The influence of the structure and sizes of jobs on the performance of co-allocation

The Influence of the Structure and Sizes of Jobs on the Performance of Co-allocation

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

Parallel and Distributed Systems Group
Faculty of Information Technology and Systems
Delft University of Technology, P.O. Box 356, 2600 AJ Delft, The Netherlands
anca@pds.twi.tudelft.nl, epema@pds.twi.tudelft.nl

Abstract. Over the last decade, much research in the area of scheduling has concentrated on single cluster systems. Less attention has been paid to multicluster systems, although they are gaining more and more importance in practice. We propose a model for scheduling rigid jobs consisting of multiple components in multicluster systems by pure space sharing, based on the Distributed ASCI Supercomputer. Using simulations, we asses the influence of the structure and sizes of the jobs on the system's performance, measured in terms of the average response time and the maximum utilization. We consider three types of requests, total requests, unordered requests and ordered requests, and compare their effect on the system's performance for two scheduling policies, First Come First Served, and Fit Processors First Served, which allows the scheduler to look further in the queue for jobs that fit. These types of job requests are differentiated by the restrictions they impose on the scheduler and by the form of co-allocation used. The results show that the performance improves with decreasing average job size and when fewer restrictions are imposed on the scheduler.

1 Introduction

Much work has been done in the area of scheduling in parallel computer systems, but most of it is related to single-cluster systems (i.e., single multiprocessors and single clusters of uniprocessors) with identical processors connected by links of the same speed. Much less research has been dedicated to multicluster systems, although many such systems are in use. We study the performance of co-allocation, that is of scheduling jobs that can be spread on more than one cluster, in multicluster systems such as the Distributed ASCI Supercomputer (DAS) [5], depending on the structure and size of jobs and on the scheduling policy. In particular, we provide a performance comparison between a single-cluster system and a multicluster system of the same total size.

Multiprocessor systems gained popularity during the last years. The increase in the computational power of the existing sytems encouraged people to design larger and larger parallel applications. The sequential solutions to many problems were replaced by parallel ones in order to become faster.

Being expensive, large parallel systems are in general not dedicated, but shared by large numbers of applications designed by many different users. For this reason, many scheduling strategies have been developed, giving solutions for how applications should

D.G. Feitelson and L. Rudolph (Eds.): JSSPP 2000, LNCS 1911, pp. 154–173, 2000.

be structured, how they should be chosen for being processed, and how they should be mapped to the resources.

One of the simplest ways to solve these problems, and yet often used in practice due to its simplicity, is to allow only rigid jobs, scheduled by pure space sharing. This means that at execution, each job requires some number of processors and is executed on those processors until its completion. The advantages are that the implementation of the application is not restricted by the system, so the users can design their applications the way they consider the best performance is obtained, and that each application can have exclusive access to and control of the assigned processors. There is also the economical advantage that the providers of service can easily and fairly charge the users for the employed resources.

Compared to single-cluster systems, multicluster systems can provide a larger computational power (more nodes). They can be geographically spread, and instead of smaller groups of users with exclusive access to single clusters, larger groups of users can share the multicluster consisting of the total of the initial single clusters. Of course, the fact that the resources are spread will entail that the connections between clusters will be slower than the ones inside the clusters. Another reason for building multicluster systems is that very large single-cluster systems are hard to manage by single schedulers.

The necessity of dividing the processors into pools in order to simplify the scheduling decisions is discussed in the literature [3]. In the case of multiclusters, the division is natural and is imposed by the architecture of the system, and not by the scheduler. The nodes cannot be treated as identical anymore because their relative position inside the clusters influences the performance of the communications between them, and this must be taken into account by the application and the scheduler.

This paper concentrates on real multicluster systems such as the DAS. We provide a model of the system and study by simulations the influence of the structure and size of the jobs on the performance of the multicluster system. We also take into account the effect of the scheduling policy on the system's performance, being aware of the modifications the policy brings to the order of the requests. The scheduling schemes implemented are First Come First Served and Fit Processors First Served, which can look further in the queue for jobs that fit. The scheduler provides co-allocation, meaning that a job can ask for the simultaneous allocation of processors in multiple clusters. The performance is measured in terms of the maximal utilization and of the response time as a function of the system's utilization.

2 The Model

Our model is a simplification of a real system, a multicluster distributed system called the Distributed ASCI Supercomputer, the performance of which we intend to evaluate, depending on the scheduling scheme and on the structure and the distribution of the requests of the incoming jobs.

2.1 The Structure of the System

We model a multicluster distributed system consisting of C clusters of processors, cluster i having N_i processors, $i = 1, \ldots, C$. The system has a single central scheduler, with one global queue (see Fig. 1).

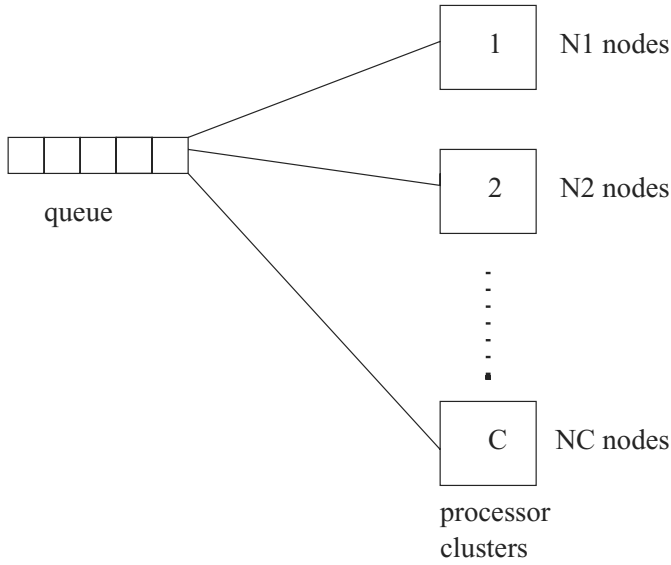

Fig. 1. The model of a multicluster system.

We assume that all processors have the same service rate. For both interarrival times and service times we use exponential distributions.

By job we understand a parallel application requiring some number of processors. A job can simultaneously ask for processors in more than one cluster (co-allocation). We will call a task the part of an application which runs on a single processor. Jobs are rigid, meaning that the numbers of processors requested by and allocated to a job are fixed, and cannot be changed during its execution. All tasks start and end at the same time, which implies that all the processors allocated to a job are being simultaneously occupied and released. Preemption is not admitted, nodes being released only when the tasks running on them end. We also assume that jobs only request processors and we do not include in the model any other types of resources.

In our simulations we make the simplification that all the clusters have an equal number N of processors. Clusters of different sizes would not change the results significantly, but would make them harder to be evaluated. Besides, factors of another nature, such as the users' preference for the larger clusters, would become relevant. When $C = 1$, the system is a single cluster. We compare the performance of the multicluster system with a single-cluster system with CN processors.

2.2 The Structure of Jobs

We consider three cases for the structure of jobs, differentiated by the structure of the system and of the job's request:

1. A request of a job is represented by a tuple of C values $(r_1, r_2, \ldots, r_C)$, each of them uniformly distributed on the interval $[n_1, n_2]$, with $0 < n_1 \leq n_2 \leq N$. By these values, the job only specifies how many nodes it needs in separate clusters, but not the precise clusters where the nodes must be allocated. We will further call this type of request "unordered request".
2. The request is again given by a tuple of C values $(r_1, r_2, \ldots, r_C)$, each uniformly distributed on the interval $[n_1, n_2]$, with $0 < n_1 \leq n_2 \leq N$, but here their positions in the tuple specifies the clusters from which the processors must be allocated. This will be called an "ordered request".
3. Here, there is only a single cluster with size CN, and a request only specifies the single number of processors it requires. An instance of this case is characterized by a number of clusters C and an interval $[n_1, n_2]$. The distribution of the numbers of processors required by jobs is the sum of C copies of the uniform distribution on the interval $[n_1, n_2]$. In this case, the requests are called "total requests". We include this case in order to compare the ordered and unordered multicluster cases above with a single-cluster case in which the job sizes have the same distribution.

As long as we do not take into account the characteristics of the applications (e.g., the amount of communication between processors), the case of total requests amounts to the same as would a case with a multicluster when the requests are given as single values and the users do not impose restrictions on the clusters they will receive processors from. In order to be able to compare the results, we choose the intervals for the uniform distributions in such a way as to have equal mean values for the request sizes, in all three cases. Ordered requests are used in practice when the user has enough information about the system, to take full advantage of the characteristics of the different clusters, for example of the data availability. Unordered requests (especially the case when grouping request components on the same cluster is allowed) model applications like FFT, where tasks in the same request component share data and need intensive communication, while tasks from different components exchange little or no information.

We also did simulations for the situation when requests are unordered and the scheduler tries to group as many components as possible on the same cluster. For the value ranges we chose the results were not much different from the case when only distinct clusters are used. However, in general this choice can much influence the performance (see also Sect. 3).

2.3 The Scheduling Policies

To observe the contribution of the scheduling scheme to the system's performance, apart from the First Come First Served (FCFS) policy, the Fit Processors First Served (FPFS) policy explained below was implemented as well.

For ordered and total requests, it is clear when a job fits on the system, either the total number of processors requested are idle, or all job components fit in the clusters they request.

When requests are unordered, for both FCFS and FPFS, the algorithm that checks whether a job fits first orders the values inside the request, and then tries to schedule them in decreasing order of their size. This ensures the maximum success for the individual job, whatever way of placement such as First Fit, Best Fit or Worst Fit, is chosen (if our placement would not succeed, no other one would—the request cannot be served for that configuration). Clusters are checked in the same order each time and the components of the request are placed into the clusters in decreasing order of their size, in the First Fit manner.

We do not consider the influence each placement has on the jobs following in the queue, although it can affect performance, especially for the FCFS policy (a placement according to Worst Fit could give better results than First Fit because it would leave in each cluster as much room as possible for the next job). Our focus is on the influence of the structure and sizes of the requests on the system's performance and less on the impact of the scheduling schemes. FCFS is the simplest scheduling scheme, processors being allocated to the job at the head of the queue. When the job at the head of the queue does not fit, the scheduler is not allowed to choose another job, further in the queue. Because of this restriction, FCFS results in a low maximal processor utilization.

In FPFS the scheduler searches further in the queue, from head to tail, and schedules the jobs which fit. It is similar to the backfilling policy [1], but the duration of jobs is not taken into account, so the requirement that the job at the head of the queue should not be delayed is not enforced. In order to avoid starvation (a job is never scheduled) we introduce counters as aging mechanism. Each job counts the number of times it was jumped over by jobs which were behind it in the queue, but were scheduled before it. When a job's counter reaches a chosen limit MaxJumps, the scheduler is not allowed to overpass that job anymore. In this way, the effectiveness of scheduling is preserved. Of course, when MaxJumps is equal to zero, FPFS becomes FCFS. FPFS has the potential advantage of an increased maximal utilization of the system compared to FCFS.

2.4 The Distributed ASCI Supercomputer

The DAS [5] is a wide-area distributed computer, consisting of four clusters of workstations located at four Dutch universities, amongst which Delft. One of the clusters contains 128 nodes, the other three contain 24 nodes each. All the nodes are identical Pentium Pro processors. The clusters are interconnected by ATM links for wide-area communications, and for local communication inside the clusters Myrinet LANs are used. The operating system employed is RedHat Linux. The system was designed by the Advanced School for Computing and Imaging (ASCI, in the Netherlands) and is used for research on parallel and distributed computing. On single DAS clusters a local scheduler called prun is used; it 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 (15 minutes).

Using the Globus toolkit [10] which we installed in the DAS system, a job can simultaneously and transparently require processors on distinct clusters. However, this form of co-allocation has not been used enough so far to let us obtain statistics on the sizes of the jobs' components.

3 The Maximal Utilization

In the model described in Sect. 2, it may happen that some processors are idle while at the same time there are waiting jobs. Of course, this phenomenon already occurs in single clusters, but in multiclusters we can expect it to occur more often or with a larger fraction of the processors remaining idle. As a consequence, if ρ_m is the traffic intensity such that the system is stable (unstable) at traffic intensities ρ with $\rho < \rho_m$ ($\rho > \rho_m$), we have $\rho_m < 1$. We will call the quantity $1 - \rho_m$ the (maximal) *capacity loss*, which we denote by L.

In this section we first discuss some important reasons for capacity loss, and then we present a very simple approximation for the capacity loss in single-cluster systems. We validate this approximation with simulations when the job sizes have a uniform or a (truncated) geometric distribution. Finally, we assess the capacity loss in multiclusters with simulations.

3.1 Reasons for Capacity Loss

The problem of unutilized processor capacity when space sharing is employed for rigid jobs in single clusters has of course been recognized before. In [8], gang scheduling is proposed as a solution to this problem. However, for multiclusters such as the DAS, gang scheduling may not be a viable solution for technical reasons and because of the distributed ownership of such systems. Even if the cluster schedulers support gang scheduling (and our local DAS scheduler does not), the separate cluster schedulers would have to synchronize and agree on the number and size of the time slices, and on the jobs that run in each time slice. Because of the wide-area latencies, the context-switching overhead will be larger than in single clusters. Clusters in a multicluster may have different systems administrators or different owners, who want to determine for themselves how their own systems are used. Setting aside some number of processors for some amount of time for (components of) foreign jobs (i.e., space sharing) does interfere much less with local jobs than gang scheduling.

There are at least three reasons for the phenomenon of capacity loss. First, it may be due to the job-size distribution, and, in multiclusters, to the jobs' structures. For instance, when in a single cluster with N processors all jobs have size $\lceil (N+1)/2 \rceil$, for large values of N the capacity loss L is close to 0.5. In multicluster systems in which ordered requests are used, much higher fractions of the capacity may be lost if many jobs have mutually conflicting requirements in one specific cluster while they do not require many processors in the remaining clusters.

Second, the scheduling policy employed may cause capacity loss. It is possible that the job at the head of the queue does not fit, while some job further down in the queue does fit, which means that the capacity loss when FCFS is employed is larger than when a policy like FPFS is used instead. In the case of total requests, this only occurs when not enough processors are idle, while in the cases of unordered and ordered requests, even when the total number of idle processors is large enough, a job may still not be accommodated because its components do not fit in the separate clusters.

A third reason for having $\rho_m < 1$ is that we are considering an *on-line* problem, which means that we take scheduling decisions without knowing when jobs will arrive

in the (near) future and what their sizes and service times are. We may expect that in multiclusters, having knowledge of the structure of jobs and the sizes of their components would be even more important to reduce capacity loss.

3.2 An Approximation of Capacity Loss in Single Clusters

We now present a procedure for computing an approximation to L in a single cluster of size N for the FCFS policy. Subsequently, we simplify this approximation, and show this simplification to yield good results for different job-size distributions. We assume that there is no correlation between the job sizes (number of processors requested) and the job service times.

When in a single cluster the traffic intensity approaches ρ_m, the job queue will be very long. This means that we can assume that whenever a job leaves the system, new jobs from the head of the queue can be started until the next job does not fit. So in fact, we can find an approximation to L by computing the average number of processors that remain idle when we put jobs one by one on a single cluster of N processors that is initially completely idle until the next job does not fit. If there was a correlation between job size and service time—for instance, when larger jobs run for a longer period of time, as has been observed in some systems [9]—the mix of the sizes of the jobs in service would be different from the general job-size distribution, and the approximation would in general not be valid.

To find the approximation, let N be the number of processors, and let F be the job-size distribution, which is a discrete distribution on the set $\{1, 2, \ldots, N\}$; $F(n)$ is the probability that a job's size does not exceed n. We assume F to be non-degenerate, for otherwise, if all jobs are of size say d, we have $L = (N \bmod d)/N$. Let f be the density of the job sizes, so $f(n) = F(n) - F(n-1), n = 1, 2, \ldots, N$, is the probability that a job is of size n. For an N-tuple $v = (v_1, \ldots, v_N)$ of jobs of sizes $1, 2, \ldots, N$, respectively, we denote by $s(v)$ the sum $\sum_n v_n n$, which is the total number of processors these jobs require. Now let

$$V = \{(v_1, \ldots, v_N) | v_n \geq 0, n = 1, \ldots, N, s(v_1, \ldots, v_N) \leq N\}$$

be the set of N-tuples of numbers of jobs that fit on N processors, and let

$$W = \{v \in V | \text{ there exists a } n \text{ with } f(n) > 0, \text{ such that } s(v) + n > N\}$$

be the subset of V of N-tuples of jobs such that an additional job may not fit. Then the set I of numbers of processors that can remain idle is

$$I = \{i | i = N - s(w), w \in W\}.$$

We are interested in the probabilities $P(i)$ that i processors remain idle, for all $i \in I$. The probability $P(i)$ is made up of the probability that when adding jobs we reach a level of $N - i$ processors, and the probability that the next job is larger than i processors. A general expression for these probabilities is (the first factor in the summation below is a multinomial coefficient)

$$P(i) = (1 - F(i)) \cdot \sum_{w=(v_1,\ldots,v_N) \in W, s(w)=N-i} \binom{\sum_n v_n}{v_1, \ldots, v_N} \cdot \left(\prod_{n=1}^{N} f(n)^{v_n} \right). \quad (1)$$

The capacity loss L can now be approximated by

$$L = \frac{1}{N} \sum_{i \in I} P(i) i. \tag{2}$$

The reason that this is an approximation rather than an exact result is that we assume that the fractions of time that i processors are idle for $i \in I$, are equal.

Equations (1) and (2) give a procedure for computing the approximation of the capacity loss, but for large values of N and a large number of possible job sizes this procedure is time-consuming. Therefore, we now present a simple approximation of L in the case of a single cluster with the FCFS scheduling scheme.

The approximation simply consists in assuming that the $P(i)$ are proportional to the first factor, $1 - F(i)$, in (1), which amounts to assuming that the value of the summation in (1) is the same for all $i \in I$. The approximated capacity loss is then given by

$$L = \frac{1}{N} \cdot \frac{\sum_{i \in I} (1 - F(i)) i}{\sum_{i \in I} (1 - F(i))}. \tag{3}$$

Let's now assume that the job size is uniformly distributed on the interval $[n_1, n_2]$, with $0 < n_1 < n_2 \leq N$. Then we have $I \subset \{0, 1, \ldots, n_2 - 1\}$. When the interval $[n_1, n_2]$ is large or n_2 is much smaller than N, the set I will not be much different from $\{0, 1, \ldots, n_2 - 1\}$, and we assume equality below. A straightforward computation shows that (3) can then be written as

$$L = \frac{1}{N} \cdot \frac{n_2^3 - n_1^3 + 3n_1^2 - n_2 - 2n_1}{3n_2^2 - 3n_1^2 + 3n_2 + 3n_1}. \tag{4}$$

In particular, when $n_1 = 1$, (4) yields

$$L = \frac{n_2 - 1}{3N}, \tag{5}$$

and so

$$\rho_m = \frac{3N - n_2 + 1}{3N}. \tag{6}$$

We have validated the approximation of (3) with two different kinds of simulations. The first kind consists of filling a single bin of size N with items of sizes drawn from the job-size distribution in the order they are drawn, until the next job does not fit. All results for simulating bin filling reported in this section give averages for $10,000$ simulation runs. The second kind is by simulating the queueing model defined in Sect. 2 and finding the utilization when the average response time is at least $1,500$ time units (the average service time is put to 1 time unit; for more on the simulations see Sect. 4). Of course, simulating bin filling is much easier than trying to simulate a queueing model close to its maximal utilization. In the latter case, it is very difficult to find out whether the simulation is still in its transient phase, and programming difficulties like running out of space for datastructures such as job queues may arise.

In Table 1 we compare the approximation and the two sets of simulation results for a cluster of 32 processors and for different uniform job-size distributions. Overall, the

results agree very well, except when the interval of job sizes is rather small and the job sizes are large relative to 32. Figure 2 shows the results of simulating the queueing model for uniform job sizes on $[1, 16]$ in a 32-processor cluster, with 95% confidence intervals, and may be compared with the entry for $n_1 = 1, n_2 = 16$ in Table 1.

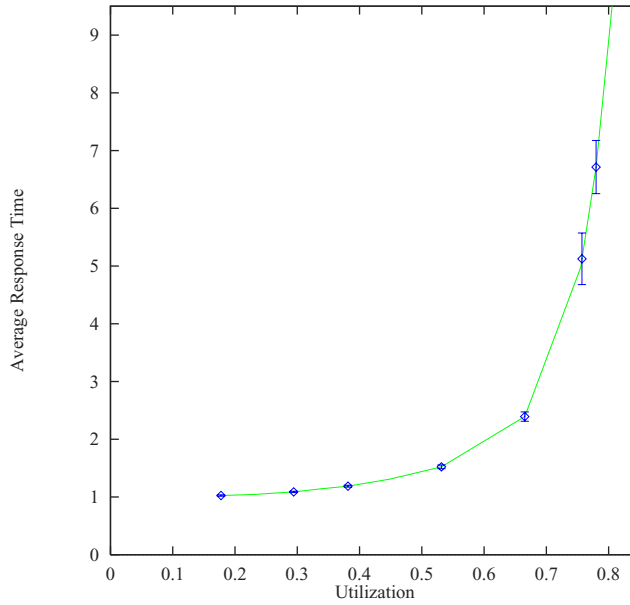

Fig. 2. The response time for a 32-processor single cluster, with job requests uniformly distributed in [1,16], for the FCFS policy.(The bars show the 95% confidence intervals)

In Table 2 we compare the approximation with only bin-filling simulations when the job sizes have a (truncated) geometric distribution on the interval $[1, 32]$ in a 32-processor cluster. In this distribution, $f(n)$ is proportional to q^n for some value q with $0 < q < 1$. This distribution gives a larger proportion of small jobs, a phenomenon that has been observed in actual systems [9], [13].

3.3 Capacity Loss in Multiclusters

For multiclusters, we have only performed a few bin-filling simulations for uniform distributions of the size of job components, for both ordered and unordered requests. It is not generally true that a multicluster performs better with unordered requests than with ordered requests. In fact, when the different job components of unordered requests all have to go to distinct clusters, First Fit may cause a much larger capacity loss than ordered requests experience because the first cluster fills up more rapidly than the last, until at some point it cannot accommodate any component of the next job. In all our

Table 1. The capacity loss in a cluster with 32 processors and with a uniform job-size distribution on the interval $[n_1, n_2]$.

job size		capacity loss		
n_1	n_2	approximation	simulation bin filling	simulation queueing model
1	4	0.031	0.031	0.033
1	5	0.042	0.042	0.044
1	13	0.125	0.124	0.138
1	16	0.156	0.154	0.166
4	5	0.056	0.049	0.052
4	13	0.132	0.132	0.145
4	16	0.163	0.159	0.175
5	13	0.137	0.137	0.150
5	16	0.166	0.163	0.178
13	16	0.212	0.094	0.095

Table 2. The capacity loss in a cluster with 32 processors and with a (truncated) geometric job-size distribution on the interval $[1, 32]$ with parameter q.

	capacity loss	
q	approximation	simulation bin filling
0.95	0.272	0.254
0.90	0.215	0.211
0.85	0.163	0.161
0.80	0.122	0.123
0.75	0.093	0.091
0.70	0.073	0.074
0.65	0.058	0.058
0.60	0.047	0.046
0.55	0.038	0.039
0.50	0.031	0.032

queueing-model simulations in Sect. 4 (in which First Fit is used) the performance with unordered requests is better than with ordered requests, which is in agreement with the results of the bin-filling simulations shown in Table 3, which presents the maximal utilization ρ_m for sets of parameters that are also used in Sect. 4.

When we use Worst Fit instead of First Fit for unordered requests (job components in decreasing order of size go to distinct clusters in decreasing order of the numbers of idle

Table 3. The maximal utilization in a multicluster with 4 clusters of 8 processors each with a uniform job-component-size distribution on the interval $[n_1, n_2]$, for ordered (O) and unordered (U) requests (First Fit, distinct clusters). These results are obtained with bin-filling simulations.

job size		maximal utilization	
n_1	n_2	O	U
1	4	0.685	0.722
1	8	0.578	0.608

Table 4. The capacity loss in a multicluster with C clusters of 32 processors each with a uniform job-component-size distribution on the interval $[n_1, n_2]$, for ordered (O) and unordered (U) requests (Worst Fit, distinct clusters). These results are obtained with bin-filling simulations.

job size		capacity loss				
n_1	n_2	$C = 1$	$C = 4$		$C = 10$	
			O	U	O	U
1	4	0.031	0.146	0.049	0.198	0.053
1	5	0.042	0.172	0.063	0.229	0.067
1	13	0.124	0.326	0.177	0.411	0.177
1	16	0.154	0.363	0.219	0.444	0.229
4	5	0.049	0.106	0.041	0.146	0.029
4	13	0.132	0.282	0.181	0.363	0.199
4	16	0.159	0.329	0.230	0.371	0.282
5	13	0.137	0.270	0.164	0.358	0.158
5	16	0.163	0.317	0.242	0.342	0.303
13	16	0.094	0.094	0.094	0.094	0.094

processors), we can expect that the performance for unordered requests is always better than for ordered requests. In Table 4 we present some results of bin-filling simulations that confirm this expectation.

4 Simulating Co-allocation

In order to estimate the performance of multicluster systems such as the DAS, for different structures and sizes of requests, we modeled the corresponding queuing systems and studied their behaviour using simulations. The simulation programs were implemented using the CSIM simulation package [4]. Simulations were performed for a single-cluster system with 32 processors and for a multicluster system with 4 clusters of 8 nodes each. We varied the distribution of the number of processors requested by jobs by changing the interval from which it was generated, in order to study the influence it has on the performance. Simulations were made for job component sizes uniformly distributed on

the intervals $[1, 4]$ and $[1, 8]$ for the multicluster system. The sizes of the total requests in the single-cluster system with 32 processors we use for comparison, are the sum of 4 numbers uniformly distributed on these intervals.

In all the simulations, the mean of the service time was maintained constant, equal to 1, and the interarrival time was varied in order to determine the response time as a function of the utilization of the system, and to approximate the saturation point (see also Sect. 3.3).

The main goal is to evaluate the performance of the model depending on the structure and distribution of the requests. We consider the performance to be better when for the same utilization the average response time is smaller, and when the maximum utilization is larger.

In Sects. 4.1 and 4.2, the scheduling policy used is FCFS. In Sect. 4.3 the two scheduling policies, FCFS and FPFS are compared. Section 4.4 presents the simulation results for a system composed of 4 clusters with 8 processors each, using ordered requests with component sizes obtained from a job-size distribution presented as being more realistic in [9] and [13].

4.1 The Influence of the Structure of the Requests

Figure 3 compares the average response time for the three types of requests, for two different distribution intervals of their component sizes. As expected, the case of total requests gives the best results from the point of view of the maximal utilization of the system and of the response time, because whenever the number of requested processors does not exceed the number of idle ones, it will be possible to accept the job for service. Less good results are given by the case with unordered requests. Still, because of the fact that it gives more freedom to the scheduler than the one with ordered requests, its performance is better than in the ordered case, in all our simulations.

When requests are ordered the results are the worst because the user imposes both the number of nodes received from distinct clusters and the actual cluster for each of those numbers. It causes a lower maximum utilization and a larger average of the response time. Because of the low utilization, the system becomes saturated faster and gets instable for a lower utilization than in the other cases. However, in general there are situations when unordered requests determine a lower maximal utilization than ordered requests (see Sect. 3.3). The saturation point can be estimated in the graphs by the fast growth of the response time depending on the utilization, as the maximum utilization is being reached.

4.2 The Influence of the Size of the Requests

We may expect that the distribution intervals of the request sizes influence the probability of having multiple jobs served simultaneously. When the upper limit of the distribution interval is decreased, the average size of the requests decreases which means that more jobs can be simultaneously admitted for service. As Fig. 3 and Fig. 4 show, this has a positive impact on the maximum utilization of the system, and on the average response time.

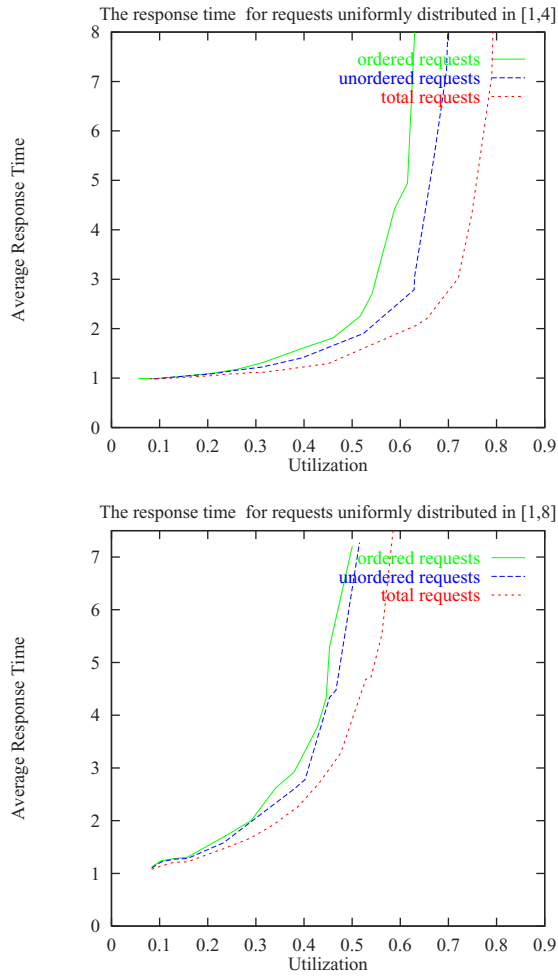

Fig. 3. The influence of the structure of the requests, for the FCFS policy.

Figure 3 shows that for each type of requests, smaller request sizes generate a decrease in the response time. For the distribution in $[1, 8]$, the mean value of the job components sizes is larger than half of the cluster's size, which makes the probability to have more jobs served simultaneously small, especially in the ordered case. Then, on average there is little more than one job in service at a time, which allows a comparison with the M/M/1 queueing model. For the ordered requests case with job components sizes in $[1, 8]$, the system behaves like a single processor, as Fig. 5 indicates. The simulation results prove the comparison to be accurate, the maximum throughput being similar to that of the M/M/1 system.

Figure 4 compares the results for the two distributions of sizes in the cases of ordered and total requests.

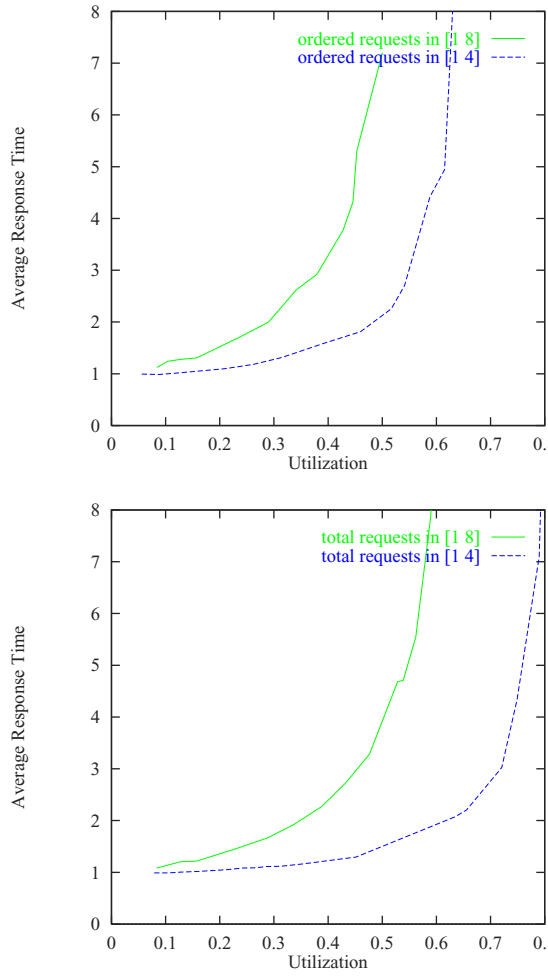

Fig. 4. The influence of the size of the requests for the FCFS policy.

4.3 The Impact of the Scheduling Policy

Another factor which influences the performance of the system is the scheduling policy. As expected, FPFS improves the maximal utilization and the response time compared to FCFS, because it can schedule jobs in an order different from the arrival order. Figure 6 also shows that the influence on performance of the distribution of jobs sizes and of the structure of the requests observed for FCFS is maintained for the FPFS scheme.

For FPFS, increasing the maximum number of times a job can be jumped over, MaxJumps, improves the performance up to a point. When this number is too large, the performance of individual jobs is negatively influenced, and even the effectiveness of the scheme can be affected (MaxJumps $\rightarrow \infty$ would cause starvation, being the same as

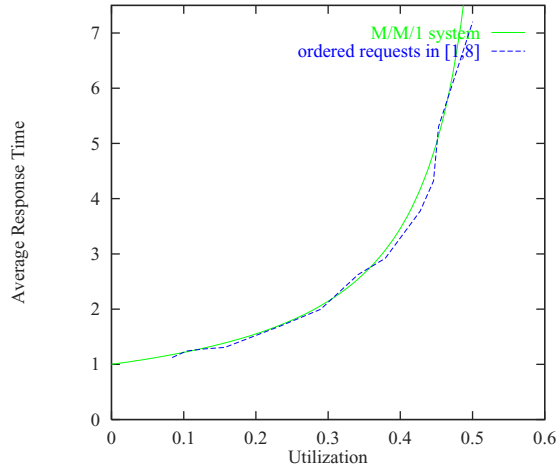

Fig. 5. A comparison between a multicluster with ordered requests in [1,8] and a M/M/1 system.

FPFS without a counter). At the other extreme, for MaxJumps=0 we return to FCFS. We performed simulations with different values for Maxjumps, and finally chose MaxJumps equal to 7. Table 5 show the sensitivity of the response time to the value of MaxJumps for different values of the utilization, in the case of total requests.

Table 5. The response time in a single cluster with 32 Processors (FPFS policy) as a function of MaxJumps.

MaxJumps	utilization	
	0.62	0.78
0	1.91246	4.8845
1	1.80850	3.8588
3	1.80102	3.3412
7	1.70349	2.9125
12	1.69023	2.8249
20	1.68996	2.7771

It can be noticed that for low utilizations, far from the saturation point, improvements given by the scheduling scheme or the distribution and the structure of the requests are very small if any. The differences show only for heavy traffic, when the arrival rate is reasonably high and the job queue is long. For a low arrival rate, under all circumstances we obtain the same results because jobs can be served immediately upon arrival.

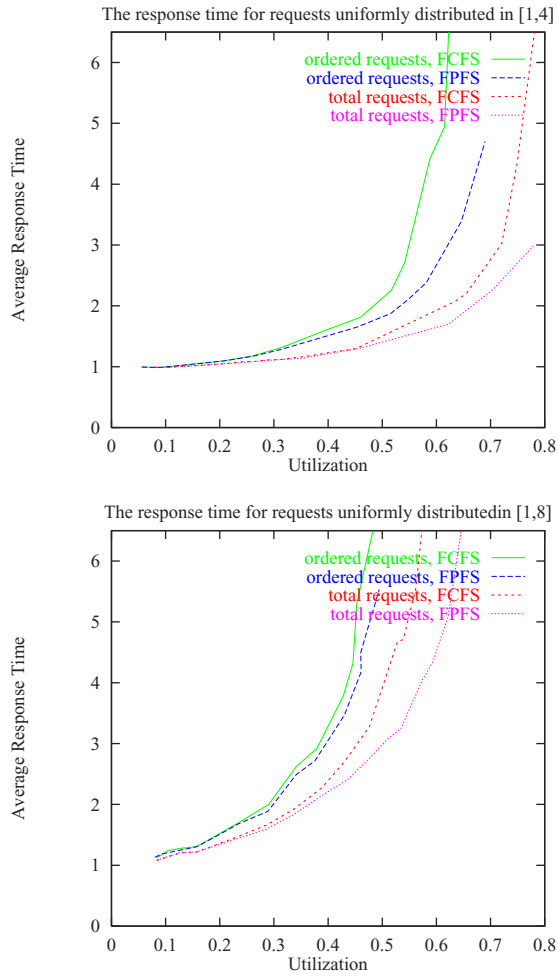

Fig. 6. The influence of the scheduling policy.

4.4 A Realistic Job-Size Distribution

The simulations described in this section use a distribution of the job component sizes that favours small values and powers of 2.

In order to achieve this, with probability $p = 0.7$ the component-size distribution on [1,8] is the normalization of $(q, 3q^2, q^3, 3q^4, \ldots, 3q^N)$, where $N = 8$ and $q = 0.90$. With probability $1 - p$ a job with components uniformly distributed in $[1, 4]$ is generated. Figure 7 shows the variation of the response time for the composed distribution together with the confidence intervals, for a confidence level of 95%.

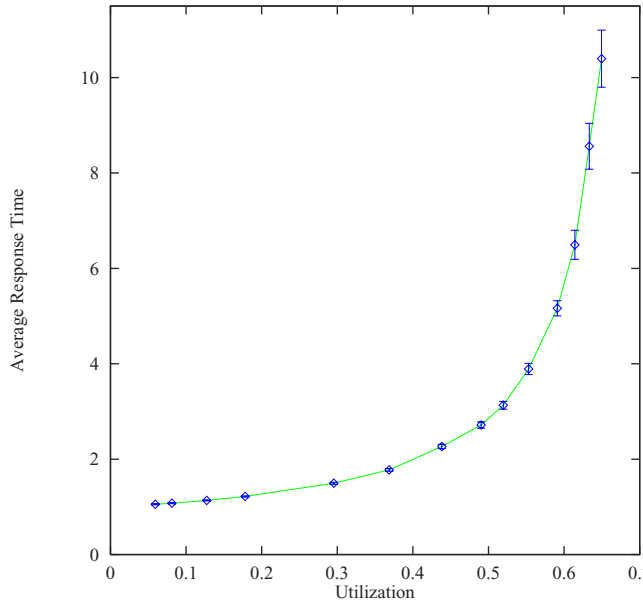

Fig. 7. The response time for a multicluster with ordered requests and with job sizes generated from a distribution that favours small values and powers of 2.

5 Related Work

The problem of scheduling rigid jobs by pure space sharing in a multiprocessor system was also the subject of [1]. It was pointed out that although more complex scheduling schemes are presented in the literature, scheduling schemes for rigid jobs using pure space sharing are still important since these are the schemes implemented on most existing multiprocessor systems. The authors implemented and compared scheduling strategies such as FCFS, FPFS, FPFS with job sorting (both decreasing and increasing), and backfilling. In order to avoid starvation in FPFS and its variations, a time limit is used rather than the maximal number of times a job can be overtaken. Simulation results were combined with performance analysis and experiments in order to verify the effectiveness and the practicality of the schemes. The performance was analysed in terms of processor utilization and stability, using queueing models and the one-dimensional bin-packing problem. As a result of the simulations, the mean response time and the utilization were represented as a function of the system's load. It was concluded that the most effective and the most practical from the schemes analysed are FPFS and Fit Processors Most Processors First Served (FPMPFS). Backfilling can improve performance as well, but it requires a-priori knowledge of the execution times of jobs, which makes it less practical. The performance of the scheduling schemes proved to be sensitive to the distribution of the number of processors requested by a job.

In [13], the optimizations of sorting the job queue according to increasing numbers of processors requested, and of backfilling, are studied with trace-driven simulations. The traces were derived from the logs of three supercomputer installations, amongst which an SP2, and showed relatively large numbers of short jobs and of jobs with sizes that are small or powers of 2. On the SP2, for a sorted job queue, increasing the Maximum Allowable Skipping Count (MASC), a parameter with the same meaning as our MaxJumps, to a large value (simulations were presented with MASC=$10, 100, 1000$) yielded a considerable decrease of the average turnaround time. This contrasts with our result that the performance of FPFS in a single cluster is not very sensitive to increasing MaxJumps beyond 7. However, on the Paragon, the sensitivity to the MASC was much smaller than on the SP2, so this sensitivity is very dependent on the job mix.

The influence of splitting the processors into groups on the performance of the system was also studied in [3]. A technique for operating system schedulers called processor pool-based scheduling, designed to assign the processes of parallel applications in multiprogrammed, shared-memory NUMA multiprocessors, is presented and evaluated. It is assumed that a job starts as a single process, and that it may grow by starting additional processes. Different policies for the initial placement of the jobs and for the placement of its additional processes when it expands were studied. Since it was assumed that the number of processors required by each job is not known when the application starts, the best strategy for initial placement was found to be Worst Fit, because it leaves the largest room for the growth of jobs inside the pool. The author noted that when the number of processors required is known by the time of the arrival, the problem of choosing which processors to allocate is similar to a bin-packing problem with multiple bins. He studied the importance of application parallelism in determining the pool size, and also the influence of the architectural configuration. The results show that although application parallelism should be considered, the optimal pool size is a function of the system's architecture.

Whereas we approach the problem of the maximal utilization from a more theoretical perspective, in [11] a study of the utilizations as observed in existing supercomputing installations is presented. Experience with a large range of machines over more than a decade shows that employing FCFS results in a $40\% - 60\%$ utilization, that more sophisticated policies such as backfilling give an improvement of about 15 percentage points, and that reducing the maximal job size allowed increases utilization.

Finally, let us briefly mention some of the other research that is being performed in the context of the DAS. Whereas the research presented in this paper is at the operating systems level, the other research on the DAS is done at the level of the run-time system [12] and of the applications [2]. In [12], a library is presented which optimizes the collective communication primitives of MPICH, a widely used version of MPI, in order to achieve fast communications in wide-area systems. Because collective communication algorithms are usually designed for LANs, they do not take into account the high latencies of wide-area links, which negatively influence their performance. The authors designed algorithms which are wide-area optimal in that an operation includes only one wide-area latency, and every data item is sent at most once across each wide-area link. They modified 14 collective operations of MPI and obtained substantial performance improvements over MPICH. As an example, in the case of the MPI_Bcast primitive, the

completion time was reduced to 50% for 32 processors divided into 4 clusters and a message size of one byte. This was obtained by sending the message only once to each cluster over the wide-area links, and then broadcasting it inside each cluster on the fast local links.

In [2], several nontrivial algorithms on a multilevel communication structure (LAN clusters connected by a WAN, such as the DAS) were analyzed and several optimization techniques were used to improve their performance. The optimizations either reduced intercluster traffic or masked the effect of intercluster communications and caused a significant improvement. The authors concluded that many medium-grain applications could be optimized to run well on a multilevel, wide-area cluster. One of the optimized applications solves the Traveling Salesman Problem. It was improved by replacing the dynamic work distribution through a centralized queue with a static distribution over the clusters, each of them having its own local queue. For 32 processors divided into 4 clusters, the speedup was 24, compared to 15 for the unoptimized solution.

6 Conclusions

We have proposed a model for scheduling rigid jobs in multicluster systems based on our DAS system, and assessed its performance for different structures and sizes of jobs in terms of average response time as a function of utilization.

We simulated two scheduling schemes, First Come First Served and Fit Processor First Served, and three types of requests. As expected, for both scheduling schemes, the average response time is smaller and the maximum utilization is larger when the requests are more flexible. The best performance was obtained for total requests, when only the total number of processors needed is provided, and when the problem is similar to a single bin-packing problem. For unordered requests, when the numbers of processors to be allocated in separate clusters is specified, the problem is similar to a set of related bin-packing problems, and because there are more restrictions, the performance decreases. It can be improved by changing the policy from FCFS to FPFS, because then the scheduler gets freedom to look further in the queue for jobs which fit, but it will still be below the total requests case. In all our simulations, ordered requests, when the exact clusters from which to satisfy the components of the requests are provided, cause even larger response times, and an even lower maximum utilization. This latter result is not universally valid using First Fit, as we did.

Decreasing the maximal size of the requests improves the performance of the system. This fact is again related to the bin-packing problem, because it is easier to schedule small jobs than large ones.

We derived an approximation for the maximal utilization in single-cluster systems and checked its validity against simulation results.

We plan future work on the effect of communication on the performance of multicluster systems (and of the applications) in comparison with single-cluster systems, because inter-cluster communication is much slower than communication inside the same cluster (in [2] a factor of 50 between these communication speeds was reported). We also intend to do simulations and performance measurements using traces from real multicluster systems (the DAS) instead of theoretical distributions.

References

1. Aida, K., Kasahara, H., Narita, S.: Job Scheduling Scheme for Pure Space Sharing Among Rigid Jobs. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1459** (1998) 98–121
2. Bal, H.E., Plaat, A., Bakker, M.G., Dozy, P., Hofman, R.F.H.: Optimizing Parallel Applications for Wide-Area Clusters. Proceedings of the 12th International Parallel Processing Symposium (IPPS'98) (1998) 784–790
3. Brecht, T.B.: An Experimental Evaluation of Processor Pool-Based Scheduling for Shared-Memory NUMA multiprocessors. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1291** (1997) 139–165
4. The CSIM18 Simulation Engine, User's Guide. Mesquite Software, Inc.
5. The Distributed ASCI Supercomputer's site. Http://www.cs.vu.nl/das/
6. Feitelson, D.G., Rudolph, L.: Toward Convergence in Job Schedulers for Parallel Supercomputers. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1162** (1996) 1–26
7. Feitelson, D.G., Rudolph, L.: Theory and Practice in Parallel Job Scheduling. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1291** (1997) 1–34
8. Feitelson, D.G., Jette, M.A.: Improved Utilization and Responsiveness with Gang Scheduling. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1291** (1997) 238–261
9. Feitelson, D.G.: Packing Schemes for Gang Scheduling. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1162** (1996) 89–110
10. The Globus site. Http://www.globus.org
11. Patton Jones, J., Nitzberg, B.: Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization. Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science **1659** (1999) 1–16
12. Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.A.F.: MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99) (1999) 131–140
13. Subhlok, J., Gross, T., Suzuoka, T.: Impact of Job Mix on Optimizations for Space Sharing Schedulers. Supercomputing '96 (1996)