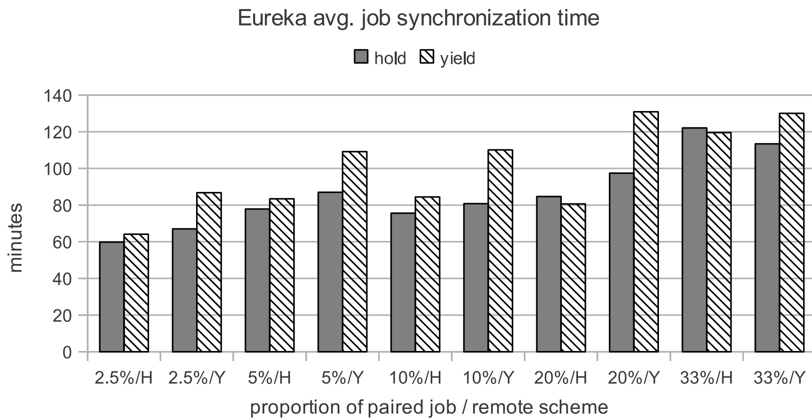
Fig. 5 Average slowdown by paired job proportion. H and Y means hold and yield, respectively. Their combination determines a scheme; the first letter represents the local scheme and the second represents the remote scheme. For example, in (**b**) HY means using hold on Eureka and yield on Intrepid

Figure 5 shows the results of average slowdown. The trend is similar to the waiting times. For Intrepid, average slowdowns of the first three proportions are all under 7.5. For the last two relatively high proportions, the slowdowns are mostly between 7.9 and 12.9, except with hold-hold. A similar trend is seen for Eureka. Note that the absolute slowdown values on Eureka are relatively high; it is because the job running times on Eureka are typically short.

Comparing Figs. 4 and 5 we can see that coscheduling will not impact system performance significantly with 20% (or less) proportion of paired jobs. When the proportion is higher than one third (33%), however, the performance degradation is noticeable. We recommend disabling coscheduling under this condition, or at least not using the hold scheme. Indeed, as shown in the figures, when the proportion



(a) Intrepid



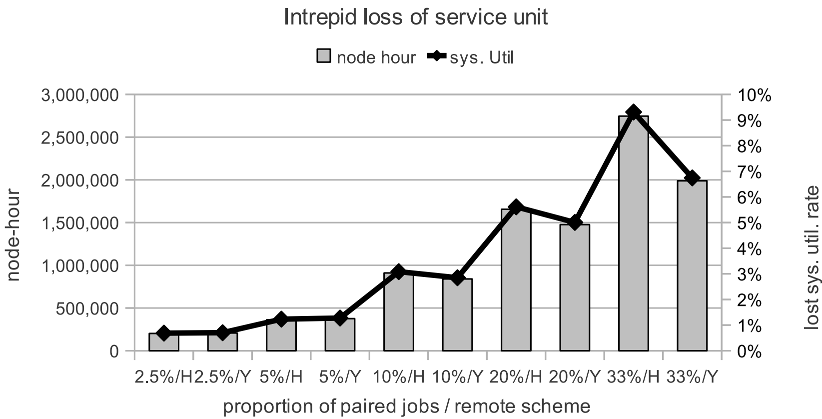
(b) Eureka

Fig. 6 Paired job average synchronization time by paired job proportion

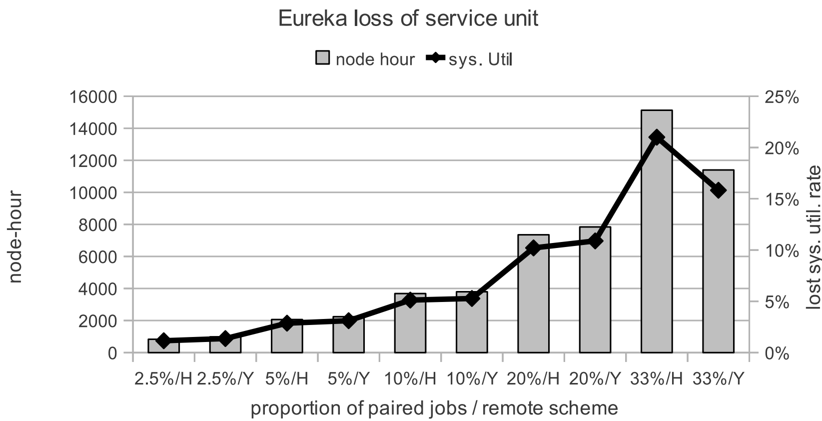
is 10% or more, the hold scheme shows more negative impact than does the yield scheme.

Figure 6 shows the average paired job synchronization time on Intrepid and Eureka. In the figure, when Intrepid uses hold, the average sync time mostly ranges between 75 and 88 minutes, except for HH with 124 minutes. Also, we observed that using hold as the local scheme consumes less average sync time than using yield. It is because a holding job can start immediately as soon as the mate job gets ready for the first time. When Eureka uses hold, the average paired job sync time varies from 60 to 122 minutes. There is a slight trend for overhead to increase as the paired job proportion increases. Similarly, all the cases using hold as the local scheme are better than using yield in terms of sync time.

Figure 7 shows the service unit loss on both Intrepid and Eureka. The x-axis is the same as in the previous figure. The primary y-axis shows the service unit loss measured by node-hours. The secondary y-axis shows the corresponding system uti-



(a) Intrepid



(b) Eureka

Fig. 7 Service unit loss by paired job proportion

lization rate loss compared with the total system node-hours over the whole time span. The figures show only the loss caused by using hold on the local machine. As shown in the figure, the loss of service unit increases as the proportion of paired jobs gets higher. Specifically, the results on Intrepid range from 0.7 to 9.3% of the total node-hours in a month; the losses on Eureka range 1 to 21% of total node-hours in a month.

We consider the service unit loss acceptable when the proportion of paired jobs is under 10%, when the loss rate is under 3% on Intrepid and 5% on Eureka. When the proportion is 20%, the loss rate is also fairly acceptable. Note that the acceptable loss rate is relative; for machines such as Eureka that have many idle cycles to begin with, more loss rate is tolerable. But when paired jobs become more than 33%, if the coscheduling feature cannot be disabled, we recommend using the yield scheme on both machines in order to avoid loss of service units.

Table 1 Results of reservation simulation

Proportion of reserved jobs	0 (base)	2.5%	5%	10%	20%	33%
Wait of Intrepid (Min.)	62.1	81.5	135	183	203	273
Slowdown of Intrepid	6.08	6.12	8.15	8.87	13.8	19.2
Wait of Eureka (Min.)	60.2	73.2	108	144	164	218
Slowdown of Eureka	27.6	28.7	39.6	43.9	64.4	89.1

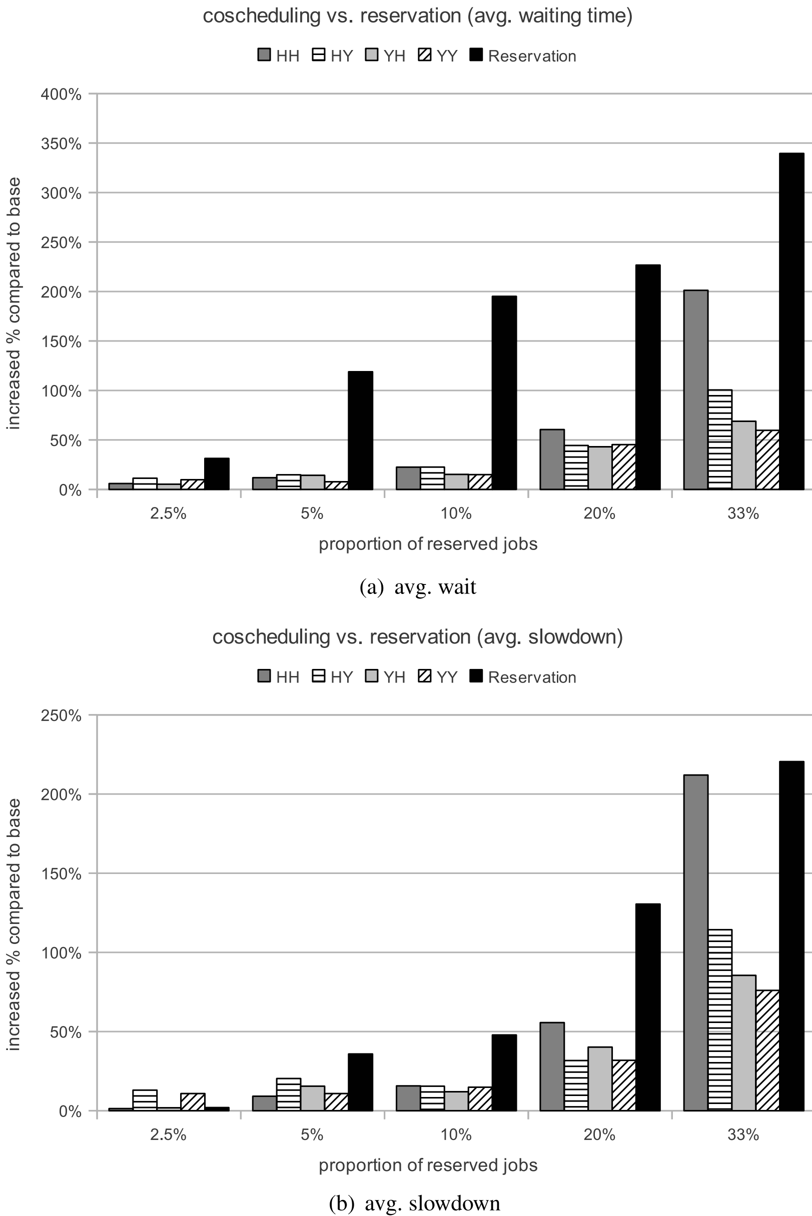
5.3 Comparison with advance reservation

As mentioned earlier, our coscheduling can save the effort of making advance reservations, which is an alternate way to start two jobs at the same time from different machines. However we will show that making advance reservations will also bring a negative impact to the system. To quantify the performance loss, we conducted experiments to evaluate the performance degradation caused by advance reservation and then compared the results with those of coscheduling schemes.

Our reservation simulation is conducted as follows: using the same mechanism specifying certain proportion of paired jobs, we specify certain proportion of reserved jobs. The reservation start times are the times when jobs actually start in the job trace. In order to avoid reservation overlaps, we use the actual job running times (instead of user estimates) as the reservation lengths. This is an idealistic assumption because in practice we cannot know the exact running times of jobs beforehand even though we have some walltime prediction schemes to get more accurate runtime estimates [20]. Doing so may increase the density of reservations but we can adjust the workload to get certain density of reservations by controlling the proportion of reservation jobs. With the reservation information, the scheduler will prevent the newly scheduled job from violating the existing reservations. The simulation job traces are the same as the ones used in previous subsection.

Table 1 shows the experimental results of reservation simulations. The results of average waiting time and slowdown for each of Intrepid and Eureka systems are shown. Each column represents a specific configuration defined by a certain proportion of reserved jobs. Proportion 0 means no jobs are reserved thus this column represents the base case. Other columns represent the corresponding proportions used in previous section. From the table we can see significant increases on all the metrics as the proportion gets higher.

To better compare advance reservations with our coscheduling schemes, we illustrate their relative increases of waiting time and slowdown compared with base in Figs. 8 and 9. In the figures, both the relative increases of coscheduling schemes and advance reservation are shown. Figure 8 shows the Intrepid results. Clearly, advance reservation results in much more significant performance degradation than coscheduling does. When the proportion is under 20%, coscheduling schemes increase waiting time by no more than 15% compared with the base. However, advance reservation results in 200% more average waiting time, meaning three times the base. Similarly, for proportion 20 and 33%, although coscheduling schemes increase of around 50%, advance reservation causes much more: the waiting time is nearly 4.5 times compared with the base when one third of jobs are reserving. As for slowdown, advance

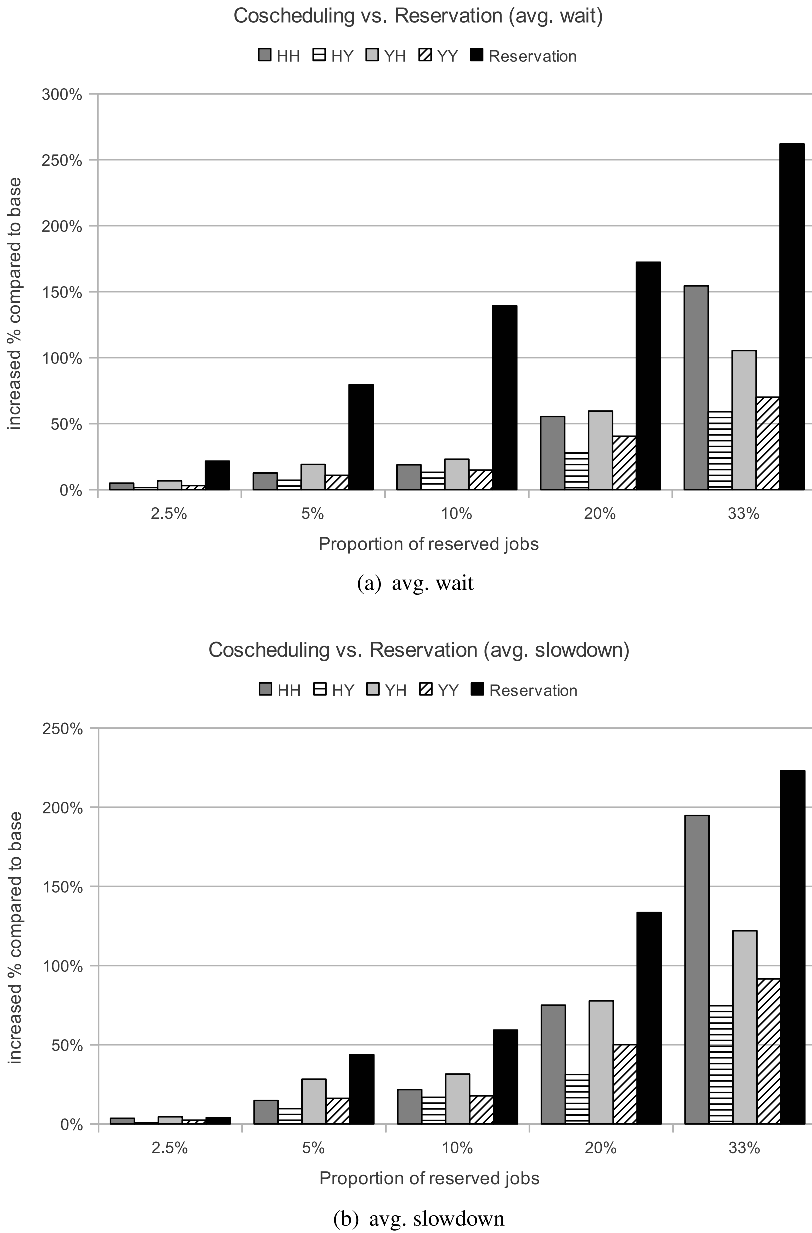


(a) avg. wait

(b) avg. slowdown

Fig. 8 Performance comparison between coscheduling schemes and advance reservation (Intrepid)

reservation also causes much more degradation than coscheduling. We noticed that the number of relative increase is less than that of waiting time, which may be because the delayed jobs are mostly long jobs which cannot fit into the fragmented resources caused by reservations. That is, short jobs are less impacted than long jobs by reservations.



(a) avg. wait

(b) avg. slowdown

Fig. 9 Performance comparison between coscheduling schemes and advance reservation (Eureka)

Very similar trends are seen for Eureka results in Fig. 8. The average waiting time is more than doubled even when only 10% jobs are reserved. The increase in slowdown is also comparably less but still very significant. In sum, to achieve co-

execution of jobs cross multiple resource management domains, our coscheduling schemes are much better than advance reservation.

6 Summary

We have designed and implemented a coscheduling mechanism used in a coupled high-performance computing system where a large-scale computing system co-resides with a special-purpose data analysis or visualization system. The goal of coscheduling is to simultaneously start associated jobs that are submitted to different machines belonging to different scheduling domains. We have proposed two basic schemes for coscheduling: hold and yield. Any combination of schemes on the coupled systems can achieve the coscheduling goal without manual intervention.

We have evaluated the coscheduling mechanism by means of trace-based simulations using real configurations and workloads from the production coupled systems deployed at Argonne. Our experimental results demonstrate that coscheduling can be achieved with limited impact on system performance under varying workloads. We also compared the performance difference of using coscheduling and advance reservation. The results show that in order to achieve job co-execution across multiple resource management domains, coscheduling results in much less impact on the system than advance reservation does.

Achieving coscheduling not only benefits a number of existing applications using the coupled systems but also can drive the emergence of new applications to take advantage of (administratively) heterogeneous resources. In the future, we plan to extended our coscheduling mechanism to support more sophisticated inter-job temporal constraints. Further, we will examine the possibility of extending our algorithm to support N-way coscheduling on more than two scheduling domains. Ultimately, we plan to deploy our coscheduling mechanism on the production systems at Argonne and measure the benefit to real applications.

Acknowledgements The work at IIT is supported in part by U.S. NSF Grants CNS-0834514, CNS-0720549, and CCF-0702737. The work at Argonne was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357 and an Argonne National Laboratory Director's Postdoctoral Fellowship.

References

1. Abbasi H, Wolf M, Eisenhauer G, Klasky S, Schwan K, Zheng F (2009) DataStager: Scalable data staging services for petascale applications. In: Proc of ACM international symposium on high performance distributed computing (HPDC)
2. Basney J, Livny M (1999) Improving goodput by co-scheduling CPU and network capacity. Int J High Perform Comput Appl 13(3):220–230
3. Binns J, Dech F, Papka M, Silverstein J, Stevens R (2005) Developing a distributed collaborative radiological visualization application. In: From Grid to HealthGrid, pp 70–79
4. Blue Gene Team (2008) Overview of the IBM Blue Gene/P project. IBM J Res Devel
5. Cobalt project. http://trac.mcs.anl.gov/projects/cobalt
6. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S (1998) A resource management architecture for metacomputing systems. In: Proc of job scheduling strategies for parallel processing (JSSPP)

7. Etsion Y, Tsafrir D (2005) A short survey of commercial cluster batch schedulers. Technical Report 2005-13, the Hebrew University of Jerusalem
8. Frachtenberg E, Feitelson D, Petrini F, Fernandez J (2003) Flexible coscheduling–mitigating load imbalance and improving utilization of heterogeneous resources. In: Proc of IEEE international parallel & distributed processing symposium (IPDPS)
9. Foster I, Kesselman C, Lee C, Lindell R, Nahrstedt K, Roy A (1999) A distributed resource management architecture that supports advance reservations and co-allocation. In: Proc of international workshop on quality of service
10. Huedo E, Montero R, Llorente I (2004) A framework for adaptive execution in grids. Softw Pract Exp 34(7):631–651
11. MacLaren J (2007) HARC: the highly-available resource co-allocator. In: Proc. of GADA'07. LNCS, vol 4804. Springer, Berlin, pp 1385–1402
12. Moab workload scheduler. http://www.adaptivecomputing.com
13. Ousterhout J (1982) Scheduling techniques for concurrent systems. In: Proc of IEEE int'l conference on distributed computing systems (ICDCS)
14. Petrini F, Feng W-C (2000) Buffered coscheduling: a new methodology for multitasking parallel jobs on distributed systems. In: Proc of IEEE int'l parallel & distributed processing symp (IPDPS)
15. Romosan A, Rotem D, Shoshani A, Wright D (2005) Co-scheduling of computation and data on computer clusters. In: Proc of int'l conf on scientific and statistical database management
16. Sobalvarro P, Pakin S, Weihl W, Chien A (1998) Dynamic coscheduling on workstation clusters. In: Proc of job scheduling strategies for parallel processing (JSSPP)
17. Sobalvarro P, Weihl W (1995) Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors. In: Proc of job scheduling strategies for parallel processing (JSSPP)
18. Smith W, Foster I, Taylor W (2000) Scheduling with advanced reservations. In: Proc of IEEE int'l parallel & distributed processing symposium (IPDPS)
19. Tang W, Lan Z, Desai N, Buettner D (2009) Fault-aware, utility-based job scheduling on Blue Gene/P systems. In: Proc of IEEE int'l conf on cluster computing
20. Tang W, Desai N, Buettner D, Lan Z (2010) Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. In: Proceedings of IEEE international parallel & distributed processing symposium (IPDPS)
21. Tang W, Desai N, Vishwanath V, Buettner D, Lan Z (2011) Job coscheduling on coupled high-end computing system. In: Proc of int'l conf on parallel processing workshops (ICPPW)
22. Teodoro G, Sachetto R, Sertel O, Gurcan M, Meira W, Catalyurek U, Ferreira R (2009) Coordinating the use of GPU and CPU for improving performance of compute intensive applications. In: Proc of IEEE int'l conf on cluster computing
23. Townsley D, Bair R, Dubey A, Fisher R, Hearn N, Lamb D, Riley K (2009) Large-scale simulations of buoyancy-driven turbulent nuclear burning. J Phys, Conf Ser 125(1)
24. Tsafrir D, Etsion Y, Feitelson D (2007) Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans Parallel Distrib Syst 18(6):789–803
25. Vadiyar S, Dongarra J (2002) A metascheduler for the grid. In: Proc of 11th IEEE international symposium on high performance distributed computing (HPDC)
26. Vishwanath V, Hereld M, Morozov V, Papka ME (2011) Topology-aware data movement and staging for I/O acceleration on BlueGene/P supercomputing systems. In: Proc IEEE/ACM international conference for high performance computing, networking, storage and analysis (SC)
27. Vishwanath V, Hereld M, Papka ME (2011) Simulation-time data analysis and I/O acceleration on leadership-class systems using GLEAN. In: Proc of IEEE symposium on large data analysis and visualization
28. Wiseman Y, Feitelson D (2003) Paired gang scheduling. IEEE Trans Parallel Distrib Syst 14(6):581–592
29. Yoshimoto K, Kavatch PA, Andrews P (2005) Co-scheduling with user settable reservations. In: Proc of job scheduling strategies for parallel processing (JSSPP)