

An Evaluation of Processor Co-Allocation for Different System Configurations and Job Structures

A.I.D. Bucur

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

D.H.J. Epema

Abstract

In systems consisting of multiple clusters of processors such as our Distributed ASCI¹ Supercomputer (DAS), jobs may request co-allocation, i.e., the simultaneous allocation of processors in different clusters. We simulate such systems ignoring communication among the tasks of jobs, and determine the response times for different types and sizes of job requests, and for different numbers and sizes of clusters. In many cases we also compute or approximate the maximum utilization. We find that the numbers and sizes of the clusters and of the job components have a strong impact on performance and that in many cases co-allocation is a viable choice.

1. Introduction

Over the last decade, clusters and distributed-memory multiprocessors consisting of hundreds or thousands of standard CPUs have become very popular. In addition, recent work in computational and data GRIDs [2, 7] enables applications to access resources in different and possibly widely dispersed locations simultaneously—that is, to employ *co-allocation*—to accomplish their goals, effectively creating single multicluster systems. Most of the research on processor scheduling in parallel computer systems has been dedicated to multiprocessors and single-cluster systems, but hardly any attention has been devoted to multicluster systems. In this paper we study the performance of processor co-allocation policies in multicluster systems employing space sharing for rigid jobs [3] depending on such parameters as the structure of jobs and the system configuration.

Because it is a simple and yet often used and very practical scheduling strategy, we only consider rigid jobs scheduled by pure space sharing, which means that jobs require fixed numbers of processors and are executed on them exclusively until their completion. Our two performance metrics are the mean job response time as a function of the utilization and the maximum utilization at which multicluster systems are stable.

We first deduce (approximate) analytic expressions for the maximal utilization of multiclusters under co-allocation. Then using simulations we find the response times for different types of job requests and numbers of job components, and for different numbers and sizes of clusters. We conclude that in many cases co-allocation yields good performance. The request type has a strong influence on the performance, and so do the numbers of jobs components. For unordered jobs, which provide the easiest and most practical way of co-allocation, the best performance is obtained in systems with clusters of equal size, or with a large number of clusters relative to the number of job components.

In the most general setting, GRID resources are very heterogeneous and wide-area connections may be very slow. In this paper we restrict ourselves to homogeneous multicluster systems and ignore communication in order to isolate the aspects mentioned above. However, if we assume that the job service times in our model do include communication in a single cluster with a fast local network, one can get an indication of the performance of processor co-allocation when communication in multiclusters is taken into account in the following way. If the service times of all jobs are extended by (about) the same factor $\alpha > 1$ due to the relatively slow intercluster communication, multiplying our response-times curves by $1/\alpha$ and α in the horizontal and vertical directions, respectively, yields the multicluster response times. In [9] it is shown that for a ratio of 100 between the intercluster and intracluster communication speeds, a value which holds for the DAS, α is below

¹In 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.

1.4 for most of the algorithms considered.

2. The Model

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

2.1. The DAS System

The DAS [1, 6] is a wide-area computer system consisting of four clusters of identical Pentium Pro processors, one with 128, the other three with 24 processors each. The 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.

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.

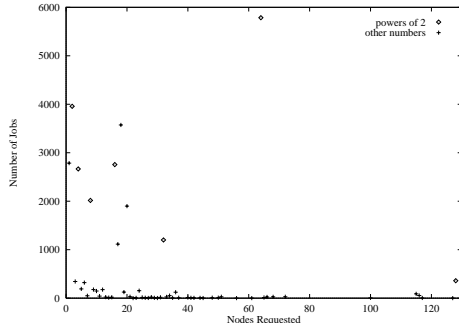


Figure 1. The density of the job-request sizes for the largest DAS cluster (128 processors)

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, 94.45% of the recorded jobs ran less than 15 minutes. Figure 2 presents the density of service time values on the DAS, as it was obtained from the log. The average service time is 356.45 seconds and the coefficient of variation is 5.37. Still, not all jobs in the log were short: the longest one took around 15 hours to complete.

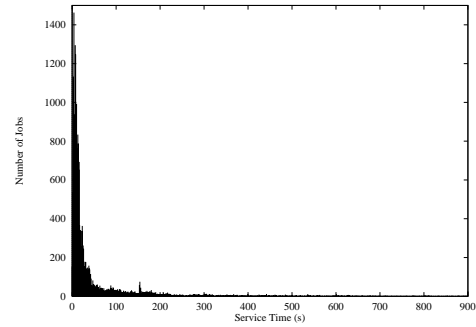


Figure 2. The density of the service times for the largest DAS cluster (128 processors)

2.2. The Structure of the System

We model a multicluster system consisting of C clusters of processors, cluster i having N_i processors, $i = 1, \dots, C$. 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. The system has a single central scheduler, with one global queue. For interarrival times we use exponential distributions.

2.3. The Structure of Job Requests

Jobs that require co-allocation 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 [5]. 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 four cases for the structure of job requests:

1. A *flexible request* specifies the total number of processors needed, obtained as the sum of the values in the tuple, letting the scheduler spread the tasks over the clusters.
2. For an *unordered request*, 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.

3. In an *ordered request* the positions of the request components in the tuple specify the clusters from which the processors must be allocated.
4. For *total requests*, there is a single cluster and a request specifies the single number of processors needed, again obtained as the sum of the values in the tuple.

2.4. Scheduling Decisions

To determine whether an unordered request fits, we try to schedule its components in decreasing order of their sizes on distinct clusters. Possible ways of placement include First Fit (FF; fix an order of the clusters and pick the first one on which a job component fits) and Worst Fit (WF; pick the cluster with the largest number of idle processors), both of which we use in our simulations. For a flexible request we first determine whether there are enough idle processors in the whole system to serve the job. If so, the job will be placed on the system in an arbitrary way.

In all the simulations the First Come First Served (FCFS) policy is used.

3. The maximal utilization

In the model described in Sect. 2, processors may be idle while there are waiting jobs because the job at the head of the queue does not fit. As a consequence, when ρ_m is the *maximal utilization*, that is, the utilization such that the system is stable (unstable) at utilizations ρ with $\rho < \rho_m$ ($\rho > \rho_m$), in general, we have $\rho_m < 1$. We define the (average) *capacity loss* l as the average fraction of the total number of processors that are idle at the maximal utilization, so $l = 1 - \rho_m$. Capacity loss may be due to the structure of job requests and to the job-component-size distribution (e.g., we expect it to be higher for ordered than for unordered jobs, and for larger component sizes). In this section we present an expression for the average maximal Multi-Programming Level (MPL, i.e., the number of jobs being served simultaneously) in multicluster systems with ordered requests, from which of course ρ_m can be derived. We also deduce an approximation for the average maximal MPL in multiclusters with unordered requests and WF component placement, which we validate with simulations. In this section we assume that the service-time distribution is exponential.

3.1. Capacity loss with ordered requests

In this section we assume that requests are ordered. Let F be the (multidimensional) job-size distribution, which (allowing components of size zero) is defined on the set

$$S_O = \left(\prod_{i=1}^c \{0, 1, \dots, N_i\} \right) \setminus \{(0, 0, \dots, 0)\}.$$

We have $F(\bar{n}) = P(\bar{s} \leq \bar{n})$ for $\bar{n} \in S_O$, where \leq denotes component-wise (in)equality. Let f be the job-size density, so $f(\bar{n})$ is the probability of having a job of size $\bar{n} \in S_O$. Denoting by $G^{(i)}$ the i -th convolution of a distribution G with itself, $F^{(i)}(\bar{N})$ and $F^{(i)}(\bar{N}) - F^{(i+1)}(\bar{N})$ are the probabilities that at least and exactly i random jobs fit on the multicluster, respectively, where $\bar{N} = (N_1, N_2, \dots, N_C)$. When the job-component sizes are mutually independent, we have $F^{(i)}(\bar{N}) = \prod_j F_j^{(i)}(N_j)$ for $i = 1, 2, \dots$, with F_j the distribution of the j -th components of jobs.

In our treatment of multiclusters with ordered requests below we follow [4]. There, a queueing model of a multiprocessor with P processors and B memory blocks is studied. The scheduling policy is First-Come-First-Loaded, in which a job is allowed from the head of the queue into the multiprogramming set when its memory requirements, taken from some discrete distribution F on $[1, B]$, can be satisfied. When the number of jobs does not exceed P , every job gets a processor to itself; otherwise processor sharing is employed. The service-time distribution (on a complete processor) is exponential. When $P \geq B$, and so every job has a processor of its own, this model coincides with our single-cluster model with memory blocks assuming the role of processors. Under the assumption that there is always a sufficiently long queue, the Markov chain V with state space (z_1, \dots, z_B) , where the z_i 's are the memory sizes of the oldest B jobs in the system, and the MPL, both immediately after a departure and the entailing job loadings, are studied. It turns out that the behaviour of V is as if FIFO is used, and, by solving the balance equations, that the associated probabilities are as if the z_i are independently drawn from F . In addition, the time-average maximal MPL is derived in terms of convolutions of F .

In our multicluster model, we also consider the sequence of the oldest jobs in the system such that it includes at least all jobs in service. Let B be some upper bound of the number of jobs that can be simultaneously in service ($\sum_i N_i$ will certainly do). Let $\bar{Z} = (\bar{z}_1, \bar{z}_2, \dots, \bar{z}_B)$ be the *processor state vector*, which is the (left-to-right ordered) sequence of the sizes of the oldest jobs in the system. Some first part of \bar{Z} describes the jobs in service, and the remainder the jobs at the head of the queue. When a job leaves, the new processor state vector is obtained by omitting the corresponding element from the current vector, shifting the rest one step to the left, and adding a new element at the end. Let V be the set of processor state vectors.

Because the service-time distribution is exponential, for $v, w \in V$, the transition of the system from state v to state w only depends on v : each of the jobs in service has equal probability of completing first, and the job at the head of the queue to be added to the state is random. So in fact, V is a Markov chain. The result of [4] explained above can be extended in a straightforward way to our situation—the im-

portant element is that the distribution F simply determines which sets of jobs can constitute the multiprogramming set, but the underlying structure of a single or of multiple resources does not matter. So also now, the stationary probability distribution π on V satisfies

$$\pi(\bar{Z}) = \prod_{i=1}^B f(\bar{z}_i), \quad (1)$$

which means that the distribution of the oldest B jobs in the system is as if they are independently drawn from F . So, because the average length of the intervals with i jobs in service is inversely proportional to i due to the exponential service, we find for the average maximal MPL M :

$$M = \frac{\sum_{i=1}^B (F^{(i)}(\bar{N}) - F^{(i+1)}(\bar{N})) \cdot (1/i) \cdot i}{\sum_{i=1}^B (F^{(i)}(\bar{N}) - F^{(i+1)}(\bar{N})) \cdot (1/i)},$$

which can be written as

$$M = \frac{1}{1 - \sum_{i=2}^B F^{(i)}(\bar{N}) / (i(i-1))}. \quad (2)$$

For single clusters this expression coincides with the formula in [4], p. 468. Denoting by s be the average total job size, the maximal utilization is given by

$$\rho_m = \frac{M \cdot s}{\sum_i N_i}. \quad (3)$$

In Sect. 3.2 we will validate an approximation for the maximal utilization for unordered requests with simulations, by taking the utilization in these simulations that yield an average response time of at least 1,500 time units (the average service time is 1 time unit; for more details on the simulations, see Sect. 4). We have validated this approach in turn by running simulations for single clusters and for multiclusters with ordered requests, for which we have the exact solution based on Eq. (3). In Tables 1 and 2 we show both exact and simulation results for a single cluster and for multicluster systems with ordered and flexible requests (and with unordered requests in Table 2, see Sect. 3.2) for different distributions of the job-component sizes (which are mutually independent). The exact and simulation results for the single cluster agree extremely well. The same is true for the simulations we did for the 4-cluster system (results not shown here).

3.2. Capacity loss with unordered requests

We now derive an approximation to the maximal utilization in multiclusters with unordered requests in case all clusters are of equal size N . For job placement, WF is used.

Table 1. The capacity loss in a single cluster ($C = 1$) of size 32 and in a multicluster with 4 clusters ($C = 4$) of size 32 for ordered and flexible requests, with job-component-size distribution $D(q)$ on $[1, 32]$

job comp. size distr.	capacity loss			
	$C = 1$		$C = 4$	
q	exact	simulation	ordered (exact)	flexible (exact)
0.95	0.293	0.295	0.564	0.226
0.90	0.249	0.251	0.555	0.157
0.85	0.188	0.188	0.494	0.112
0.80	0.134	0.135	0.416	0.085
0.75	0.097	0.097	0.348	0.067
0.70	0.073	0.074	0.296	0.056
0.65	0.057	0.058	0.257	0.048
0.60	0.046	0.047	0.227	0.042
0.55	0.038	0.039	0.203	0.038
0.50	0.032	0.032	0.182	0.034

The job-size distribution F is now defined on

$$S_U = \{(s_1, s_2, \dots, s_C) \mid 1 \leq s_1 \leq N, s_{i+1} \leq s_i, \\ i = 1, 2, \dots, C-1\},$$

that is, job requests are represented by a vector with non-increasing components. Compared to the case of ordered requests, the case of unordered requests presents some difficulty for two reasons. First, given a set of unordered jobs, it is not possible to say whether they simultaneously fit on a multicluster, because that depends also on the order of arrival of the jobs. For instance, if $C = 2$ and $N = 5$, then if three jobs of sizes $(2, 1)$, $(3, 1)$, $(2, 1)$ arrive in this order, they can all be accommodated, while if they arrive in the order $(2, 1)$, $(2, 1)$, $(3, 1)$, only the first two jobs can run simultaneously. Second, a departure can leave the system in a state that cannot occur when it is simply filled with jobs from the empty state. If with the first sequence of arrivals above the job of size $(3, 1)$ leaves, both $(2, 1)$ -jobs will have their largest component in cluster 1. Our result below deals with the first problem—by running over all possible arrival sequences—but not with the second, which is why it is an approximation.

We now define the i -fold WF-convolution $F \star G$ of two distributions F, G on S_U in the following way. Let for any C -vector $\bar{s} = (s_1, s_2, \dots, s_C)$ the reversed vector $rev(\bar{s})$ be defined as $rev(\bar{s}) = (s_C, s_{C-1}, \dots, s_1)$, and the ordered vector $ord(\bar{s})$ as the vector with the elements of \bar{s} permuted such that they form a non-increasing sequence. Now if f, g are the densities of F and G , respectively, $F \star G$ has density

Table 2. The capacity loss in a single cluster ($C = 1$) of size 32 and in a multicluster with 4 clusters ($C = 4$) of size 32 for ordered, unordered, and flexible requests, with job-component-size distribution $U[n_1, n_2]$

job comp. size distr.		capacity loss					
		$C = 1$		$C = 4$			
n_1	n_2	exact	simulation	ordered (exact)	unordered approximation	simulation	flexible (exact)
1	4	0.032	0.033	0.149	0.050	0.053	0.038
1	5	0.043	0.044	0.176	0.065	0.067	0.047
1	13	0.139	0.139	0.345	0.187	0.192	0.120
1	16	0.169	0.169	0.380	0.233	0.239	0.148
4	5	0.051	0.052	0.111	0.043	0.048	0.043
4	13	0.145	0.145	0.302	0.186	0.188	0.149
4	16	0.174	0.175	0.337	0.250	0.255	0.167
5	13	0.149	0.150	0.292	0.170	0.175	0.146
5	16	0.177	0.178	0.321	0.260	0.260	0.186
13	16	0.094	0.095	0.094	0.094	0.094	0.094

h defined by:

$$h(\bar{s}) = \sum_{\bar{t}, \bar{u} \in S_U, \bar{s} = \text{ord}(\bar{t} + \text{rev}(\bar{u}))} f(\bar{t}) \cdot g(\bar{u}).$$

Then, putting $F^{[2]} = F \star F$, we define inductively $F^{[i]} = F^{[i-1]} \star F$ for $i = 3, 4, \dots$. What this amounts to is that $F^{[i]}$ is the distribution of the non-increasingly ordered numbers of processors in the clusters of the multicluster that are in use when i unordered requests are put on an initially empty system with WF. Now our approximation of the maximum average MPL M is (cf. Eq. (2)):

$$M = \frac{1}{1 - \sum_{i=2}^B F^{[i]}(\bar{N}) / (i(i-1))}, \quad (4)$$

where again B is some upper bound to the number of jobs that can simultaneously be served, from which an approximation of the maximal utilization (or the capacity loss) can be derived with Eq. (3). In addition, we have again performed simulations along the lines presented in Sect. 3.1 to find the maximal utilization. The results, which are in Table 2, show that the approximation is very accurate. Also, the results for multiclusters in Table 2 show the reduction in capacity loss when going from ordered to unordered to flexible requests.

Extending the results in this section to unequal cluster sizes is cumbersome, because then adding an unordered request to a multicluster depends on the numbers of idle rather than used processors. So one would have to replace the WF-convolution by a sort of convolution that depends on the cluster sizes.

4. Simulating Co-Allocation

To measure the performance of multicluster systems such as the DAS for different structures and sizes of requests, different numbers of clusters and different clusters sizes, we modeled the corresponding queuing systems and studied their behaviour using simulations. The simulation programs were implemented using the CSIM simulation package [8]. In all the simulations the mean service time is 1. In all the graphs in this section we include confidence intervals; they are at the 95%-level. In all cases where the service times are exponential we can find (an approximation to) the maximal utilizations based on our results in Sect. 3; we show these only for a limited number of cases.

In this section various multicluster systems with a total of 128 nodes are considered; job-component sizes are obtained from the distribution $D(0.9)$ on [1, 8].

4.1. Equal versus Different Cluster Sizes

A question we try to answer is whether a system with equal clusters provides a better performance than a system where nodes are unevenly divided among the clusters, for equal total system sizes. For this we look at two systems with 4 clusters, with total numbers of processors of 128: a system with equal clusters of size 32, and one with uneven clusters of sizes 64, 32, 16, 16. (FF uses the clusters in this order; see Sect. 4.2 for the reverse order.) Simulations were performed for ordered and unordered requests, and for total requests in a single 64-processor cluster to see whether only using the largest cluster in the uneven system can give better performance than co-allocation. For unordered requests both FF and WF were simulated; since the results were similar, only those for FF are shown.

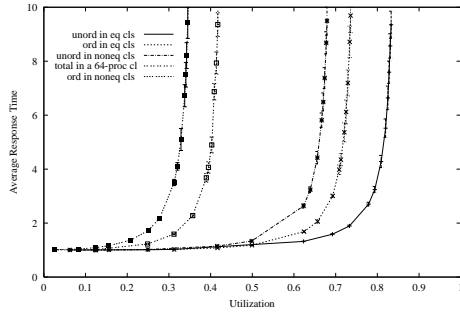


Figure 3. The performance of ordered and unordered requests in a multicluster with equal clusters (32,32,32,32), and in a multicluster with unequal clusters (64,32,16,16)

Although the effect is less pronounced for unordered requests, for which the scheduler can deal well with unequal cluster sizes, Fig. 3 shows that for both ordered and unordered requests, the system with equal clusters provides a much better performance. This can be explained by the fact that both types of requests need nodes from all the 4 clusters (the number of components is equal to the number of clusters) and once the smallest clusters get filled to the extent that they cannot accommodate components of a new job, the idle processors from the big clusters cannot be used either. Since the small clusters behave as a bottleneck for the utilization, we can expect that the performance deteriorates with the increase in the difference between the clusters' capacities. For ordered requests in uneven clusters the performance is even worse than for total requests in a system with half the capacity (64 processors, the utilization on the horizontal axis is then still relative to the whole 128-processor system), which means that we can ignore the smaller clusters and only use the largest.

4.2. Comparing Systems with Different Numbers of Clusters

For the same total size of its clusters, we can expect that the performance of a system varies with the number of clusters. We study this variation for systems with 128 nodes with at least 4 clusters, in the presence of unordered requests consisting of 4 components, which the scheduler can now put on any 4 different clusters. We compare the results for FF and WF.

Figure 4 compares two system configurations with clusters of equal sizes: a system with 4 clusters of 32 processors, and a system with 8 clusters of 16 nodes. Although in the second case the clusters are much smaller, the results show that the performance is similar at low utilizations and even slightly better at moderate and high utilizations, which sug-

gests that the performance can be better when the number of clusters is larger than the number of request components. This can be explained by the fact that FF, which we use here, first fills up the first cluster with the largest job components. Then, when such a component does not fit in the first cluster anymore, FF in the 4-cluster case is still forced to use the first cluster for a job component. However, in the 8-cluster case FF may skip the first cluster(s) completely for a whole job once in a while, but will always try to put small job components on them. Apparently, this determines a better packing when the number of clusters is larger, even if those clusters are smaller.

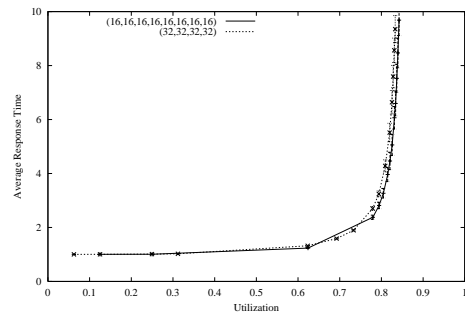


Figure 4. The influence of the number and sizes of clusters for equal clusters and unordered requests, with FF component placement

Figure 5 shows the performance of systems with different numbers of clusters of uneven sizes; the difference between the graphs is the order in which clusters are used by the scheduler. Taking as a reference the case (32, 32, 32, 32) in the two graphs, we notice that performance is better when the scheduler starts with the largest clusters. For a system where the user can express preference for certain clusters, this can be translated into the fact that preferring the larger clusters is a good option and improves both the response time and the maximal utilization. From Figure 5 (and Figure 4), we conclude that the performance is only good in systems with clusters of equal size and, for both orders of using the clusters with FF, in systems with clusters of unequal sizes whose numbers of clusters are considerably larger than the number of job components.

Figures 6 and 7 show the results for the same configurations, when WF is used for placing the jobs. We conclude that systems with large equal clusters give much better performance. For systems with unequal clusters, we should still prefer to have many (although relatively small) clusters, since the performance will be better than for few, large clusters.

Comparing WF with FF (large-small order), we notice that WF is much better for systems with few, equal, large

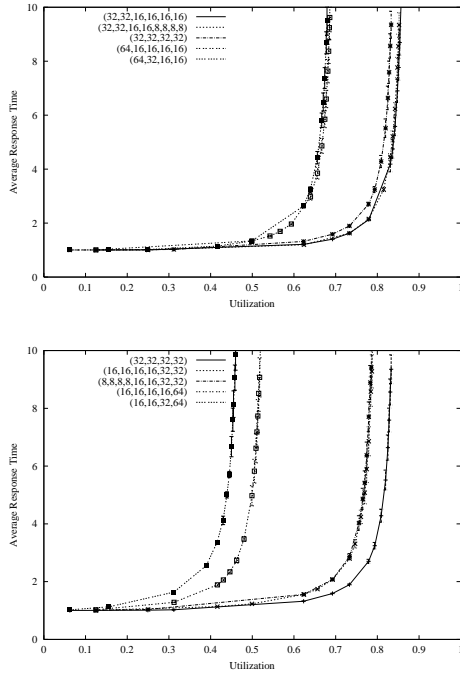


Figure 5. The influence of the number and sizes of clusters for unordered requests; component placement uses FF and starts with the large clusters (top), and with the small clusters (bottom)

clusters. Using WF instead of FF slightly improves the performance also for systems with unequal clusters, when the number of clusters is relatively small compared to the number of job components. However, comparing Fig. 6 with Fig. 4 we can see that for systems with small, equal clusters and with numbers of clusters considerably larger than the number of job components, WF has significantly worse performance than FF. This is also valid for systems with unequal clusters, as Fig. 8 shows. Summing up, FF should be chosen as placement policy when dealing with systems with many but relatively small clusters, while WF is to be preferred for multiclusters with few, large clusters.

5. Co-Allocation versus No-Coallocation

In this section we study workloads consisting of jobs with different numbers of components. We consider a multicluster system with 4 clusters of 32 processors each and evaluate its performance for jobs having between 1 and 4 components. Mixes of jobs are also considered. This corresponds to a system in which co-allocation is combined with jobs requiring processors from a single-cluster, but both sin-

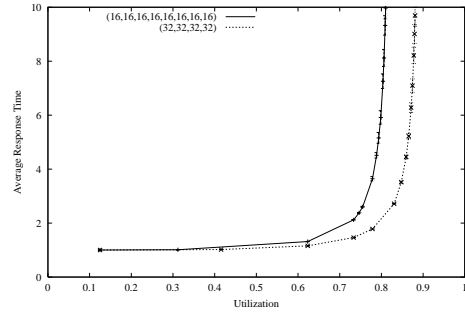


Figure 6. The influence of the number and sizes of clusters for equal clusters and unordered requests, with WF component placement

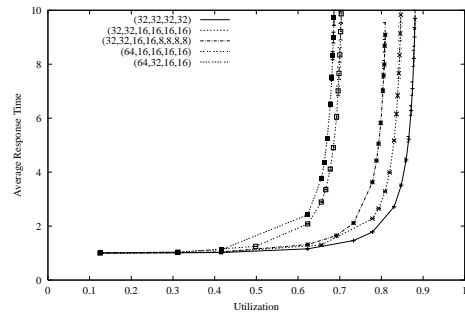


Figure 7. The influence of the number and sizes of clusters for unordered requests; component placement uses WF

gle cluster and multicluster jobs are submitted to the same central scheduler. We model unordered requests and two ways of generating components: either each component is obtained from the distribution $D(0.9)$, which means that jobs with fewer components are in general smaller, or jobs have the same total request size and their components are obtained as a sum of values from $D(0.9)$. In this second case a job with fewer components has in general larger components.

5.1. Jobs with Components from the Same Distribution

Figure 9 compares the performance of the system for jobs with 1, 2, 3 and 4 components, and mixes of such jobs in different percentages. Each component is obtained from the distribution $D(0.9)$ and jobs are placed using FF. The performance is better for jobs with fewer components since they are smaller and also allow more ways of placement. For mixes the performance is somewhere between those of

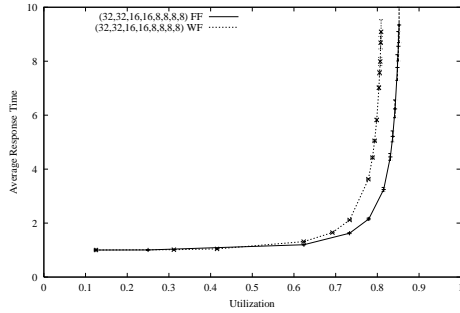


Figure 8. Comparison between FF and WF for placing unordered requests, in systems with many, unequal clusters

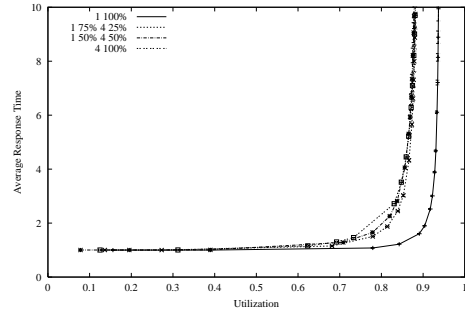


Figure 10. Different number of components, the same job-component size; WF placement

the participants in the mix, depending on the percentage of each of them. Figure 10 shows the performance for jobs with 1 and 4 components and for mixes of the two when WF is used. Compared to FF, WF brings visible improvements for jobs with 4 components and mixes with high percentages of such jobs, but has little effect on jobs with 1 component or on the mix with 75% jobs with a single component.

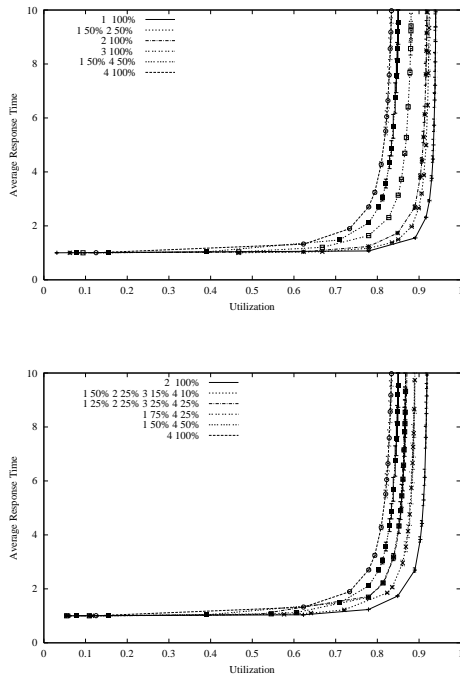


Figure 9. Different number of components, the same job-component size; FF placement

5.2. Jobs with the Same Total Request Size

In this section each job has the same total request size. Figure 11 compares jobs with 1, 2 and 4 components and mixes of such jobs for the FF policy. For jobs with 2 components, each component is the sum of two values from $D(0.9)$. The worst performance is displayed by jobs with one component since that component is large compared with the clusters' sizes. Significantly better performance is obtained for jobs with four components, although they have the disadvantage of placing a component in each cluster. For this reason, the best performance is shown by systems hosting jobs with two components since these components are still relatively small and need only two of the clusters (more possibilities for the placement). For mixes of 1 and 4 the performance of the system is even worse than for jobs with 1 component since the requirements of jobs with 1 and 4 components are opposed to each other: jobs with 4 components need to place one component in each cluster, while jobs with 1 component need enough room in one cluster to place their large component and tend to fill in individual clusters. When the mix also contains jobs with 2 components the performance of the system improves, since they are easier to fit. Figure 12 shows the results for systems with jobs consisting of 1 component, 4 components, and mixes of 1 and 4 when WF is used. WF improves the performance for jobs with 4 components compared to FF, but leaves almost unchanged the results for jobs with 1 component and for mixes where such jobs are predominant.

6. Conclusions

In this paper we have studied the performance of processor co-allocation in multicenter systems with space sharing for rigid multi-component jobs of different request types that impose various restrictions on their placement. Our main conclusions are as follows:

- *Multiclusters with equal clusters provide better performance.* For the same number of clusters and total num-

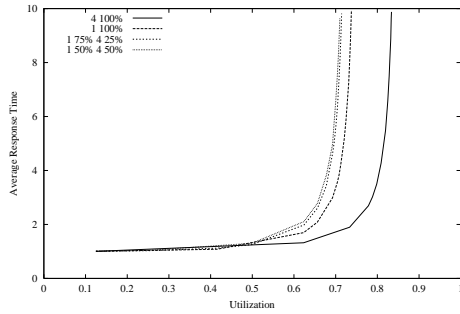
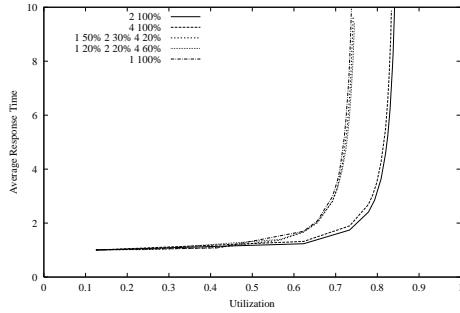


Figure 11. Different number of components, the same total request size; FF placement

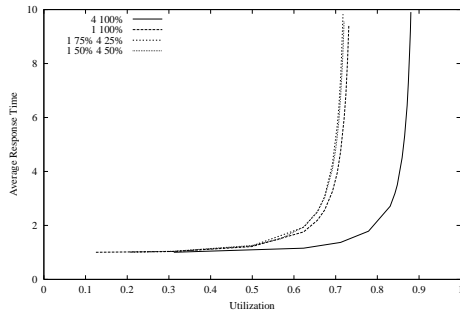


Figure 12. Different number of components, the same total request size; WF placement

ber of processors, (un)ordered request types fit better in systems with equal clusters.

- *In general, a system with few, large clusters should be preferred.* For the same total number of processors and unordered requests, we can obtain a better performance if the system is only divided into a small number of clusters, especially if jobs are placed using WF.
- *For systems with few, large clusters WF should be used. For systems with many clusters, FF is better.* If a system has a small number of clusters compared to the number of job components, WF policy should be

chosen for placing the jobs on clusters. On the contrary, for systems with many clusters FF gives better performance.

- *For (un)ordered jobs the performance is better when either the cluster sizes are equal, or the number of clusters is large relative to the number of job components.*
- *Large jobs or jobs with large components have lower performance.* The performance is better when the jobs submitted to the system are small. For jobs with large (even if few) components compared to the cluster sizes the performance is worse than for jobs with more but smaller components. Spreading large single-component jobs over more clusters (co-allocation) can generate better results.
- *Jobs with many components yield poor performance.* Even when the cost of communication is not considered, the performance is lower for jobs with numbers of components close to the number of clusters in the system.

References

- [1] *The Distributed ASCI Supercomputer (DAS) site.* <http://www.cs.vu.nl/das>.
- [2] *The Global Grid Forum.* <http://www.gridforum.org>.
- [3] K. Aida, H. Kasahara, and S. Narita. Job Scheduling Scheme for Pure Space Sharing Among Rigid Jobs. In D. Feitelson and L. Rudolph, editors, *4th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of LNCS, pages 98–121. Springer-Verlag, 1998.
- [4] R. Bryant. Maximum Processing Rates of Memory Bound Systems. *J. of the ACM*, 29:461–477, 1982.
- [5] S.-H. Chiang and M. Vernon. Characteristics of a Large Shared Memory Production Workload. In D. Feitelson and L. Rudolph, editors, *7th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of LNCS, pages 159–187. Springer-Verlag, 2001.
- [6] H. B. et al. The Distributed ASCI Supercomputer Project. *ACM Operating Systems Review*, 34(4):76–96, 2000.
- [7] I. Foster and C. K. (eds). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [8] Mesquite Software, Inc. *The CSIM18 Simulation Engine, User's Guide*.
- [9] A. Plaatz, 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.