Trace-based simulations of processor co-allocation policies in multiclusters

DOI:
10.1109/HPDC.2003.1210017

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)

Trace-Based Simulations of Processor Co-Allocation Policies in Multiclusters

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 employ space sharing for scheduling jobs, such as our Distributed ASCI[1] *Supercomputer (DAS), co-allocation, i.e., the simultaneous allocation of processors to single jobs in multiple clusters, may be required. In this paper we study the performance of several scheduling policies for co-allocating unordered requests in multiclusters with a workload derived from the DAS. We find that beside the policy, limiting the total job size significantly improves the performance, and that for a slowdown of jobs due to global communication bounded by* 1.25*, co-allocation is a viable choice.*

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 [2] enables applications to access resources in different and possibly widely dispersed locations simultaneously—that is, to employ processor *co-allocation*—to accomplish their goals, effectively creating single multicluster systems. Scheduling parallel jobs in single-cluster systems has received very much attention. In this paper we study scheduling in multicluster systems, and we restrict ourselves to rigid jobs (which have predefined, fixed sizes) and to space sharing, in which jobs run to completion on exclusively allocated processors.

We evaluate the performance (in terms of average response times and maximal utilization) of several scheduling policies for co-allocating unordered requests—these specify the numbers of processors to be allocated in different clusters, but not the actual clusters—in multiclusters with a workload derived from a real system. For comparison, we assess the performance of total requests, which only specify the total numbers of processors needed, equal to the numbers required by unordered requests, in a single cluster of the same total size. The best multicluster policy turns out to be LS (for each cluster there is a local queue and all jobs go to those queues; multi-component jobs are spread across the clusters, single-component jobs are scheduled on the local clusters); if we consider the gross utilization (includes the wide-area communication), in some cases LS even comes close to using FCFS for total requests in a single cluster. However, in multiclusters a large amount of utilization is spent on global communication, and when comparing the net utilization (takes into account only the computations and the local communication) the single-cluster system performs better. Although the choice of policy does matter for the performance, we found that the largest improvement can be obtained from simply limiting the total job size.

In previous papers [6, 7, 8], we have assessed the influence on the mean response time and maximal utilization of the job structure and size, the sizes of the clusters in the system, the ratio of the speeds of local and wide-area communications, and of the presence of a single or of multiple queues in the system.

Our five-cluster second-generation Distributed ASCI Supercomputer (DAS) [1, 12] (and its predecessor), which was an important motivation for this work, was designed to assess the feasibility of running parallel applications across wide-area systems [14]. In the most general setting, grid resources are very heterogeneous; in this paper we restrict ourselves to homogeneous multicluster systems such as the DAS. 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.

[1] In 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.

2 The model

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

2.1 The DAS system

The DAS (in fact the DAS2, the second-generation system which was installed at the end of 2001 when the first-generation DAS1 system was discontinued) [1, 12] is a wide-area computer system consisting of five clusters of dual-processor nodes, one with 72, the other four with 32 nodes each. The clusters are interconnected by the Dutch university backbone for wide-area communications, while for local communications inside the clusters Myrinet LANs are used. The system was designed for research on parallel and distributed computing. On single DAS clusters the PBS scheduler [4] is used, while jobs spanning multiple clusters can be submitted with Globus [3].

2.2 The structure of the system

We model a multicluster system consisting of C clusters of processors, of possibly different sizes. We assume that all processors have the same service rate. By a job we understand a parallel application requiring some number of processors, possibly in multiple clusters (*co-allocation*). Jobs are rigid, so the numbers of processors requested by and allocated to a job are fixed. We call a task the part of a job that runs on a single processor. We assume that jobs only request processors and we do not include in the model other types of resources. For interarrival times we use exponential distributions.

2.3 The structure of job requests and the placement policies

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 go to the separate clusters. A job is represented by a tuple of C values, at least one of which is strictly positive.

We consider *unordered requests*, where 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. For comparison we introduce *total requests*, where there are only single-component jobs, scheduled in a single-cluster system with FCFS. 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. To determine whether an unordered request fits, we try to schedule its components in decreasing order of their sizes on distinct clusters. We use Worst Fit (WF) to place the components on clusters.

Table 1. The fractions of jobs with sizes powers of two

total job size	fraction of the jobs
1	0.091
2	0.130
4	0.087
8	0.066
16	0.090
32	0.039
64	0.190
128	0.012

2.4 The workload

Although co-allocation is possible on the DAS, so far it has not been used enough to let us gather statistics on the sizes of the jobs' components. Before we started our simulations we could not obtain a relevant log from the recently installed DAS2. However, from the log of the largest cluster (128 processors) of the DAS1 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 and a coefficient of variation of 1.11; their density is presented in Fig. 1. The results are in line with those presented in the literature for other systems, in that there is an obvious preference for small numbers and powers of two [10] (see also Table 1).

By sampling the job-size distribution as measured on the DAS1 we derive two distributions which we use in our simulations for the total job sizes. DAS-s-128 is based on the entire log, while DAS-s-64 is obtained from the log cut at 64 (the average job size for DAS-s-64 is 21.46 and the coefficient of variation is 1.06); the maximum size of a job is reduced to half excluding only 2% of the jobs from the log—the percentage of jobs that require more than 64 processors. We introduce DAS-s-64 to check whether limiting the total job size improves the performance.

When simulating total requests in a single cluster the total job-size distribution is directly used. In the multicluster case, a size limit is set for the job components; the number of components of a job is computed depending on the size limit so that no component is larger than it, as long as the number of components does not exceed the number of clusters. Once we decide on the number of components of the job, we split it into components of sizes as equal as possible. We consider three size limits for the components: 16, 24 and 32; with the size limit we vary the numbers and the sizes of the job components (see Table 2).

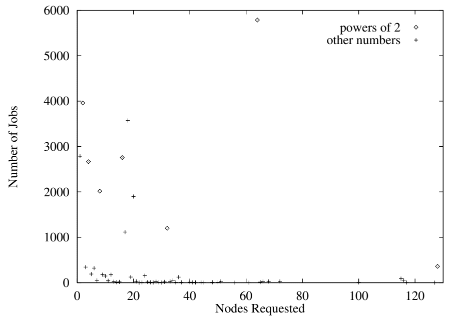

Figure 1. The density of the job-request sizes for the largest DAS1 cluster (128 processors)

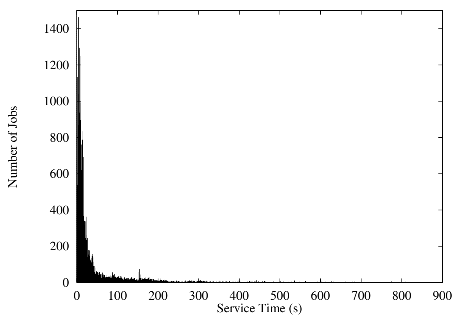

Figure 2. The density of the service times for the largest DAS1 cluster (128 processors)

Table 2. The fractions of jobs with the different numbers of components for the DAS-s-128 distribution and the three job-component-size limits

Job-component-size limit	Number of job components			
	1	2	3	4
16	0.513	0.267	0.090	0.211
24	0.738	0.051	0.194	0.017
32	0.780	0.200	0.003	0.017

In the DAS1 log, $28,426$ jobs were recorded with both their starting and ending times, and we could compute their service times. Influenced by the fact that during working hours jobs are restricted to 15 minutes of service (they are automatically killed after that period), 94.45% of the recorded jobs ran for less than 15 minutes. Figure 2 presents the density of service time values on the DAS1, as it was obtained from the log. The average service time is 356.45 seconds and the coefficient of variation is 5.37. In our simulations, we use for the service-time distribution the distribution derived from the log of the DAS, cut off at 900 seconds (DAS-t-900). The average service time for the jobs in the cut log is 62.66 and the coefficient of variation is 2.05.

We include wide-area communication in the model by extending the service times of all multi-component jobs with a factor, independent on the number of components. In [14], the performance of four parallel applications in wide-area systems is assessed by comparing the speedups of the original applications on a 64-cluster system with the speedups of versions of the applications optimized for wide-area execution on a 4x16 multicluster system. For three of the four applications, these speedups are 60.6 and 53.9, 60.9 and 61.1 (which is counter-intuitive, but the reason is explained in [14]), and 55.4 and 52.5, respectively; the fourth application (All-pair Shortest Paths) has very poor multicluster performance. So for these three applications, the extension factor for multicluster operation does not exceed 1.12. In [5], experiences are presented with distributing a large, wide-area optimized application solving Einstein's equations across four supercomputers in two locations using Globus. For a total of 1140 processors, an efficiency of 88% is cited, which amounts to an extension factor of 1.14. In addition, in [11] it was concluded that it pays to use co-allocation when the extension factor is 1.25. We use 1.25 as the extension factor of the service times of multi-component jobs, which is a realistic upper bound for many application.

We call "gross utilization" the utilization obtained for the system with extended service times. Since there is no preemption for communication, the processors are considered busy also when the jobs running on them are involved in inter-cluster communication. The "net utilization" takes into account only the processor usage for computations and fast local communication, i.e., the non-extended service times. In Sect. 4 we compare the two types of utilization for several policies and component-size limits.

2.5 The scheduling policies

In a multicluster system where co-allocation is used, jobs can be either single-component or multi-component, and in a general case both types are simultaneously present in the system. A scheduler dealing with the first type of jobs can be local to a cluster and does not need any knowledge about the rest of the system. For multi-component jobs, the scheduler needs global information for its decisions.

Treating both types of jobs equally or keeping single-component jobs local and scheduling only multi-component jobs globally over the entire system, having a single global scheduler or schedulers local to each cluster, all these are decisions that influence the performance of the system. In [8] we have studied several policies, some of which with multiple variations; in this paper we consider the following approaches:

1. **[GS]** The system has one global scheduler with one

global queue, for both single- and multi-component jobs. All jobs are submitted to the global queue. The global scheduler knows at any moment the number of idle processors in each cluster and based on this information chooses the clusters for each job.

2. **[LS]** Each cluster has its own local scheduler with a local queue. All queues receive both single- and multi-component jobs and each local scheduler has global knowledge about the numbers of idle processors. However, single-component jobs are scheduled only on the local cluster. The multi-component jobs are co-allocated over the entire system. When scheduling is performed all enabled queues are repeatedly visited, and in each round at most one job from each queue is started. When the job at the head of a queue does not fit, the queue is disabled until the next job departs from the system. At each job departure the queues are enabled in the same order in which they were disabled.

3. **[LP]** Each cluster has its own local scheduler with a local queue and all the single-component jobs are distributed among the local queues, and there is a global scheduler with a global queue where all the multi-component jobs are placed. The local schedulers have priority: the global scheduler can schedule jobs only when at least one local queue is empty. When a job departs, if one or more of the local queues are empty both the global queue and the local queues are enabled. 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 when at least one of the local queues gets empty. When both the global queue and the local queues are enabled at job departures, they are always enabled starting with the global queue. The order in which the local queues are enabled does not matter since the jobs in them are only started on the local clusters.

In all the cases considered, both the local and the global schedulers use the FCFS policy to choose the next job to run.

For comparison, we consider the single-cluster case where there are only single-component jobs and we use FCFS as scheduling policy (**[SC]**).

3 Performance evaluation

In this section we assess the performance of multicluster systems for the scheduling policies introduced (Sect. 2.5), for the DAS-t-900, DAS-s-128 and DAS-s-64 workload distributions, for several job-component-size limits and for different ways of distributing the jobs across the local queues. We compare the results to those for a single cluster with FCFS.

The simulations are for a multicluster with 4 clusters of 32 processors each and a single cluster with 128 processors. For the policies where local queues are defined, we compare the balanced case (all local queues receive the same percentage of jobs) to the unbalanced case when one local queue receives 40% and the other three 20% of the jobs submitted locally. The simulation programs were implemented using the CSIM simulation package [13].

In this section we only look at the gross utilization, and depict the response time as a function of this utilization because that is a fair basis for comparing the policies. For SC there is no wide-area communication and the net utilization is equal to the gross utilization. In the figures, the legends describe the curves in decreasing order of their performance (right-left).

Section 3.1 makes a general comparison of the policies. In Sect. 3.2 we study the effect of limiting the total job size. Section 3.3 discusses the impact on performance of the job-component-size limit. Section 4 compares for all the policies and size limits the gross and the net utilization, which shows how efficient the global applications use the gross utilization offered.

3.1 Comparing the policies

In this section we compare the three policies defined for multiclusters to the FCFS policy in a single cluster. For multiclusters, with the maximum job-component size we vary the numbers and sizes of job components; it influences the performance by modifying the way jobs fit together and, through the percentage of single-component jobs, the percentage of jobs with extended service times.

3.1.1 Multiclusters versus single clusters

In this section we compare the four policies for each of the three job-component-size limits. For SC the size limit does not influence the results, so the SC curves can be used as a reference for the graphs in Fig. 3. With the workload considered the performance is poor for all policies and all size limits. Even for total requests the maximal utilization is below 0.65. This seems to indicate that the bad performance is caused by the total sizes of the jobs; we further investigate this aspect in Sect. 3.2.

Even though it restricts single-component jobs to the local clusters and uses co-allocation to schedule its many multi-component jobs (48.7%) for which the service times are extended due to the global communication, LS performs much better than the other multicluster policies for a size limit of 16; it also provides a higher maximal utilization than SC, but a part of that utilization is spent waiting for the wide-area communication, so SC is still significantly

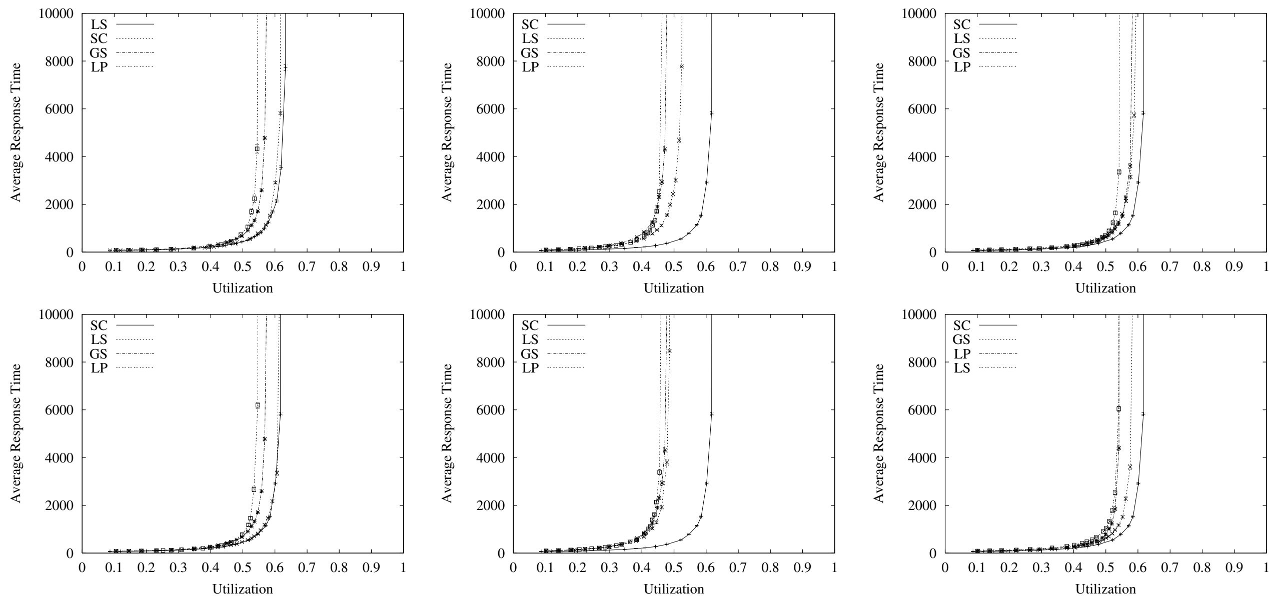

Figure 3. The performance of the policies for job-component-size limits of 16, 24 and 32 (left-right); for LS and LP we depict results with balanced local queues (top) and unbalanced local queues (bottom)

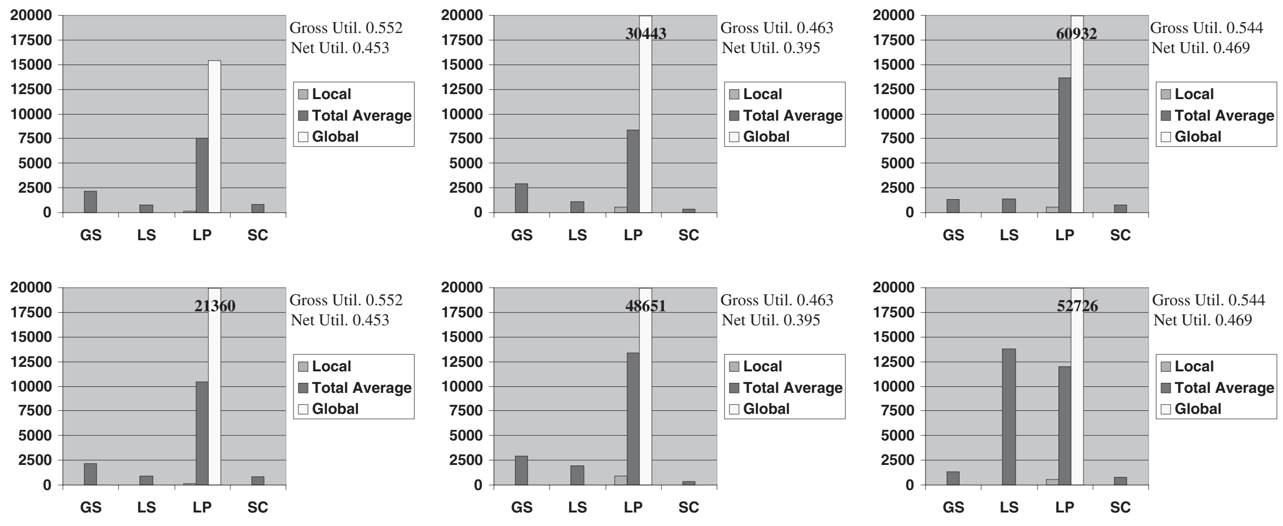

Figure 4. The response times for job-component size limits of 16, 24 and 32 (left-right) close to LP's saturation point; for LS and LP the local queues are balanced (top) and unbalanced (bottom)

better (see also Sect. 4). The main advantage of LS is that it distributes the multi-component jobs among the local queues allowing the local schedulers to spread them across the clusters; at each moment a job can be chosen from any of the local queues, which generates a form of backfilling with a window equal to the number of clusters. For size limits of 24 and 32, where there are only 26.2% and 22.0% multi-component jobs respectively, the performance of LS is worse. In all the graphs LP displays the worst results because all the multi-component 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 single 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; this is due to the fact that it has the freedom to choose the clusters for the single-component jobs.

In itself, the fact that a multicluster policy can have similar performance to SC in terms of gross utilization makes co-allocation a good option, showing that the fragmentation caused by having multiple clusters can be overcome. Even when possible, admitting only single-component jobs would result in a lower utilization as our previous results have shown (see [7]). Combining these results, it seems that a multicluster system without co-allocation deals worse with the fragmentation. The internal loss of utilization caused by the slow global communication should be handled by structuring the applications to use the high utilization offered by some of the policies, while minimizing the amount of time spent with wide-area communication. An application involved in less global communication can benefit better from a policy that uses co-allocation.

3.1.2 Balanced versus unbalanced local queues

Comparing the balanced and unbalanced cases for LS and LP (see Fig. 3) we notice that an unbalanced load for the local queues has a negative impact on performance. Since for both GS and SC all jobs go to the global queue, their curves can be used as a reference when comparing the pairs of balanced-unbalanced graphs for LS and LP.

The worsening of the performance is more pronounced for LS, especially for larger job-component-size limits, when there is a higher percentage of local jobs. One 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 in a queue means a higher load for the local cluster; besides, their cluster can be used with equal priority by global jobs from the other local queues. As a result, the more loaded local queue saturates faster than in the balanced case, decreasing this way the maximal utilization of the system. For a size limit of 32 and unbalanced local queues, LS performs worse than GS and similarly to LP.

For LP the performance deterioration due to the unbalance of the local queues is small for all size limits. All global jobs go to the global queue, and the local queues have priority on their local clusters as long as none of them is empty. 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% 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 multi-component jobs from the global queue are started, the use of WF insures that the least loaded clusters are chosen.

3.1.3 LP approaching saturation

For LP, we can expect that the performance differs between the global and local queues. In Fig. 4 we depict for the LP policy, beside the total average response times, the average response times for the local queues and for the global queue. Beside the gross utilization we display in each chart the corresponding net utilization. The figure shows the same results as Fig. 3, but for a small set of utilization values chosen so that at least one of the policies approaches saturation. This representation makes it easier to compare the response times of the policies but has the limitation of providing less information. At the chosen utilization values, LP, the policy with the worst performance, is approaching saturation—the global queue grows without bounds. As a consequence, we can easily see for LP the differences in response times, for example among the balanced and unbalanced cases, even though as the graphs in Fig. 3 show these differences are small. On the other hand, if we compare the response times for the "better" policies like LS, the charts do not show the significant performance loss from the unbalanced case. At the utilizations from Fig. 4, LS is on a very low point on its response-time curve and its performance is very good in both the balanced and the unbalanced cases. LP delays the large multi-component jobs much longer than LS and its performance bottleneck is the global queue and not the local ones. This is shown in Fig. 4 where for LP the average response times for the global queue are much larger than those for the local queues. The response times for the global queue are displayed on the corresponding bars in the charts.

3.1.4 Conclusions

We conclude that for a system with a workload similar to that of the DAS, and an overhead due to the wide-area communication covered by the 1.25 extension factor, the LS policy is the best option. In all the cases the performance is very poor: for all job-component-size limits, both in the balanced and the unbalanced cases, and even for total requests. 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 multi-component jobs in a multicluster system. We verify this assumption in the next section.

3.2 Limiting the total job size

In the job-size distribution derived from the DAS, only 2% of the jobs require more than 64 processors (out of which 1.2% need 128 processors to run, which is the entire system). Our assumption is that eliminating from the distribution this small percentage of very large jobs can bring important changes to our previous results.

Figure 5 compares the performance of the DAS-s-128 and the DAS-s-64 distributions for the four policies discussed before, a job-component-size-limit equal to 16 and balanced local queues. We choose this set of parameters because they previously provided the most interesting results (LS better than SC).

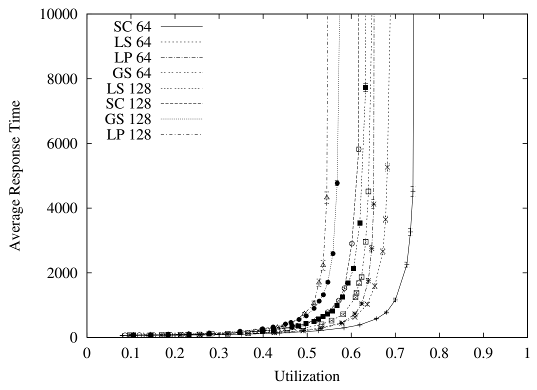

Figure 5. The response times for maximal total job size 64 and 128 (job-component-size limit 16, balanced local queues)

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 top 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 at the top large jobs 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 it gives the local queues priority, with LP the very large jobs in the global queue are much postponed for DAS-s-128. For DAS-s-64 jobs are smaller, and LP outperforms GS because it can benefit from having more queues (also a form of backfilling but worse than in the case of LS since all global jobs go to the global queue).

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, for the users whose jobs are larger than the limit allowed, complying to this restriction translates into reconfiguring their jobs to use fewer processors and accepting the consequence of having longer service times. The extension of the service times is highly dependent on the specific application.

3.3 Setting the job-component-size limit

In this section we evaluate the performance of GS, LS and LP depending on the maximum size admitted for the job components. We compare three size limits: 16, 24 and 32 (see Fig. 6), and use the DAS-s-128 distribution.

Having smaller job components improves the system's performance, but having many components relative to the number of clusters deteriorates the performance. A smaller size limit also means more multi-component jobs, so more jobs with extended service times, which also worsens the performance; for a size limit of 16 there are 26.7% more multi-component 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 multi-component jobs can be spread across any of the clusters, while the single-component jobs are restricted to the local clusters, a smaller size limit becomes an important advantage. For this policy a size limit of 16 is a much better choice than a size limit of 32. Comparing the balanced and unbalanced cases for LS we notice that the performance deteriorates more when the local queues are not balanced for a size limit of 32. The queue receiving extra load can place the multi-component jobs also on other clusters, but it has to keep local the single-component jobs, overloading the local cluster. It means that the more single-component jobs there are, the more will the system be affected by the unbalance.

For LP, all multi-component jobs go to the global queue and only the single-component jobs are spread among the local queues. Smaller local jobs fit better, and fewer local jobs mean fewer jobs restricted to the local clusters. Fewer local jobs also mean that 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 low priority when no local queue is empty, and the more global jobs, the higher the average response time for those jobs. As Fig. 6 shows, LP performs better for a size limit of 16 than for 32, but the difference is small in both the balanced and the unbalanced cases.

For all the policies, the worst results are obtained 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 very large difference in performance should be in the way jobs fit together in the system, depending on the size limit of their components. For a size limit of 24, the jobs split up differently are only those with sizes (integers) in the intervals $[17, 24]$ and $[33, 72]$ when compared to a size limit of 16, and the intervals $[25, 32]$, $[49, 64]$ and $[73, 96]$ when compared to 32. Since for GS and LP the performance is very similar for size

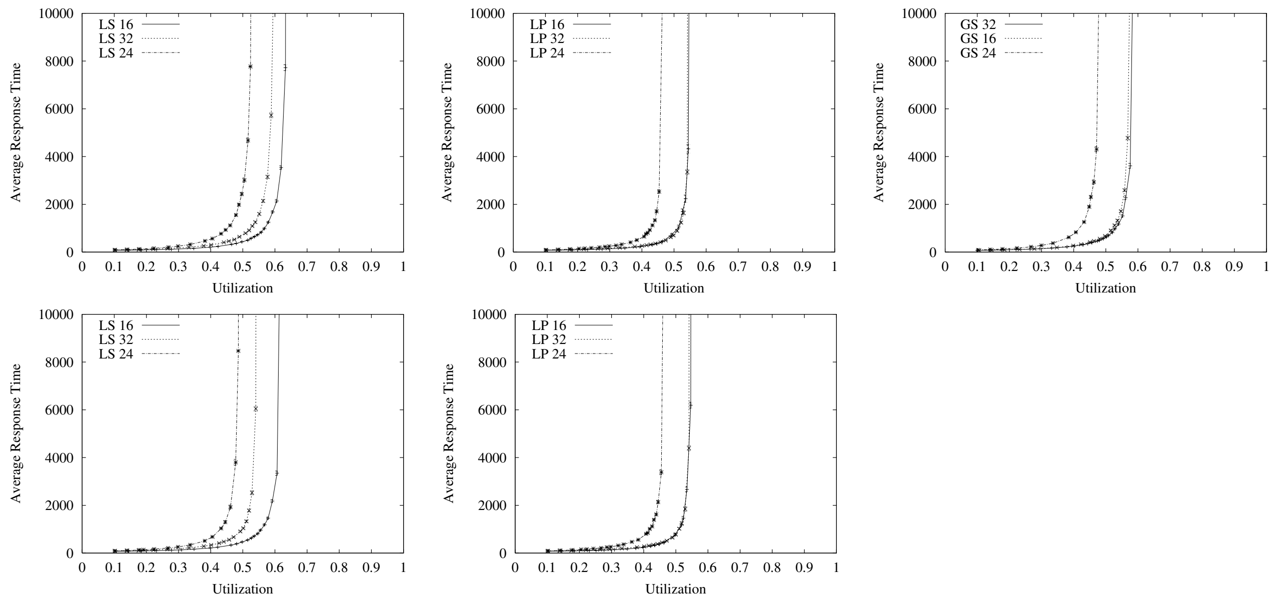

Figure 6. The performance of LS, LP and GS (left-right) depending on the size limit of the job components. For LS and LP both the balanced (top) and unbalanced (bottom) cases are depicted

limits of 16 and 32, it makes sense to look for the reason of the much worse performance when the size limit is 24 in the interval $[49, 64]$, which contains the jobs split differently compared to both the other limits. Checking the log, we found that the most relevant size in that interval is 64: 19.0% of the jobs in the log have this size. 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 we obtain $(22, 21, 21, 0)$. 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 with a size of 64 would also fit in the first two cases, but not in the third.

4 Gross versus net utilization

In Sect. 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, and quantify this difference for the cases considered in Sect. 3. We have defined in Sect. 2.4 the net and the gross utilization based on the job service times in single clusters with fast local communication, and on the extended service times (at least, for multi-component jobs) to account for the slow wide-area communications, respectively. The difference between these utilizations is the capacity lost internally in multi-component jobs due to slow wide-area links. This internal capacity loss might be reduced by restructuring applications or by having them use (collective-) communication operations optimized for wide-area systems.

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 (or of the throughput) may be poor. This ”real” performance of a multicluster policy would improve with more efficient applications or with faster global communication (smaller extension factor). 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. 3).

In Fig. 7 we depict the average response time for our three policies, for the DAS-s-128 job-size distribution, and for the three job-component-size limits as a function of both the gross and the net utilization. For LS and LP the local queues are balanced; the results for unbalanced local queues do not bring additional information. To assess the difference between the two utilizations for a specific policy and job-component-size limit at a certain response time, one should compare the graphs in the horizontal direction. Of course, for the same workload (defined by the arrival rate, and so, by the net utilization) and job-component-size limit, the difference between the gross and the net utilization is the same for all scheduling policies, albeit at possibly different

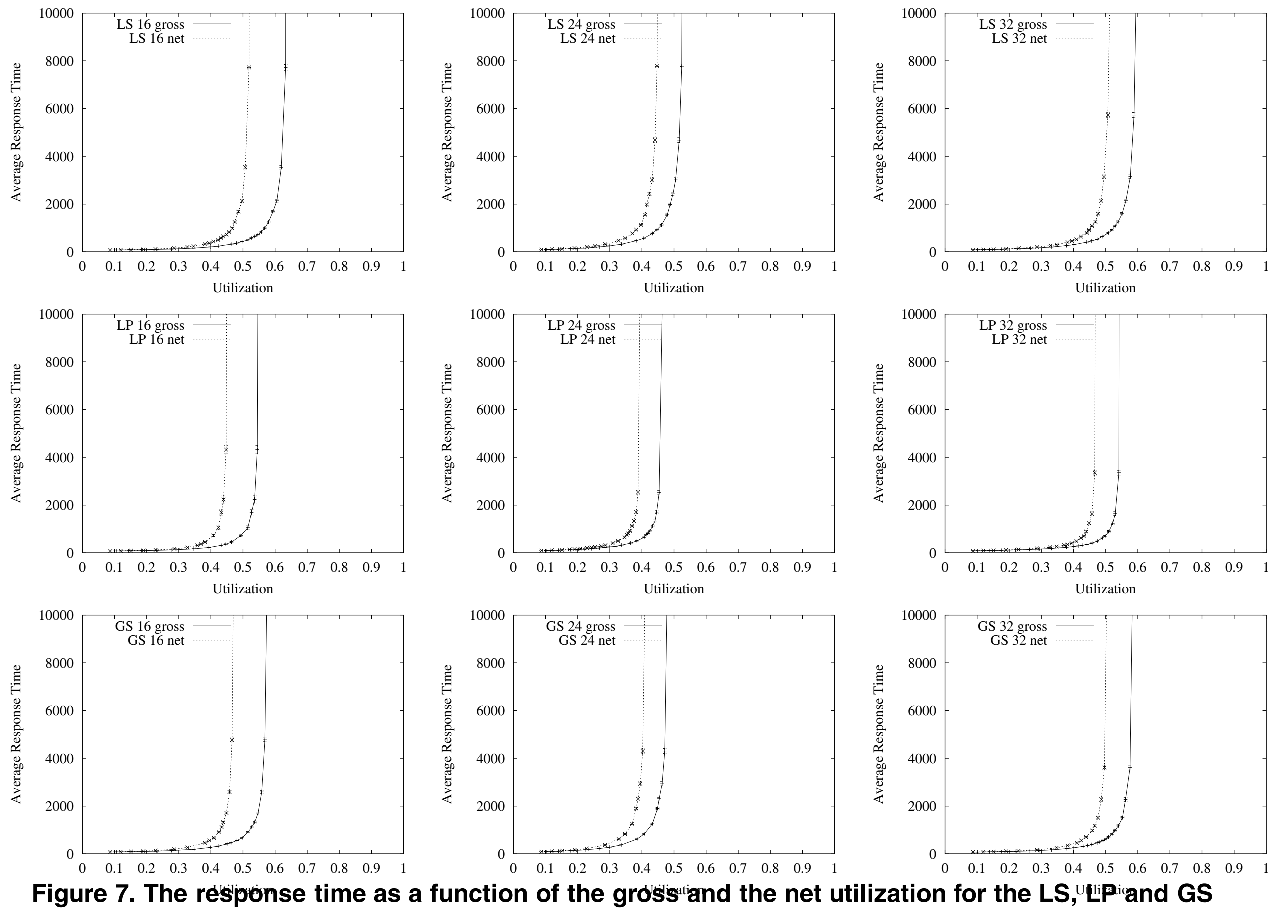

Figure 7. The response time as a function of the gross and the net utilization for the LS, LP and GS policies and for the three job-component-size limits (balanced local queues for LS and LP)

response times.

In our model, job sizes and job service times are independent. 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 multi-component jobs having weight 1.25 (the extension factor) and the 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.

The difference between the gross and the net utilization grows with a decrease of the job-component-size limit—the more multi-component jobs the higher the amount of global, slow communication—and with the number of (multi-component) jobs that fit on the system—when jobs fit better, the workload can be higher, entailing more global communication. For these reasons, for all the policies a job-component-size limit of 16 yields a larger distance between the curves of the net and gross utilizations than 24 and 32. For a size limit of 24 there are 4.2% more multi-component jobs than for 32, but since jobs fit very poorly, the gross and net utilizations are closer together when the size limit is 24. Comparing the policies, we notice that for a size limit of 16, LS yields a much better utilization, and so also more wide-area communication, than the other policies, and as a consequence, it has the largest difference between the gross and the net utilization. For component-size limits of 24 and 32 the amount of utilization spent in global communication is rather similar for all three policies.

In [9], we have studied the maximal utilization (beyond which the system gets unstable) of co-allocation with analytic means (when the service times are exponential) and with simulations. Both methods only work when there is a single, global queue, so in the cases of GS and SC. In these simulations, we maintain a constant backlog and observe the time-average fraction of processors being busy, which yields the maximal gross utilization. For GS, the simula-

Table 3. The maximal gross and net utilizations for different job-component-size limits for the GS policy

	maximal utilization	
job-component-size limit	gross	net
16	0.588	0.483
24	0.491	0.419
32	0.595	0.513

tions produced the values for the maximal gross utilization as in Table 3; the maximal net utilizations are then computed with the ratios between the two types of utilization. These values are in very good agreement with the graphs for GS in Figure 7. For SC, our simulations gave us a maximal utilization of 0.625 (cf. the curve for SC in Fig. 3).

5 Conclusions

In this paper we have evaluated the performance of several scheduling policies for co-allocating unordered requests in multiclusters with a workload derived from the DAS. The best policy proved to be LS, with a performance comparable in some cases to using FCFS for total requests in a single cluster.

Studying the job sizes, we have observed that a small percentage of jobs with total sizes close to the size of the system can strongly impact on performance. Although the choice of policy does matter for the performance, when dealing with very large jobs requiring (almost) the entire system to run, we found that by far the largest improvement in performance can be obtained from simply limiting the total job size.

When choosing the job-component-size limit, care should be taken that the jobs of the total sizes that occur most often (in our log 19% of the jobs are of size 64!) are split up in a way that makes them easy to fit in the system. When both the clusters' sizes and the most popular total job sizes are powers of two, a limit that is also a power of two makes sense.

In multicluster systems we have to deal with a significant amount of processor time spent waiting for the global communication. However, co-allocation remains a viable option while the duration of the global communication is covered by an extension factor of 1.25.

For some policies, such as LS, a higher amount of co-allocation—going from component size limit of 32 to 16—can significantly improve the performance, while for others (LP, GS) it does not. More co-allocation brings along more flexibility in spreading the jobs over the system, but also more global communication. For a policy that can take advantage of it, this flexibility can compensate the disadvantage of slower communication.

References

[1] *The Distributed ASCI Supercomputer (DAS).* `www.cs.vu.nl/das2`.

[2] *The Global Grid Forum.* `www.gridforum.org`.

[3] *Globus.* `www.globus.org`.

[4] *The Portable Batch System.* `www.openpbs.org`.

[5] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with CACTUS and Globus. In *Proc. of Supercomputing*, 2001.

[6] A. Bucur and D. Epema. The Influence of the Structure and Sizes of Jobs on the Performance of Co-Allocation. In D. Feitelson and L. Rudolph, editors, *6th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *LNCS*, pages 154–173. Springer-Verlag, 2000.

[7] A. Bucur and D. Epema. The Influence of Communication on the Performance of Co-Allocation. In D. Feitelson and L. Rudolph, editors, *7th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *LNCS*, pages 66–86. Springer-Verlag, 2001.

[8] A. Bucur and D. Epema. Local versus Global Queues with Processor Co-Allocation in Multicluster Systems. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *8th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2537 of *LNCS*, pages 184–204. Springer-Verlag, 2002.

[9] A. Bucur and D. Epema. The Maximal Utilization of Processor Co-Allocation in Multicluster Systems. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2003 (to appear; available from www.pds.its.tudelft.nl/ epema).

[10] S.-H. Chiang and M. Vernon. Characteristics of a Large Shared Memory Production Workload. In D. Feitelson and L. Rudolph, editors, *7th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *LNCS*, pages 159–187. Springer-Verlag, 2001.

[11] 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.

[12] H.E. Bal et al. The Distributed ASCI Supercomputer Project. *ACM Operating Systems Review*, 34(4):76–96, 2000.

[13] Mesquite Software, Inc. *The CSIM18 Simulation Engine, User's Guide.*

[14] R. van Nieuwpoort, J. Maassen, H. Bal, T. Kielmann, and R. Veldema. Wide-Area Parallel Programming Using the Remote Method Invocation Method. *Concurrency: Practice and Experience*, 12(8):643–666, 2000.