On the benefit of processor coallocation in multicluster grid systems

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

On the Benefit of Processor Coallocation in Multicluster Grid Systems

Omer Ozan Sonmez, Hashim Mohamed, and Dick H.J. Epema

Abstract—In multicluster grid systems, parallel applications may benefit from processor coallocation, that is, the simultaneous allocation of processors in multiple clusters. Although coallocation allows the allocation of more processors than available in a single cluster, it may severely increase the execution time of applications due to the relatively slow wide-area communication. The aim of this paper is to investigate the benefit of coallocation in multicluster grid systems, despite this drawback. To this end, we have conducted experiments in a real multicluster grid environment, as well as in a simulated environment, and we evaluate the performance of coallocation for various applications that range from computation-intensive to communication-intensive and for various system load settings. In addition, we compare the performance of scheduling policies that are specifically designed for coallocation. We demonstrate that considering latency in the resource selection phase improves the performance of coallocation, especially for communication-intensive parallel applications.

Index Terms—Coallocation, grid, multicluster, parallel job scheduling.

✦

---

1 INTRODUCTION

OVER the last decade, multicluster grids have become the mainstream execution environment for many large-scale (scientific) applications with varying characteristics. In such systems, parallel applications may benefit from using resources such as processors in multiple clusters simultaneously, that is, they may use *processor coallocation*. This potentially leads to higher system utilizations and lower queue wait times by allowing parallel jobs to run when they need more processors than are available in a single cluster. Despite such benefits, with processor coallocation, the execution time of parallel applications may severely increase due to wide-area communication overhead and processor heterogeneity among the clusters. In this paper, we investigate the benefit of processor coallocation (hereafter, we use "coallocation" to refer to "processor coallocation"), despite its drawbacks, through experiments performed in a real multicluster grid environment. In addition, we have performed simulation-based experiments to extend our findings obtained in the real environment.

From the perspective of a single parallel application, coallocation is beneficial if the intercluster communication overhead is lower than the additional queue wait time, the application will experience if it is instead submitted to a single cluster. However, this additional queue wait time is neither known a priori nor can it be predicted easily due to the heterogeneity and the complexity of grid systems. Therefore, in this work, we do not rely on predictions. We aim to assess the effect of various factors such as the communication requirements of parallel applications, the

communication technology and the processor heterogeneity of the system, and the scheduling policies of a grid scheduler on the coallocation performance of single parallel applications in terms of the execution time, in particular in a real multicluster grid system. In addition, we aim to investigate the benefit of coallocation from the perspective of scheduling workloads of parallel applications in terms of the average job response time.

In our previous work, we have focused on the implementation issues of realizing support for coallocation in our KOALA grid scheduler [1], and implemented scheduling policies for parallel applications that may need coallocation. The Close-to-Files (CFs) policy [2] tries to alleviate the overhead of waiting in multiple clusters for the input files of applications to become available in the right locations. We have shown that the combination of the CF policy and file replication is very beneficial when applications have large input files. The (Flexible) Cluster Minimization (FCM) policy [3] minimizes the number of clusters to be combined for a given parallel application in order to reduce the number of intercluster messages between the components of a coallocated application, which turns out to improve the performance of coallocation for communication-intensive applications.

In this paper, we extend our previous work with the following contributions. First, we present an analysis of the impact of the intercluster communication technology and the impact of the processor speed heterogeneity of a system on the coallocation performance of parallel applications. Second, we investigate when coallocation in multicluster grids may yield lower average job response times through experiments that run workloads of real MPI applications as well as synthetic applications which vary from computation-intensive to communication-intensive. Finally, we extend the scheduling policies of KOALA with the Communication-Aware (CA) policy that takes either intercluster bandwidth or latency into account when deciding on coallocation, and we compare its performance to that of FCM, which only takes the numbers of idle processors into account when coallocating jobs.

- *The authors are with the Parallel and Distributed Systems Group, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands. E-mail: {o.o.sonmez, h.h.mohamed, d.h.j.epema}@tudelft.nl.*
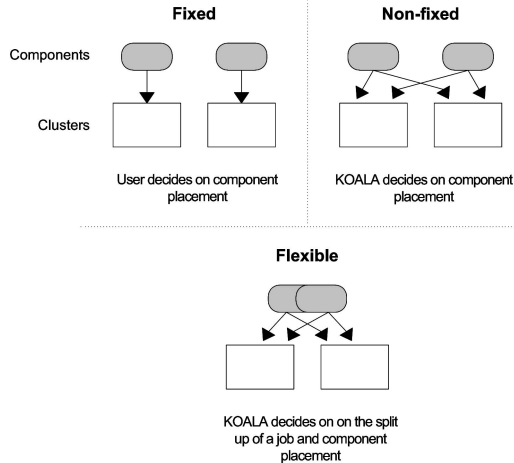
Fig. 1. The job request types supported by KOALA.

The rest of the paper is organized as follows: Section 2 presents a job model for parallel applications that may run on coallocated resources. In Section 3, we explain the main mechanisms of our KOALA grid scheduler and our testbed environment. In Section 4, we present the scheduling policies of KOALA that we consider in this paper. In Sections 5, 6, and 7, we present the results of our experiments. In Section 8, we discuss the challenges and issues of realizing coallocation in real multicluster grids. Section 9 reviews related work on coallocation. Finally, Section 10 ends the paper with some concluding remarks.

2 A JOB MODEL FOR PARALLEL APPLICATIONS

In this section, we present our job model for processor coallocation of parallel applications in multicluster grids. In this model, a job comprises either one or multiple *components* that can be scheduled separately (but simultaneously) on potentially different clusters, and together execute a single parallel application. A job specifies for each component its requirements and preferences, such as its size (the number of processors or nodes it needs) and the names of its input files. We assume jobs to be rigid, which means that the number of processors allocated to a job (and to each of its components) remains fixed during its execution. A job may or may not specify the execution sites where its components should run. In addition, a job may or may not indicate how it is split up into components. Based on these distinctions, we consider three job request structures: *fixed* requests, *nonfixed* requests, and *flexible* requests (see Fig. 1).

In a fixed request, a job specifies the sizes of its components and the execution site on which the processors must be allocated for each component. On the other hand, in a nonfixed request, a job also specifies the sizes of its components, but it does not specify any execution site, leaving the selection of these sites, which may be the same for multiple components, to the scheduler. In a flexible request, a job only specifies its total size and allows the scheduler to divide it into components (of the same total size) in order to fit the job on the available execution sites. With a flexible request, a user may impose restrictions on the number and sizes of the components. For instance, a user may want to specify for a job a lower bound on the component size or an

upper bound on the number of components. By default, this lower bound is one and this upper bound is equal to the number of execution sites in the system. Although it is up to the user to determine the number and sizes of the components of a job, some applications may dictate specific patterns for splitting up the application into components; hence, complete flexibility is not suitable in such a case. So, a user may specify a list of options of how a job can be split up, possibly ordered according to preference. In the experiments in this paper, we do not include this feature.

These request structures give users the opportunity of taking advantage of the system considering their applications' characteristics. For instance, a fixed job request can be submitted when the data or software libraries at different clusters mandate a specific way of splitting up an application. When there is no such affinity, users may want to leave the decision to the scheduler by submitting a nonfixed or a flexible job request. Of course, for jobs with fixed requests, there is nothing a scheduler can do to schedule them optimally; however, for nonfixed and flexible requests, a scheduler should employ scheduling policies (called job placement policies in this paper) in order to optimize some criteria.

3 THE SCHEDULER AND THE SYSTEM

In this section, first, we briefly describe the KOALA grid scheduler [1], which is the basis of the work presented in this paper, and second, we describe our testbed, the DAS3 [4].

3.1 The KOALA Grid Scheduler

The KOALA grid scheduler has been designed for multicluster systems such as the DAS-3 which have in each cluster a head node and a number of compute nodes. The main distinguishing feature of KOALA is its support for coallocation.

Upon submission of a job, KOALA uses one of its job placement policies (see Section 4) to try to place job components on suitable execution sites. If the placement of the job succeeds and input files are required, the scheduler informs the job submission tool to initiate the third-party file transfers from the selected file sites to the execution sites of the job components. If a placement try fails, KOALA places the job at the tail of the placement queue, which holds all jobs that have not yet been successfully placed. The scheduler regularly scans the queue from head to tail to see whether it is able to place any job.

For coallocation, KOALA uses an atomic transaction approach [5] in which job placement only succeeds if all the components of a job can be placed at the same time. This necessitates the simultaneous availability of the desired numbers of idle nodes in multiple clusters. KOALA tries to allocate nodes using the resource managers of the clusters in question. If all the allocation attempts for all components succeed, the job is initiated on the allocated nodes after the necessary file transfers. In this study, we map two application processes per node, since all the clusters in our testbed comprise nodes of dual processors.

Currently, KOALA is capable of scheduling and coallocating parallel jobs employing either the Message Passing Interface (MPI) or Ibis [6] parallel communication libraries. In this paper, we only consider MPI jobs, which have to be compiled with the Open-MPI [7] library. Open-MPI, built

TABLE 1
Properties of the DAS-3 Clusters

Cluster Location	Nodes	Speed	Interconnect
Vrije University	85	2.4 GHz	Myri-10G & GbE
U. of Amsterdam	41	2.2 GHz	Myri-10G & GbE
Delft University	68	2.4 GHz	GbE
MultimediaN	46	2.4 GHz	Myri-10G & GbE
Leiden University	32	2.6 GHz	Myri-10G & GbE

upon the MPI-2 specification, allows KOALA to combine multiple clusters to run a single MPI application by automatically handling both intercluster and intracluster messaging.

3.2 The DAS-3 Testbed

Our testbed is the third-generation Distributed ASCI Supercomputer (DAS-3) [4], which is a wide-area computer system in The Netherlands that is used for research on parallel, distributed, and grid computing. It consists of five clusters of, in total, 272 dual-processor AMD Opteron compute nodes. The distribution of the nodes over the clusters and their speeds is given in Table 1. As can be seen, the DAS-3 has a relatively minor level of processor speed heterogeneity. The clusters are connected by both 10 Gb/s Ethernet and 10 Gb/s Myri-10G links both for wide-area and for local-area communications, except for the cluster in Delft, which has only 1 Gb/s Ethernet links. On each of the DAS-3 clusters, the Sun Grid Engine (SGE) [8] is used as the local resource manager. SGE has been configured to run applications on the nodes in an exclusive fashion, i.e., in space-shared mode. As the storage facility, NFS is available on each of the clusters.

4 JOB PLACEMENT POLICIES

The KOALA job placement policies are used to decide where the components of nonfixed and flexible jobs should be sent for execution. In this section, we present three job placement policies of KOALA, which are the Worst Fit, the Flexible Cluster Minimization, and the Communication-Aware placement policy. Worst Fit is the default policy of KOALA which serves nonfixed job requests. Worst Fit also makes perfect sense in the absence of coallocation, when all jobs consist of a single component. The two other policies, on the other hand, serve flexible job requests and only apply to the coallocation case.

4.1 The Worst Fit Policy

The Worst Fit (WF) policy aims to keep the load across clusters balanced. It orders the components of a job with a nonfixed request type according to decreasing size and places them in this order, one by one, on the cluster with the largest (remaining) number of idle processors, as long as this cluster has a sufficient number of idle processors. WF leaves in all clusters as much room as possible for later jobs, and hence, it may result in coallocation even when all the components of the considered job would fit together on a single cluster.

4.2 The Flexible Cluster Minimization Policy

The FCM policy is designed with the motivation of minimizing the number of clusters to be combined for a
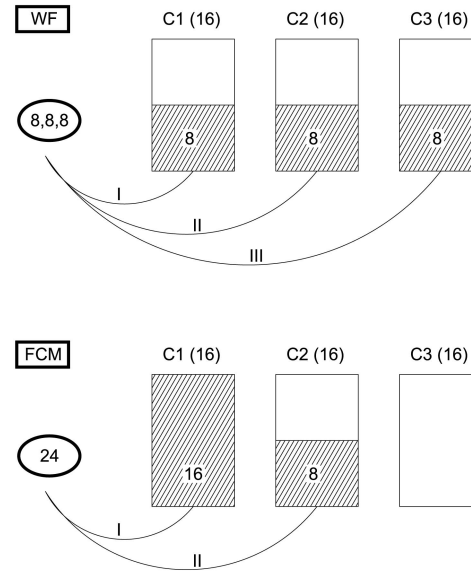

Fig. 2. An example comparing the WF and the FCM placement policies.

given parallel job in order to reduce the number of intercluster messages. FCM first orders the clusters according to decreasing number of idle processors and considers component placement in this order. Then, FCM places on clusters one by one a component of the job of size equal to the number of idle processors in that cluster. This process continues until the total processor requirement of the job has been satisfied or the number of idle processors in the system has been exhausted, in which case the job placement fails (the job component placed on the last cluster used for it may be smaller than the number of idle processors of that cluster).

Fig. 2 illustrates the operation of the WF and the FCM policies for a job of total size 24 in a system with three clusters, each of which has 16 idle processors. WF successively places the three components (assumed to be of size 8 each) of a nonfixed job request on the cluster that has the largest (remaining) number of available processors, which results in the placement of one component on each of the three clusters. On the other hand, FCM results in combining two clusters for a flexible job of the same total size (24), splitting the job into two components of sizes 16 and 8, respectively.

4.3 The Communication-Aware Policy

The CA placement policy takes either bandwidth or latency into account when deciding on coallocation. The performance of parallel applications that need relatively large data transfers is more sensitive to bandwidth, while the performance of parallel applications which are dominated by interprocess communication is more sensitive to latency. In this paper, we only consider the latter case, and run the CA policy with the latency option.

The latencies between the nodes of each pair of clusters in the system are kept in the information service of KOALA and are updated periodically. CA first orders the clusters according to increasing intracluster latency, and checks in this order whether the complete job can be placed in a single cluster. If this is not possible, CA computes for each cluster the average of all of its intercluster latencies, including its own intracluster latency, and orders the clusters according to increasing value of this average

TABLE 2
The Average Bandwidth (in Megabyte Per Second,
Top Numbers) and Latency (in Millisecond, Bottom Numbers)
between the Nodes of the DAS-3 Clusters
(For Delft-Leiden, See Text)

| Clusters	Vrije	Amsterdam	Delft	MultimediaN	Leiden
Vrije	561	185	45	185	77
	0.03	0.4	1.15	0.4	1.0
Amsterdam	185	526	53	512	115
	0.4	0.03	1.1	0.03	0.6
Delft	45	53	115	10	-
	1.15	1.1	0.05	1.45	-
MultimediaN	185	512	10	560	115
	0.4	0.03	1.45	0.03	0.6
Leiden	77	115	-	115	530
	1.0	0.6	-	0.6	0.03

Fig. 3. The execution time of a synthetic coallocated MPI application, depending on the interconnect technology used and the number of clusters combined.

latency. As in the FCM policy, CA then splits up the job into components of sizes equal to the numbers of idle processors of the clusters in this order (again, the last component of the job may not completely fill up the cluster on which it is placed).

In fact, the CA policy does not guarantee the best solution to the problem of attaining the smallest possible execution time for a coallocated parallel application, since this problem is NP-complete. However, it is a reasonable heuristic for small-scale systems. For larger systems, a clustering approach can be considered, in which clusters with low intercluster latencies are grouped together, and coallocation is restricted to those groups separately.

5 THE IMPACT OF SYSTEM PROPERTIES ON COALLOCATION PERFORMANCE

In this section, we evaluate the impact of the intercluster communication characteristics and the processor speed heterogeneity of a multicluster system on the execution time performance of a single parallel application that runs on coallocated processors.

5.1 The Impact of Intercluster Communication

In a multicluster grid environment, it is likely that the intercluster communication is slower than the intracluster communication in terms of latency and bandwidth, which are the key factors that determine the communication performance of a network. This slowness, in fact, depends on various factors such as the interconnect technology that enables the intercluster communication among the processes of a parallel application, the distance between the clusters, the number and capabilities of the network devices, and even the network configuration. Therefore, depending on the communication requirements of a parallel application, the intercluster latency and bandwidth may have a big impact on its execution time performance.

In this section, we first present the results of experiments for measuring the communication characteristics of our testbed, and then, we present the results of experiments for assessing the impact of intercluster communication on execution time performance.

With the DAS-3 system, we have the chance to compare the performance of the Myri-10G and the Gigabit Ethernet
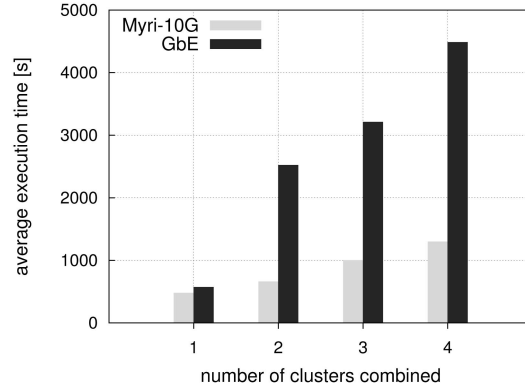
(GbE, 1Gb/s) interconnect technologies. When the cluster in Delft is involved in the coallocation of a parallel job, GbE is used for the entire intercluster communication, since it does not support the faster Myri-10G technology. For all other cluster combinations, for coallocation, Myri-10G is used, even though they all support GbE. Table 2 shows the average intracluster and intercluster bandwidth (in megabyte per second) and the average latency (in millisecond) as measured between the compute nodes of the DAS-3 clusters (the values are diagonally symmetric). These measurements were performed with an MPI ping-pong application that measures the average bidirectional bandwidth, sending messages of 1 MB, and the average bidirectional latency, sending messages of 64 KB, between two (co)allocated nodes. The measurements were performed when the system was almost empty. With Myri-10G, the latency between the nodes is lower and the bandwidth is higher in comparison to the case with GbE. The measurements also indicate that the environment is heterogeneous in terms of communication characteristics even when the same interconnection technology is used. This is due to characteristics of the network structure such as the distance and the number of routers between the nodes. For example, the clusters Amsterdam and MultimediaN are located in the same building, and therefore, they achieve the best intercluster communication. We were not able to perform measurements between the clusters in Delft and Leiden due to a network configuration problem; hence, we excluded either the cluster in Delft or the cluster in Leiden in all of our experiments.

The synthetic parallel application that we use in our execution time experiments performs one million MPI_AllGather all-to-all communication operations each with a message size of 10 KB. The job running this application has a total size of 32 nodes (64 processors), and we let it run with fixed job requests with components of equal size on all possible combinations of one-four clusters with the following restrictions. We either exclude the cluster in Delft and let the intercluster communication use the Myri-10G network, or we include the cluster in Delft, exclude the one in Leiden, and let the intercluster communication use GbE.

Fig. 3 shows the execution time of the synthetic application averaged across all combinations of equal numbers of clusters. Clearly, the execution time increases with the increase of the number of clusters combined. However, the

TABLE 3
Execution Time of a Synthetic Application When Coallocating
the Cluster in Leiden with Each of the Other Clusters

Cluster Comb.	Leiden	Leiden-Vrije	Leiden-Delft	Leiden-MultiMediaN	Leiden-Amsterdam
Exec.[s]	30	32	32	32	35
Increase	-	7%	7%	7%	17%

increase is much more severe, and the average execution time is much higher, when GbE is used—coallocation with Myri-10G adds much less execution time overhead. These results indicate that the communication characteristics of the network are a crucial element in coallocation, especially for communication-intensive parallel applications. However, the performance of coallocation does not solely depend on this aspect for all types of parallel applications, as we will explain in the following section.

5.2 The Impact of Heterogeneous Processor Speeds

Unless an application developer does take into account processor speed heterogeneity and optimizes his applications accordingly, the execution time of a parallel application that runs on coallocated clusters will be limited by the speed of the slowest processor, due to the synchronization of the processes. This is a major drawback of coallocation especially for computation-intensive parallel applications which do not require intensive intercluster communications.

We have run a synthetic parallel application combining the cluster in Leiden (which has the fastest processors, see Table 1) with each of the other clusters in the DAS-3 system and quantified the increase in the execution time over running the application only in Leiden. The synthetic parallel application performs 10 million floating point operations without any I/O operations and interprocess communications except the necessary MPI initialization and finalization calls. As the results in Table 3 indicate, there is a slight increase in the execution time ranging from 7 to 17 percent due to the minor level of processor speed heterogeneity in DAS-3. Therefore, in this paper, we do not consider the slowdown due to heterogeneous processor speeds in our policies. Nevertheless, the FCM policy can easily be enhanced such that it does consider the processor speeds when coallocating in systems where this slowdown can be high.

6 COALLOCATION VERSUS NO COALLOCATION

In this section, we investigate when coallocation for parallel applications may be beneficial over disregarding coallocation. In Section 6.1, we present the applications that we have used in our experiments. In Section 6.2, we present and discuss the results of the experiments conducted in the DAS-3 system. We have performed additional experiments in a simulated DAS-3 environment, in order to investigate the performance of coallocation for a wide range of situations. We present and discuss the results of these simulation-based experiments in Section 6.3.

6.1 The Applications

For the experiments, we distinguish between computation- and communication-intensive parallel applications. We have used three MPI applications: *Prime Number* [9], *Poisson*
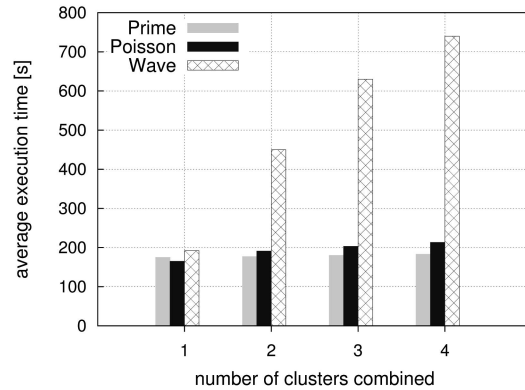

Fig. 4. The average execution times of the applications depending on the number of clusters combined.

[10], and *Concurrent Wave* [11], which vary from computation-intensive to communication-intensive.

The Prime Number application finds all the prime numbers up to a given integer limit. In order to balance the load (large integers take more work), the odd integers are assigned cyclically to processes. The application exhibits embarrassing parallelism; collective communication methods are called only to reduce the data of the number of primes found, and the data of the largest prime number.

The Poisson application implements a parallel iterative algorithm to find a discrete approximation to the solution of a two-dimensional Poisson equation on the unit square. For discretization, a uniform grid of points in the unit square with a constant step in both directions is considered. The application uses a red-black Gauss-Seidel scheme, for which the grid is split up into "black" and "red" points, with every red point having only black neighbors and vice versa. The parallel implementation decomposes the grid into a two-dimensional pattern of rectangles of equal size among the participating processes. In each iteration, the value of the each grid point is updated as a function of its previous value and the values of its neighbors, and all points of one color are visited first followed by the ones of the other color.

The Concurrent Wave application calculates the amplitude of points along a vibrating string over a specified number of time steps. The one-dimensional domain is decomposed by the master process, and then, distributed as contiguous blocks of points to the worker processes. Each process initializes its points based on a sine function. Then, each process updates its block of points with the data obtained from its neighbor processes for the specified number of time steps. Finally, the master process collects the updated points from all the processes.

The runtimes of these applications in the DAS-3 are shown in Fig. 4. Each application has been run several times on all combinations of clusters (excluding the cluster in Delft; the interconnect technology is Myri-10G) as fixed job requests with a total size of 32 nodes and components of equal size (except for the case of three clusters in which we submit components of sizes 10-10-12 nodes), and the results have been averaged. The results demonstrate that as the Concurrent Wave application is a communication-intensive application, its execution time with multiple clusters increases markedly, from 200 seconds as a single cluster to 750 seconds when combining four clusters. The Poisson application suffers much less from the wide-area communication overhead, while the Prime Number application is
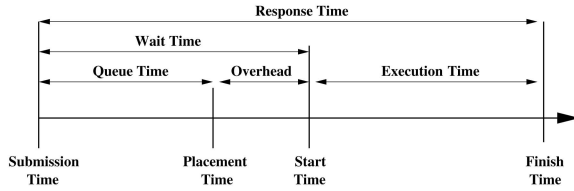
Fig. 5. The timeline of a job submission in Koala.

not affected by it at all, since it is a computation-intensive parallel application.

6.2 Experiments in the Real Environment

In this section, we present our experiments in the DAS-3. We first explain our experimental setup, and then, discuss the results.

6.2.1 Experimental Setup

In our experiments, we use three workloads that each contains only one of the applications presented in Section 6.1. In the experiments in which no coallocation is employed, the workloads are scheduled with the WF policy, and in the experiments in which coallocation is used, the workloads are scheduled with the FCM policy (and all job requests are flexible).

We consider jobs with total sizes of 8, 16, and 32 nodes so that the jobs can fit on any cluster in the system in case of no coallocation; the total sizes of the jobs are randomly chosen from a uniform distribution. For every application, we have generated a workload with an average interarrival time determined in such a way that the workload is calculated to utilize approximately 40 percent of the system on average. The real (observed) utilization attained in the experiments depends on the policy being used, since the theoretical calculation of the utilization (i.e., the net utilization) is based on the average single-cluster execution times of the applications. When there is no coallocation, there is no wide-area communication, and the real and the net utilizations coincide. The job arrival process is Poisson.

We use the tools provided within the GrenchMark project [12] to ensure the correct submission of our workloads to the system, and run each workload for 4 hours, under the policy in question. We have excluded the cluster in Delft, and the interconnect technology is Myri-10G.

In the DAS-3 system, we do not have control over the background load imposed on the system by other users. These users submit their (nongrid) jobs straight to the local resource managers, bypassing KOALA. During the experiments, we monitored this background load and tried to maintain it between 10 and 30 percent across the system by injecting or killing dummy jobs to the system. We consider our experimental conditions no longer to be satisfied when the background load has exceeded 30 percent for more than 5 minutes. In such cases, the experiments were aborted and repeated.

In order to describe the performance metrics before presenting our results, we first discuss the timeline of a job submission in KOALA, as shown in Fig. 5. The time instant of the successful placement of a job is called its *placement time*. The *start time* of a job is the time instant when all components are ready to execute. The total time elapsed from the submission of a job until its start time is the *wait time* of a job. The time interval between the submission and the placement of a job shows the amount of time it spends in the placement queue, i.e., the *queue time*. The time interval between the placement time and the start time of a job is its *start-up overhead*.

6.2.2 Results

We will now present the results of our experiments for comparing the performance with and without coallocation with the WF and FCM policies, respectively, for workloads of real MPI applications. Fig. 6a shows the average job response time broken down into the wait time and the execution time for the workloads of all three applications, and Fig. 6b shows the percentages of coallocated jobs.

First of all, we have observed in our experiments that the start-up overhead of jobs is 10 seconds, on average, regardless of the number of clusters combined for it, and hence, from the values of the wait time shown in Fig. 6a, we conclude that the wait time is dominated by the queue time. Compared to what we have observed in [3] with Globus DUROC [13] and the MPIGH-2 [14] library for coallocation, the DRMAA-SGE [15] interface and the Open-MPI [7] library for coallocation yield a much lower start-up overhead, by a factor of 5 on average.

Fig. 6a indicates that for the workloads of the Prime and Poisson applications, the average job response time is lower
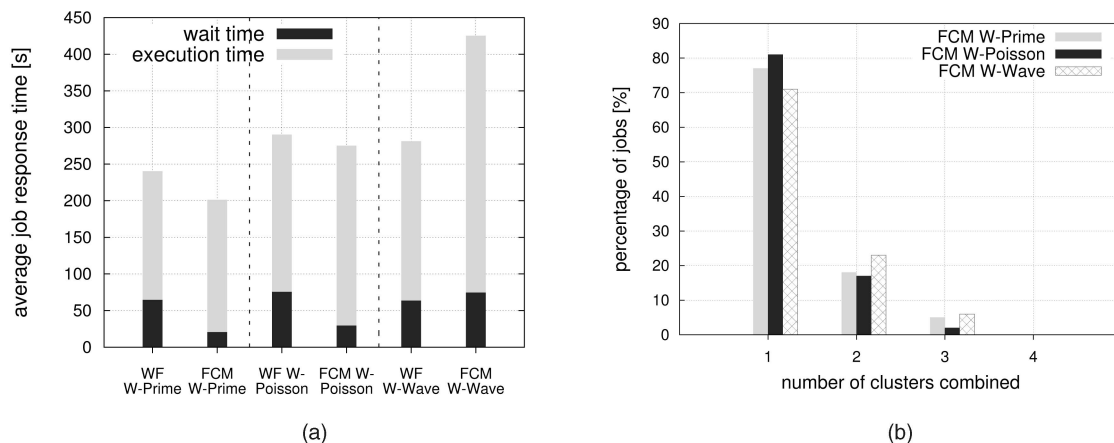

(a)

(b)

Fig. 6. Real experiments: (a) the average job response times and (b) the percentages of coallocated jobs of the workloads.
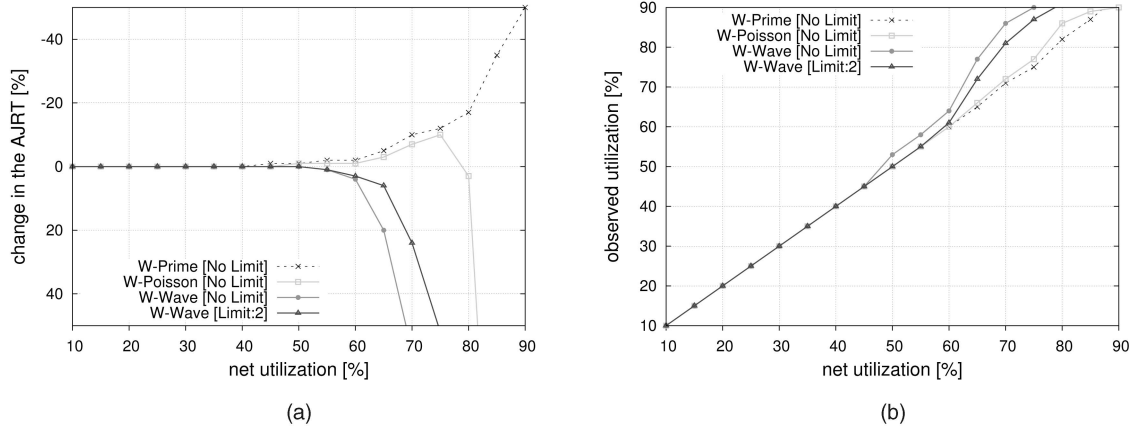
Fig. 7. Simulation experiments: (a) percentage of change in the average job response time for the workloads of the MPI applications when they are scheduled with FCM in comparison to when they are scheduled with WF, and (b) the observed utilization versus the net utilization when the workloads are scheduled with FCM.

when the workloads are scheduled with FCM compared to when they are scheduled with WF; however, the average job response time is higher for the workload of the Wave application with FCM. The FCM policy potentially decreases the job wait times since it is allowed to split up jobs in any way it likes across the clusters. Given that the execution times of the Prime Number and Poisson applications only slightly increase with coallocation, and the substantial reduction in wait time results in a lower average job response time.

For the Wave application, coallocation severely increases the execution time. As a consequence, the observed utilization also increases, causing higher wait times. Together, this leads to higher response times. As Fig. 6b indicates, a relatively small fraction of coallocation is responsible for the aforementioned differences in the average job response times between no coallocation and coallocation.

We conclude that in case of moderate resource contention (i.e., 40 percent workload + 10-30 percent background load), coallocation is beneficial for computation-intensive parallel applications (e.g., Prime) and communication-intensive applications whose slowdown due to the intercluster communication is low (e.g., Poisson). However, for very communication-intensive parallel applications (e.g., Wave), coallocation is disadvantageous due to the severe increase in the execution time. In the next section, we further evaluate the performance of no coallocation versus coallocation under various workload utilization levels using simulations.

6.3 Experiments in the Simulated Environment

In this section, as in the previous section, we first explain the experimental setup, and then, present and discuss the results of our simulations.

6.3.1 Experimental Setup

We have used the DGSim grid simulator [16] for our simulation-based experiments. We have modeled the KOALA grid scheduler with its job placement policies, the DAS-3 environment, and the three MPI applications based on their real execution times in single clusters and combinations of clusters. We have also modeled a synthetic application whose communication-to-computation ratio (CCR) can be modified. We define the CCR value for a parallel application as the ratio of its total communication time to its total computation time,

when executed in a *single cluster*. We set the total execution time of the application to 180 s in a single cluster irrespective of its CCR. For instance, for a CCR value of 1.0, both the communication and the computation part of the application take 90 s; for a CCR value of 0.5, these values are 60 and 120 s. When the application runs on coallocated clusters, the communication part is multiplied by a specific factor that is calculated from the real runs of the synthetic application on the corresponding coallocated clusters, and the total execution time of the application increases accordingly.

As in Section 6.2.1, we use workloads that each contains only one of MPI or the synthetic applications. In the experiments in which no coallocation is employed, the workloads are scheduled with the WF policy, and in the experiments in which coallocation is used, the workloads are scheduled with the FCM policy. For the workloads of the Wave application, we also consider the case in which FCM is limited to combine two clusters at most.

We consider jobs with total sizes of 8, 16, and 32 nodes so that the jobs can fit on any cluster in the system in case of no coallocation; the total sizes of the jobs are randomly chosen from a uniform distribution. For every application, we have generated 17 workloads with net utilizations ranging from 10 to 90 percent in steps of 5 percent. The job arrival process is Poisson. We assume that there is no background load in the system. Each workload runs for 24 simulated hours, under the policy in question, and we have again excluded the cluster in Delft.

6.3.2 Results

Fig. 7a shows the percentage of change in the average job response time (AJRT) for the workloads of the MPI applications when they are scheduled with FCM in comparison to when they are scheduled with WF. Fig. 7b illustrates the observed utilization versus the net utilization for the same workloads when they are scheduled with FCM. In Table 4, for each policy-workload pair, we present the net utilization interval in which saturation sets in and jobs are piled up in the queue and the wait times constantly increase without bounds.

When the resource contention is relatively low (up to 40 percent), with the job sizes included in the workloads, most jobs are placed in single clusters without a need for coallocation; hence, we observe no difference in the average job response times. For the computation-intensive Prime

TABLE 4
The Net Utilization Intervals in Which the
Policy-Workload Pairs Induce Saturation

Policy-Workload	Utilization Interval
WF {W-Prime, W-Poisson, W-Wave}	85-90%
FCM W-Prime	90-95%
FCM W-Poisson	80-85%
FCM W-Wave [No Limit]	70-75%
FCM W-Wave [Limit:2]	75-80%

TABLE 5
The Net Utilizations in the Real and the Simulation Experiments
Where the Changes in AJRTs Match

Workload	Change in the AJRT	Net Util. (in Real Exp.)	Net Util. (in Sim. Exp.)
W-Prime	-16%	40% + BG Load	75-80%
W-Poisson	-5%	40% + BG Load	65%
W-Wave	+50%	40% + BG Load	70%

application, the performance benefit of coallocation increases with the increase of the contention in the system, since jobs have to wait longer in the placement queue in case of no coallocation. In addition, as Table 4 shows that the workload of the Prime application causes saturation at lower utilizations when coallocation is not considered; the saturation point is in between 85-90 percent net utilization for WF W-Prime, and between 90-95 percent for FCM W-Prime.

We observe that for the Poisson application, coallocation is advantageous up to 75 percent net utilization, since the lower wait times compensate for the increase of the execution times. However, beyond this level, saturation sets in, and consequently, the average job response times increase.

For the Wave application, the extreme execution time increase of the jobs with coallocation increases the observed utilization in the system, as shown in Fig. 7b, which as a result causes an early saturation (see also Table 4). In addition, we see that limiting coallocation to two clusters yields a better response time performance than in case of no limit. However, the benefit is minor.

In order to compare real and simulation experiments, in Table 5, we present the net utilizations imposed by the workloads in the real and the simulation experiments where the percentages of change in the average job response times match. It turns out that the net utilization in the real experiments is lower than the net utilization in the corresponding simulation experiments, which is probably due to the background load in the real experiments having different characteristics than the workloads of MPI applications.

Fig. 8 shows the change in the average job response time for the workloads of the synthetic application with various CCR values. Comparing the results to those of the real MPI applications, we see that W-Prime matches CCR-0.1, W-Poisson matches CCR-0.25, and W-Wave matches CCR-4. The results with the workloads of the synthetic application exhibit the following. First, parallel applications with very low CCR values (i.e., 0.10) always benefit from coallocation. Second, for applications with CCR values between 0.25 and 0.50, coallocation is beneficial to a certain extent; with the increase of the contention in the system, the performance benefit of coallocation decreases and after some point, it becomes disadvantageous. Finally, for applications with
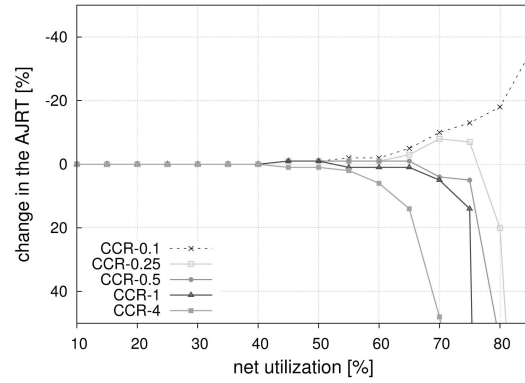

Fig. 8. Simulation experiments: percentage of change in the average job response time for the workloads of the synthetic application (with different CCR values) when they are scheduled with FCM in comparison to when they are scheduled with WF.

CCR values higher than 0.50, coallocation is disadvantageous since it increases the job response times severely.

7 PERFORMANCE OF THE PLACEMENT POLICIES

Although we have observed that it would be really advantageous to schedule communication-intensive applications on a single cluster from the perspective of the execution time (see in Fig. 4), users may still prefer coallocation when more processors are needed than available on a single cluster. In this section, we compare the FCM and CA policies in order to investigate their coallocation performance for communication-intensive parallel applications.

7.1 Experiments in the Real Environment

In this section, we present our experiments in the DAS-3. We first explain our experimental setup, and then, discuss the results.

7.1.1 Experimental Setup

In our experiments in this section, we use workloads comprising only the Concurrent Wave application [11], with a total job size of 64 nodes (128 processors). We have generated a workload with an average interarrival time determined in such a way that the workload is calculated to utilize approximately 40 percent of the system on average. The job arrival process is Poisson.

We handle the background load in the way mentioned in Section 6.2.1. We run the workload for 4 hours, under the policy in question. In the first set of experiments, we have excluded the cluster in Delft, and in the second set of experiments, we have excluded the cluster in Leiden and included the one in Delft; the interconnect technology used by a job is GbE when the cluster in Delft is involved in its coallocation, and Myri-10G otherwise.

7.1.2 Results

Fig. 9 shows the performance of the FCM and CA policies when scheduling the workload of the Wave application on the sets of clusters without and with the one in Delft.

In terms of the average job response time, the CA policy outperforms the FCM policy, irrespective of the involvement of the cluster in Delft, which has a slow intercluster communication speed. The difference in response time is moderate (50 s) or major (230 s) depending on whether the
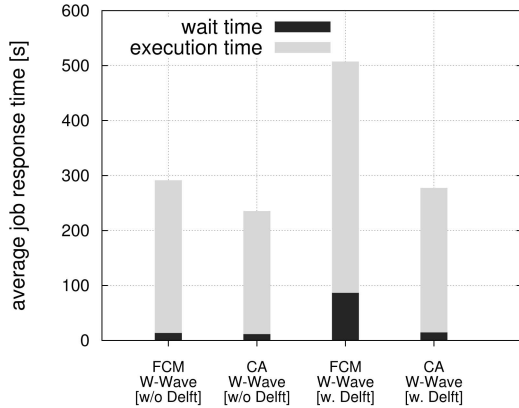
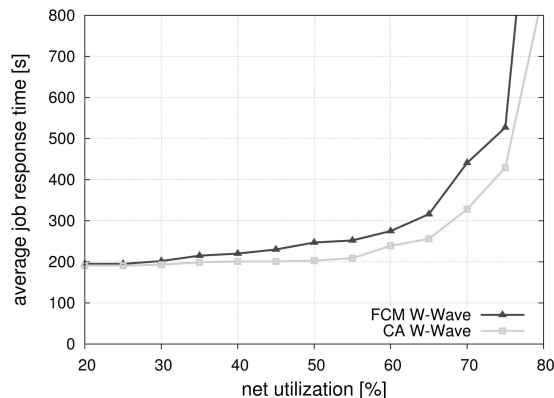Fig. 9. Real experiments: performance comparison of the FCM and CA policies.

cluster in Delft is excluded (communication speed has a low variability across the system) or included in the experiments (communication speed has a high variability across the system), respectively.

The CA policy tries to combine clusters that have faster intercluster communication (e.g., the clusters in Amsterdam and MultimediaN). However, as it is insensitive to communication speeds, the FCM policy may combine clusters with slower intercluster communication, which consequently increases the job response times. The increase is more severe when the cluster in Delft is included in the experiments, since it is involved in many of the coallocations for the jobs due to its large size.

We conclude that considering intercluster latency in scheduling communication-intensive parallel applications that require coallocation is useful, especially when the communication speed has a high variability across the system. In the following section, we extend our findings in the real environment by evaluating the performance of the FCM and CA policies under various resource contention levels in a simulated DAS-3 environment.

7.2 Experiments in the Simulated Environment

In this section, again, we first explain the experimental setup, and then, present and discuss the results of our simulations.
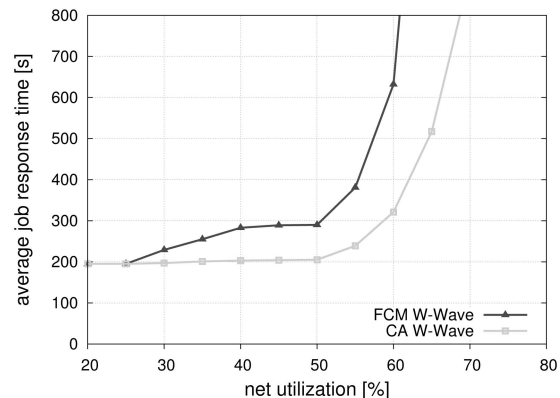
7.2.1 Experimental Setup

In our simulations, we use workloads comprising only the Concurrent Wave application, with total job sizes of 32, 48, and 64 nodes. The total sizes of the jobs are randomly chosen from a uniform distribution. We have generated 13 workloads with net utilizations ranging from 20 to 80 percent in steps of 5 percent. The job arrival process is Poisson. We assume that there is no background load in the system. Each workload runs for 24 simulated hours, under the policy in question.

In the first set of experiments, we have excluded the cluster in Delft, and in the second set of experiments, we have included the cluster in Delft and excluded the one in Leiden.

7.2.2 Results

Figs. 10a and 10b illustrate the average job response time results of the FCM and CA policies scheduling the workloads of the Wave application on the set of clusters either excluding the one in Delft or including it, respectively.

The CA policy outperforms the FCM policy for almost all utilization levels in both sets of experiments. As the utilization increases, the gap between the results of the two policies becomes wider. When the cluster in Delft is excluded, the system is saturated between 75 and 80 percent net utilization level; however, when it is included, the system is saturated between 60 and 70 percent net utilization, which is much less. The reason is that coallocating the cluster in Delft increases the job response times more severely.

We also see that the simulation results are consistent with the real experiments as the difference in the performance of the two policies is much larger when the cluster in Delft is included than when the cluster in Delft is excluded. This fact supports our claim that taking into account intercluster communication speeds improves the performance, especially when the communication speed has a high variability across the system.

To conclude, the results provide evidence that we should omit clusters that have slow intercluster communication speeds when coallocation is needed. In other words, in large systems, we should group clusters with similar intercluster

(a)

(b)

Fig. 10. Simulation experiments: performance comparison of the FCM and CA policies. (a) Cluster in Delft excluded. (b) Cluster in Delft included.

communication speeds, and restrict coallocation to those groups separately.

8 CHALLENGES WITH SUPPORTING COALLOCATION

Although we have demonstrated a case for supporting coallocation in a real environment with our KOALA grid scheduler, there are still many issues to be considered before processor coallocation may become a widely used phenomenon in multicluster grids and grid schedulers. In this section, we discuss some of these issues, related to communication libraries, processor reservations, and system reliability.

8.1 Communication Libraries

There are various communication libraries available [6], [7], [14], [17], [18] that enable coallocation of parallel applications. However, all these libraries have their own advantages and disadvantages; there is no single library we can name as the most suitable for coallocation. Some include methods for optimizing intercluster communication, some include automatic firewall and NAT traversal capabilities, and some may depend on other underlying libraries. Therefore, it is important to support several communication libraries as we do with the KOALA grid scheduler (e.g., MPICH-G2 [14], OpenMPI [7], and IBIS [6]).

8.2 Advance Processor Reservations

The challenge with simultaneous access to processors in multiple clusters of a grid lies in guaranteeing their availability at the start time of an application. The simplest strategy is to reserve processors at each of the selected clusters. If the Local Resource Managers (LRMs) of the clusters do support advance reservations, this strategy can be implemented by having a grid scheduler obtain a list of time slots from each LRM, reserve a common time slot for all job components, and notify the LRMs of this reservation. Unfortunately, a reservation-based strategy in grids is currently limited due to the fact that only few LRMs support advance reservations (e.g., PBS-pro [19], Maui [20]). In the absence of an advance reservation mechanism, good alternatives are required in order to achieve coallocation; for instance, with KOALA, we use a reservation mechanism (see [1]) that has been implemented on top of the underlying LRM.

8.3 System Reliability

The single most important distinguishing feature of grids as compared to traditional parallel and distributed systems is their multiorganizational character, which causes forms of heterogeneity in the hardware and software across the resources. This heterogeneity, in turn, makes failures appear much more often in grids than in traditional distributed systems. In addition, grid schedulers or resource management systems do not actually own the resources they try to manage, but rather, they interface to multiple instances of local schedulers in separate clusters which are autonomous and have different management architectures, which makes the resource management a difficult challenge.

We have experienced in our work on KOALA that even only configuring sets of processors in different administrative domains in a cooperative research environment are not a trivial task. Due to incorrect configuration of some of the nodes, during almost all our experiments, hardware failed and jobs were inadvertently aborted. To accomplish the experiments that we have presented in this study, we have spent more than half a year and have submitted more than 15,000 jobs to get reliable results. We claim that coallocation in large-scale dynamic systems such as grids require good methods for configuration management as well as good fault-tolerance mechanisms.

9 RELATED WORK

Various advance reservation mechanisms and protocols for supporting processor coallocation in grid systems have been proposed in the literature [21], [22], [23], [24], [25], [26]. Performance studies on coallocation, however, mostly studied in simulated environments; only a few studies investigate the problem in real systems. In this section, we discuss some of the studies that we find most related to our work.

In [27], [28], [29], we study through simulations processor coallocation in multiclusters with space sharing of rigid jobs for a wide range of such parameters as the number and sizes of the job components, the number of clusters, the service time distribution, and the number of queues in the system. The main results of our experiments are that coallocation is beneficial as long as the number and sizes of job components, and the slowdown of applications due to the wide-area communication, are limited.

Ernemann et al. [30] present an adaptive coallocation algorithm that uses a simple decision rule to decide whether it pays to use coallocation for a job, considering the given parameters such as the requested runtime and the requested number of resources. The slow wide-area communication is taken into account by a parameter by which the total execution time of a job is multiplied. In a simulation environment, coallocation is compared to keeping jobs local and compared to only sharing load among the clusters, assuming that all jobs fit in a single cluster. One of the most important findings is that when the application slowdown does not exceed 1.25, it pays to use coallocation.

Röblitz and coworkers [31], [32] present an algorithm for reserving compute resources that allows users to define an optimization policy if multiple candidates match the specified requirements. An optimization policy based on a list of selection criteria, such as end time and cost, ordered by decreasing importance, is tested in a simulation environment. For the reservation, users can specify the earliest start time, the latest end time, the duration, and the number of processors. The algorithm adjusts the requested duration to the actual processor types and numbers by scaling it according to the speedup, which is defined using speedup models or using a database containing reference values. This algorithm supports so-called fuzziness in the duration, the start time, the number of processors, and the site to be chosen, which leads to a larger solution space.

Jones et al. [33] present several bandwidth-aware coallocation metaschedulers for multicluster grids. These schedulers consider network utilization to alleviate the slowdown associated with the communication of coallocated jobs. For each job modeled, its computation time and

average per-processor bandwidth requirement are assumed to be known. In addition, all jobs are assumed to perform all-to-all global communication periodically. Several scheduling approaches are compared in a simulation environment consisting of clusters with globally homogeneous processors. The most significant result is that coallocating jobs when it is possible to allocate a large fraction (85 percent) of a single cluster provide the best performance in alleviating the slowdown impact due to intercluster communication.

The Grid Application Development Software (GrADS) [34] enables coallocation of grid resources for parallel applications that may have significant interprocess communication. For a given application, during resource selection, GrADS first tries to reduce the number of workstations to be considered according to their availabilities, computational and memory capacities, network bandwidth, and latency information. Then, among all possible scheduling solutions, the one that gives the minimum estimated execution time is chosen for the application. Different from our work, GrADS assumes that the performance model of the applications and mapping strategies is already available or can be easily created. While they present their approach's superiority over user-directed strategies, we handle the coallocation problem for various cases and present a more in-depth analysis.

In addition to the benefit of coallocation from a system's or users' point of view, various works also address the performance of a single coallocated parallel application [35], [36], [37]. A recent study by Seinstra and Geusebrock [38] presents a work on the coallocation performance of a parallel application that performs the task of visual object recognition by distributing video frames across coallocated nodes of a large-scale Grid system, which comprises clusters in Europe and Australia. The application has been implemented using the Parallel-Horus [39] tool, which allows researchers in multimedia content analysis to implement high-performance applications. The experimental results show the benefit of coallocation for such multimedia applications that require intensive computation and frequent data distribution.

10 CONCLUSION

In this paper, we have investigated the benefit of processor coallocation in a real multicluster grid system using our KOALA grid scheduler [1] as well as in a simulated environment using our DGSim tool [16]. Initially, we have assessed the impact of intercluster communication characteristics of a multicluster system on the execution time performance of a single coallocated parallel application. Then, we have evaluated the coallocation performance of a set of parallel applications that range from computation- to communication-intensive, under various utilization conditions. Finally, we have evaluated two scheduling policies for coallocating communication-intensive applications. We conclude the following.

First, the execution time of a single parallel application increases with the increase of the number of clusters combined. This increase depends very much on the communication characteristics of the application, and on the intercluster communication characteristics and the processor speed heterogeneity of the combined clusters.

Second, for computation-intensive parallel applications, coallocation is very advantageous provided that the differences between the processor speeds across the system are small. For parallel applications whose slowdown due to the intercluster communication is low, coallocation is still advantageous when the resource contention in the system is moderate. However, for very communication-intensive parallel applications, coallocation is disadvantageous since it increases execution times too much.

Third, in systems with a high variability in intercluster communication speeds, taking network metrics (in our case, the latency) into account in cluster selection increases the performance of coallocation for communication-intensive parallel applications.

Although there is a large opportunity for many scientific parallel applications to benefit from coallocation, there are still many issues that need to be overcome before coallocation can become a widely employed solution in future multicluster grid systems. The difference between inter- and intracluster communication speeds, efficient communication libraries, advance processor reservations, and system reliability is some of these challenges.

REFERENCES

[1] H. Mohamed and D. Epema, "Koala: A Co-Allocating Grid Scheduler," *Concurrency and Computation: Practice and Experience,* vol. 20, no. 16, pp. 1851-1876, 2008.
[2] H.H. Mohamed and D.H.J. Epema, "An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '04),* pp. 287-298, 2004.
[3] O.O. Sonmez, H.H. Mohamed, and D.H.J. Epema, "Communication-Aware Job Placement Policies for the KOALA Grid Scheduler," *Proc. Second IEEE Int'l Conf. e-Science and Grid Computing (E-SCIENCE '06),* p. 79, 2006.
[4] "The Distributed ASCI Supercomputer," http://www.cs.vu.nl/das3/, 2009.
[5] K. Czajkowski, I. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids," *Proc. Eighth IEEE Int'l Symp. High Performance Distributed Computing (HPDC '99),* p. 37, 1999.
[6] R.V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H.E. Bal, "Ibis: An Efficient Java-Based Grid Programming Environment," *Proc. Joint ACM ISCOPE Conf. Java Grande (JGI '02),* pp. 18-27, 2002.
[7] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," *Proc. 11th European PVM/MPI Users' Group Meeting,* pp. 97-104, Sept. 2004.
[8] Sun Grid Computing, http://wwws.sun.com/software/grid/, 2009.
[9] The Prime Number Application, http://www.mhpcc.edu/training/workshop/mpi/samples/C/mpi_prime.c, 2009.
[10] H.H. Mohamed and D.H.J. Epema, "The Design and Implementation of the KOALA Co-Allocating Grid Scheduler," *Proc. European Grid Conf.,* pp. 640-650, 2005.
[11] G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, and D.W. Walker, *Solving Problems on Concurrent Processors. Vol. 1: General Techniques and Regular Problems.* Prentice-Hall, Inc., 1988.

[12] A. Iosup and D.H.J. Epema, "Grenchmark: A Framework for Analyzing, Testing, and Comparing Grids," *Proc. Sixth IEEE Int'l Symp. Cluster Computing and the Grid (CCGRID '06)*, pp. 313-320, 2006.

[13] "The Dynamically-Updated Request Online Coallocator (DUR-OC)," http://www.globus.org/toolkit/docs/2.4/duroc/, 2009.

[14] "MPICH-G2," http://www3.niu.edu/mpi/, 2009.

[15] "Distributed Resource Management Application Api," http://www.drmaa.net/w/, 2008.

[16] A. Iosup, O. Sonmez, and D. Epema, "DGSim: Comparing Grid Resource Management Architectures through Trace-Based Simulation," *Proc. 14th Int'l Euro-Par Conf. Parallel Processing (Euro-Par '08)*, pp. 13-25, 2008.

[17] "Grid Ready MPI Library: MC-MPI," http://www.logos.ic.i.u-tokyo.ac.jp/h_saito/mcmpi/, 2008.

[18] "GridMPI," http://www.gridmpi.org/, 2009.

[19] "Portable Batch System-PRO," http://www.pbspro.com/platforms.html, 2009.

[20] "Maui Cluster Scheduler," http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php, 2009.

[21] F. Azzedin, M. Maheswaran, and N. Arnason, "A Synchronous Co-Allocation Mechanism for Grid Computing Systems," *Cluster Computing*, vol. 7, no. 1, pp. 39-49, 2004.

[22] J. Sauer, "Modeling and Solving Multi-Site Scheduling Problems," *Planning in Intelligent Systems: Aspects, Motivations and Methods*, A.M. Meystel, ed., pp. 281-299, Wiley, 2006.

[23] A.C. Sodan, C. Doshi, L. Barsanti, and D. Taylor, "Gang Scheduling and Adaptive Resource Allocation to Mitigate Advance Reservation Impact," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGRID)*, pp. 649-653, 2006.

[24] J. Li and R. Yahyapour, "Negotiation Model Supporting Co-Allocation for Grid Scheduling," *Proc. IEEE/ACM Int'l Conf. Grid Computing*, pp. 254-261, 2006.

[25] C. Qu, "A Grid Advance Reservation Framework for Co-Allocation and Co-Reservation across Heterogeneous Local Resource Management Systems," *Proc. Int'l Conf. Parallel Processing and Applied Math. (PPAM)*, pp. 770-779, 2007.

[26] C. Castillo, G.N. Rouskas, and K. Harfoush, "Efficient Resource Management Using Advance Reservations for Heterogeneous Grids," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '08)*, pp. 1-12, 2008.

[27] A.I.D. Bucur and D.H.J. Epema, "The Maximal Utilization of Processor Co-Allocation in Multicluster Systems," *Proc. 17th Int'l Symp. Parallel and Distributed Processing (IPDPS '03)*, p. 60.1, 2003.

[28] A.I.D. Bucur and D.H.J. Epema, "The Performance of Processor Co-Allocation in Multicluster Systems," *Proc. Third Int'l Symp. Cluster Computing and the Grid (CCGRID '03)*, p. 302, 2003.

[29] A.I.D. Bucur and D.H.J. Epema, "Scheduling Policies for Processor Coallocation in Multicluster Systems," *IEEE Trans. Parallel Distributed Systems*, vol. 18, no. 7, pp. 958-972, July 2007.

[30] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "On Advantages of Grid Computing for Parallel Job Scheduling," *Proc. Second IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGRID '02)*, pp. 39-46, May 2002.

[31] T. Roblitz and A. Reinefeld, "Co-Reservation with the Concept of Virtual Resources," *Proc. Fifth IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '05)*, pp. 398-406, 2005.

[32] T. Röblitz, F. Schintke, and A. Reinefeld, "Resource Reservations with Fuzzy Requests: Research Articles," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 13, pp. 1681-1703, 2006.

[33] W. Jones, L. Pang, W. Ligon, and D. Stanzione, "Bandwidth-Aware Co-Allocating Meta-Schedulers for Mini-Grid Architectures," *Proc. IEEE Int'l Conf. Cluster Computing*, pp. 45-54, 2004.

[34] H. Dail, F. Berman, and H. Casanova, "A Decoupled Scheduling Approach for Grid Application Development Environments," *J. Parallel and Distributed Computing*, vol. 63, no. 5, pp. 505-524, 2003.

[35] A. Plaat, H.E. Bal, and R.F.H. Hofman, "Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects," *Future Generation Computer Systems*, vol. 17, no. 6, pp. 769-782, 2001.

[36] T. Kielmann, H.E. Bal, S. Gorlatch, K. Verstoep, and R.F. Hofman, "Network Performance-Aware Collective Communication for Clustered Wide-Area Systems," *Parallel Computing*, vol. 27, no. 11, pp. 1431-1456, 2001.

[37] R.V. van Nieuwpoort, T. Kielmann, and H.E. Bal, "Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications," *Proc. Eighth ACM SIGPLAN Symp. Principles and Practices of Parallel Programming (PPoPP '01)*, pp. 34-43, 2001.

[38] F.J. Seinstra and J.M. Geusebroek, "Color-Based Object Recognition by a Grid-Connected Robot Dog," *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2006.

[39] F.J. Seinstra, J.-M. Geusebroek, D. Koelma, C.G. Snoek, M. Worring, and A.W. Smeulders, "High-Performance Distributed Video Content Analysis with Parallel-Horus," *IEEE MultiMedia*, vol. 14, no. 4, pp. 64-75, Oct.-Dec. 2007.

Omer Ozan Sonmez received the BSc degree in computer engineering from Istanbul Technical University, Turkey, in 2003, and the MSc degree in computer science from the Koc University, Turkey, in 2005. He is currently working toward the PhD degree at the Parallel and Distributed Systems Group, Delft University of Technology, The Netherlands. His research interests focus on resource management and scheduling in multicluster systems and grids.

Hashim Mohamed received the BSc degree in computer science from the University of Dar-es-Salaam, Tanzania, in 1998, and the MSc degree in technical informatics and the PhD degree in 2001 and 2007, respectively, from the Delft University of Technology, The Netherlands, where he currently works as a software programmer. Between June 1998 and January 1999, he worked at the University of Dar-es-Salaam Computing Center as a systems analyst/programmer. His research interests are in the areas of distributed systems, multicluster systems, and grids in general.

Dick H.J. Epema received the MSc and PhD degrees in mathematics from Leiden University, The Netherlands, in 1979 and 1983, respectively. From 1983 to 1984, he was with the Computer Science Department, Leiden University. Since 1984, he has been with the Department of Computer Science, Delft University of Technology, where he is currently an associate professor in the Parallel and Distributed Systems Group. During the academic year 1987-1988, the Fall of 1991, and the Summer of 1998, he was also a visiting scientist at the IBM T.J. Watson Research Center, Yorktown Heights, New York. In the Fall of 1992, he was a visiting professor at the Catholic University of Leuven, Belgium. His research interests are in the areas of performance analysis, distributed systems, peer-to-peer systems, and grids.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.