The performance of processor co-allocation in multicluster systems

Citation for published version (APA):

Bucur, A. I. D., & Epema, D. H. J. (2003). The performance of processor co-allocation in multicluster systems. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 12-15, 2003 (pp. 302-309). Institute of Electrical and Electronics Engineers.
<https://doi.org/10.1109/CCGRID.2003.1199381>

DOI:

[10.1109/CCGRID.2003.1199381](https://doi.org/10.1109/CCGRID.2003.1199381)

Document status and date:

Published: 01/01/2003

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

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](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

The Performance of Processor Co-Allocation in Multicenter Systems

A.I.D. Bucur and D.H.J. Epema
Faculty of Information Technology and Systems
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, The Netherlands
e-mail: A.I.D.Bucur, D.H.J.Epema@its.tudelft.nl

Abstract

In systems consisting of multiple clusters of processors which are interconnected by relatively slow communication links and which employ space sharing for scheduling jobs, such as our Distributed ASCI¹ Supercomputer (DAS), co-allocation, i.e., the simultaneous allocation of processors to single jobs in different clusters, may be required. We study the performance of co-allocation by means of simulations for the mean response time of jobs depending on the structure and sizes of jobs, the scheduling policy, and the communication speed ratio. Our main conclusion is that for current communication speed ratios in multicenter systems, co-allocation is a viable option.

1 Introduction

Over the last decade, clusters and distributed-memory multiprocessors consisting of hundreds or thousands of standard CPUs have become very popular. In addition, recent work in computational and data GRIDs [7, 2] enables applications to access resources in different and possibly widely dispersed locations simultaneously—that is, to employ processor *co-allocation* [4]—to accomplish their goals, effectively creating single multicenter systems. Most of the research on processor scheduling in parallel computer systems has been dedicated to multiprocessors and single-cluster systems, but hardly any attention has been devoted to multicenter systems. In this paper we study through simulations the performance of processor co-allocation policies in multicenter systems employing space sharing for rigid jobs [3] depending on such parameters as the structure and the sizes of job requests, and the *communication speed ratio*, which is the ratio between the intracluster and the intercluster communication speeds. The scheduling policies we consider are First Come First Served (FCFS) and Fit Processors First Served (FPFS). Our performance metric is the mean job response time as a function of the utilization. In addition to synthetic workloads, we consider a workload de-

rived from measurements of a real application on our five-cluster Distributed ASCI Supercomputer (DAS) (for details, see Sect. 2.1). This system was designed to show the feasibility of running parallel applications across wide-area systems [9, 11]. We find that the request types and the slow intercluster communication have a severe impact on response times, that co-allocation can be a good choice in many situations, and that for a mix of request types, the performance is determined not only by their separate behaviours, but also by the way in which they interact.

In the most general setting, GRID resources are very heterogeneous and wide-area connections may be slow. In this paper we restrict ourselves to homogeneous multicenter systems such as the DAS, with relatively low communication speed ratios. Showing the viability of co-allocation in such systems may be regarded as a first step in assessing the benefit of co-allocation in more general GRID environments.

Not much work has been done related to the performance of co-allocating rigid jobs with space sharing in multicenter systems. In [5], co-allocation (called multi-site computing there) is studied, with as performance metric the (average weighted) response time. There, jobs only specify a total number of processors, and are split up across the clusters. The slow wide-area communication is accounted for by a factor r by which the total execution times are multiplied. Co-allocation is compared to keeping jobs local and to only sharing load among the clusters, assuming that all jobs fit in a single cluster. One of the most important findings is that for r less than or equal to 1.25, it pays to use co-allocation. In [12], a performance comparison of two meta-schedulers is presented. It is shown that dedicating parts of subsystems to jobs that need co-allocation is not a good idea.

2 The Model

In this section we describe our model of multicenter systems based on the DAS system.

2.1 The DAS System

The (second-generation) DAS [1, 8] is a wide-area computer system consisting of five clusters of identical processors (in total 400). The clusters are interconnected by the

¹In this paper, ASCI refers to the Advanced School for Computing and Imaging in The Netherlands, which came into existence before, and is unrelated to, the US Accelerated Strategic Computing Initiative.

Dutch University backbone for wide-area communications, while for local communication inside the clusters Myrinet LANs are used. The ratio of the communication speeds is about two orders of magnitude. So far, co-allocation has not been used enough on the DAS to let us obtain statistics on the sizes of the jobs' components. However, from the log of the largest cluster of the (first-generation) system (of 128 processors) we found that over a period of three months, the cluster was used by 20 different users who ran 30,558 jobs. The sizes of the job requests took 58 values in the interval $[1, 128]$, for an average of 23.34; their distribution is presented in Fig. 1. The results comply with one of the distributions we use for the job-component sizes (see below) in that there is an obvious preference for small numbers and powers of two.

Figure 1. The job-request-size distribution for the largest cluster in the first-generation DAS (128 processors)

2.2 The Structure of the System

We model a multicluster consisting of C clusters of identical processors, cluster i having N_i processors, $i = 1, \dots, C$. The workload consists of rigid jobs that require fixed numbers of processors, possibly in multiple clusters simultaneously (*co-allocation*). We call a task the part of a job that runs on a single processor. The system has a single central scheduler, with one global queue. For the interarrival times we use exponential distributions. All intra-cluster communication links have the same speed, as do all the intercluster links, with the former assumed to be (much) faster. In our model, the *communication speed ratio* is defined as the ratio between the time needed to complete a single synchronous send operation between processors in different clusters and in the same cluster.

2.3 The Structure of Job Requests

Jobs that require co-allocation specify the number and the sizes of their components, i.e., of the sets of tasks that have to go to the separate clusters. The distribution J of job-component sizes is either the uniform distribution $U[n_1, n_2]$ on some interval $[n_1, n_2]$, or the distribution $D(q)$ on some interval $[n_1, n_2]$ with the probability of having job-component size i equal to $p_i = 3q^i/Q$ (q^i/Q) if i is (is not) a power of 2, and with Q such that the p_i sum to 1. This distribution favours small sizes and powers of 2, which has

been found to be realistic [6]. A job is represented by a tuple of C values, each generated from J . We will consider four cases for the structure of job requests:

1. In an *ordered request* the positions of the request components in the tuple specify the clusters from which the processors must be allocated.
2. For an *unordered request*, by the components of the tuple the job only specifies the numbers of processors it needs in the separate clusters, allowing the scheduler to choose the clusters for the components.
3. A *flexible request* specifies the total number of processors needed, obtained as the sum of the values in the tuple.
4. For *total requests*, there is a single cluster and a request specifies the single number of processors needed, again obtained as the sum of the values in the tuple.

As long as we do not take into account the communication between processors, the cases of flexible and total requests amount to the same. Ordered requests are used in practice when a user has enough information about the complete system to take full advantage of the characteristics of the different clusters. For example, the data available at the different clusters may dictate a specific way of splitting up an application. Unordered requests model applications like FFT, where tasks in the same job component share data and need intensive communication, while tasks from different components exchange little or no information.

2.4 The Communication Pattern and the Scheduling Decisions

As a model for the structure of jobs, we consider a parallel application implementing an iterative method, each process of which performs the same algorithm on its subdomain, alternating between computation and communication steps; in the latter, we assume that an all-to-all personalized message exchange is performed.

To determine whether an unordered request fits, we try to schedule its components in decreasing order of their sizes on distinct clusters. Possible ways of placement include First Fit (FF) and Worst Fit (WF), both of which we use in our simulations.

If for a flexible request there are enough idle processors in the whole system, the job is placed on the system according to one of three placement policies, *Cluster-Filling* (CF), and *Load-Balancing* on the *Smallest* number of (on *All*) clusters (LB-S and LB-A). Both CF and LB-S place a job on the smallest set of clusters with enough idle processors. CF completely fills the clusters with the largest numbers of idle processors until all the tasks are distributed; LB-S distributes the request over these clusters in such a way as to balance the load. LB-A does the same with all clusters.

In all the simulations except in Sect. 3.2, the First Come First Served (FCFS) policy is used. To observe the contribution of the scheduling scheme to the system's performance,

we have also implemented the Fit Processors First Served (FPFS) policy, in which the scheduler searches the queue from head to tail, and schedules all the jobs that fit. It is similar to backfilling [3], except that the durations of jobs (which we assume unknown) are not taken into account, so the requirement that the job at the head of the queue should not be delayed is not enforced. To avoid starvation, we use counters: when a job's counter reaches a chosen limit, the scheduler is not allowed to overpass that job anymore.

3 Simulating Co-Allocation

To measure the performance of multicluster systems for different structures and sizes of requests and the two scheduling policies, we modeled the corresponding queuing systems and studied their behaviour using simulations. To isolate these aspects, in this section we ignore the communication (so flexible and total requests amount to the same). The simulation programs were implemented using the CSIM simulation package [10]. When confidence intervals are included, they are at the 95%-level.

3.1 The Influence of the Component Size and Service Time Distributions

We assess the performance of a system consisting of 4 clusters with 32 processors each, for flexible, unordered (with WF) and ordered requests depending on the distribution of the job-component sizes and on the service-time distribution. We consider the five distributions for the sizes of the job components described in Table 1. Notice that $U[1, 7]$, $D(0.9)$ and $D(0.768)$ have approximately the same mean 4, and $U[1, 14]$ and $D(0.894)$ have mean 7.5.

Table 1. The job-component-size distributions considered in Sect. 3.1

distribution	mean	coeff. of var.
$U[1, 7]$	4.000	0.500
$D(0.9)$ on $[1, 8]$	3.996	0.569
$D(0.768)$ on $[1, 32]$	3.996	0.829
$U[1, 14]$	7.500	0.537
$D(0.894)$ on $[1, 32]$	7.476	0.884

We first fix the service-time distribution to be exponential with mean 1. From Fig. 2 we can conclude that independent of the component-size distribution, flexible requests generate the best performance and ordered requests the worst. For both the uniform distributions and the D distributions, a larger mean implies a worse performance. For the same mean, uniform distributions for the job components yield better performance than the D distributions, because they provide better packing due to a smaller maximum value for the components and a smaller coefficient of variation. For the most rigid type of requests, ordered requests, the mean of the distribution matters less than its range and variance, since although its mean is 7.5, $U[1, 14]$ has a better performance than $D(0.768)$ which has mean 4. Similarly, even though $D(0.9)$ on $[1, 8]$ and $D(0.768)$ on $[1, 32]$ have both mean 4, the first distribution displays better performance, especially for ordered and unordered requests

Figure 2. The influence of the component-size distribution for flexible (top), unordered (middle) and ordered (bottom) requests, for exponential service times

where not only the total size of the job request matters, but also the sizes of the components and the way they fit in the system.

We now consider three distributions for the service time, deterministic, exponential and hyperexponential (with coefficient of variation 3), all with mean 1. Figures 3 and 4 show the performance of the system for these distributions, for flexible, unordered and ordered requests, for component-size distributions $D(0.768)$ on $[1, 32]$ and $U[1, 14]$, respectively. The results are similar for the other distributions in Table 1. For all types of requests and for both component-size distributions, the best performance is obtained for deterministic service time and the worst for hyperexponential service time. The graphs indicate that for ordered requests the performance is better when components are from the uniform distribution, for all service time distributions, which is consistent with what we observed above for exponential service times.

In the rest of the paper (except for Sect. 5) we will only use exponential distributions for the service times (with mean 1).

Figure 3. The influence of the service time distribution for flexible (top), unordered (middle) and ordered (bottom) requests, for job-component distribution $D(0.768)$ on $[1, 32]$

3.2 The Impact of the Scheduling Policy

Before we can compare FCFS and FPFS, we have to choose the value of MaxJumps—the maximum number of times a job can be jumped over in FPFS. When MaxJumps is too large, the performance of individual jobs is negatively influenced, to the point that when $\text{MaxJumps} \rightarrow \infty$, which amounts to FPFS without counters, starvation may occur under heavy loads. At the other extreme, with MaxJumps equal to 0 we return to FCFS. Based on simulation results concerning the sensitivity of the response time to the value of MaxJumps for different values of the utilization, we chose MaxJumps equal to 7. Simulations are reported for a single-cluster system with 32 processors and for a multicluster system with 4 clusters of 8 nodes each. Job component sizes are from $U[1, 4]$ and $U[1, 8]$. Figure 5 shows that FPFS improves the performance of the system. In addition, the influence of the structure and sizes of the requests observed for FCFS is maintained with FPFS.

4 Co-Allocation with Communication

In this section we assess the performance of multicluster systems for different types of requests in the pres-

Figure 4. The influence of the service time distribution for flexible (top), unordered (middle) and ordered (bottom) requests, for job-component distribution $U[1, 14]$

ence of communication. The simulations, unless otherwise specified, are for systems with 4 clusters of 8 processors each, and for single clusters with 32 processors. The job-component-size distribution we use is $D(0.9)$ on $[1, 8]$.

The effect of including communication in our model is that the service times of tasks are extended by the times needed for communication—which we suppose to be the sum of the times needed for all send operations to the other tasks. We only simulate a single iteration. We still assume the duration of the computation steps to be exponential with mean 1, and we assume the time needed for a single local send operation to be 0.001. With our parameter settings, when the communication speed ratio is 100, a task of an (un)ordered request has an average total service time of about 2.2 (1 computation + $12 \cdot 0.1$ wide-area communication). In [11], the results of running a version optimized for wide-area systems of the Barnes-Hut algorithm for the N -body simulation problem which uses personalized all-to-all communication on the DAS are presented. It turns out that when the ratios of wide-area latency to local latency, and of local bandwidth to wide-area bandwidth are 100, the

Figure 5. The impact of the scheduling policy for job-component sizes from $U[1, 4]$ (top) and $U[1, 8]$ (bottom)

speedup of the algorithm in a multicluster with 4 clusters of size 8 relative to a single cluster of size 32 is about 45%, which matches the extension by the factor of 2.2 of the service time derived from our parameters.

4.1 The Communication Speed Ratio

From Fig. 6 we find that for ordered, unordered (with WF component placement) and flexible requests (with LB-S), the increase of the communication speed ratio increases the response time. The performance is affected more for (un)ordered requests than for flexible ones. This suggests that in those cases the average amount of intercluster communication per job is higher. A higher communication ratio decreases the maximal utilization as well, but the deterioration is smaller than in the case of the response time. (Going from a ratio of 10 to 100, the response times at a utilization of 0.4 are increased by a factor of about 1.8 (1.6) for (un)ordered (flexible) requests, while the maximal utilizations are reduced by about 0.05, which is about 10%.)

4.2 Co-Allocation versus no Co-Allocation

In this section we discuss the benefits of using co-allocation over using no co-allocation when there is a choice (when all jobs can be scheduled without co-allocation, on a single cluster). We show that rather than waiting for enough idle processors in one cluster, spreading requests over clusters can bring better performance, despite the extra communication. We consider a multicluster system consisting of 4 clusters with 32 processors each. In the co-allocation case we schedule flexible requests using LB-A, in the no-co-allocation case we have jobs consisting of a single component. The request sizes are obtained as the sum of 4 components generated from $D(0.9)$ on $[1, 8]$. Note that LB-A is

Figure 6. The average response time as a function of the utilization for the four request types for communication speed ratios of 10 (top), 50 (middle), and 100 (bottom)

not very favourable for co-allocation, as it may entail much time spent in intercluster communication (see Sect. 4.3).

Figure 7 (top) compares co-allocation with flexible requests with no co-allocation with single-component jobs using FF (the results for WF are slightly worse). At low utilizations avoiding co-allocation can bring benefits when the speed ratio is large. However, for intermediate and high system loads, co-allocation is significantly better, even for speed ratios larger than 50. We conclude that co-allocation is a very good choice for systems with speed ratios that are not extremely large.

We studied further why co-allocation gives better results by looking at the way the scheduler works. The scheduler only makes an attempt at scheduling jobs when a job departs and the queue is not empty, and when a job arrives at an empty system. We call a scheduling attempt unsuccessful when no job can be scheduled. Figure 7 (bottom) shows the percentages of unsuccessful scheduling attempts with and without co-allocation, with a communication speed ratio of 50, as a function of the utilization (the graph stops

Figure 7. Co-allocation for flexible requests with LB-A and different communication speed ratios versus no co-allocation with FF: response times (top) and percentages of unsuccessful scheduling attempts (bottom)

just before the no-co-allocation case becomes unstable). Especially at average and high utilizations, in the absence of co-allocation, it occurs much more often that no job can be scheduled at a job departure.

4.3 Load-Balancing versus Cluster-Filling

To reduce intercluster communication, placement policies for flexible requests should not spread jobs unnecessarily over the clusters. Therefore, both CF and LB-S choose the minimal number of clusters, but whereas CF aims to reduce the number of intercluster messages, LB-S may have a larger number of intercluster messages while balancing loads. Since LB-A potentially spreads jobs over more clusters, in smaller components, it may cause a still larger number of such messages. It seems that there is no benefit from using LB-S or LB-A when there are only flexible requests, and they should only be used when there are also (un)ordered requests, because then keeping an equilibrium between the loads of the clusters becomes essential (see Sect. 4.4). However, in Figure 8, even for only flexible requests, LB-S proves to be better. The explanation is that although CF has fewer intercluster messages per job, the communication time is not evenly spread across the tasks, and on average, CF causes jobs to spend more time on communication.

In a similar comparison of LB-S and LB-A, we found that at low utilizations, LB-A gives a slightly higher response time due to more communication, but at higher utilizations this difference attenuates, being within the confidence intervals. For smaller jobs and higher speed ratios we expect this difference in performance to grow, implying that

Figure 8. Performance comparison between LB-S and CF for flexible requests, for communication speed ratios of 50 (top) and 100 (bottom)

when only flexible jobs are admitted in the system, there is no reason to choose LB-A.

As an example, consider the consecutive arrival of two flexible requests for 18 and 12 processors at an empty system with 4 clusters of 8 processors. For CF, LB-S and LB-A, the tuples of the numbers of tasks on the clusters for the first and the second job will be $(8, 8, 2, 0)$, $(6, 6, 6, 0)$, $(5, 5, 4, 4)$, and $(0, 0, 4, 8)$, $(0, 2, 2, 8)$, and $(3, 3, 3, 3)$, respectively. The total numbers of intercluster messages in a communication step for the first and the second job are 192, 216, 242, and 64, 72, 108, while the longest tasks in the two jobs have 16, 12, 14, and 8, 10, 9 such messages. This shows that CF does not entail more time spent in communication than LB-S for every job, although in Fig. 8 on average it apparently does.

When CF and LB-S are faced with a system with the same numbers of idle processors in the clusters when placing a flexible request, CF entails a higher maximal number of intercluster messages across all tasks in the job, but a lower total number of such messages than LB-S.

4.4 Mixes of Job Request Types

Real systems have to schedule multiple request types, and the overall performance of the system is determined by the way the scheduler deals with all these types. We study multiclusters with both flexible and ordered requests, and compare the performance of using CF, LB-S and LB-A. Figure 9 shows that for both CF and LB-S the average response time grows and the maximal utilization decreases with the increase of the percentage of ordered jobs: going from a low to a high percentage of ordered requests, the performance is ever more determined by this type of requests.

Figure 9. The performance of a system with mixed requests for different percentages of flexible and ordered requests, with LB-S (top) and CF (bottom)

LB-S, and in particular CF, tend to obstruct ordered requests because they use the minimum number of clusters for flexible requests. As Fig. 9 indicates, for up to 50% ordered requests the system behaves better with LB-S. This is due mostly to the smaller amount of time spent in communication by the flexible requests. Because the jobs are large enough not to permit in average the presence in the system of more than two jobs simultaneously (average job size 16), there is little benefit for ordered jobs from load balancing.

One may ask whether the job sizes play a role here. Figure 10 depicts the variation of the response time for a job mix of 50% ordered requests and 50% flexible requests, with job components obtained from $U[1, 4]$ (average job size 10). In this case the difference in performance is larger, so LB-S brings more benefit when jobs are smaller.

Figure 10. The performance of a system with mixed requests for equal percentages of flexible and ordered requests with job-component sizes obtained from $U[1, 4]$

We saw in Sect. 4.3 that with only flexible requests, there is no reason to use LB-A. However, we expect that

in a system where flexible requests are mixed with ordered requests, LB-A can bring relevant performance improvements. Comparing the performance of a system with mixed requests for the two types of LB we concluded that since the job sizes are large compared to the clusters' sizes, the benefit from using LB-A is insignificant, and even when 50% of the jobs are flexible, the performance is dominated by the ordered requests. Figure 11, where job-component sizes are obtained from $U[1, 4]$, shows a different situation. When jobs are small, the advantage of using LB-A is obvious, to the extent that LB-A with 30% percent flexible requests is better than LB-S with 50% percent flexible requests.

Figure 11. The performance of a system with mixed requests (ordered and flexible in different percentages) for LB-S and LB-A with job-component sizes from $U[1, 4]$

5 Simulations Based on a Real Application

In this section we present simulation results for ordered and total requests based on data obtained from measurements when running a real application on the DAS.

5.1 Description of the Application

Our application implements a parallel iterative algorithm to solve the Poisson equation with a red-black Gauss-Seidel scheme. It searches for a solution in its computational domain, which is the unit square split up into a grid of points with constant step. In each iteration, each grid point has its value updated as a function of its previous value and the values of its neighbours. The domain of the problem is split into rectangles among the participating processes, which communicate in order to exchange the values of the grid points on the borders and to compute a global stopping criterion. For the same grid size and the same initial data, we ran the application on a single DAS cluster with 8 processes (4x2 and 8x1 rectangles) and 16 processes (2x8 and 4x4), and on the 4-cluster DAS for configurations 4x2 and 4x4, scheduling an equal number of processes on each cluster. In Table 2 we report the number of iterations needed to reach convergence, the duration of the computation steps, and the total time needed for exchanging borders; the time cost of diffusing the local errors and computing the global error is always about 0.014 s.

Table 2. Measurements of running our application on the DAS

Config.	Nr. of Iterat.	Update (ms)	Exchange Borders (ms)	
			single-cluster	multi-cluster
4x2	2436	0.953-0.972	0.408-0.450	5.9-7.3
8x1	2418	0.970-0.994	0.260-0.315	—
4x4	2132	0.480-0.515	0.350-0.425	6.3-7.7
8x2	2466	0.470-0.525	0.337-0.487	—

5.2 Simulation Results

We use the application structure described and the data collected from the DAS for job sizes 8 and 16 (cases 4x2 and 4x4, respectively) in our simulations of a single cluster of size 32 and a multicluster with 4 clusters of size 8. The distribution of the job sizes is an equal mix of sizes 8 and 16, and of sizes (2,2,2,2) and (4,4,4,4), in the single cluster and the multicluster, respectively. The co-allocation on the DAS corresponds to the case of ordered requests in our simulations. Since the job-component sizes are equal, identical performance would be obtained for unordered requests.

The results are presented in Fig. 12 (we set the number of iterations to 10% of the values in Table 2). Since for the multicluster simulations the components of the requests are of equal sizes, which sum up to the two values of the request sizes used for the single-cluster case, and the total size of the multicluster is equal to the size of the single cluster, the two cases would have identical performance in the absence of communication. This implies that the worse performance displayed by the ordered requests compared to the total requests is caused by the slow intercluster communication and not by the rigidity of the ordered requests. We conclude that in practice, the performance penalty of running multicluster jobs may not be prohibitive.

Figure 12. The average response time as a function of the utilization for ordered and total requests, with data obtained from an application on the DAS

6 Conclusions

In this paper we have studied the performance in terms of the response time of processor co-allocation in multicluster systems with space sharing for rigid multi-component jobs

of different request types that impose various restrictions on their placement. Our main conclusions are as follows:

- *Restrictive job request types yield worse performance.* Independent of the component-size and service-time distributions, when a large fraction of the workload consists of (un)ordered jobs (specifying the numbers of processors needed in distinct clusters, or, worse yet, in each cluster), the response time is higher.
- *Slow intercluster communication deteriorates performance, but does not preclude co-allocation.* A high communication speed ratio between intercluster and local communication has a severe impact on performance. However, even when this ratio is 50, co-allocation for workloads with a large fraction of flexible requests yields acceptable performance, and is to be preferred over no co-allocation even when all jobs would fit in a single cluster.
- *The scheduling decisions should take into account the presence of mixes of request types.* When the workload consists of flexible and ordered requests, the former should be placed in a balanced way across the clusters to keep room for the latter, even though this brings them an increased amount of intercluster communication.

References

- [1] The DAS. www.cs.vu.nl/das2.
- [2] The Global Grid Forum. www.gridforum.org.
- [3] K. Aida, H. Kasahara, and S. Narita. Job Scheduling Scheme for Pure Space Sharing among Rigid Jobs. In D. Feitelson and L. Rudolph, editors, *4th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of LNCS, pages 98–121. Springer-Verlag, 1998.
- [4] K. Czajkowski, I. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. In *8th IEEE Int'l Symp. on High Perf. Distrib. Comp.*, pages 219–228, 1999.
- [5] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On Advantages of Grid Computing for Parallel Job Scheduling. In *2nd IEEE/ACM Int'l Symposium on Cluster Computing and the GRID (CCGrid2002)*, pages 39–46, 2002.
- [6] D. Feitelson and L. Rudolph. Toward Convergence in Job Schedulers for Parallel Supercomputers. In D. Feitelson and L. Rudolph, editors, *2nd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of LNCS, pages 1–26. Springer-Verlag, 1996.
- [7] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [8] H.E. Bal et al. The Distributed ASCI Supercomputer Project. *ACM Operating Systems Review*, 34(4):76–96, 2000.
- [9] T. Kielmann, R. Hofman, H. Bal, A. Plaat, and R. Bhoedjang. MagPie: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 131–140, 1999.
- [10] Mesquite Software, Inc. *The CSIM18 Simulation Engine*.
- [11] A. Plaat, H. Bal, R. Hofman, and T. Kielmann. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. *Future Generation Computer Systems*, 17:769–782, 2001.
- [12] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-Scheduling. In D. Feitelson and L. Rudolph, editors, *6th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of LNCS, pages 137–153. Springer-Verlag, 2000.