Scheduling policies for processor coallocation in multicluster systems


DOI:
10.1109/TPDS.2007.1036

Document status and date:
Published: 01/01/2007

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.

Download date: 21. Oct. 2023
Scheduling Policies for Processor Coallocation in Multicluster Systems

Anca I.D. Bucur and Dick H.J. Epema

Abstract—Building multicluster systems out of multiple, geographically distributed clusters interconnected by high-speed wide-area networks can provide access to a larger computational power and to a wider range of resources. Jobs running on multiclusters and, more generally, in grids, may require (processor) coallocation, i.e., the simultaneous allocation of resources (processors) in different clusters or subsystems of a grid. In this paper, we propose four scheduling policies for processor coallocation in multiclusters, and we assess with simulations their performance under a wide variety of parameter settings. In particular, in our simulations we use synthetic workloads and workloads derived from the logs of actual systems and from runtime measurements. We conclude that although coallocation makes scheduling more difficult and the wide-area communication critically impacts the performance, there is a wide range of realistic applications that may benefit from coallocation. However, unrestricted coallocation is not recommended: Limiting the total job size or the number or the sizes of their components improves performance.

Index Terms—Multicluster systems, coallocation, scheduling policies, simulation.

1 INTRODUCTION

Over the last decade, clusters and distributed-memory multiprocessors consisting of hundreds or thousands of standard CPUs have become very popular. Compared to single-cluster systems, multicluster systems made up of multiple, geographically distributed clusters can provide a larger computational power. Instead of smaller groups of users with exclusive access to their single clusters, larger groups of users can share the multicluster. This potentially leads to lower job turn-around times and higher system utilizations, and makes larger job sizes (in terms of the number of processors) possible by allowing jobs to use processors in multiple clusters simultaneously, that is, to employ (processor) coallocation. Of course, coallocation entails wide-area communication, which may increase application runtimes and so may reduce the potential benefits of coallocation. In this paper, we design and study the performance of scheduling policies for processor coallocation in multicluster systems.

An example of a multicluster system is the Distributed ASCI Supercomputer (DAS) [3], [7], which is an important motivation for our work. In particular, on the DAS the performance of scheduling policies for processor coallocation runtimes and so may reduce the potential benefits of coallocation. In this paper, we propose four scheduling policies for processor coallocation in multiclusters, and we assess with simulations their performance under a wide variety of parameter settings. In particular, in our simulations we use synthetic workloads and workloads derived from the logs of actual systems and from runtime measurements. We conclude that although coallocation makes scheduling more difficult and the wide-area communication critically impacts the performance, there is a wide range of realistic applications that may benefit from coallocation. However, unrestricted coallocation is not recommended: Limiting the total job size or the number or the sizes of their components improves performance.

A simple and yet, due to its practicality, often used scheduling strategy in single parallel systems is to allow only rigid jobs scheduled by pure space sharing, i.e., jobs requiring fixed numbers of processors that are executed on these processors exclusively, until completion. This is also the strategy considered here. In our case, jobs consist of one or more components, each of which has to be scheduled on a different cluster, and each of which has a fixed size. Single-component jobs go to a single cluster, and multicomponent jobs are scheduled across a number of sites equal to the number of their components. We design and assess four scheduling policies (two of which with multiple variations) for three different queuing structures: one with only a single global queue, and two with both. In the latter case, single-component jobs go to the local queues and multicomponent jobs to the global queue, and either the local queues or the global queue have priority. These policies and queuing structures cover a wide range of situations that may occur in practice. A single global scheduler may be used when a single organization brings its clusters together into a single system to have a complete view of the whole system and to relieve the users from the burden of choosing the appropriate clusters for their applications. When independent departments or organizations join their clusters, they may want to retain their independence with their own local schedulers, but in addition, they may also want to give special treatment (preferential or the opposite) to multicomponent jobs. In our analysis, the performance metric is the mean job response time as a function of the utilization.
This paper is structured as follows: In Section 2, we define our model of processor co-allocation in multiclusters. In Section 3, we address the general question as to what performance can be expected from policies for co-allocation by investigating all variations of our four policies with synthetic workloads. In the remainder of the paper, we restrict ourselves to what turns out to be the best variation of the policies with multiple variations, and to only one policy for each of the three queuing structures; in the case of both local and global queues, we omit the policy that gives the global queue priority as this would not be a realistic option in an actual system. In Section 4, we perform simulations based on traces of actual systems, and impose restrictions to co-allocation by limiting the job-component sizes, and the total job size. Section 5 contains the results of simulations based on the multicluster runtime measurements of two applications on the DAS; here, we also introduce the element of restricting co-allocation by setting a limit to the number of job components. Finally, in Section 6, we present our conclusions and ideas for future work. Our two main conclusions are that processor co-allocation may be beneficial, at least when the overhead due to the wide-area communication is not too high, and that the restrictions to job-component sizes and to the number of job components markedly improve the performance of co-allocation.

2 A MODEL FOR COALLOCATION IN MULTICLUSTERS

In this section, we present our model of processor co-allocation in multicluster systems. In Section 2.1, we introduce the structure of the system and of the job requests. Section 2.2 describes how we model the slowdown experienced by jobs due to the wide-area communication. Section 2.3 discusses our scheduling policies, and in Section 2.4 we present the workloads used in our simulations. In Section 2.5, we present related work.

2.1 The System and Job Requests

Our model of the system is inspired by such systems as our Distributed ASCI Supercomputer (DAS), which is a wide-area computer system located at a number of Dutch universities. The DAS was designed and deployed by the Advanced School for Computing and Imaging (ASCI) in the Netherlands, and is used for research in parallel and distributed computing [3]. The DAS2, the second-generation DAS system which was installed at the end of 2001 [7], consists of five clusters of identical dual-processor nodes, one with 72, the other four with 32 nodes each, for a total of 400 processors. For local communications inside the clusters, Myrinet LANs (1,200 Mbit/s) are used, while at the time of our measurements of application performance (see Section 5.2), the clusters were interconnected by a 100 Mbit/s wide-area network. On single DAS clusters the PBS scheduler [17] is employed, while jobs spanning multiple clusters can be submitted with Globus [12].

In our model, a multicluster system consists of $C$ clusters of possibly different numbers of processors which all have the same service rate. By a job, we mean a parallel application requiring some number(s) of processors, possibly in multiple clusters simultaneously (co-allocation). We assume that jobs only request processors and we do not include in the model any other types of resources. Jobs are rigid, so the numbers of processors requested by and allocated to a job are fixed and cannot be changed during its execution. A task is the part of a job that runs on a single processor. Tasks can communicate by exchanging messages over the network. There is no preemption, and all nodes allocated to a job are only released simultaneously when all of its tasks have finished. When $C$ is equal to 1, the system is a single cluster. In many cases we compare the performance of a multicluster system with that of a single cluster system with the same total number of processors. We refer to a system with $C$ clusters of identical size $n$ with the notation $C \times n$.

Jobs that require co-allocation have to specify the number and the sizes of their components, i.e., of the sets of tasks that have to be run in separate clusters. A job is described by $C$ integers, at least one of which is nonzero. We consider two cases for the structure of jobs, one for multiclusters and one for single clusters:

1. For an unordered request, with the $C$ integers a job specifies the numbers of processors it needs in the separate clusters, allowing the scheduler to choose the clusters for the components.
2. For a total request, there is a single cluster and a job specifies only the total number of processors it needs, which is obtained as the sum of the $C$ integers.

Jobs with unordered requests are placed on the system with Worst Fit (WF) without repetitions. That is, we order the job components according to decreasing size and the clusters according to decreasing number of idle processors, and go one by one through both lists trying to fit job components on clusters. A potential advantage of WF is that it leaves in each cluster as much room as possible for subsequent jobs. All our simulations in this paper are either for a $4 \times 32$ multicluster or for a 128-processor single cluster.

2.2 Wide-Area Communication within Jobs

In general, co-allocation introduces communication over the relatively slow wide-area links, which may increase the runtimes of jobs. There are two ways in which we account for wide-area communication in our model.

2.2.1 Extending Single-Cluster Service Times

Here, in order to obtain the service time of a multi-component job, we multiply the single-cluster service time of a job of the same total size by a so-called extension factor, independent of the number of components. In [19], the performance of four parallel applications in wide-area systems is assessed by comparing the speedups of the original applications on a 64-processor single-cluster system with the speedups of versions of the applications optimized for wide-area execution on a $4 \times 16$ multicluster system. For three of the four applications, the extension factor for multicluster operation does not exceed 1.12; the fourth application (All-pair Shortest Paths) has very poor multicluster performance. In [8], it is concluded that it pays off to use co-allocation when the extension factor does not exceed 1.25. Based on these results and on our measurements (see Section 5.2), we will use an extension factor of 1.25 in the simulations in Section 4.
2.2.2 Using Total Runtime Measurements

A second way to include wide-area communication in the model is to measure the execution times of applications on the DAS and to use the results of these measurements in the simulations. We use this method in Section 5.

We define the gross utilization as the utilization computed from the actual service times experienced by jobs, which for multicomponent jobs includes the time spent in the slow wide-area communication. The net utilization is defined as the utilization computed from the single-cluster service times of jobs of the same total size, which gives a measure of the throughput of the system. When there is no coallocation, there is no wide-area communication, and the gross and net utilizations coincide.

2.3 The Scheduling Policies

Multicluster systems may come into existence either by design, or because organizations join their separate clusters. In the first case, only a single global scheduler may be installed. In the second case, the separate clusters already have their own schedulers which the different organizations are probably not willing to give up. In order then to make use of the complete multicluster system, either the local schedulers may be fitted with a facility to submit multicomponent jobs, or an additional global scheduler for this purpose may be introduced. Therefore, in order to cover all these possibilities, in our model the system may have a single central scheduler with one global queue, a local scheduler with its own queue for each cluster, or both global and local schedulers. In the latter case, all the multicomponent jobs are stored in the global queue and all the single-component jobs in the local queues. We define four policies for processor coallocation in multiclusters, one with only a global queue, one with only local queues, and two with both. For two of the policies, we define several variations, depending on the order in which jobs are scheduled from the queues. The rules of the scheduling policies are invoked either when a job departs or when a job arrives at an empty queue. We will describe below only the behavior of the system at job departures since job arrivals at empty queues are a degenerate case of it.

In our description of the policies, we say that a queue is enabled when its corresponding scheduler is allowed to start jobs from that queue. In the initial state when all queues are empty, all queues are disabled. When a queue is enabled, the job at its head is scheduled if it fits. If the job does not fit, the queue (and its corresponding scheduler) becomes disabled and may only get enabled again at a subsequent job departure. Queues that schedule their last job and become empty are also disabled.

When a job departs from the system all or only some of the nonempty queues are enabled, depending on the policy rules; empty queues are not enabled at departures. The enabled queues are then repeatedly visited in some order, starting in each round at most one job from each (enabled) queue, until no job can be scheduled anymore according to the policy rules. At this point all queues, both the empty and the nonempty ones, are disabled. When a job arrives at an empty queue, the queue becomes enabled and the job can be scheduled right away if it fits, unless there is a policy rule saying that the queue should stay disabled. The scheduling discipline in each queue is FCFS. We now define our policies as follows:

1. [GS]. In the Global Scheduler policy, the system has one global scheduler with a global queue for both single and multicomponent jobs.

2. [LS]. In the Local Schedulers policy, each cluster has its own local scheduler with a local queue for both single and multicomponent jobs. The former are scheduled only on the local cluster, while the latter are coallocated across the entire system. At each departure, all nonempty queues are enabled. It is a priori not obvious in which order the queues should be enabled with LS. Therefore, we define four variations of LS, the first two of which are nonadaptive, while the last two are adaptive. The aim of all these variations is to give fair treatment to the queues and to achieve a low response time.

   - [LS-OR]. The queues are enabled in a fixed order, starting with the same queue.
   - [LS-RD]. The queues are enabled in a fixed order, starting with a queue randomly and uniformly chosen.
   - [LS-RO]. The queues are enabled in the order in which the processors in the corresponding clusters are released by the departing job, which is assumed to be in the decreasing order of the job-component sizes (the same as the order in which the processors are allocated). If the departing job has fewer components than the number of clusters, the queues local to the clusters not holding any component are enabled last.
   - [LS-DO]. The queues are enabled in the same order in which they were disabled last, so local schedulers that have not scheduled jobs for the longest time get the first chance to do so.

The two remaining policies define both a global queue for the multicomponent jobs and local queues for the single-component jobs, which have to be scheduled on the corresponding clusters. For these policies, the order in which the local queues are enabled does not matter since the jobs in them are only started on the local clusters.

3. [GP]. In the Global Priority policy, the local queues are enabled only when the global queue is empty, so the global queue has priority.

4. [LP]. In the Local Priority policy, the local queues have priority: The global queue is only enabled when at least one local queue is empty. In a real multicluster system, some form of LP would probably be favored by the separate cluster owners, but requiring all local queues to be empty before enabling the global queue would be too extreme an option. If no local queue is empty only the local queues are enabled and repeatedly visited; the global queue is enabled and added to the list of queues which are visited as soon as at least one of the local queues becomes empty. If one or more of the local queues are empty both the global queue...
and the (nonempty) local queues are enabled. Depending on the order in which this happens, we define three variations of LP. The aim of defining these variations is to be able to make a trade-off between the different levels of priority for local and global jobs.

- [LP-LF]. The local queues are first enabled and then the global queue.
- [LP-GF]. The global queue is first enabled and then the local queues.
- [LP-RD]. Either the global queue is enabled first or the local queues, with equal probability.

For comparison, we consider the single-cluster case where there are only single-component jobs and where we also use FCFS as the scheduling policy ([SC]). In fact, SC is identical to GS in a single-cluster system.

One of the questions that has to be answered when multiple schedulers are active in a single system is to what extent they are autonomous or to what extent they are coupled and synchronize their actions. Clearly, our policies LS, GP, and LP are at the more tightly coupled end of the spectrum, and an implementation of these policies would require message exchanges about scheduling actions and queue lengths. However, in a multicluster system as the DAS with a rather low frequency of submissions of (large) jobs, this is feasible. However, it is certainly imaginable that in many multicluster systems, the cluster owners would like their local schedulers to be more autonomous.

### 2.4 The Workloads

We define the workloads in our model by specifying the job-component sizes or the (total) job sizes (in terms of the number of processors they require), the arrival process of jobs, and the service times of the jobs. The arrival process is always assumed to be Poisson. For both job sizes and service times, we use synthetic distributions, distributions derived from logs of actual systems amongst which the DAS, or the values as measured with our applications on the DAS. The latter are presented in Section 5.2.

#### 2.4.1 Job Sizes

The synthetic distribution used for the (nonzero) sizes of the job components is the distribution $D(q)$ defined as follows:

$$D(q) \text{ takes values on some interval } [n_1, n_2] \text{ with } 0 < n_1 \leq n_2,$$

and the probability of having job-component size $i$ is $p_i = q^i/Q$ if $i$ is not a power of $2$, and $p_i = 3q^i/Q$ if $i$ is a power of $2$, with $Q$ such that the $p_i$ sum to 1. This distribution favors small sizes and sizes that are powers of two, which has been found to occur in practice [5]. For total requests, we use the sum of multiple copies of $D(q)$.

For the job sizes of actual systems, we use the log of a three-month period of the largest cluster (128 processors) of the first-generation DAS1 system. The reasons for using this DAS1 log were that at the time when we started our simulations, no logs of the DAS2 were available yet, and coallocation had not been used enough to derive meaningful statistics on the sizes of the jobs’ components. As the log is for a single-cluster system, it only contains the total sizes of jobs. We will describe below how we derive the sizes of multicomponent jobs from these total sizes. The results of our simulations with trace-based workloads are presented in Section 4. All these simulations are for systems with a total of 128 processors.

In one of our simulations, we investigate whether limiting the total job size improves the performance (see Section 4.3). The job-size limit we use is 64, and we refer by DAS-s-64 to the log obtained from the full log DAS-s-128 by cutting it at size 64, which only excludes 2 percent of the jobs. Statistics of the two logs are presented in Table 1 (cv stands for coefficient of variation). The density of DAS-s-128, which is presented in Fig. 1a, also shows a preference for small numbers and for powers of two [5].

When we use the logs for total requests in a single cluster, the total job-size distribution is directly used. In the multicluster case, we will only consider systems with clusters of equal size, and we generate the (dependent) job-component sizes in the following way. We set a size

---

**Fig. 1.** The density of (a) the job-request sizes and of (b) the service times for the largest DAS1 cluster (128 processors).
limit for the job components, and compute the number of components depending on this limit, with jobs having as few components as possible. As long as this number of components is smaller than the number of clusters, no component is larger than the size limit; only when the number of components is equal to the number of clusters may the size limit be exceeded. Once we have decided on the number of components of a job, we split the job into components of sizes as equal as possible. In Section 4.2, we will assess the effect of limiting the component sizes on the performance. There, we will consider three size limits for the components: 16, 24, and 32; in Table 2 we show how the numbers and the sizes of the job components vary with this size limit on the DAS.

2.4.2 Job Service Times
The synthetic job service-time distribution we use is the exponential distribution with mean 1. We assume that any intercomponent communication is also included in this service time.

For data from actual systems, we again use the DAS1 log. In this log, 28,426 jobs were recorded with both their starting and ending times, and we could compute their service times. We refer by DAS-t to the subset of the entire log with only the jobs for which the service times could be computed, and by DAS-t-900 to the yet smaller subset with only the jobs with service times at most equal to 900 seconds, which is the service-time limit during working hours. We use DAS-t-900 in our simulations. Fig. 1b shows the density of the service times on the DAS1 as it was obtained from the log and Table 3 gives statistics of the service times in the logs.

2.5 Related Work
In [8], processor coallocation (called multisite execution there) is studied with only trace-based simulations of different cluster configurations. Coallocation is compared to having all clusters operate in isolation and to only sharing the load among the clusters, assuming that all jobs fit in a single cluster. In the coallocation and load-sharing cases, there is only a single global scheduler that decides where to run jobs. When coallocation is employed, jobs only specify their total size, and are split up across the clusters by the scheduler. The main result of [8] is, as mentioned already in Section 2.2, that coallocation is beneficial when the overhead incurred by the wide-area execution is below 25 percent. In [9], two improvements to the basic coallocation policy of [8] are proposed, viz., setting a limit to the number of components into which a job can be split based on its total size, and adaptively choosing between coallocation and single-site execution based on the estimated completion times (locally, backfilling is used). In this paper, we study a much wider range of coallocation policies, as the models of [8], [9] only fit the model of our GS policy and of our LS policy with only single-component jobs. Moreover, we also perform simulations with synthetic and measured workloads.

In [13], bandwidth-aware coallocation in multiclusters is simulated with synthetic workloads with an emphasis on the contention on the network links due to coallocated jobs. When a local job arrives, it either stays local, is migrated, or is coallocated with component sizes determined by the scheduler according to different policies based on processor availability and network link usage. The main result is that assigning a very large part of jobs to a single cluster gives the best performance. In contrast to our work, there is only a single global scheduler. In [20], an algorithm for finding the minimum execution time of a set of parallel tasks that require coallocation in grids is simulated. The algorithm is based on the list-scheduling approach, finds for each task its critical resource which determines its start time, and uses task priorities calculated from the precedence among tasks due to data transfers. In [6], the Globus component DUROC for coallocation in real systems is presented. This component is one of the building blocks of our KOALA scheduler [15], [16] that we have designed and implemented in the DAS as a follow-up to our simulation study of coallocation, and that adds resource brokering and fault tolerance to DUROC.

3 Simulations with Synthetic Workloads
In this section, we assess the performance of the four scheduling policies defined in Section 2.3 with synthetic workloads. Section 3.1 presents the compositions of the job streams used in terms of the fractions of jobs with certain numbers of components. As the Local Schedulers (LS) and Local Priority (LP) policies have several variations, we first evaluate these in Sections 3.2 and 3.3. Then we compare all four policies, choosing the best variations of LS and LP. Because when the job-stream composition only contains single-component or multicomponent jobs some of our policies coincide, we treat the cases with both these types of jobs and with only one of these types separately in Sections 3.4 and 3.5. In Section 3.6, we briefly present some overall conclusions to this section.

In this section, it is our aim to assess the ordering of policies according to the response time at a fixed utilization at which at least one policy is close to or just over its saturation point. As these utilizations vary among the different cases we consider, there is no point in comparing...
the response times in the distinct charts in the figures in this section. Indeed, we choose the appropriate scale on the vertical (response-time) axis in each chart independently. For policies with both local and global queues we expect that the performance differs between these queues. Therefore, for GP and LP, we depict besides the total average response time also the average response times for the local queues and the global queue.

### 3.1 Job-Stream Compositions

Jobs can have between one and four components. We express the job-stream composition as a tuple of four values representing, in this order, the percentages of 1, 2, 3, and 4-component jobs submitted to the system. We consider nine job-stream compositions, which cover a wide range of possibilities of combinations of single and multicomponent jobs (see Table 4). For all of these, we assume first that the local queues are balanced, in the sense that they receive the same percentages of jobs submitted locally. For job-stream composition (80, 0, 0, 20), we add a case with unbalanced local queues: One queue receives 40 percent and the other three 20 percent of the jobs submitted locally. The 10 cases that result are presented in Table 4.

### 3.2 The Variations of the Local Schedulers Policy

In this section, we compare the four variations of LS defined in Section 2.3 for nine of the cases in Table 4: The case with 100 percent single-component jobs is not relevant here because in that case all variations of LS amount to the same.

---

**Fig. 2.** Response times for the four variations of LS, several job-stream compositions, and balanced local queues.

**Table 4**

<table>
<thead>
<tr>
<th>1-comp.</th>
<th>2-comp.</th>
<th>3-comp.</th>
<th>4-comp.</th>
<th>bal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>yes</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>50</td>
<td>50</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>yes</td>
</tr>
<tr>
<td>90</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>yes</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>no</td>
</tr>
</tbody>
</table>
queues schedule their jobs sooner, finally being enabled first at each departure until its job fits. Therefore, this variation keeps in balance the numbers of jobs run from each queue and, as the queues receive equal percentages of the job stream, it achieves a good queue-load balance.

The worst performance in Figs. 2a, 2b, 2c, 2d, 2e, 2f, and 2g is shown by LS-OR, which always enables the queues in the same order. It favors the queues visited earlier (especially the first queue), while allowing the queues visited last to grow. When no queue is empty, there are up to four jobs from which to choose one that fits. Emptying the queues visited first, LS-OR reduces the set of jobs among which to find one that fits.

LS-RD, which randomly chooses the queue to enable first, displays very good performance in Figs. 2a, 2b, 2c, 2d, 2e, 2f, and 2g. It maintains in general similar queue lengths, but it can delay large jobs that are hard to fit.

In Figs. 2a, 2b, 2c, 2d, 2e, and 2f, the performance of LS-RO is similar to that of LS-RD. However, for a high percentage of local jobs, as Figs. 2g and 2h show, LS-RO performs poorly. The explanation resides in the way the local and the global jobs interact there: At high loads, a queue that becomes disabled with a multicomponent job at its head, while all the other queues have single-component jobs that fit at next departures, will be visited last at each such next departure.

This behavior may indefinitely delay the multicomponent job and allow its corresponding queue to grow.

Comparing the variations of LS for job-stream compositions with at least 80 percent local jobs, we find that DO balances the queue lengths and adapts to the workload, RD balances the load of the queues but does not adapt to the jobs in the system, OR keeps the queue lengths unbalanced due to the visiting order, and RO causes the worst unbalance by avoiding the queues with jobs that do not fit. LS-OR improves for a very low percentage of global jobs (see Fig. 2h) because when there are just single-component jobs, the visiting order does not matter. Since it has the best performance for balanced queues, we select LS-DO for further performance evaluation in Sections 3.4 and 3.5.

Fig. 3 shows the average response time for the variations of LS, for 80 percent single-component jobs and unbalanced queues—one queue receives 40 percent of the jobs arriving to the system. LS-OR displays the best performance because it gives priority exactly to the queue with the highest load, visiting it first after each job departure. The other variations have higher response times because they do not take into account that one queue receives twice as many jobs and let that queue grow.

3.3 The Variations of the Local Priority Policy

In this section, we compare the three variations of LP defined in Section 2.3. With LP the local queues only get single-component jobs, which are restricted to the local clusters, so the relative order in which these queues are enabled does not matter. Only seven of the cases in Table 4 are relevant here: For 100 percent single-component jobs, there are only local queues, while for 100 percent multicomponent jobs, there is only the global queue. In Fig. 4, the local queues are balanced, while in Fig. 5, the unbalanced case is assessed.

In all the charts in Fig. 4, LP-GF displays the best total performance and the best performance for the global queue. Enabling the global queue first rather than last deteriorates very little the performance of the local queues; in most
cases, there is a very small increase in response time for the local queues compared to LP-LF. On the other hand, enabling first the local queues with LP-LF causes a large increase in response time for the global queue compared to LP-GF.

For LP-GF, the performance of the local queues becomes worse than for LP-LF when the global jobs have fewer components, because they fit better on the system and leave less room for the local jobs (see job-stream composition (50, 50, 0, 0)). For a high percentage of global jobs and with many components, enabling the global queue first does not hamper much the local jobs and has a good effect on the global jobs.

In all the cases in Fig. 4, LP-RD has a total performance and a performance for the global queue worse than those of LP-GF and better than those of LP-LF. The reason is that LP-RD randomly chooses at each departure whether to enable first the global queue or the local queues. We conclude that LP-GF is the best choice for the job-stream compositions in Fig. 4, and LP-LF the worst. We will further assess the LP-GF variation in Sections 3.4 and 3.5.

When the local queues are unbalanced (see Fig. 5), LP-LF has the best total performance because always enabling first the local queues allows the queue with a higher load to fit its jobs without being perturbed by the multicomponent jobs. However, LP-LF has a bad performance for the global queue. LP-GF provides a high total average, and a very high average response time for the local queues. LP-RD has a slightly worse total performance than LP-LF and a higher response time for the local queues, but a much lower response time for the global queue. If we aim at a good total performance or a low response time for the local jobs, LP-LF is the best option in this case, but if we also want a low response time for the global queue, LP-RD should be chosen.

3.4 Policy Comparison for Job-Stream Compositions with Both Single and Multicomponent Jobs

In this section, we assess the performance of the four policies for the balanced cases when both single and multicomponent jobs are simultaneously present in the system. From here on in this paper, we only consider the LS-DO variation of the LocalSchedulers policy and the LP-GF variation of the Local Priority policy, and simply denote them by LS and LP. The results are depicted in Fig. 6. When for a certain policy saturation has been reached for the utilization considered, the response time bar is marked with SAT. Then the height of the bar is of course irrelevant.

In Fig. 6a, we compare the policies for equal fractions of jobs with one, two, three, and four components. The best performance is displayed by LS, for which at each departure the system tries to schedule up to four jobs (when no queue is empty), one from each of the four local queues; the FCFS policy is transformed this way into a form of backfilling with a window of size 4. A disadvantage of LS compared to GS is that LS can only place single-component jobs on the cluster where they were submitted, while GS can choose from the four clusters one where such jobs fit. However, in the case in Fig. 6a, only 25 percent of jobs have one component, so their negative influence on the performance of LS is small.
GP and LP may schedule up to five jobs at a time, but 75 percent of the jobs in the system are multicomponent and they all go to the global queue. This yields worse performance for GP and LP than that of LS. GP gives priority to the global queue and only allows jobs from the local queues to run when the global queue is empty. As a result, GP displays the best average response time for the global queue, but the average response time for the local queues and the overall performance are much worse than for the other policies. LP also runs mostly jobs from the global queue, but it does not delay the jobs from the local queues when the job at the head of the global queue does not fit. When none of the local queues is empty, the LP policy strongly favors those queues. When at least one local queue is empty the global scheduler is enabled first at job departures (LP-GF), which has a positive effect on the overall performance, and only slightly deteriorates the performance of the single-component jobs.

Figs. 6b, 6c, and 6d show that for GP the performance improves with the decrease of the percentages of jobs with three and four components. Jobs with many components are more difficult to combine with other jobs, and therefore not allowing jobs from the local queues to run when the job at the head of the global queue does not fit deteriorates the performance. The lowest response times in Figs. 6b, 6c, and 6d are displayed by LS followed by LP. The low response time of LP indicates that, when none of the local queues is empty and at most 50 percent of jobs are local, delaying the global jobs to wait for the local jobs to fit has little impact on performance.

GS performs well for a high percentage of single-component jobs because it does not restrict the local jobs to the corresponding clusters (see Fig. 6f). Further increasing the percentage of single-component jobs would improve the performance of GS and deteriorate the performance of all the other policies (when there are 100 percent single-component jobs GP and LP become LS and the clusters operate in isolation). Increasing the percentage of multicomponent jobs would improve the performance of LS, but worsen it for the rest (when there are only multicomponent jobs, GP and LP become GS).

### 3.5 Policy Comparison for Job-Stream Compositions with Only Single or Multicomponent Jobs

In this section, we assess our four policies in the three (balanced) cases when there are either only single-component or only multicomponent jobs in the system. In the former situation, GP and LP are reduced to LS, while in the latter, both these policies become GS. Therefore, in Figs. 7a, 7b, and 7c we only compare the GS and LS policies.

When there are only single-component jobs in the system (Fig. 7a), GS has better performance because it performs load balancing across the clusters (using WF), while with LS the clusters operate in isolation. For a system with only multicomponent jobs (Figs. 7b and 7c), LS is better because at any time there are up to four jobs from which to choose one that fits in the system rather than only one with GS.

### 3.6 Conclusions

Based on the detailed performance assessment presented in the previous sections, we can conclude that in general policies with multiple queues yield better performance, and that when single and multicomponent jobs are simultaneously present in the system, policies that do not strictly favor one of the two job types provide better results. Also, the performance of a policy is strongly influenced by the job-stream composition, so it is not possible to choose a policy that always outperforms the others for every workload. Therefore, a policy chosen in practice should either be highly workload-aware (i.e., it should be adaptive or tuned to the workload), or it should achieve a good enough compromise for the various job-stream compositions.

### 4 Trace-Based Simulations

In this section, we assess the performance of the Global Scheduler, the Local Schedulers (that is, LS-DO), and the Local Priority (LP-GF) policies with simulations based on traces of the DAS (see Section 2.4). We omit the GP policy from here onward as we think that in actual multiclusters, cluster owners will prefer LP over GP when both local queues and a global queue are maintained. For the job sizes and the service times, we use the DAS-s-128 distribution (in Section 4.3 also the DAS-s-64 distribution) and the DAS-t-900 distribution, respectively. For multicluster execution, we employ the first method of accounting for the wide-area communication by extending the single-cluster service times with an extension factor of 1.25 (see Section 2.2). For the policies with local queues (LS and LP), we consider both the case with balanced local queues and the unbalanced case when one local queue receives 40 percent and the other three 20 percent of the jobs submitted locally. We compare the results to those for a single cluster (policy SC).

Section 4.1 makes a general comparison of the policies, while Sections 4.2 and 4.3 discuss the impact on the
performance of imposing a limit on the size of the job components and on the total job size, respectively. In all these sections, we depict the overall response time as a function of the gross utilization because that is a fair basis for comparing the policies. In Section 4.4, we discuss the relation between the gross and net utilizations, which has to do with how efficient the global applications use the gross utilization offered. For SC, there is no wide-area communication and, so, the net utilization is equal to the gross utilization. In all the figures in this section and the next, we follow the convention that the order of the legends corresponds to the right-to-left order of the response-time curves, i.e., from high to low maximal utilization. The graphs also include confidence intervals, which are always at the 95 percent level.

4.1 Comparing the Policies

We first present a general comparison of the performance of the policies. The job-component size is either unlimited (32), or limited to 24 or 16. With setting a maximum job-component size, we vary the numbers and sizes of job components (see Table 2), which influences the performance by modifying the way jobs fit together and by changing the percentage of jobs with extended service times. For SC, the size limit is irrelevant, so the SC curves can be used as a reference in Fig. 8.

For a size limit of 16, when there are many multi-component jobs (48.7 percent), LS performs better than the other policies. It even provides a similar or somewhat higher maximal utilization than SC, although a part of that utilization is spent on wide-area communication and SC has a significantly better throughput (see Section 4.4). In itself, the fact that a multiclient policy can have a similar performance to SC in terms of gross utilization makes coallocation a good option, showing that the fragmentation caused by having multiple clusters can be overcome. The main advantage of LS is that it distributes the multicomponent jobs among the local queues, leading to a form of backfilling with a window of size equal to the number of clusters. For size limits of 24 and 32, when there are only 26.2 percent and 22.0 percent multicomponent jobs, respectively, the performance of LS is worse. In all the graphs, LP displays the worst results because all the multicomponent jobs are placed in a single global queue, and all the single-component jobs are restricted to the local clusters. Although GS has only a global queue, it is consistently better than LP, and for a size limit of 32 (when there are many local jobs), it even approaches LS, because it may choose the clusters for the single-component jobs.

Comparing the balanced and unbalanced cases for LS and LP (see Fig. 8), we find that an unbalanced load for the local queues has a negative impact on performance. This effect is more pronounced for LS, especially for larger job-component-size limits, when there is a higher percentage of local jobs. A cause is that the least loaded queues get empty sooner, which decreases the backfilling window for the global jobs. The deterioration is stronger when there is a high percentage of local jobs, which indicates a second cause: The local jobs are restricted to their corresponding clusters, and a higher percentage of local jobs means a higher load for the local cluster. For a size limit of 32 and unbalanced local queues, LS performs worse than GS and similarly to LP.

For LP, the performance decrease due to the unbalance of the local queues is small for all size limits. When there are few local jobs, the loads of all local queues are low, even in the unbalanced case (for a size limit of 16 the most loaded local queue receives 20.5 percent of all jobs). For a high percentage of local jobs the local queues access their clusters with priority, which reduces the impact of the unbalance. Even when a local queue gets empty and multicomponent jobs from the global queue are started, the use of WF insures that the least loaded clusters are chosen.

We find that, in all the cases, the performance is very poor: For all job-component-size limits, for balanced and
unbalanced local queues, and for all policies. Even for total requests, the maximal utilization is below 0.65. The main cause seems to be the total job-size distribution, and not the job-component-size limit, the global communication, the policies, or the extra fragmentation introduced by scheduling multicomponent jobs in a multicluster system. We will verify this assumption in Section 4.3.

4.2 Setting the Job-Component-Size Limit

In this section, we focus on the impact of the maximum allowed job-component size on the performance of GS, LS, and LP. The results are in Fig. 9, which is a rearrangement of Fig. 8 (balanced local queues for LS and LP).

Smaller job components improve the system's performance, but having many components relative to the number of clusters deteriorates the performance. A smaller job-component-size limit also means more multicomponent jobs, and so more jobs with extended service times, which also worsens the performance: For a size limit of 16, there are 26.7 percent more multicomponent jobs than for a size limit of 32. Adding up all these factors, for GS, a size limit of 32 provides slightly better results than 16.

For LS, due to the backfilling effect and to the fact that multicomponent jobs can be spread across any of the clusters while the single-component jobs are restricted to the local clusters, a smaller size limit is an important advantage. For this policy, a size limit of 16 is a much better choice than a size limit of 32.

For LP, all multicomponent jobs go to the global queue and only the single-component jobs are spread among the local queues. Fewer local jobs means that fewer jobs are restricted to the local clusters and, so, the local queues empty faster and the global queue gets more often the chance to start jobs. On the other hand, the global queue has a low priority when no local queue is empty, and the more global jobs, the higher the average response time for those jobs. As Fig. 9 shows, LP performs better for a size limit of 32 than for 16, but the difference is small.

For all the policies, the performance is worst for a job-component-size limit of 24. Since all the factors discussed above would place a system with limit 24 in between the other two cases, the reason for this anomaly should be in the way jobs fit together in the system. Checking the DAS log, we found that this anomaly is caused by the fact that 19 percent (!) of all jobs has size 64. For a size limit of 16, the corresponding job request is (16, 16, 16, 16), for a size limit of 32, it is (32, 32, 0, 0), and for a limit of 24, it is (22, 21, 21, 0). Considering an empty system which receives a job of size 64, in the first two cases after the job is placed there are many jobs, with different numbers of components and sizes up to 64, that would still fit in the system. However, in the third case only single-component jobs with maximum sizes of 10 and 11 can fit in three of the clusters, and single-component jobs with a maximum size of 24 (due to the size limit) in the fourth, empty cluster. A second job of size 64 would also fit in the first two cases, but not in the third.

4.3 Limiting the Total Job Size

In the job-size distribution derived from the DAS, only 2 percent of the jobs require more than 64 processors (out of which 1.2 percent need 128 processors to run, which is the entire system). Our hypothesis is that eliminating this small percentage of very large jobs from the distribution will improve performance.

Fig. 10 compares the performance of the three policies considered in this section (and SC) for the DAS-s-128 and the DAS-s-64 job-size distributions, for a job-component-size-limit of 16, and for balanced local queues. We choose this set of parameters because they provided the most interesting results (LS better than SC) in Section 4.1.

With DAS-s-64, the improvements in performance are large for LS, and even more so for SC. When a job requiring 128 (or a similarly large number of) processors is at the head of the queue, SC waits for the entire system to become empty, which yields a very low utilization. LS, on the other hand, can run jobs from the other queues and postpone the large job until either the other queues are empty, or they also have large

Fig. 9. The performance of GS, LS, and LP depending on the size limit of the job components. In all cases, the DAS-s-128 job-size distribution is used.

Fig. 10. The response times for maximal total job sizes 64 and 128 (job-component-size limit 16, balanced local queues). Both DAS-s-128 and DAS-s-64 job-size distributions are used.
jobs at their heads that do not fit. For DAS-s-64, the largest jobs in the system require only half of the processors, which improves the utilization of SC and diminishes the advantage of LS.

LP and GS also perform better for DAS-s-64. Since the local queues have priority, for DAS-s-128 LP postpones much the very large jobs in the global queue. For DAS-s-64 jobs are smaller, and LP outperforms GS because it can benefit from having more queues.

The results in this section indicate that a very small percentage of very large jobs can significantly worsen the performance, and rather than designing a complicated policy to deal with such jobs, simply imposing a maximum size for the jobs submitted to the system brings more important improvements. Of course, this means that jobs may have to be configured to use fewer processors, possibly leading to longer service times; this extension of the service times is highly dependent on the specific application.

4.4 Gross versus Net Utilization
In Sections 4.1, 4.2, and 4.3 we have studied the average response time as a function of the gross utilization. In this section we discuss the difference between gross and net utilization, which were defined in Section 2.2, and we quantify this difference for the cases considered in the previous sections.

The difference between the two utilizations is the capacity lost internally in multicomponent jobs due to slow wide-area links, which might be reduced by restructuring applications or by having them use (collective) communication operations optimized for wide-area systems [2], [4], [14], [18]. The performance of a multicluster policy may look good when considering the response time as a function of the gross utilization, but, when there is much internal capacity loss, the performance as a function of the net utilization may be poor. In the extreme case, if the global communication was as fast as the local communication, LS would sometimes provide even better performance than SC (see Fig. 8). Of course, for the same workload (defined by the arrival rate and, so, by the net utilization) and the same job-component-size limit, the difference between the gross and the net utilization is the same for all scheduling policies, albeit at possibly different response times.

In our model, the job sizes are independent of the job service times. This means that we can compute the ratio between the gross and the net utilization, independent of the scheduling policy, as the quotient of the weighted average total job size with single-component jobs having weight 1 and multicomponent jobs having weight 1.25 (the extension factor), and the (nonweighted) average total job size. For the DAS-s-128 job-size distribution, we find for the job-component-size limits of 16, 24, and 32 ratios of gross and net utilization of 1.218, 1.173, and 1.159, respectively. These ratios should be applied as scaling factors to the job-component-size limits of 16, 24, and 32 ratios of gross size. For the DAS-s-128 job-size distribution, we find for the extension factor), and the (nonweighted) average total job weight 1 and multicomponent jobs having weight 1.25 (the

5 Simulations Based on Runtime Measurements
In this section, we assess the performance of the Global Scheduler (GS), the Local Schedulers (LS), and the Local Priority (LP) policies with simulations using the runtime measurements on the DAS of two applications. This is our second method of accounting for wide-area communication defined in Section 2.2. Our aim in this section is to assess the effect on the performance of restrictions on co-allocation in terms of the numbers and sizes of job components. In this section, we first discuss these restrictions, then we present the workloads in the simulations, and finally we compare with simulations the performance of the policies and the restrictions.

5.1 Co-allocation Rules
We now formulate the restrictions that we impose on co-allocation in so-called co-allocation rules. We will compare a no co-allocation case, when only single-component jobs are admitted, to several co-allocation cases. We define the following co-allocation rules:

1. [no]. Only single-component jobs are admitted, and there is no co-allocation.
2. [co]. Both single and multicomponent jobs are admitted, without any restriction on the sizes or numbers of job components.
3. [rco]. Both single and multicomponent jobs are admitted with the restriction that the size of job components is limited to half the clusters’ sizes (which are all 32 in our simulations).
4. [fco]. Both single and multicomponent jobs are admitted with full restrictions: The size of job components is limited to half the clusters’ sizes, and the number of job components is limited to 2.

5.2 The Workloads
In our simulations, we use the runtimes of two applications called Ensflow and Poisson, which we present below (for a more extensive description, see [4]). The Ensflow application [10] simulates the streams and eddies in the ocean near the southern tip of Africa with the data-assimilation technique, in which information from observations of the ocean is combined with the simulation itself. The application calculates the evolution of a large number (typically 50-500) of different states. Periodically, an analysis and an update of the states is done; only then communication is performed. In our case, there are 60 states that evolve over a period of 20 days and an analysis and an update are only done twice. We take the numbers of processors to be divisors of 60 and distribute the states evenly over the processors.

The Poisson application implements a parallel iterative algorithm to find a discrete approximation to the solution of the two-dimensional Poisson equation with as domain the unit square. In every iteration, each point in the uniform grid that we define in the domain for the discretization has its value updated as a function of its previous value and the values of its four neighbors. The domain of the problem is split up into rectangles of equal size among the participating processors. When we execute the Poisson application on multiple clusters, the clusters are responsible for adjacent vertical strips of equal width of the domain.

Tables 5a and 5b display the execution times measured on the DAS for the two applications (see also [4]). For the Ensflow application, the performance of multicluster execution for all numbers of clusters considered compared to single-cluster
execution is very good, because this application has a relatively small communication component. For the Poisson application (in which we use a grid size of $4,000 \times 4,000$), the speedup when going from 8 to 32 processors for one and two clusters is 3.45 and 3.46, respectively, and the speedup when going from 16 to 32 processors for four clusters is 1.74. The largest value of the extension factor due to wide-area communication we find here is 1.23.

In the simulations, each of the jobs is supposed to run one of our two applications. We consider three workloads, one in which all jobs in the system run the Ensflow application, one in which all jobs run the Poisson application and one in which each of the two applications is run by half of the jobs. In all our three workloads, the fractions of jobs of all total sizes that occur are equal, and for the same total size, the various ways of splitting jobs into components have equal probability.

### 5.3 Evaluation

In Fig. 11, we show the response time as a function of the (gross) utilization for the three workloads, the three scheduling policies, and the four coallocation rules. Because our two applications have very different service times, we assess the performance in terms of the point where the system saturates (where the response-time curves rise very steeply) rather than in terms of the actual response times.

Overall, the performance is the best for only the Poisson application and the poorest for the mix of the two applications. The reason for this is that for the Poisson application, all the job sizes are powers of two, like the

---

**TABLE 5**

The Execution Times (in Seconds) for (a) the Ensflow Application and (b) the Poisson Application, Depending on the Total Job Size and the Number of Components, Used in the Simulations

| total job size | number of job components
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>3485.0</td>
</tr>
<tr>
<td>15</td>
<td>2836.0</td>
</tr>
<tr>
<td>20</td>
<td>1935.0</td>
</tr>
<tr>
<td>30</td>
<td>1563.0</td>
</tr>
</tbody>
</table>

(a)

| total job size | number of job components
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1230.0</td>
</tr>
<tr>
<td>16</td>
<td>649.0</td>
</tr>
<tr>
<td>32</td>
<td>357.0</td>
</tr>
</tbody>
</table>

(b)

---

![Fig. 11. The performance of GS, LS, and LP (top to bottom) for the Ensflow application, the Poisson application, and a mix of the two in equal proportions (left to right), for the four coallocation rules.](image-url)
clusters’ sizes, which makes them fit very well in the system. For the Ensflow application, and apparently even more so for the mix, the different sizes of jobs are more difficult to fit on the system in an efficient way.

In all the graphs in Fig. 11, the \([co]\) coallocation rule yields the poorest performance. So, although in general, coallocation provides more flexibility in placing jobs on the system, jobs with conflicting requirements can make the performance worse than that in the absence of coallocation. The poor performance is due to the simultaneous presence in the system of large single-component jobs using (almost) entire clusters and of jobs with many components (even equal to the number of clusters). In all the cases considered, restricting the maximum size of job components (coallocation rule \([rco]\)) or in addition also limiting the number of job components \((fco)\) significantly improves the performance.

For GS, coallocation does not enhance the performance, but only maintains it with the full restrictions (the curves for \([no]\) and \([co]\) are almost identical), and deteriorates it in the other cases; the potential advantage of more flexibility brought by coallocation does not compensate for the limitation of scheduling in a FCFS manner from a single queue or for the disadvantage of longer service times due to the wide-area communication. For LS and LP, the performance for both the \([rco]\) and \([co]\) restrictions proves to be much better than for the no coallocation and the unrestricted coallocation cases \((no)\) and \((co)\). When there are only single-component jobs (coallocation rule \([no]\)) LP becomes LS, and for LP, the curve for LS is depicted.

When comparing the three policies in Fig. 11, we conclude that LS and LP provide the best results for the coallocation cases, whereas when there are only single-component jobs, the performance of GS is better. The reasons are that when there are multicomponent jobs, spreading these among the local queues in the case of LS and having a separate queue for them in the case of LP allow those policies to outperform GS. However, with only a job-component-size limitation \((fco)\), LS displays much better results than LP. The reason is that then the single queue in LP for the multicomponent jobs is a big obstacle. This difference is much smaller for \([co]\) because then the number of job components is limited to 2. Overall, we conclude that the best results are obtained with LS and LP, with the \([co]\) restrictions.

6 Conclusions

In this paper, we have designed four policies for processor coallocation in multicluster systems, and we have assessed their performance with simulations with synthetic workloads, and (for three of these policies) with workloads derived from traces of our DAS system and from runtime measurements of two applications on this system. We conclude the following.

First, in multicluster systems we may have to deal with a significant amount of processor time spent waiting for wide-area communication. However, coallocation is a viable option at least as long as the extension factor covering this time does not exceed 1.25.

Second, LS-DO, which remembers which queues have jobs at their heads that are difficult to fit, and LP-GF, which gives only limited priority to local jobs over global jobs, turned out to be the best variations of the LS and LP policies.

Third, in general, we found that the policies with local queues (possibly with a global queue for multicomponent jobs) yield better performance than having only a single global queue. Especially in the simulations with workloads derived from the DAS, the best policy proved to be LS, which effectively provides a form of backfilling also for multicomponent jobs with a window size equal to the number of clusters. In some cases, its performance is even comparable to using FCFS for total requests in a single cluster. With the LP policy, the presence of multicomponent jobs that are coallocated across the system does not impact much the response time of local jobs.

Fourth, a small percentage of very large jobs with total sizes close or equal to the size of the entire system can strongly impact the performance. Although the choice of policy then does matter, when dealing with such jobs, by far the largest improvement in performance can be obtained by simply limiting the total job size. More generally, simply allowing coallocation without any restriction is not a good idea. One should at least limit the job-component size, and preferably also the number of job components.

There are many opportunities for future work on the subject of coallocation of processors (and other types of resources). For instance, our model of coallocation can be extended to include heterogeneous systems with clusters of different sizes and different processor architectures and capacities, and more complex application types than simple parallel applications. Also, in our current model the local schedulers are rather tightly coupled, but models with more autonomous schedulers are certainly worthwhile to be studied. Finally, other criteria for placing jobs than only the numbers of instantaneously available processors can be investigated, such as the lengths of the local queues.

References

Anca I.D. Bucur received the MSc degree in computer science from the Technical University of Bucharest, Romania, in 1997, and the PhD degree from the Delft University of Technology, The Netherlands, in 2004. Since 2003, she has been with Philips Research in Eindhoven, The Netherlands, currently working as a senior scientist in the Healthcare Systems Architecture Group. Her research interests include software architectures and computationally intensive applications in the healthcare domain, high performance computing, distributed systems, and grids.

Dick H.J. Epema received the MSc and PhD degrees in mathematics from Leiden University, Leiden, 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.