Local versus Global Schedulers with Processor Co-Allocation in Multicluster Systems

A.I.D. Bucur D.H.J. Epema
Faculty of Information Technology and Systems
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, The Netherlands

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 different clusters, may be required. We study the performance of co-allocation by means of simulations for the mean response time of jobs depending on a set of scheduling decisions such as the number of schedulers and queues in the system, the way jobs with different numbers of components are distributed among these queues and the priorities imposed on the schedulers, and on the composition of the job stream.

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, 8] enables applications to access resources in different and possibly widely dispersed locations simultaneously—that is, to employ processor *co-allocation* [5]—to accomplish their goals, effectively creating single multicluster systems. Most of the research on processor scheduling in parallel computer systems has been dedicated to multiprocessors and single-cluster systems, but hardly any attention has been devoted to multicluster systems.

In this paper we study through simulations the performance of processor co-allocation policies in multicluster systems employing space sharing for rigid jobs [3], depending on several scheduling decisions and on the composition of the job stream. The scheduling decisions we consider are the number of schedulers and queues in the system, the way jobs with different numbers of components are distributed among queues and the priorities and restrictions imposed on the schedulers. Our performance metric is the mean job response time as a function of the utilization.

Using co-allocation does not mean that all jobs have to be split into components and spread over the clusters, small jobs can also be submitted as single-component jobs and go to a single cluster. In general, there is in the system a mix of jobs with different numbers of components. In this context, an important decision to make is whether there will be one global scheduler with one global queue in the system, or more schedulers and in the second case how jobs will be divided among schedulers.

Our results show that a multicluster which employs co-allocation and treats all job requests as unordered requests, i.e., the user specifies the numbers of processors needed in separate clusters but not the clusters, also improves the performance of single-component jobs by not restricting them to a cluster, and choosing from all the clusters in the system one where they fit. Evaluating different scheduling decisions, we find the best choice to be a system where there is one scheduler for each cluster, and all schedulers have global information and place jobs using co-allocation over the entire system.

Our four-cluster Distributed ASCI Supercomputer (DAS) [6] was designed to assess the feasibility of running parallel applications across wide-area systems [4, 9, 13]. In the most general setting, GRID resources are very heterogeneous; in this paper we restrict ourselves to homogeneous multicluster systems, such as 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 [1, 6] is a wide-area computer system consisting of four clusters of identical Pentium Pro processors, one with 128, the other three with 24 processors each. The clusters are interconnected by ATM links for wide-area communications, while for local communication inside the clusters Myrinet LANs are used. The system was designed for research on parallel and distributed computing. On single DAS clusters a local scheduler is used that allows users to request a number of processors bounded by the cluster's size, for a time interval which does not exceed an imposed limit.

2.2 The Workload

Although co-allocation is possible on the DAS, so far it has not been used enough to let us obtain statistics on the sizes of the jobs' components. However, from the log of the largest cluster of the system 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 comply with the distributions we use for the job-component sizes in that there is an obvious preference for small numbers and powers of two.

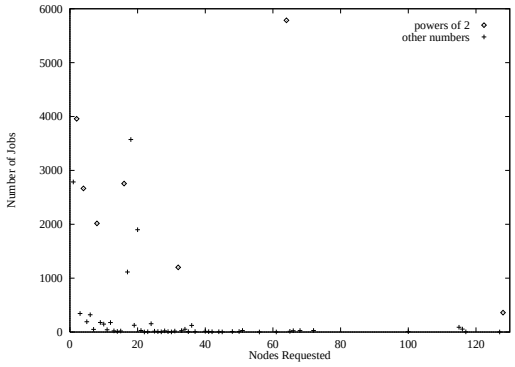

Figure 1. The density of the job-request sizes for the largest DAS cluster (128 processors)

From the jobs considered, $28,426$ were recorded in the log with both starting and ending time, and we could compute their service time. Due to the fact that during working hours jobs are restricted to at most 15 minutes of service (they are automatically killed after that period), 94.45% of the recorded jobs ran less than 15 minutes. Figure 2 presents the density of service time values on the DAS, as it was obtained from the log. The average service time is 356.45 seconds and the coefficient of variation is 5.37. Still, not all jobs in the log were short: the longest one took around 15 hours to complete. Figure 3 divides the service times of the jobs into eight intervals: $< 10s$, $10 - 30s$, $30 - 60s$, $60 - 300s$, $300 - 900s$, $900 - 1800s$, $1800 - 3600s$, and $> 3600s$, each segment in the graph parallel to the horizontal axis corresponds to an interval. The vertical axis coordinate of any point of a segment represents the total number of jobs in that interval.

In our simulations, beside an exponential distribution with mean 1 we also use for the service-time distribution the distribution derived from the log of the DAS, cut off at 900 seconds (which is the run-time limit during the day). The average service time for the jobs in the cut log is 62.66 and the coefficient of variation is 2.05. We made the choice to use both distributions because with the DAS distribution we obtain a more accurate, realistic evaluation of the DAS performance, but in the same time this distribution might be very specific and make our results hard to compare to those from other systems. On the other hand, the exponential distribution is less realistic but more general and more suited for analysis.

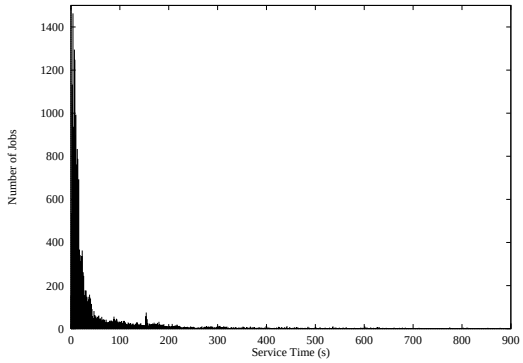

Figure 2. The density of the service times for the largest DAS cluster (128 processors)

2.3 The Structure of the System

We model a multicluster system consisting of C clusters of processors, cluster i having N_i processors, $i = 1, \ldots, C$. 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

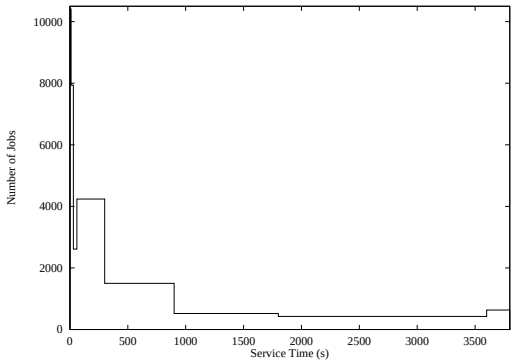

Figure 3. The service times of jobs divided into eight main intervals

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.

2.4 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. The distribution of the sizes of the job components is $D(q)$ defined as follows: $D(q)$ takes values on some interval $[n_1, n_2]$ 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 favours small sizes, and sizes that are powers of two, which has been found to be a realistic choice [7]. A job is represented by a tuple of C values, each of which is either generated from the distribution $D(q)$ or is of size zero. We consider only *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. 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; pick the cluster with the largest number of idle processors) to place the components on clusters.

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. It is useful to make this division since the single-component jobs do not use co-allocation while multi-component jobs do. 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 multicluster system, having a single global scheduler or schedulers local to each cluster, all these are decisions that influence the performance of the system. 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.

3. **[EQ]** The system has both a global scheduler with a global queue, and local schedulers with local queues. Multi-component jobs go to the global queue and are scheduled by the global scheduler using co-allocation over the entire system. Single-component jobs are placed in one of the local queues and are scheduled by the local scheduler only on its corresponding cluster. There is no direct priority policy imposed on the schedulers when accessing the clusters.

   When both the global scheduler and some of the local schedulers are blocked at "full system" (the jobs at the top of their queues do not fit) and a job departs, the system will first unblock the local schedulers associated to the clusters where the job ran. This favours the local schedulers allowing them to try to place jobs before the global scheduler, but since with the chosen job stream compositions the load of the local queues is low (each of them receives maximum 12.5% of the jobs in the system — see Sect. 3), it is a bearable burden for the global scheduler. The opposite choice would be much

to the disadvantage of the jobs in the local queues because, depending on the job stream composition, up to 75% of the jobs can be multi-component and go to the global queue; unblocking first the global scheduler would give little chance to the local schedulers to fit their jobs.

4. **[GP]** Again both global and local schedulers with their corresponding queues. Like before, the global queue receives the multi-component jobs while the single-component jobs are placed in the local queues. The local schedulers are allowed to start jobs only when the global scheduler has an empty queue.

5. **[LP]** Both global and local schedulers, but this time the local schedulers have priority: the global scheduler gets the permission to work only when at least one local queue is empty. When both the global scheduler and some of the local schedulers are blocked at "full system" and a job departs, the system first unblocks the global scheduler if one or more of the local queues are empty. If no local queue is empty only the local schedulers are unblocked.

6. **[LQ]** Both global and local schedulers; at any moment either the local schedulers are allowed to work, or the global one, depending on the lengths of their queues. The global scheduler is enabled if its queue is longer than all the local queues, otherwise the local schedulers are enabled. This strategy might seem to favour the local schedulers (the global scheduler is only permitted to schedule jobs when its queue is longer than all the others), but our results show that this is not the case. It only takes into account the fact that each of the local schedulers accesses just one cluster, so they can be simultaneously enabled. To allow the local schedulers to work only when more of their queues are longer than the global queue would be much to the disadvantage of the local schedulers, especially if the load of their queues is unbalanced.

When the local queues receive only single-component jobs, the local schedulers manage disjoint sets of resources (a local scheduler starts jobs on a single cluster) and there is no need for coordination among them. However, for systems with both a global scheduler and local ones, or when the local schedulers also deal with the multi-component jobs and may use more clusters, the access to the data structures used in the process of scheduling (numbers of idle processors, queue lengths) has to be mutually excusive. The global scheduler always uses global information since it does co-allocation over the entire system; except for the case when they also schedule multi-component jobs, the local schedulers only need access to the data associated to their own cluster.

In the extreme case, GP can indefinitely delay the single-component jobs, and LP can do the same with the multi-component jobs. In practice, an aging mechanism has to be implemented in order to prevent this behaviour.

Once a scheduler gets to run, we allow it to start as many jobs as possible: it only suspends itself when its queue is empty or when the job at the head of the queue does not fit. This choice reduces the scheduling overhead when coordination among schedulers is required. In all the cases considered, both the local and the global schedulers use the First Come First Served (FCFS) policy to choose the next job to run.

Scheduling jobs from queues with equal priorities is done in the following way. Whenever a job leaves the system, and after all higher-priority queues in the system have had the opportunity to schedule jobs, the non-empty equal-priority queues are enabled in a sequence which only changes when a queue turns empty or when a job arrives at an empty queue. In the first case, the empty queue is simply removed from the sequence of queues, and in the second case, the queue with the new arrival is appended at the end of it. As a consequence, the queues at the head of the sequence are favored, but as they will become empty once in a while and then be appended to it, in the long run the equal-priority queues are treated evenly.

All the local schedulers are assumed to have the same load.

We choose not to include communication in our model because it would not change the quality of the results since all policies are tested with identical job streams (the same numbers of components).

3 Performance Evaluation for the Different Scheduling Decisions

In this section we assess the performance of multicluster systems for the six scheduling policies introduced (Sect. 2.5) depending on the composition of the job stream. Jobs can have between 1 and 4 components, and the percentages of jobs with the different numbers of components influence the performance of the system. We consider the following cases:

- $(25\%, 25\%, 25\%, 25\%)$ Equal percentages of 1-, 2-, 3- and 4-component jobs are submitted to the system.
- $(100\%, 0\%, 0\%, 0\%)$ There are only 1-component jobs.
- $(50\%, 0\%, 0\%, 50\%)$ Only 1- and 4-component jobs are present in the system, in equal percentages.
- $(0\%, 0\%, 0\%, 100\%)$ There are only 4-component jobs.

- $(50\%, 25\%, 25\%, 0\%)$ No 4-component jobs, half of the jobs are single-component ones.
- $(0\%, 50\%, 50\%, 0\%)$ Just 2- and 3-component jobs in equal proportions.
- $(50\%, 50\%, 0\%, 0\%)$ Only 1- and 2-component jobs are submitted.

The simulations in this section are for a system with 4 clusters of 32 processors each, and the job-component sizes are generated from $D(0.9)$ on $[1, 8]$. The simulation programs were implemented using the CSIM simulation package [11]. For all the graphs in this section we computed confidence intervals; they are at the 95%-level. For the distribution of service times we use an exponential distribution with mean 1 in Sect. 3.1 and a distribution derived from the DAS log (see also Sect 2.2) in Sect. 3.2.

3.1 Simulations with exponential service

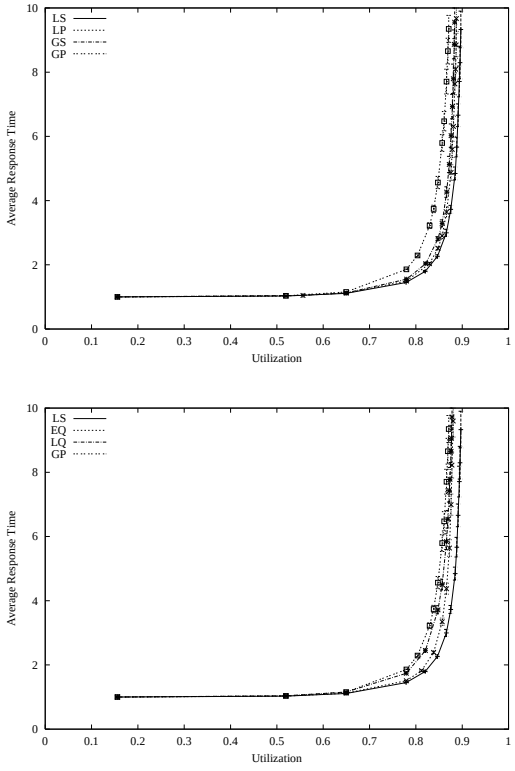

Figure 4. A performance comparison of the scheduling strategies for a job stream with composition $(25\%, 25\%, 25\%, 25\%)$ and exponential service times

Figure 4 compares the different scheduling strategies for a job stream containing 1-, 2-, 3- and 4-component jobs in equal proportions. The best performance is obtained for LS, where all jobs go to the local schedulers and all four schedulers are allowed to spread the multi-component jobs over

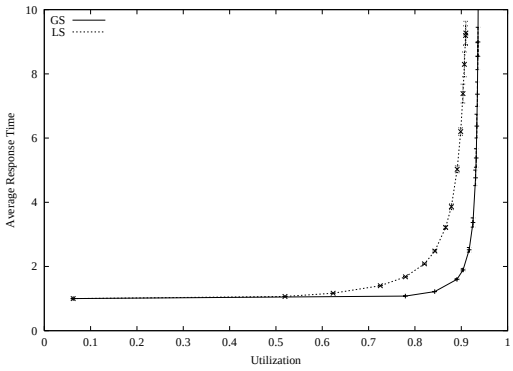

Figure 5. A performance comparison of the scheduling strategies for a job stream with composition $(100\%, 0\%, 0\%, 0\%)$ and exponential service times

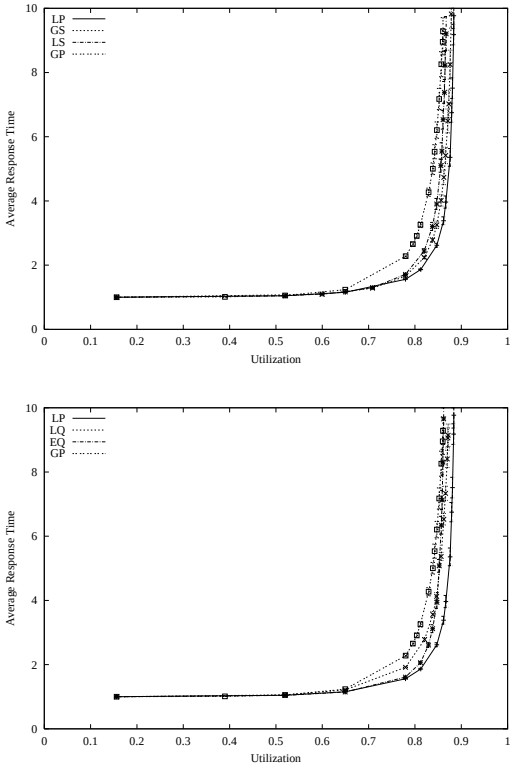

Figure 6. A performance comparison of the scheduling strategies for a job stream with composition $(50\%, 0\%, 0\%, 50\%)$ and exponential service times

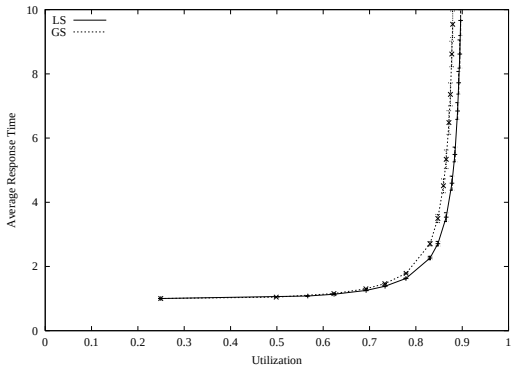

Figure 7. A performance comparison of the scheduling strategies for a job stream with composition $(0\%, 0\%, 0\%, 100\%)$ and exponential service times

the entire system. At any moment, the system tries to schedule up to four jobs (if no queue is empty), one from each of the four local queues, and the FCFS policy is transformed this way into a form of backfilling with a window of size 4. This explains why LS is better than GS. A disadvantage for LS compared to GS is that LS can place 1-component jobs only on the cluster where they were submitted, while the other can choose from the four clusters one where the job fits. However, in the case from Fig. 4, only 25% of jobs have one component, so their negative influence on the performance of LS is small.

GP, LP, EQ, and LQ try to schedule up to 5 jobs at a time, but since 75% of the jobs in the system are multi-component and they all go to the global queue, and only the rest of 25% is distributed among the local queues, their performance is worse than that of LS. GP displays the worst performance; it gives priority to the global scheduler and only allows the local schedulers to run jobs when the global queue is empty. Even if the job at the head of the global queue does not fit, the policy does not allow jobs from the local queues to run and this deteriorates the performance. Since most of the jobs are multi-component, the global queue is the longest in most of the cases and LQ behaves similarly to GP and its performance is the second worst. LP and EQ also run mostly jobs from the global queue, but they do not delay the jobs from the local queues when the job at the top of the global queue does not fit and this improves their performance.

Figures 5, 7 and 9 compare only the GS and LS strategies. The system in Fig. 5 contains only single-component jobs, so EQ, GP, LP, and LQ are reduced to LS. In the other two cases there are only multi-component jobs, so EQ, GP, LP and LQ become GS. We also used these cases to check our simulations and gain confidence in the results.

When there are only single-component jobs in the system (Fig. 5), GS has better performance due to the fact that it chooses the clusters for the jobs (with WF), while with LS jobs can be scheduled only on the clusters they were submitted to. With single-component jobs GS does a sort of load balancing over the entire system (it does not look at the actual loads however) while LS keeps the clusters in isolation.

In Figures 7 and 9 LS proves to be better because for multi-component jobs the local schedulers are not restricted to their own clusters and there are up to four jobs at a time from which to choose one that fits in the system.

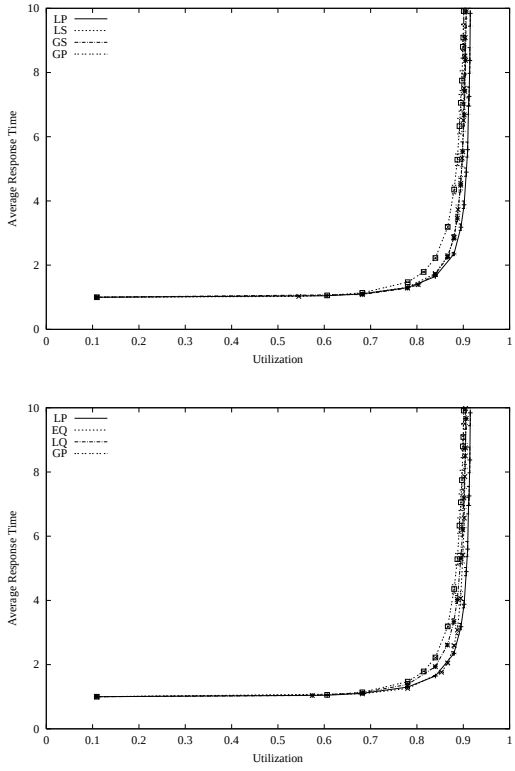

Figure 8. A performance comparison of the scheduling strategies for a job stream with composition $(50\%, 25\%, 25\%, 0\%)$ and exponential service times

Figures 6, 8 and 10 show that for GP the performance decreases with the increase of the percentage of jobs with 3 and 4 components. Since jobs with more components cause a higher capacity loss, it is a bad choice not to allow the local schedulers to try to fit jobs from their own queues when the job at the head of the global queue does not fit. Waiting for enough idle processors in multiple clusters for that job results in a deterioration of the performance. This is shown also by the fact that LQ has worse performance when the

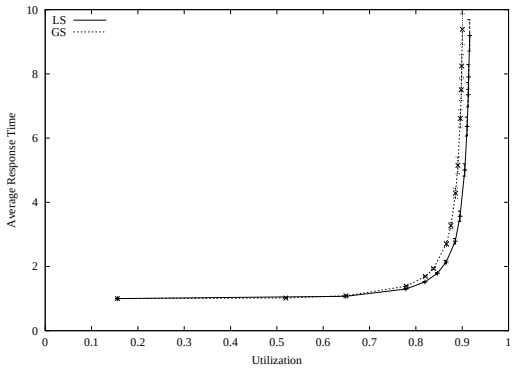

Figure 9. A performance comparison of the scheduling strategies for a job stream with composition $(0\%, 50\%, 50\%, 0\%)$ and exponential service times

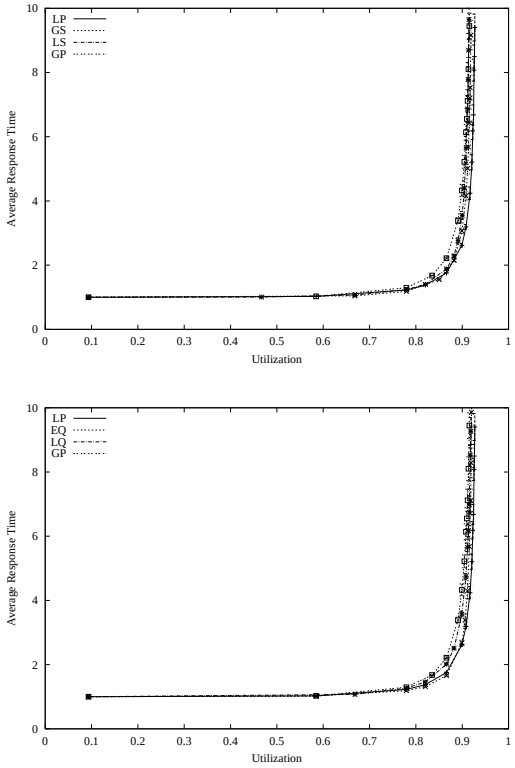

Figure 10. A performance comparison of the scheduling strategies for a job stream with composition $(50\%, 50\%, 0\%, 0\%)$ and exponential service times

percentage of multi-component jobs is higher. EQ has a good performance for all chosen job mixes because it tries to fit as many jobs at possible from all queues without taking into account the characteristics of the job stream.

The best performance in Figs. 6, 8 and 10 is displayed by LP. This suggests that allowing first the 1-component jobs, which are restricted to a certain cluster, to be placed and only then trying to schedule multi-component jobs for which the scheduler can shuffle the components to fit them, improves the utilization of the system. It also seems that when none of the local queues is empty it is a good choice to delay the global jobs waiting for the local jobs to fit, since LP constantly gives better results than EQ, while the opposite decision taken in the case of GP made this policy constantly worse than EQ. The disadvantage of LP is that it tends to delay the multi-component jobs, similarly as GP delays the single-component ones. The differences in performance are larger in Fig. 6 where there are 50% 1-component jobs and 50% 4-component jobs. In Figs. 8 and 10 where there are no 4-component jobs, all strategies display more similar performance. In these two cases there are 50% 1-component jobs and the rest are 2- and 3-component jobs.

Increasing the percentage of 1-component jobs would improve the performance of GS and deteriorate all the others (when there are 100% single-component jobs GP, EQ, LQ and LP all become LS). Increasing the percentage of multi-component jobs would improve the performance of LS, but worsen it for the rest (when there are only multi-component jobs GP, EQ, LQ and LP become GS).

In all the graphs discussed so far we looked at the total (average) response time. However, when there are both local and global queues in the system we can expect that the performance differs between the global and local queues and is dependent on the policy. Figures 11 — 14 show beside the total average response time, the average response times for the local queues and the global queue for the EQ, GP, LP and LQ policies and for the four job compositions which include both single- and multi-cluster jobs. For each utilization value where we approximated before an average response time for the entire system, now we also depict the average response times for the jobs in the global and local queues respectively.

While LP and EQ provide much better performance for local jobs, GP and LQ are better for the global jobs. We cannot say that LQ favours the global jobs in general, since in a system with many single-cluster jobs it would be exactly the opposite. LQ is also fair to all jobs from the perspective that if there is a large job, be it single- or multi-cluster, which is difficult to fit on the system, not only that LQ will give that job a chance to run probably sooner than with other policies (unless they directly favour that type of jobs), but it will also limit the delay for the jobs behind it in the queue. In fact, LQ keeps the load of the queues balanced, switching its behaviour between GP and LP depending on the queue lengths.

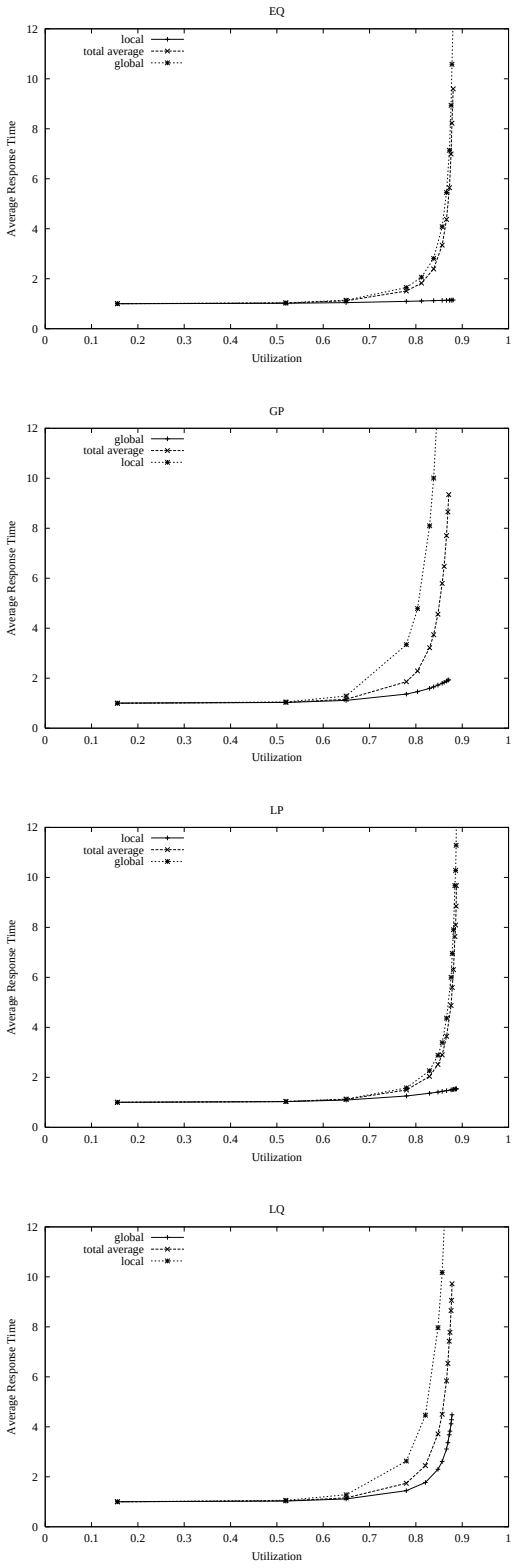

Figure 11. Comparing EQ, GP, LP and LQ for a job stream with composition $(25\%, 25\%, 25\%, 25\%)$ and including the separate performance for the local and global queues

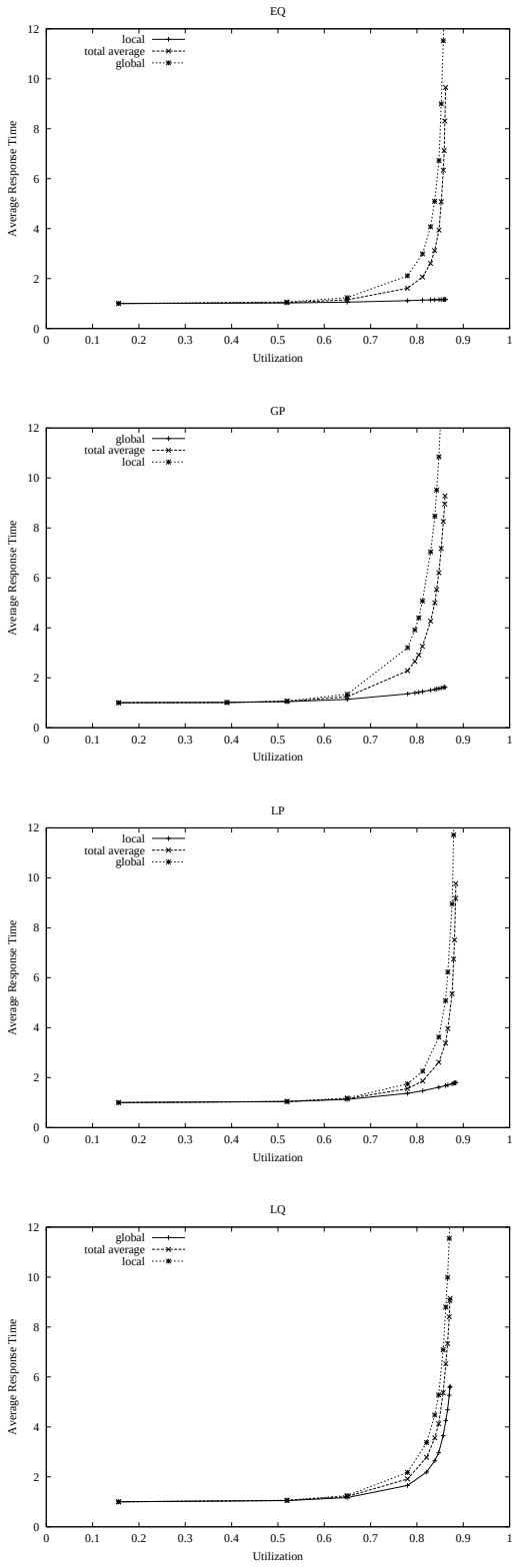

Figure 12. Comparing EQ, GP, LP and LQ for a job stream with composition $(50\%, 0\%, 0\%, 50\%)$ and including the separate performance for the local and global queues

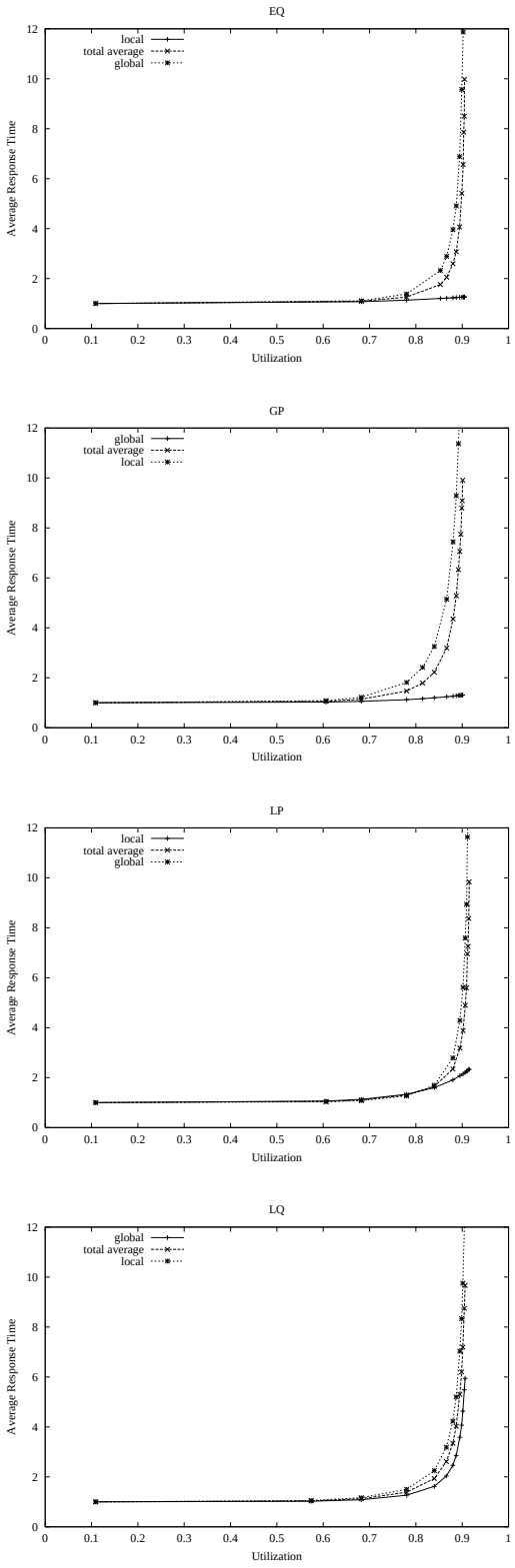

Figure 13. Comparing EQ, GP, LP and LQ for a job stream with composition $(50\%, 25\%, 25\%, 0\%)$ and including the separate performance for the local and global queues

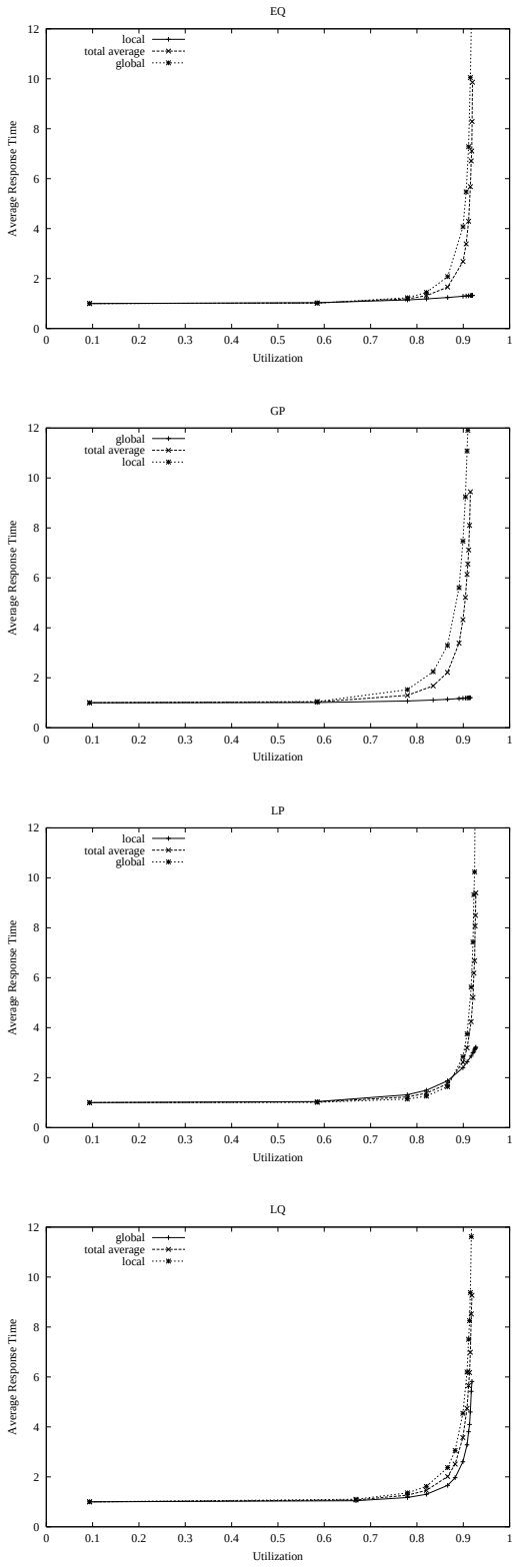

Figure 14. Comparing EQ, GP, LP and LQ for a job stream with composition $(50\%, 50\%, 0\%, 0\%)$ and including the separate performance results for the local and global queues

On the negative side, with LQ the performance of jobs of one type is more sensitive to the performance of jobs of the other type than for EQ, GP or LP.

The figures show that EQ has better performance for the local queues and worse for the global queue than LP. A reason for this is that for EQ when both the global and some of the local schedulers are blocked at ”full system” and a job departs the local schedulers are allowed to try to schedule jobs first. If this decision is reversed the average response time for the local queues increases, for the global queue decreases and the overall performance of the system is improved. When none of the local queues is empty the LP policy strongly favours the local schedulers by not letting the global scheduler run. However, when at least one local queue is empty, the global scheduler is blocked at ”system full” and a job departs, first the global scheduler is allowed to try to place jobs. This decision has a positive effect on the overall performance but slightly deteriorates the performance of the local queues and makes it dependent on the global jobs: the better the global jobs fit, the worse the performance of the local jobs is.

From these four policies the most practical would be LP or EQ since the other two tend to delay the local jobs and it can be expected that the organizations owning the different clusters would not like their local jobs to be delayed in favour of the global, multi-component jobs. Our results show that, for policies like LP and EQ, even a high percentage of global jobs in the system does not deteriorate the performance of the local jobs. However, the users submitting multi-component jobs to a system implementing such a policy should be aware that the performance of their jobs is much influenced by the local jobs and it can be significantly lower than the overall performance of the system.

In most of our graphs, at high utilizations some of the curves are rather close and one might think that it means that the performance is very similar. However, it only shows that the maximum utilizations are close, and not that the average response times are similar. Due to the steepness of the curves at high utilizations, for the same utilization the corresponding response times on two curves that seem very close are very different. To show this, Fig. 15 compares the average response time for the six policies considered for the four job stream compositions which contain both global and local jobs and a utilization high enough to be on the steep side of the curves for all policies, and close to the maximum utilization for the policy with the worst performance. The values for the utilization in all cases correspond to a system that is not saturated for any of the policies. Although they only depict the response time values at a single utilization value each, the charts in Fig. 15 are useful to show that there are large differences in response times for utilization points where the curves in Figs. 4 — 8 are hardly distinguishable.

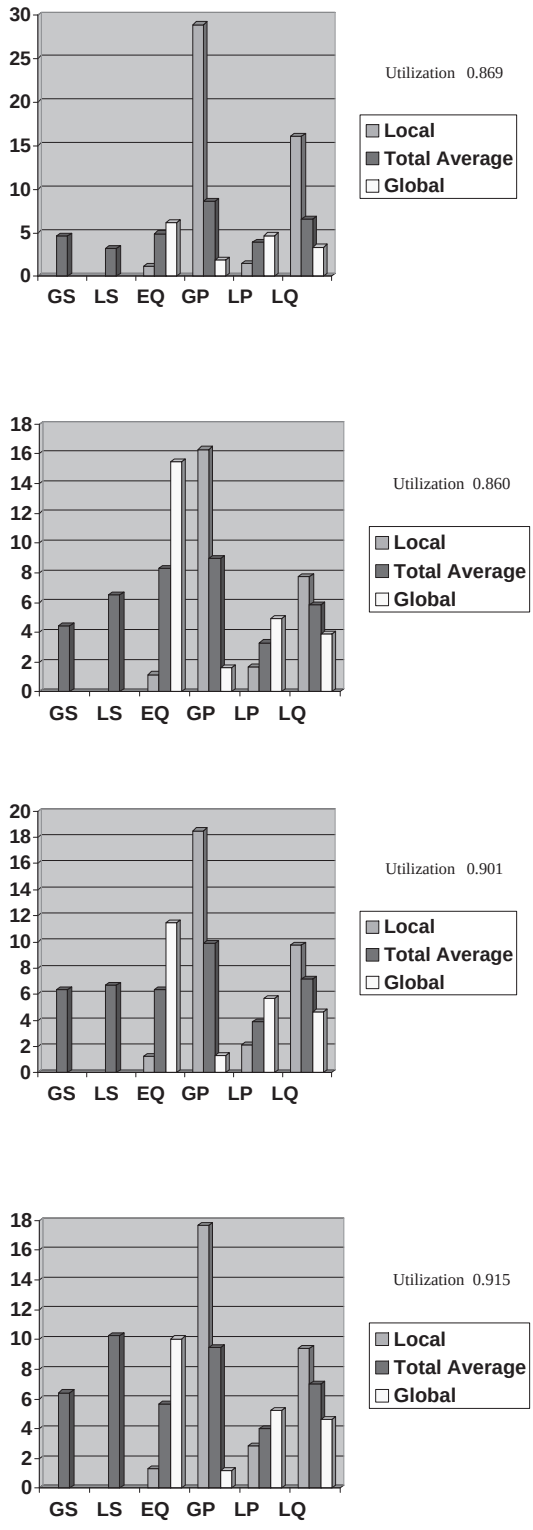

Figure 15. A comparison of the average response times for the policies considered, for job stream compositions (from top to bottom) of $(25\%, 25\%, 25\%, 25\%)$**,** $(50\%, 0\%, 0\%, 50\%)$**,** $(50\%, 25\%, 25\%, 0\%)$ **and** $(50\%, 50\%, 0\%, 0\%)$

Since the displayed results are at different utilizations it is not meaningful to compare the bar charts in Fig. 15 to each other.

3.2 Simulations with the DAS service-time distribution

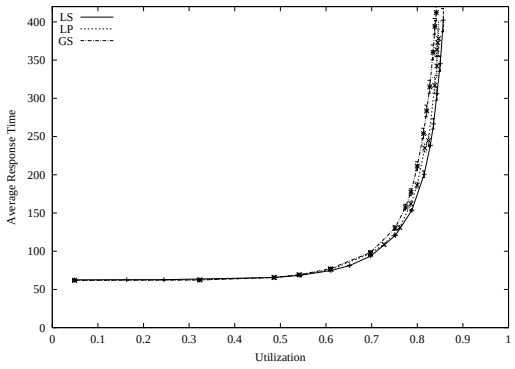

Figure 16. Performance comparison of the scheduling strategies for a job-stream with composition $(25\%, 25\%, 25\%, 25\%)$ and a service time distribution from the DAS

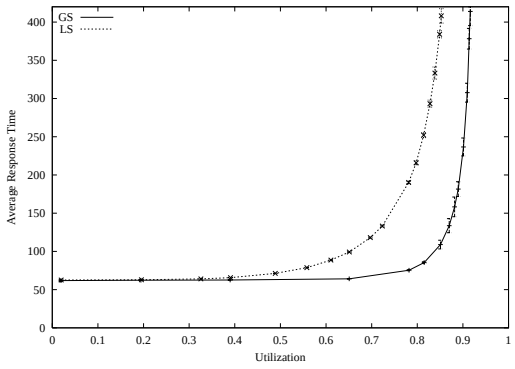

Figure 17. Performance comparison of the scheduling strategies for a job-stream with composition $(100\%, 0\%, 0\%, 0\%)$ and a service time distribution from the DAS

In this section for the service-time distribution we use the cut distribution derived from the DAS log. We only present simulations for LS, LP and GS and the first four job-stream compositions. The results are very much in line with those from the previous section: in Figs. 16 and 19, LS displays the best performance, in Fig. 17 GS is the best and in Fig. 18 is LP. This shows that the previous use of exponential

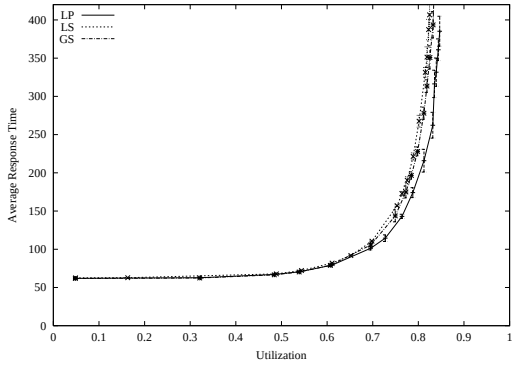

Figure 18. Performance comparison of the scheduling strategies for a job-stream with composition $(50\%, 0\%, 0\%, 50\%)$ and a service time distribution from the DAS

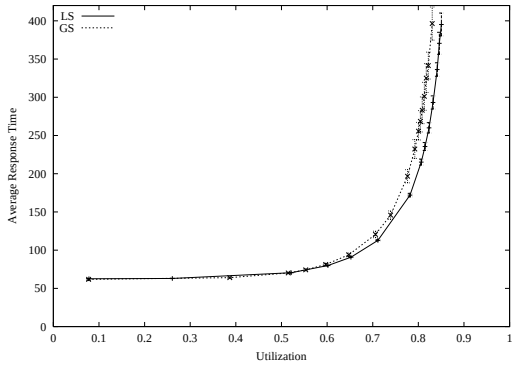

Figure 19. Performance comparison of the scheduling strategies for a job-stream with composition $(100\%, 0\%, 0\%, 0\%)$ and a service time distribution from the DAS

distributions did not alter the results and that our conclusions are valid for systems such as the DAS.

4 Related Work

Not much work has been done related to co-allocating rigid jobs with space sharing in multicluster systems. In [12], a queueing system in which jobs require simultaneous access to multiple resources is studied. The interarrival and service-time distributions are only required to be stationary. Feasible job combinations are defined as the sets of jobs that can be in service simultaneously. A linear-programming problem based on an application of Little's formula for these feasible job combinations is formulated for finding the maximal utilization, regardless of the scheduling pol-

icy employed. In [14], a performance comparison of two meta-schedulers is presented. It is shown that dedicating parts of subsystems to jobs that need co-allocation is not a good idea. In [15], NUMA multiprocessors are split up into processor pools of equal sizes along architectural lines. The number of threads into which a job is split, and the number of pools—the ones with the lowest loads are chosen—across which it is spread—a parallel job incurring more overhead when it spans multiple pools—is controled with parameters. The main result is that using intermediate pool sizes and limiting the number of pools a job is allowed to span yields the lowest response times, as this entails the best locality. In [10], simulations of two offline algorithms for multidimensional bin-packing, a problem that resembles scheduling ordered jobs without communication with deterministic service times, are presented. These algorithms search for items that will reduce the imbalance in the current bin. In order to relate these algorithms to scheduling in multiclusters with deterministic service demands, the algorithms are also simulated for short item lists, with replacement of items before a new bin is started.

5 Conclusions

In this paper we looked at different scheduling policies for co-allocation in multicluster systems and evaluated the performance of the system in terms of response time as a function of the utilization of the system.

Co-allocation with unordered requests is a good choice not only for large jobs, which can get to run faster if split into more components and spread over the clusters, it also deals well with small single-component jobs. For a high percentage of single-component jobs, allowing them to run on any of the clusters, even if scheduled by a single global scheduler, proved to be a better choice than keeping them local to the cluster they were submitted to.

For multi-component jobs, having more schedulers in the system and distributing the jobs among them improves the performance; any of the jobs at the heads of the queues can be chosen to run if it fits, which generates a form of backfilling with a window equal to the number of queues in the system, and increases the utilization.

When there are separate queues for single- and multi-component jobs, favouring the multi-component jobs lowers the performance. In order to improve the system's performance it is good to employ as many processors as possible, so if the job at the head of the global queue does not fit it is better to try to run jobs from the other queues even if it might delay that job, than to wait for enough free processors for it.

If single-component jobs are restricted to one cluster, it is better to try to place them first and when they do not fit to try to schedule multi-component jobs since their components can be shuffled (unordered requests) and there is a higher chance for them to fit this way, than to fit the same set of jobs starting with the multi-component ones.

Considering at one extreme a system with one global scheduler which manages all the jobs using co-allocation over the entire system, and at the other a system with a local scheduler for each cluster, where the schedulers have no global information and only provide resources from the cluster they are associated to, we choose for a combination of the two.

Our results show that from all the strategies we considered the best is to have more schedulers (for example one for each cluster), and to drop the requirement of keeping single-component jobs local. As long as we treat all jobs the same and we do not know the composition of the job stream, there is no reason to separate single- and multi-component jobs in different queues and it is better to distribute jobs evenly among queues. Our choice would be for the LS without restricting jobs to the local clusters, since this strategy is both simple and brings good performance. However, we might expect that if the clusters have different owners LP or a version of LS that favours the local jobs would be preferred in order to give priority to their own local jobs.

References

[1] *The Distributed ASCI Supercomputer (DAS) site*. `http://www.cs.vu.nl/das`.

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

[3] K. Aida, H. Kasahara, and S. Narita. Job Scheduling Scheme for Pure Space Sharing Among Rigid Jobs. In D.G. Feitelson and L. Rudolph, editors, *4th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 98–121. Springer-Verlag, 1998.

[4] H.E. Bal, A. Plaat, M.G. Bakker, P. Dozy, and R.F.H. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. In *Proc. of the 12th International Parallel Processing Symposium*, pages 784–790, 1998.

[5] K. Czajkowski, I. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. In *8th IEEE Int'l Symp. on High Perf. Distrib. Comp.*, pages 219–228, 1999.

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

[7] D.G. Feitelson and L. Rudolph. Theory and Practice in Parallel Job Scheduling. In D.G. Feitelson and

L. Rudolph, editors, *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291, pages 1–34. Springer-Verlag, 1997.

[8] I. Foster and C. Kesselman (eds). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[9] T. Kielmann, R.F.H. Hofman, H.E. Bal, A. Plaat, and R.A.F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 131–140, 1999.

[10] W. Leinberger, G. Karypis, and V. Kumar. Multi-capacity Bin Packing Algorithms with Applications to Job Scheduling under Multiple Constraints. In *Int'l Conf. on Parallel Processing*, pages 404–412, 1999.

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

[12] K.J. Omahen. Capacity Bounds for Multiresource Queues. *J. of the ACM*, 24:646–663, 1977.

[13] A. Plaat, H.E. Bal, R.F.H. Hofman, and T. Kielmann. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. *Future Generation Computer Systems*, 17:769–782, 2001.

[14] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-Scheduling. In D.G. Feitelson and L. Rudolph, editors, *6th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *LNCS*, pages 137–153. Springer-Verlag, 2000.

[15] S. Zhou and T. Brecht. Processor Pool-Based Scheduling for Large-Scale NUMA Multiprocessors. In *ACM Sigmetrics '91*, pages 133–142, 1991.