Priorities among multiple queues for processor co-allocation in multicluster systems

Priorities among Multiple Queues for Processor Co-Allocation in Multicluster Systems

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

Abstract

In systems consisting of multiple clusters of processors which 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 order to handle both single-cluster (local) jobs and multi-cluster (global) jobs, such systems may have only local schedulers (which then need to be aware of the whole system), or only a single global scheduler, or both, and each scheduler has its own queue. In this paper we assess with simulations the response times of both local and global jobs in multi-cluster systems for different configurations of queues, for different priority orders in which the associated schedulers are allowed to schedule jobs, and for different job-stream compositions.*

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 [3, 13] enables applications to access resources in different and possibly widely dispersed locations simultaneously—that is, to employ processor *co-allocation* [8]—to accomplish their goals, effectively creating single multicluster systems. In such systems, jobs may be submitted with numbers of components that vary between one and the number of clusters. Whereas single-component jobs may still be handled by local cluster schedulers, for the multi-component ones, either a separate global scheduler has to be introduced (which may then also deal with single-component jobs), or the local schedulers have to be made aware of the whole multicluster system. In this paper we assess with simulations the average response time of both single- and multi-component jobs in multicluster systems for different configurations of queues, for different priority orders in which the associated schedulers are allowed to schedule jobs, and for several ways in which jobs with different numbers of components are distributed among the queues.

Most of the research on processor scheduling in parallel computer systems has been dedicated to multiprocessors and single-cluster systems (see, e.g., [12]), but hardly any attention has been devoted to multicluster systems. Two important elements of co-allocation that are absent when scheduling jobs in single clusters are the structure of jobs and the way they are spread across the clusters, and the number of schedulers in the system and how they interfere. Of course, using co-allocation does not mean that all jobs have to be split up into components and spread over the clusters, small jobs can still go to a single cluster. In this paper we consider what we call *unordered* job requests: Jobs specify the numbers of processors they need (exclusively) in the separate clusters—so we consider space sharing for rigid jobs [4]—but they are indifferent as to the clusters in which these numbers of processors are obtained. In general, there is in the system a mix of jobs with different numbers of job components.

We design six scheduling policies, some of which have several versions: one with only a global queue, one with only local queues, and four with both, in which case single-component jobs go to the local queues and multi-component ones to the global queue. An important conclusion is that in the latter case, which will be the most common in practice, the best performance is often obtained by giving priority to the local queues, but in a restricted way taking care that global jobs do get some chance to run.

Other important factors influencing the performance of

[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.

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

co-allocation in multiclusters which we studied previously are the job structure and sizes, and the sizes of the clusters in the system [6], and the ratio of the speeds of local and wide-area communications [7]. Also in [9], co-allocation (called multi-site computing there) is studied, with as performance metric the (average weighted) response time. There, jobs only specify a total number of processors, and are split up across the clusters. The slow wide-area communication is accounted for by a factor r by which the total execution times are multiplied. Co-allocation is compared to keeping jobs local and to only sharing load among the clusters, assuming that all jobs fit in a single cluster. One of the most important findings is that for r less than or equal to 1.25, it pays to use co-allocation.

Our five-cluster second-generation Distributed ASCI Supercomputer (DAS) [1, 10] (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 [5, 14, 16]. 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.

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 [2, 10] 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.

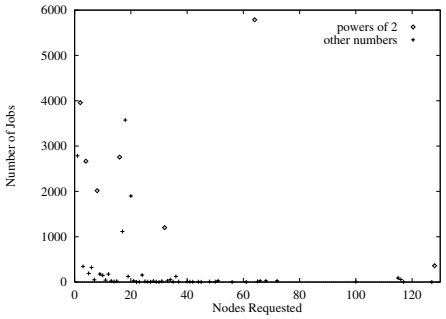

Figure 1. The density of the job-request sizes for the largest DAS cluster (128 processors)

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.

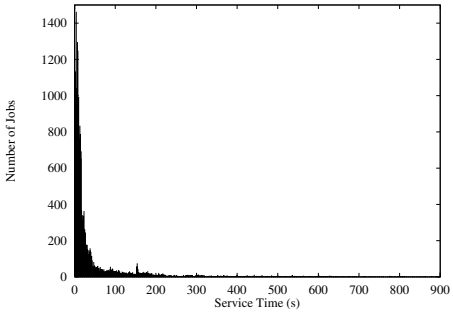

Figure 2. The density of the service times for the largest DAS cluster (128 processors)

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 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 [11]. 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) 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. 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.

   Depending on the order in which the queues are enabled at job departures we define four variations of LS.

   [LS-OR] At each job departure all the queues are enabled in a fixed order, starting with the same queue.

   [LS-RD] At each job departure all the queues are enabled, in a fixed order, starting with a queue randomly chosen. All queues have the same probability to be enabled first.

   [LS-RO] At each job departure all the queues are enabled, in the same order in which the processors on the corresponding clusters are released by the departing job (which is the same as the order in which the processors were allocated with WF): the queues associated to the clusters holding larger job components are enabled first. If the job has fewer components than the number of clusters, the queues local to the clusters not holding any component are enabled last.

   [LS-DO] At each job departure the queues are enabled in the same order in which they were disabled.

3. **[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.

4. **[LP]** Both global and local schedulers, but this time 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, depending on the order in which this happens we differentiate the following variations.

[LP-LF] Always, first the local queues are enabled and then the global queue.

[LP-GF] The queues are always enabled starting with the global queue.

[LP-RD] At each job departure, either the global queue is enabled first or all the local queues, with equal probability.

5. **[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.

   When a job departs all queues are enabled and repeatedly visited in a pre-defined order. We consider three different ways in which queues are enabled at departures.

   [EQ-LF] First the local queues are enabled and then the global queue

   [EQ-GF] First the global queue is enabled, followed by the local queues.

   [EQ-RD] At each job departure, either first the local schedulers are enabled and then the global one, or the other way around, both choices occurring with equal probability.

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 queue is enabled if it is longer than all the local queues, otherwise the local queues 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 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 their queues are unbalanced.

For the policies with both local and global schedulers, 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 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.

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. 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

In this section we assess the performance of multicluster systems for the six scheduling policies introduced (Sect. 2.5), depending on the job-stream composition and the way the single-component jobs are spread among the local queues.

The simulations are for a system with 4 clusters of 32 processors each, and the job-component sizes are generated from $D(0.9)$ on the interval $[1, 8]$. The simulation programs were implemented using the CSIM simulation package [15]. For the distribution of service times we use an exponential distribution with mean 1.

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 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 depending on the percentages of jobs with different numbers of components. For all these compositions we consider first that 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 the case when the local queues are unbalanced, one of them receiving 40% and the other three 20% of the jobs submitted locally. All the ten cases which result are presented in Table 1.

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. This is why for the EQ, GP, LP and LQ policies we depict beside the total average response time, the average response times for the local queues and the global queue.

When comparing the bar charts in this section to each other, one must be aware that the displayed results are at different utilizations. In each chart, the utilizations are chosen high enough so that at least one of the policies is close to the maximum utilization.

Table 1. The ten cases considered, depending on the job-stream compositions and the way jobs are spread among the local queues

% 1-comp.	% 2-comp.	% 3-comp.	% 4-comp.	bal.
25	25	25	25	yes
100	0	0	0	yes
50	0	0	50	yes
0	0	0	100	yes
50	25	25	0	yes
0	50	50	0	yes
50	50	0	0	yes
80	0	0	20	yes
90	0	0	10	yes
80	0	0	20	no

3.1 Policies with multiple versions

For the LS, LP and EQ policies the order in which the queues are enabled at job departures has a significant influence on the performance, as the results in this section show. Depending on this order, we defined four versions of LS and three versions of LP and EQ, which are being compared below.

3.1.1 The LS policy

We compare the four versions of LS for nine of the ten cases introduced in Table 1: when all jobs are single-component they are restricted to their corresponding cluster and the order of enabling the queues does not influence the results.

In Fig. 3 the local queues are balanced, while in Fig. 4 they are not. In all the cases with balanced queues the LS-DO version of the policy, where queues are enabled in the order in which they were disabled, displays the best performance. It treats all jobs and queues fairly, keeping in balance the numbers of jobs run from each queue, and since the queues receive equal percentages of the job stream, their lengths stay similar. When after a departure the queues are repeatedly visited, those which get disabled earlier managed to schedule fewer or at most the same number of jobs as the queues disabled later, so enabling them first is a good way to keep the queues balanced.

The worst performance in Fig. 3a - g is shown by LS-OR. Here the queues are enabled each time in the same order, an approach which tends to favour and keep empty the queues visited earlier (especially the first queue), while allowing the queues visited last to grow. When scheduling decisions are taken, there are up to four jobs (when no queue is empty) from which to choose one that fits in the system. The OR version unbalances the lengths of the queues and, emptying the queues visited first, it also reduces the set of jobs among which the system searches for one that fits.

LS-RD displays very good performance in Fig. 3a - h due to the fact that at each departure it randomly chooses the queue to be enabled first, which maintains in general balanced queue lengths. However, it does not take into account the job stream and it can delay large jobs that are hard to fit, decreasing the performance. Unlike LS-RD, LS-DO remembers the jobs that did not fit and enables the corresponding queues in the same order helping this way the large jobs to run: a queue repeatedly disabled because of the same (large) job advances in the visiting order if the other queues manage sooner to schedule their jobs, finally being enabled first at each departure until its job fits. This yields a better queue lengths balance when the DO version is used, and as a consequence a better performance.

In the cases in Fig. 3a - f, the performance of LS-RO is close to that of LS-RD. The RO variation of the policy is good for local jobs: when the load of the system is high and a job releases its processors, there is a good chance that the clusters hosting larger components will have larger numbers of idle processors. Enabling first the queues corresponding to those clusters gives the single-component jobs in the queues a better chance to run, before a multi-component job takes the processors away. This variation of the LS policy looks at the load of the system and not at the lengths of the queues. It assumes that keeping the load of the system balanced keeps the queues balanced as well, and that scheduling first the local jobs, restricted to their cluster, yields better performance since for the global jobs the components can be shuffled.

When local jobs do not represent the majority, LS-RO displays good performance. However, for a high percentage of local jobs, as Fig. 3g, h show, LS-RO is a bad choice. The explanation resides in the way the local and the global jobs interact there: at high loads, a queue that gets disabled with a multi-component job at its top while all the other queues have local jobs that fit has very little chance to have its job scheduled until another multi-cluster job appears in another queue and gets scheduled, or some of the other queues become empty. When the local jobs scheduled from the other queues end, those queues will be visited first, leaving little room for the multi-component job.

Comparing the versions of LS for job-stream compositions with more than 80% local jobs, we find that DO balances the queue lengths (and also the load of the system, since all queues receive the same percentages of jobs with different numbers of components) adapting to the workload, RD balances the load but does not adapt to the jobs in the system, OR keeps the queue lengths unbalanced due to the way queues are visited, and RO causes the worst unbalance for the queues, taking the worst decisions by avoiding exactly the queues which have jobs that do not fit. LS-OR improves for a very low percentage of global jobs (see Fig.

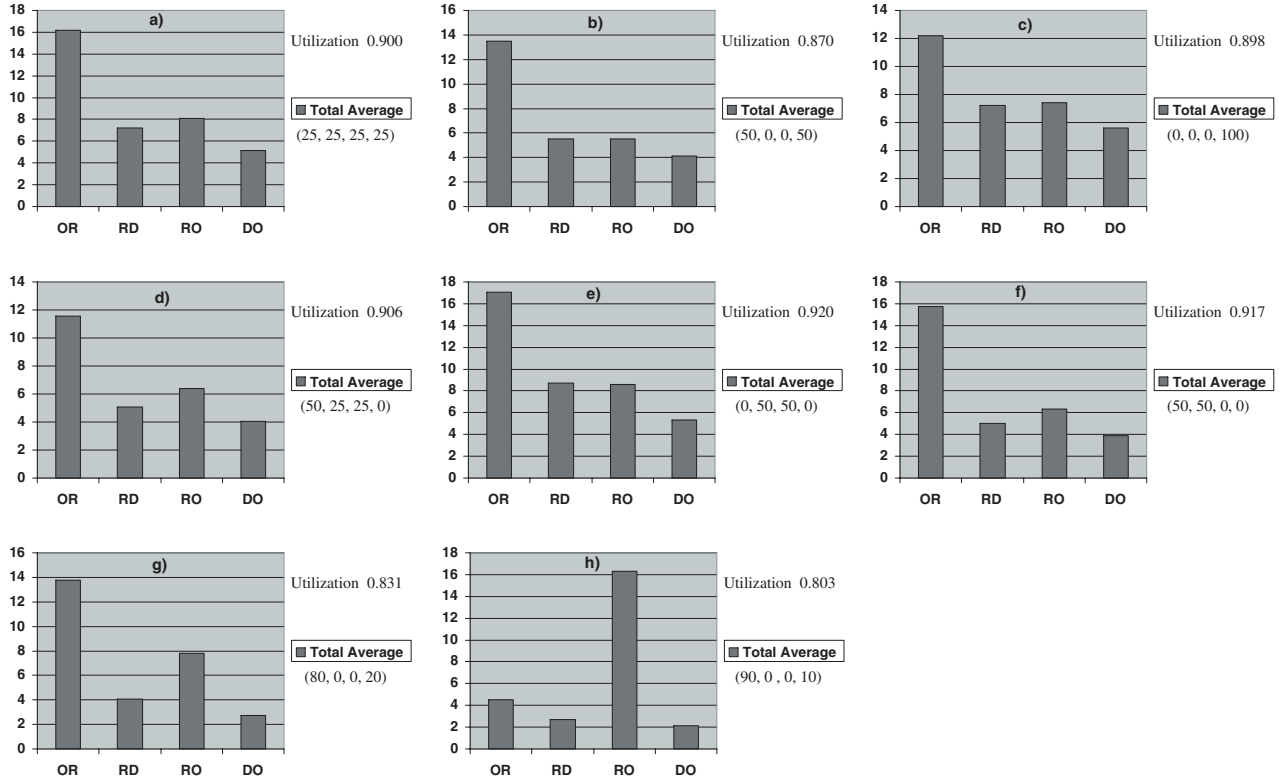

Figure 3. Response times for the four versions of LS, several job-stream compositions, and balanced local queues

3h) because when there are just single-component jobs it does not matter in which order the queues are visited.

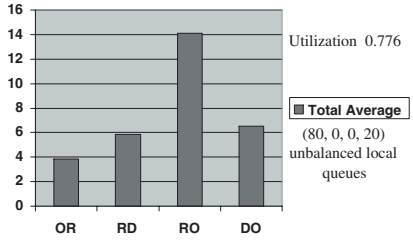

Figure 4. Response times for the four versions of LS, job-stream composition $(80, 0, 0, 20)$ and unbalanced local queues

Figure 4 shows the average response time for the variations of the LS policy when 80% of jobs are single-component and the queues are unbalanced — one queue receives 40% of the jobs arriving to the system. OR displays the best performance because it gives priority exactly to the queue receiving more jobs, visiting it first after each job departure. DO, RD and RO have higher response times because they treat all queues the same, ignoring the fact that one of them receives twice as many jobs and letting the corresponding queue grow. RO has the worst performance due to the high percentage of single-component jobs.

3.1.2 The LP policy

In this section we compare the three versions of the LP policy defined. Since with LP the local queues only get single-component jobs, which are restricted to the local cluster, the relative order in which they are enabled does not matter.

Only seven of the cases considered are relevant here: for 100% single-component jobs there are only local queues, while for 100% multi-component jobs there is only the global queue. In Fig. 5 the local queues are balanced, while in Fig. 6 the unbalanced case is assessed.

All charts in Fig. 5 display the best total performance for LP-GF when at each job departure the global queue is enabled first; also the average response time for the global queue is the smallest for the GF variation of the policy.

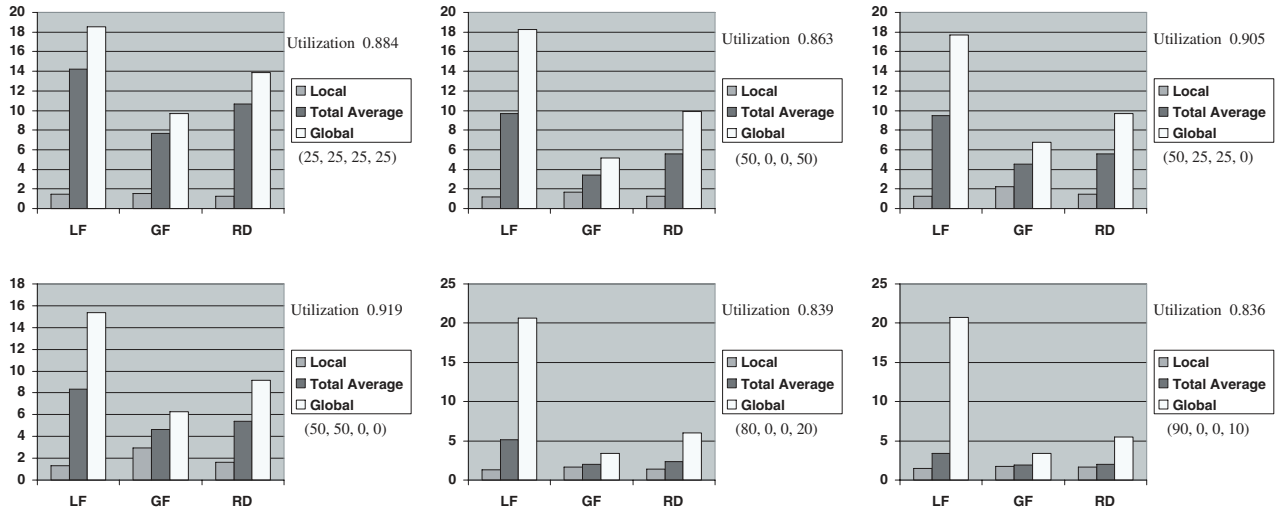

Figure 5. Response times for the three versions of LP, several job-stream compositions, and balanced local queues

Looking at the local queues, we notice that enabling first the global queue deteriorates very little their performance: in most cases there is a very small increase in response time for the local queues compared to LP-LF. On the other hand, for the global queue enabling first the local queues with LP-LF causes a large increase in response time compared to LP-GF.

For LP-GF, the performance of the local queues gets worse compared to LP-LF when the global jobs have fewer components, since they fit better on the system and leave less room for the local jobs (see job-stream composition $(50, 50, 0, 0)$). When there is a high percentage of global jobs and they have many components, enabling the global queue first does not bother much the local jobs and has a good effect on the global jobs.

We can conclude that LP-GF is the best choice for the job-stream compositions in Fig. 5, while having a higher total average response time and a very bad performance for the global jobs makes LP-LF the worst option. In all the cases from the figure, LP-RD has a total performance worse than LP-GF and better than LP-LF, and the average response times for the global and local queues have values situated between those displayed by the GF and LF versions. This is due to the fact that LP-RD randomly chooses at each departure whether to enable first the global queue or the local queues, treating both types of queues equally.

When the local queues are unbalanced (see Fig. 6), LP-LF has the best total performance because always enabling the local queues first allows the queue with a higher load to fit its jobs without being bothered by the multi-component jobs from the global queue. No other choice can improve

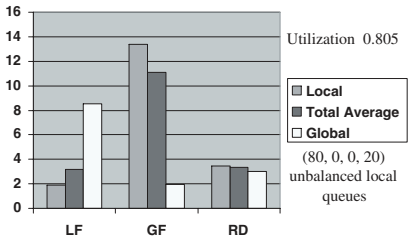

Figure 6. Response times for the three versions of LP, job-stream composition $(80, 0, 0, 20)$ and unbalanced local queues

the situation for the most loaded queue as long as local jobs are restricted to the local cluster. On the negative side, LP-LF has a bad performance for the global queue. Here, 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 are interested to have 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 since its total performance and average response time for the local queues are not much worse than those of LP-LF, and it gives a much lower response time for the global jobs.

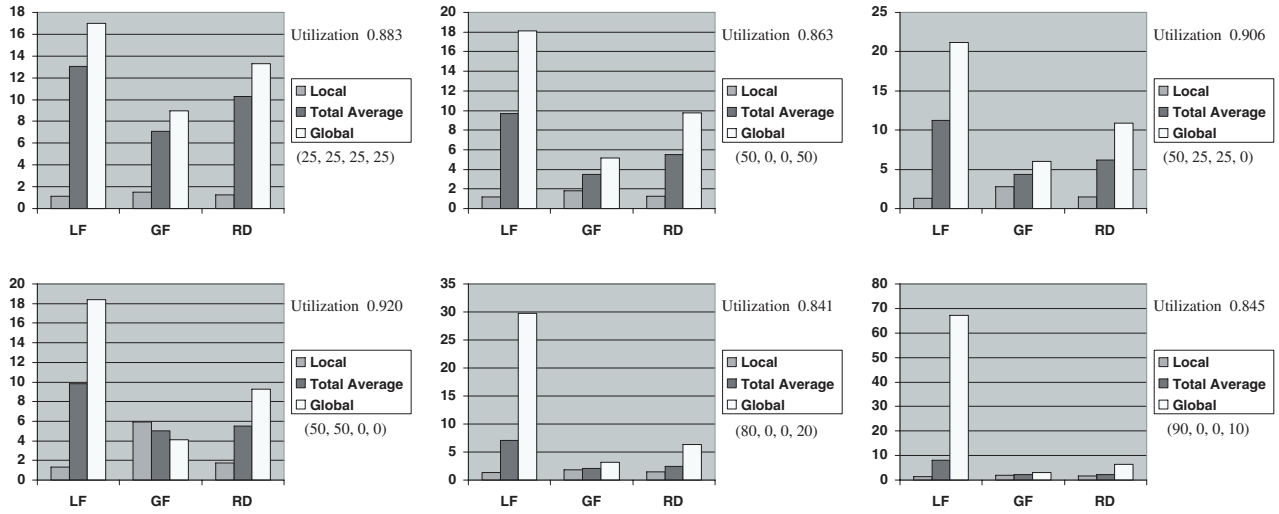

Figure 7. Response times for the three versions of EQ, several job-stream compositions, and balanced local queues

3.1.3 The EQ policy

Like with the LP policy, for the EQ policy we defined three different versions; also in this case local queues only get single-component jobs and the order in which these queues are enabled does not matter. Again, only seven of the cases are relevant since for 100% single-component jobs there are only local queues, while when all jobs are multi-component there is only the global queue. In the charts in Fig. 7 the local queues are balanced, while in Fig. 8 they are not.

Similar to the LP policy, all the charts in Fig. 7 display the best total performance for the GF version of the EQ policy; also the average response time for the global queue is the smallest for EQ-GF. In most cases, enabling first the global queue deteriorates very little the response time of the local queues compared to EQ-LF, while for the global queue enabling first the local queues with EQ-LF, causes a large increase in response time.

For EQ-GF, the average response time of the local queues gets worse compared to EQ-LF when the global jobs have fewer components, since they fit better on the system and leave less room for the local jobs (see composition $(50, 50, 0, 0)$). Compared to LP, for EQ this deterioration is significantly larger. While even the GF version of LP gives priority to local jobs, EQ-GF does not. Here, when the global jobs have few components the response time is lower for the global queue than for the local queues, which can be explained by the fact that the local jobs are restricted to their clusters and global jobs can be scheduled on any clusters where they fit. For a high percentage of global jobs with many components, enabling the global queue first does not bother much the local jobs and has a very good effect on the global jobs. In such cases EQ-GF is the best option, while EQ-LF which has a higher total average response time, and a very bad performance for the global jobs is the worst.

Similarly as for LP, in all these cases EQ-RD has a total performance worse than EQ-GF and better than EQ-LF, and the average response times for the global and the local queues have values situated between those displayed by the GF and LF versions. For the job-stream composition $(50, 50, 0, 0)$ EQ-RD can be the most appropriate choice because its total performance is only slightly worse than that of EQ-GF, it is almost as good as EQ-LF for the local queues and much better for the global queue.

Similarly to LS, for unbalanced local queues EQ-LF has the best performance. On the other hand, EQ-LF has a rather bad performance for the global queue. Here, EQ-GF provides a much worse total performance and a very high average response time for the local queues. EQ-RD has a slightly worse total performance than EQ-LF and slightly higher response time for the local queues, but a much lower response time for the global queue. If we look for a good total performance or a low response time for the local jobs EQ-LF is the best option for this case, but if we also want a low response time for the global queue, EQ-RD should be chosen.

3.2 Performance comparison of the policies

Figures 9 — 18 compare the average response time for the six policies and the ten cases considered. When a policy has more versions, the one with the best total performance is depicted, for each job-stream composition. For Figs. 9 —

14 LS-DO, LP-GF and EQ-GF were chosen, while in Fig. 18 LS-OR, LP-LF and EQ-LF are represented.

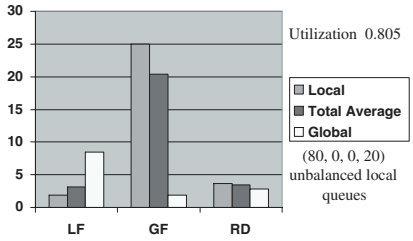

Figure 8. Response times for the three versions of EQ, job-stream composition $(80, 0, 0, 20)$ and unbalanced local queues

3.2.1 Dealing with both single- and multi-component jobs

For all the cases in this section we depict the response times at two utilization values; at the second value the system is already saturated for some of the policies (indicated by SAT).

Figure 9 compares the policies for a job stream containing 1-, 2-, 3- and 4-component jobs in equal proportions, at two utilization values. 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 the entire system. At any moment, the system tries to schedule up to four jobs (when 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 the other policies. 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 GS can choose from the four clusters one where the job fits. However, in the case in Fig. 9, 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. The average response time for the global queue is the best from all the policies, but the average response time for the local queues is much worse than for the

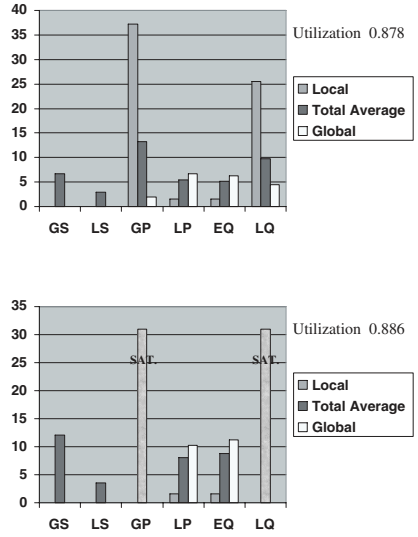

Figure 9. Response times for the scheduling policies for job-stream composition $(25, 25, 25, 25)$ and balanced local queues

other policies. Since most of the jobs are multi-component, the global queue is the longest in most of the cases when a scheduling decision has to be taken and LQ behaves similarly to GP, its performance being the second worst. For the utilization value in the second chart, the system is saturated for both GS and LQ.

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. LP has a slightly better total average, and although it favours the local queues enabling the global queue only when at least one local queue is empty, it also has a better average response time for the global queue than EQ. Since there are few local jobs in the system, LP favouring them does not delay the multi-component jobs, on the contrary, imposing an order among queues and not randomly mixing jobs from the local and global queues allows jobs to fit better.

Figures 10, 11 and 12 show that for GP the performance improves with the decrease of the percentages of jobs with 3 and 4 components: the local, global and total average response times for GP are smaller in Fig 12. 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

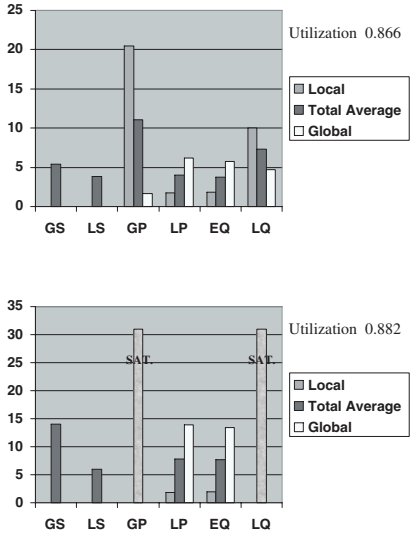

Figure 10. Response times for the scheduling policies for job-stream composition $(50, 0, 0, 50)$ and balanced local queues

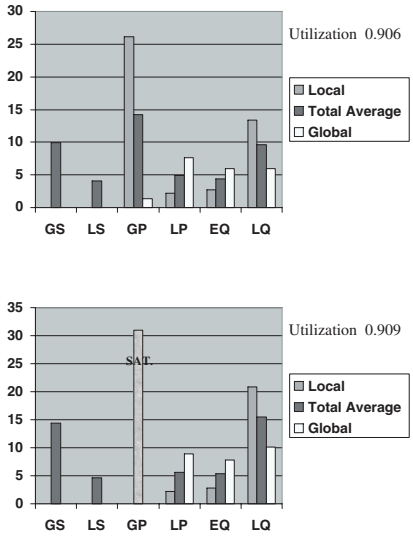

Figure 11. Response times for the scheduling policies for job-stream composition $(50, 25, 25, 0)$ and balanced local queues

LQ has worse performance when the percentage of multi-component jobs is higher.

The best performance in Figs. 10 and 11 is displayed by LP and EQ. 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, is a good choice when there are many jobs with 3 and 4 components. It also seems that when none of the local queues is empty, delaying the global jobs to wait for the local jobs to fit does not deteriorate much the performance of LP when at most 50% of jobs are local; this choice is an advantage for LS when the percentage of local jobs is smaller (see Fig. 9).

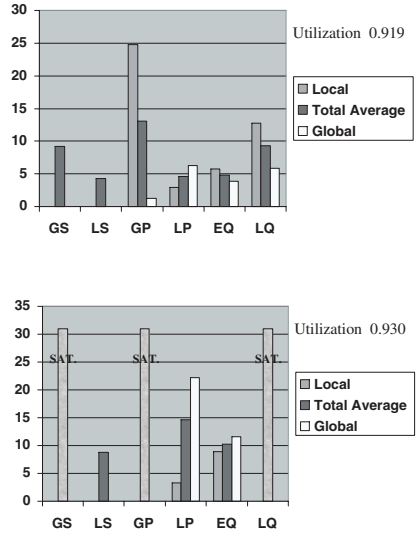

Figure 12. Response times for the scheduling policies for job-stream composition $(50, 50, 0, 0)$ and balanced local queues

The differences in performance are larger in Fig. 10 where there are 50% 4-component jobs. In Figs. 11 and 12, where there are no 4-component jobs, all policies display more similar performance.

EQ has a good performance for all chosen job mixes because it tries to fit as many jobs as possible from all queues without taking into account the characteristics of the job stream. Favouring the multi-component jobs by enabling the global queue first at job departures also has a positive influence on performance. In Fig. 12 EQ is slightly better than LP due to the fact that there are many 1-component jobs and the 2-component jobs fit very well on the system.

Also when there is a higher percentage of local jobs (see Figs. 13 and 14) EQ displays the best performance; here LP is worse because for 80% and 90% local jobs the policy significantly delays the global jobs. GS also shows good results for a high percentage of single-component jobs due to the fact that it does not restrict the local jobs to the cor-

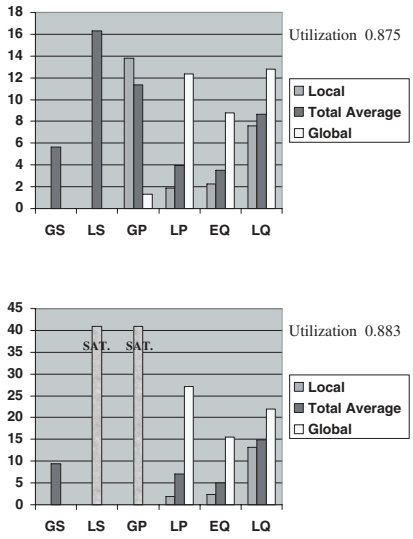

Figure 13. Response times for the scheduling policies for job-stream composition $(80, 0, 0, 20)$ and balanced local queues

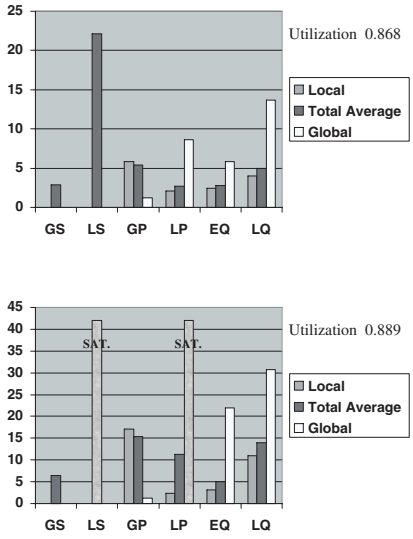

Figure 14. Response times for the scheduling policies for job-stream composition $(90, 0, 0, 10)$ and balanced local queues

responding cluster; at 90% local jobs its performance approaches that of EQ. LQ has a rather good performance in these two cases, balancing the lengths of the local and global queues.

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).

3.2.2 Dealing with only single- or multi-component jobs

Figures 15, 16 and 17 compare only the GS and LS strategies. The system in Fig. 15 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.

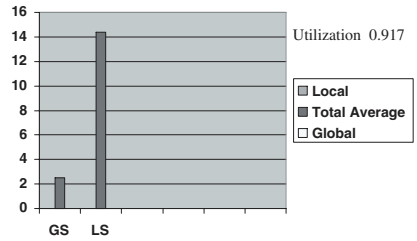

Figure 15. Response times for the scheduling policies for job-stream composition $(100, 0, 0, 0)$ and balanced local queues

When there are only single-component jobs in the system (Fig. 15), 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 while LS keeps the clusters in isolation.

In Figures 16 and 17 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.

3.2.3 The unbalanced case

For the policies defining local queues, Fig. 18 compares the performance for a job-stream composition with 80% local jobs and unbalanced local queues (40% of the local jobs go to one queue).

Although LS with its OR version favours the local queue receiving the highest percentage of jobs, always enabling it first at job departures, its performance is worse than that of EQ and LP. Due to the separate global queue, EQ and

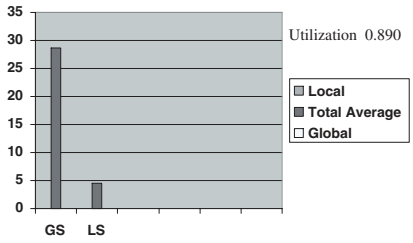

Figure 16. Response times for the scheduling policies for job-stream composition $(0, 0, 0, 100)$ and balanced local queues

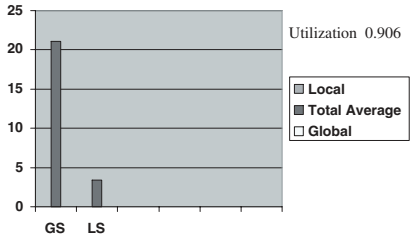

Figure 17. Response times for the scheduling policies for job-stream composition $(0, 50, 50, 0)$ and balanced local queues

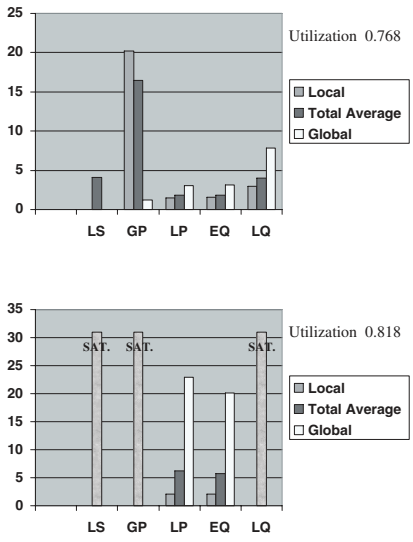

Figure 18. Response times for the scheduling policies for job-stream composition $(80, 0, 0, 20)$ and unbalanced local queues

LP look at the top of five queues for a job that fits, compared to only four queues in the case of LS. Both LP and EQ favour the local queues at job departures (the LF versions) but LP gives slightly worse results because it delays more the multi-component jobs by not letting them run unless at least one local queue is empty.

3.3 Local versus global queues

In this section we discuss the performance of global and local jobs when there are both global and local queues, i.e., for LP, EQ, GP and LQ.

Considering the separate results for local and global queues, we notice that while LP provides the best results for local jobs, GP is the best for the global jobs. LP and EQ display a low response time for both the local and the global queues, with EQ showing better performance than LP for the global queue, while LP is better for the local queues.

If there is a high percentage of multi-component jobs LQ yields a smaller average response time for the global queue, while when there is a high percentage of single-component jobs it provides better response time for the local queues. LQ is 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 will LQ give that job a chance to run 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 lengths of the queues balanced, switching its behaviour between GP and LP depending on the queue lengths. 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.

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 and a job departs, the global scheduler is enabled first (LP-GF). 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. This dependency is even stronger for EQ.

From these four policies the most practical would be either LP or EQ, since the other two 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. In most of the cases, for both EQ and LP the GF version gave better results: at job departures, enabling first the global queue improves the total average response time and has little influence on the local 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, 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.

4 Conclusions

In this paper we evaluated different scheduling policies for co-allocation, with unordered requests, in multicluster systems.

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.

When dealing with unbalanced local queues, the scheduling policy should give priority to the local queues if there are both local queues and a global queue in the system. When there are only local queues (the global jobs are also submitted locally) the queue receiving a higher percentage of jobs should be favoured by the policy.

We might expect that if the clusters have different owners a policy that favours the local jobs would be preferred. For this reason, although EQ-GF gives in most of the cases with balanced queues slightly better results, a version of LP can be more suited. We can then choose LP-GF because although at job departures it favours the global queue, we have shown that this does not significantly worsen the performance for the local jobs, while it is better for both the total performance and the average response time for the global jobs. Choosing LP-LF instead would bring too little improvement for the local jobs to make the loss in total performance and in the global jobs' performance worthwhile.

References

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

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

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

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

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

[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] 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.

[9] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On Advantages of Grid Computing for Parallel Job Scheduling. In *Cluster Computing and the GRID, 2nd IEEE/ACM Int'l Symposium CCGRID2002*, pages 39–46, 2002.

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

[11] D. Feitelson and L. Rudolph. Theory and Practice in Parallel Job Scheduling. In D. Feitelson and L. Rudolph, editors, *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291, pages 1–34. Springer-Verlag, 1997.

[12] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In D. Feitelson and L. Rudolph, editors, *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *LNCS*, pages 1–34. Springer-Verlag, 1997.

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

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

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

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