The Influence of Communication on the Performance of Co-Allocation

A.I.D. Bucur D.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

In systems consisting of multiple clusters of processors interconnected by relatively slow connections such as our Distributed ASCI[1] Supercomputer (DAS), jobs may request co-allocation, i.e., the simultaneous allocation of processors in different clusters. The performance of co-allocation may be severely impacted by the slow intercluster connections, and by the types of job requests. We distinguish different job request types ranging from ordered requests that specify the numbers of processors needed in each of the clusters, to flexible requests that only specify a total. We simulate multicluster systems with the FCFS policy—and with two policies for placing a flexible request, one tries to balance cluster loads and one tries to fill clusters completely—to determine the response times under workloads consisting of a single or of different request types for different communication speeds across the intercluster connections. In addition to a synthetic workload, we also consider a workload derived from measurements of a real application on the DAS. We find that the communication speed difference has a severe impact on response times, that a relatively small amount of capacity is lost due to communication, and that for a mix of request types, the performance is determined not only by the separate behaviours of the different types of requests, but also by the way in which they interact.

1 Introduction

Over the last decade, most of the research on scheduling in parallel computer systems has been dedicated to homogeneous multiprocessors and single-cluster systems. Much less attention has been devoted to multicluster systems consisting of single clusters with fast internal communication interconnected by relatively slow links, although many such systems are in use. One such system is the Distributed ASCI Supercomputer (DAS) [7], which was designed and deployed by the Dutch Advanced School for Computing and Imaging (ASCI) in the Netherlands. In such systems, jobs may ask for, or, due to the numbers of processors in the separate clusters and the number of processors requested, need co-allocation, i.e., the simultaneous allocation of processors in more than one cluster. In this paper, we assess the performance of co-allocation in a model of multicluster systems based on the DAS, with a special emphasis on the effect of the differences in communication speeds between intracluster and intercluster links, for different (combinations of) job structures, and for a workload based on actual measurements on the DAS.

Because of the potentially large computational power they offer at a low cost, multiclusters are an attractive option. Possibly widely distributed groups of users, each with exclusive access to their own clusters of processors, may join forces in order to share the multicluster consisting of the total of the original single clusters. However, together with this advantage of a more powerful system at a lower cost, this solution also has the potential disadvantage of increased execution times of jobs co-allocated across multiple clusters, and a decreased system utilization as a consequence, due to the slow communication among the clusters.

The possibility of creating multiclusters fits in with the recent interest in computational and data GRIDs [12, 17], in which it is envisioned that applications can access resources—hardware resources such as processors, memory, and special instruments, but also data resources—in many different locations at the same time to accomplish their goal. In addition to many other problems that need to be solved in order to realize this vision, such as security issues and re-

[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.

source discovery, these GRIDs need schedulers that work across the boundaries of the domains governed by single resource managers to achieve co-allocation. Such schedulers have been called metaschedulers and superschedulers. Globus [13] is one of the software systems incorporating a co-allocation component.

In our model we specify the structure of the jobs, and the way jobs communicate this structure to the scheduler. Many scheduling strategies have been developed for parallel systems in an attempt to improve their performance, such as gang scheduling and dynamic jobs. However, a simple and yet often used and very practical strategy is to allow only 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. Therefore, we also only consider rigid jobs scheduled by pure space sharing. However, we do allow different request types in that jobs have the option to specify only the total number of processors they need, or also the numbers of processors in each of the clusters in the multicluster system. We assess the performance of FCFS scheduling depending on the amount of communication and on the ratio between the speeds of intracluster and intercluster links, for workloads consisting of a single or of a mix of different request types. The performance is evaluated in terms of the average response time as a function of the utilization.

This paper is a follow-up to our previous paper [19], in which we focused on different scheduling policies—in addition to FCFS we also considered some forms of backfilling—and on the system capacity lost due to none of the (first few) jobs in the queue fitting on the processors left idle by the jobs in execution; communication was not considered. Here we do include communication, we also perform simulations based on actual measurements of a real application on the DAS, we include a new type of requests—flexible requests which the scheduler is allowed to split up in any way across the clusters—and two different ways of scheduling these, we also consider workloads consisting of a mix of request types, and we assume a more realistic distribution of the sizes of jobs.

2 The Model

In this section we describe our model of multicluster systems. This model is based on the Distributed ASCI Supercomputer, the performance of which we intend to evaluate depending on the structure of the jobs, on the amount and pattern of communication, and on the scheduling decisions.

2.1 The Distributed ASCI Supercomputer

The DAS [6, 7] is a wide-area distributed computer system consisting of four clusters of workstations located at four Dutch universities. 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, while 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.

We have created an interface between prun and Globus, and we have installed the Globus toolkit [13] on the DAS system. However, there is also a way around using Globus for submitting multicluster jobs to the DAS, which was used for the measurements in Section 4. So far, co-allocation has not been used enough on the DAS to let us obtain statistics on the sizes of the jobs' components.

2.2 The structure of the system

We model a multicluster distributed system consisting of C clusters of processors, cluster i having N_i processors, $i = 1, \ldots, C$. We assume that all processors have the same service rate. As to the communication and the communication network, we assume that any two processors can communicate with each other, and that the communication between the tasks of a parallel application is synchronous, so that a task doing a send operation can only continue when it knows the receiving task has received the message. All intracluster communication links are assumed to be of the same capacity, as are all the intercluster links. Since LAN links are faster than WAN links, we assume the speed of intracluster links to be significantly higher than the speed of intercluster links. This implies that the amount of time needed for sending a message between two processors within a cluster is smaller than the time required to send the same message between two processors from different clusters. The two parameters related to communication in our model are the total time needed to complete a single synchronous send operation between processors in the same and in different clusters.

By a 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 that runs on a single processor. Tasks can communicate by exchanging messages over the network. Jobs are rigid, meaning that the numbers of processors requested by and allocated to a job are

fixed, and cannot be changed during their execution. Job requests can have different degrees of flexibility in the way they allow their components to be spread over the clusters (see Sect. 2.3). All tasks of a job start and end at the same time, which implies that all the processors allocated to a job are being simultaneously occupied and released. We also 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 both interarrival times and service times we use exponential distributions.

In our simulations we make the simplification that all the clusters have an equal number N of processors. In order to assess the performance loss due to the wide-area links, we also compare the performance of a multicluster system with C clusters with a single-cluster system with CN processors.

2.3 The structure of job requests

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 D we use for the sizes of these job components is defined as follows: D takes values on some interval $[n_1, n_2]$, with $0 < n_1 \leq n_2 \leq N$, 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 sum of the probabilities equals 1, and with $q = 0.95$. This distribution favours small sizes, and sizes that are powers of two, which has been found to be a realistic choice [8].

We will consider four cases for the structure of jobs, which are differentiated by the flexibility of their requests:

1. An *ordered request* is represented by a tuple of C values $(r_1, r_2, \ldots, r_C)$, each generated from distribution D. The positions of the request components in the tuple specify the clusters from which the processors must be allocated. This is the most restrictive of the request types considered.

2. An *unordered request* is again specified by a tuple of C values $(r_1, r_2, \ldots, r_C)$, each of them obtained from D, but now by these values, the job only specifies the numbers of nodes it needs in separate clusters, and not the precise clusters where the nodes must be allocated. Because it leaves the scheduler the freedom to choose the clusters on which to place each of the C components, this request type is more flexible than the previous one.

3. A *flexible request* is represented by a single number obtained as a sum of C values $r_1, r_2, \ldots, r_C$, each of them obtained from the distribution D. With this request the job specifies only the total number of processors it requires; it leaves to the scheduler the decision about how to spread the tasks over the clusters.

4. For *total requests*, there is 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 cluster number C. The distribution of the numbers of processors required by jobs is again the sum of C copies of the distribution D. We include this case in order to compare the multicluster cases above with a single-cluster case in which the total 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 the case of flexible requests in multiclusters. The speed difference between intercluster and intracluster links makes the two cases distinct also from the performance point of view. The way we determine the job component sizes in ordered and unordered requests and the total job sizes for flexible and total requests, makes the the results for the four cases comparable.

Ordered requests are used in practice when a user has enough information about the complete system to take full advantage of the characteristics of the different clusters. For example, the data available at the different clusters may dictate a specific way of splitting up an application. Unordered requests (especially when grouping request components on the same cluster would be allowed) are modeled by 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. The flexible requests are the best from the system's point of view because their lack of restrictions concerning the placement of their tasks gives the scheduler the possibility to improve the overall performance of the system.

2.4 The communication pattern

The communication between the tasks of single jobs is an important part of our model, one of our aims being to study the influence of the amount of communication on the performance of the system for the different request types, and its sensitivity to the ratio between the intercluster and intracluster speeds. As a model for the structure of jobs we consider a general parallel application that can be solved by an iterative method (it can be anything from a Poisson problem solved by an iterative method to an implementation of surface rendering using the Marching Cubes technique). Such applications have in common that the

space of the problem is divided among the processes making up the application, with each of them performing the algorithm on its subdomain, alternating computation and communication steps. The communication is necessary for example to exchange border information or to compute a global error as a stopping criterion. The communication steps realize also the synchronization between the processes.

For the simulations in Section 3, we will assume that each task does a fixed number of iterations. Each iteration consists of a communication step in which a message of a fixed size is sent to every other task, and a computation step. We assume that a task sends its messages successively and synchronously, which means that it waits until its first message has been received before it sends the second, etc., and that in the mean time, it receives all messages sent to it by the other tasks. When all these messages have been sent and received, the tasks do their computation step. Although the sizes of the messages are equal, the time costs will not be the same, depending on whether the sender and the receiver are in the same cluster or not. Since we assume that jobs release all their processors at once, the slowest task will determine the job duration.

The simulations in Section 4.2 are based on measurements of the execution of a real application on the DAS. This application has a structure that deviates somewhat from the general structure outlined above. Details on this application can be found in Section 4.1.

2.5 The scheduling decisions

In all our simulations the First Come First Served (FCFS) policy is used. FCFS is the simplest scheduling scheme, processors being allocated to the job at the head of the queue. When this job does not fit, the scheduler is not allowed to choose another job further down in the queue. This restriction has a negative influence on the maximal processor utilization, since processors may have to stay idle even when one or more jobs in the queue do fit. For ordered and total requests it is clear when a job fits or not, and there is basically only one way of allocating processors.

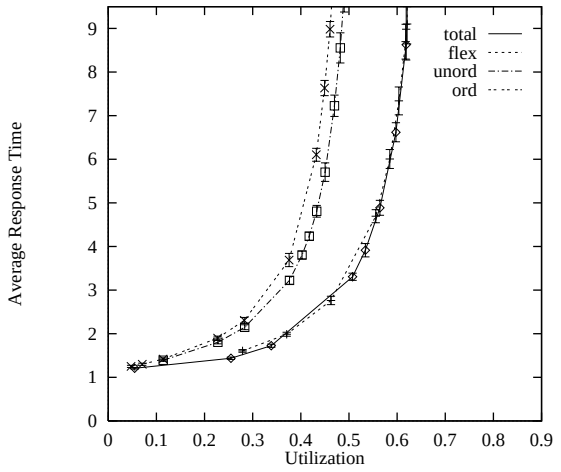
In order to determine whether an unordered request fits, one can first order the job component sizes, and then try to schedule the components in decreasing order of their sizes. Whatever way of placement is used, if placing the job components in this order does not succeed, no other order will. Possible ways of placement include First Fit (fix an order of the clusters and pick the first one on which a job component fits), Best Fit (pick the cluster with the smallest number of idle processors on which the component fits), or Worst Fit (pick the cluster with largest number of idle processors). In our simulations, we employ Worst Fit. If we consider the influence each placement has on the jobs following in the queue, Worst Fit can be expected to give better results than the other placement methods when combined with the FCFS policy, because it leaves in each cluster as much room as possible for subsequent jobs.

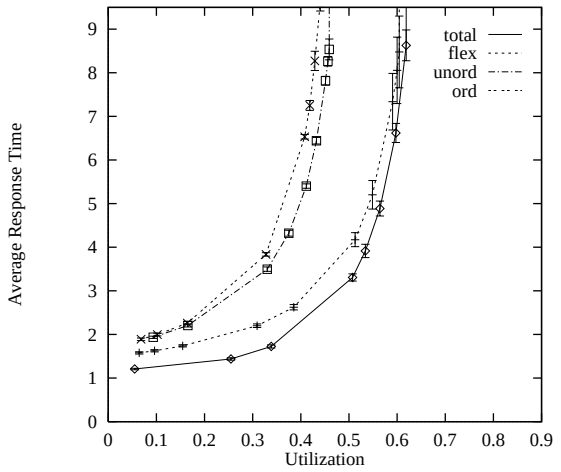
For flexible requests the scheduling algorithm first determines whether there are enough idle processors in the whole system to serve the job at the head of the queue. If so, the clusters on which the job will be scheduled are again chosen in a Worst Fit manner by taking the smallest set of clusters with enough idle processors. (One can simply order the clusters according to their numbers of idle processors, in decreasing order, and add a cluster at a time until enough idle processors are obtained.) The reason for this selection of clusters is that because of the higher cost of intercluster communication it is preferable to schedule the job on as few clusters as possible. The only decision still to be taken is how to spread the request over the selected clusters. We implemented two ways of doing so, both starting to use clusters in the decreasing order of their number of idle processors. The first method, called *cluster-filling*, completely fills the least loaded clusters until all the tasks are distributed; the second, *load-balancing*, distributes the request over the clusters in such a way as to balance the load.

Cluster-filling has the potential advantage of a smaller number of intercluster links among the tasks in a job, and so of a smaller communication overhead, while load-balancing potentially improves the performance of the system when the workload includes more types of requests. As an example, consider a flexible request of 18 processors coming in an empty system consisting of four clusters with 8 processors each. When load-balancing is used, there are 216 intercluster messages in a single all-pairs message exchange among the job's tasks, and only 192 for cluster-filling (see Figure 3). When all the requests in the system are flexible, balancing the load does not change the maximal utilization, but when combined with ordered requests for example, the advantage of having comparable loads and avoiding completely filling some clusters while leaving some others emptier is obvious. Using cluster-filling for flexible requests would be very obstructionist towards both ordered and unordered requests, and would result in a higher capacity loss.

3 Co-allocation in the presence of communication

In order to estimate the performance of multicluster systems such as the DAS, for different types of requests in the presence of communication, we modeled the corresponding queuing systems and studied their behaviour using simulations.





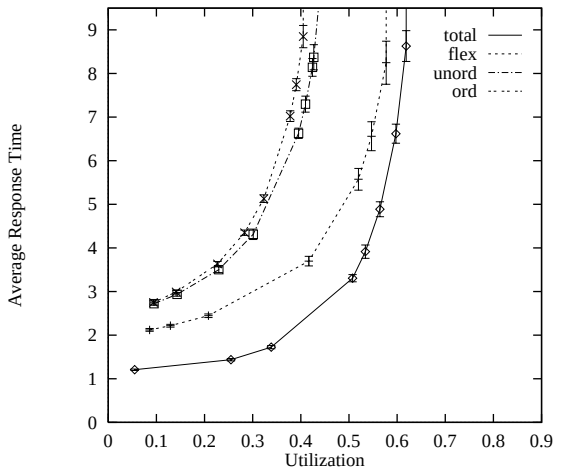
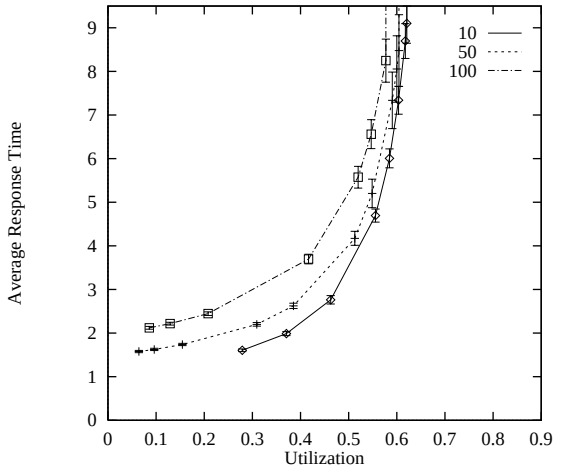
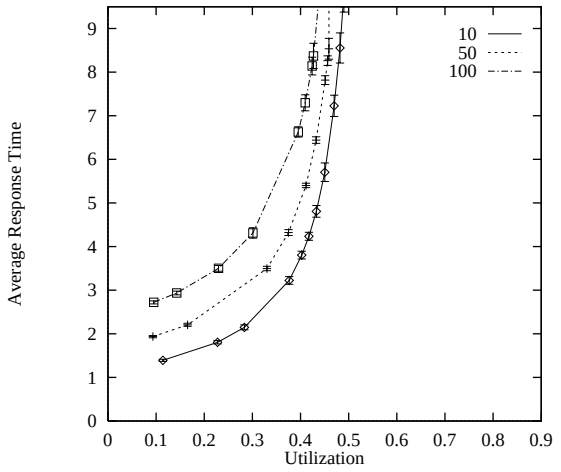


Figure 1: The average response time as a function of the utilization for the four request types and for communication speed ratios of 10 (top), 50 (middle), 100 (bottom).





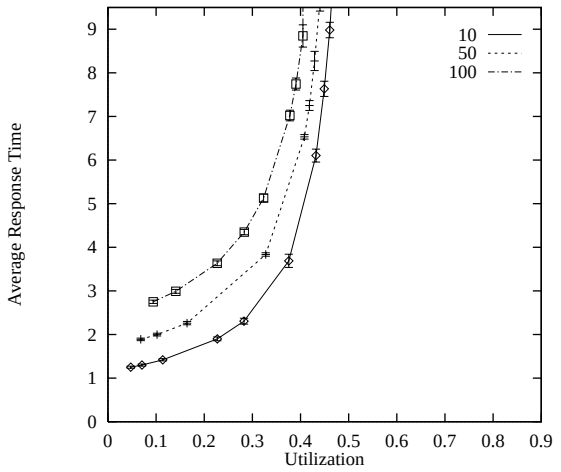


Figure 2: The average response time as a function of the utilization for three communication speed ratios, for flexible (top), unordered (middle) and ordered (bottom) requests.

The simulation programs were implemented using the CSIM simulation package [5]. Simulations were performed for a single-cluster system with 32 processors and for a multicluster system with 4 clusters of 8 nodes each. The job component sizes were generated from the distribution presented in Sect. 2.3. The sizes of the total requests in the single-cluster system with 32 processors we use for comparison, are the sum of 4 numbers obtained from the same distribution. In Section 3.4.2 we also use for the job components a uniform distribution on the interval $[1, 4]$.

In all the simulations reported in this section, the mean of the service time is equal to 1, and the interarrival time is varied in order to determine the response time as a function of the utilization of the system. We consider the performance to be better when either for the same utilization, the average response time is smaller, or when the maximum utilization is larger. The simulations for this section were done for tasks performing only a single iteration consisting of a communication step followed by a computation step, because it is the ratio of communication to computation that matters. The (deterministic) amount of time needed for the synchronous communication between two tasks in the same cluster is set to 0.001.

3.1 The influence of the ratio between intercluster and intracluster communication speed

Even more than the total amount of communication, the ratio between the speeds of intercluster and intracluster links, included in our model as the ratio between the time intervals necessary for sending the same message inside a cluster and between two processors from different clusters, is an interesting factor to consider because of its different influence on each of the distinct types of requests. Below we evaluate the influence of this ratio on the performance of our model for the different request types.

Figure 1 compares the average response time for the four types of requests, for communication ratios of 10, 50, and 100. It turns out that for ordered, unordered and flexible requests, the increase of the ratio between intercluster and intracluster communication deteriorates the performance, increasing the response time and decreasing the maximal utilization. The performance seems to be affected more for ordered and unordered requests than for flexible ones. This suggests that in those cases the average amount of intercluster communication per job is higher. The variation of the communication ratio does not affect the results for total requests since in single clusters there is only intracluster communication and they can be used as a reference in the graphs.

In Figure 2 we show the same results (except for total requests), but ordered in a different way: now we depict the performance per request type for different communication speed ratios in the same graph. It can be noticed that the change in performance is significant for all utilizations. For low utilizations, where in the absence of communication all the request types have similar response times, taking as a reference the curve of total requests, now we can see a large increase of the average response time for all the other request types. The results indicate that the communication ratio decreases the maximal utilization as well, but the deterioration is smaller than in the case of the response time. An approximation for the extra capacity loss due to communication is presented in the following section.

3.2 The extra capacity loss due to communication

In both single-cluster and multicluster systems, it may of course happen that some processors are idle while there are waiting jobs in the queue. For single clusters this happens only when the number of idle processors is smaller than the number of processors requested by the job at the head of the queue. The same is true for jobs with flexible requests in multiple clusters. For the more restrictive types of requests this phenomenon occurs more often, and a larger fraction of the processors may stay idle because even if the total number of idle processors covers the number of processors requested by the job, it may happen that their division over the clusters does not satisfy the requirements of the job. If ρ_m is the average utilization of the system such that the system is unstable at utilizations ρ with $\rho > \rho_m$ and stable for $\rho < \rho_m$, we have $\rho_m < 1$. The quantity $L = 1 - \rho_m$ is called the capacity loss of the system. Capacity loss and the utilization of high-performance computer systems has been a major concern [14]. In [19], we presented an approximation of capacity loss in single clusters, and some simulations for capacity loss in multiclusters, in the absence of communication.

In the presence of communication there is an additional component L_c of the capacity loss, which is due to the fact that jobs are not preempted during communication steps. If the queue is not empty during such a step, one or more jobs at the head of the queue might have fit on the system if we did preempt jobs when they start communicating. Such jobs might have taken part or all of the processors freed by the preempted job, in which case we talk about *internal capacity loss*, denoted by L_i. There is also an external capacity loss L_e resulting from the fact that if the processors would be released during communication, another job at the head of the queue could be scheduled, which would take the processors freed by the communicating job, but also some other processors which are idle otherwise. The idle processes which

would be occupied if the job involved in communication gave up its processors generate this additional external component to the capacity loss.

Table 1 contains the estimated values for the capacity loss due to communication L_c for different communication ratios and for all four types of requests. They are obtained by subtracting the maximal utilization in the presence of communication from the maximal utilization for a system without communication. We can conclude that the capacity loss increases with the ratio between intercluster and intracluster communication speed and it is higher for the job request types which generate more communication (a larger number of intercluster messages). For total requests in single clusters the extra capacity loss is small since all the communication is fast intracluster one.

request type	communication ratio		
	10	50	100
ord.	0.00553	0.02075	0.03100
unord.	0.00393	0.01581	0.02415
flex.	0.00013	0.00034	0.00050
total	0.00006	0.00006	0.00006

Table 1: The extra capacity loss due to communication for the four request types.

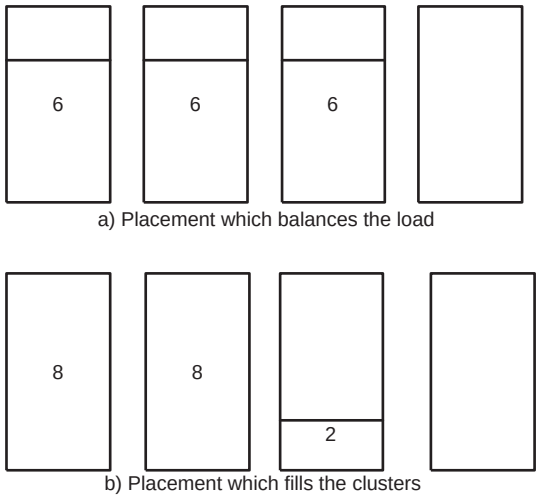


Figure 3: A numerical example comparing load-balancing and cluster-filling.

3.3 A comparison between balancing the load and filling the clusters

As we stated before, both ways of dealing with flexible requests choose the minimal number of clusters on which the request can be scheduled using Worst Fit. They differ however in the way the jobs are spread over this minimal number of clusters.

The placement which fills the least loaded clusters is aiming to reduce the number of intercluster messages taking into account that this type of communication is more costly than the intracluster one. The placement which balances the load accepts to have a larger number of intercluster messages but leaves as much room as possible in the clusters for other jobs. If there are only flexible requests in the system, filling the clusters does not affect the performance and there is no benefit from balancing the load, but if besides flexible requests the system also schedules more restrictive ones, such as ordered or unordered requests, keeping an equilibrium between the loads of the cluster becomes essential. Ordered and unordered requests specify the number of processors they need in distinct clusters and if one of those clusters is full such a job will not be scheduled even if the total number of free processors in the system is large enough. This would determine an extra capacity loss that can be avoided by an even loading of the clusters.

In our case, since jobs are rigid, even when only flexible requests are present in the system, load-balancing brings a better performance then cluster-filling. Figure 5 shows the evolution of the response time function of the utilization of the system for both techniques, for communication speed ratios of 50 and 100. Although in the system are only flexible requests, for both ratios load-balancing proves to be better. This can be explained by the fact that in the case of rigid jobs it is not the average duration of tasks which matters but the longest task because a job will only finish when its last task ends. Although cluster-filling has a smaller average for the communication time, having fewer intercluster messages per job, this communication time is not evenly spread between the tasks. This is why, on average, cluster-filling generates jobs with a higher time average spent with communication.

As an example, consider that an empty system with four clusters of eight processors each receives a flexible request asking for 18 processors. Figure 3 shows the decisions for each technique. The average communication time is in favour of cluster-filling with only 192 intercluster messages which gives an average of 10.7 intercluster messages per task compared to 12 for load-balancing. However, if we look at the longest task in both cases, which sets the amount of time spent with communication by the job, it has only 12 intercluster messages when using load-balancing, compared to 16 intercluster messages in the other case.

3.4 Performance evaluation for a mix of job request types

Real systems have to be able to schedule multiple request types and not just a single type. The overall performance of the system is determined by the way the scheduler deals with all types of requests, in such a way as to give acceptable performance to all of them and to improve the total performance. It is a fact that the different request types in the system influence each other. Results obtained for a system with just one type of job requests are not always valid also for a system where more types of requests are being submitted. In order to obtain a more general model we study a multicluster system where both flexible and ordered requests are present. To show how much the scheduling decisions for jobs with one type of request can influence the results for jobs with other requirements we compare the performance of the system when for flexible requests is chosen the way of placement which balances the load to the one when tasks are distributed to the clusters in such a way as to completely fill the least loaded ones.

We study the performance of the system for different ratios between flexible and ordered jobs. The communication ratio in all the simulations in this section is 50.

3.4.1 The influence of the ratio between ordered and flexible requests

Figure 4 shows the average response time as a function of the utilization of the system for different ratios of ordered and flexible requests. In the first graph the flexible requests were spread over the clusters in such a way as to fill the clusters, in the second one the load was balanced. The results show that the average response time grows with the increase in the percentage of ordered jobs, indifferent of the manner in which the flexible jobs are spread over the clusters. The maximal utilization of the system decreases when there is a higher percentage of ordered jobs.

At low utilizations the increase of the response time with the percentage of ordered jobs is low but it gradually grows when the utilization becomes larger. For less than 10% ordered jobs, the load is still dominated by the flexible jobs and that is why the deterioration of the performance is small. When the percentage of ordered and flexible requests is the same, the average response time is already significantly larger than for a system with just flexible requests, especially at high utilizations. When 90% or more of the job requests are ordered, in both cases the performance is determined by the ordered requests.

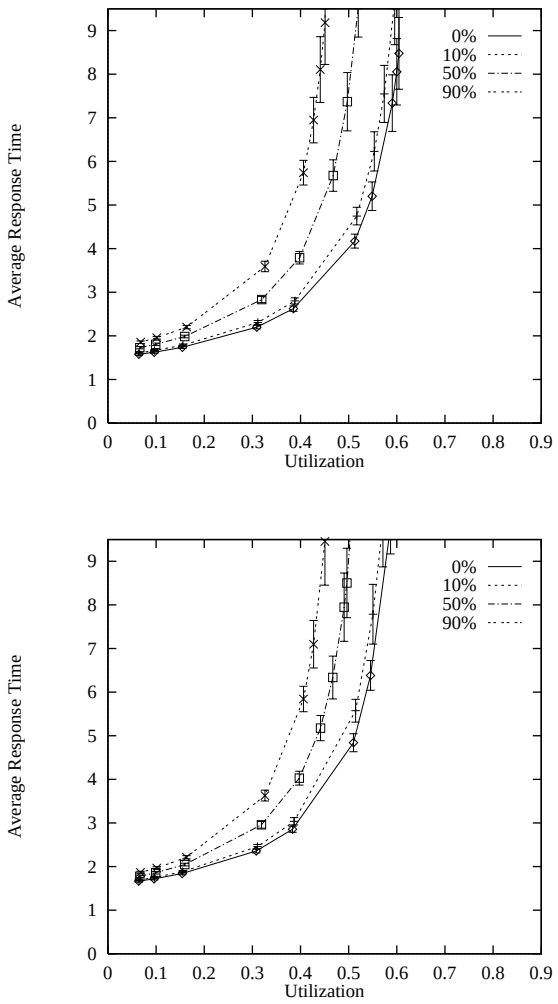
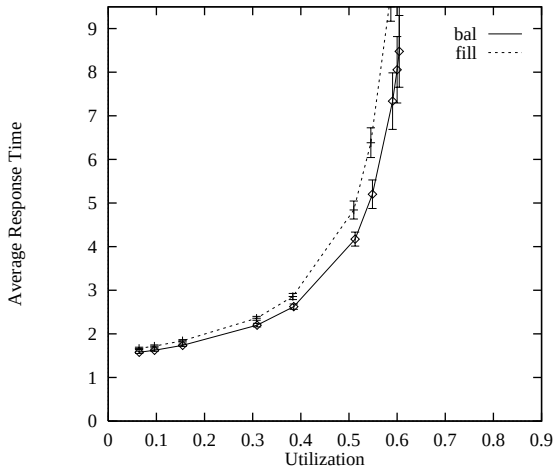


Figure 4: The performance of a system with mixed requests for different percentages of flexible and ordered requests, with load-balancing (top) and cluster-filling (bottom)

3.4.2 Load-balancing versus cluster-filling for flexible requests

In this section we evaluate the two ways of placement used for flexible requests in the more general case of a system with mixed requests. As figure 5 indicates, for up to 50% ordered jobs the system behaves better when flexible requests are placed in such a way as to balance the load. This better performance is due mostly to the better response time obtained by the flexible jobs. Because the jobs are large enough not to permit in average the presence in the system of more than two jobs simultaneously, there is little or no benefit for the ordered jobs from the fact that the load is balanced.

There are more factors which can influence the performance of the system. Both types of placement for flexible requests tend to deteriorate the perfor-



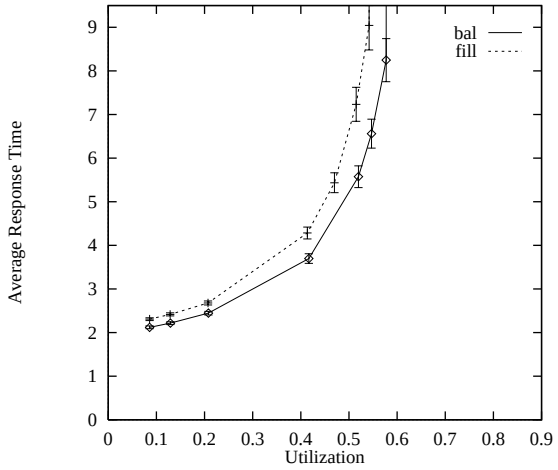


Figure 5: Performance comparison between load-balancing and and cluster-filling for flexible requests, for communication ratios of 50 (top) and 100 (bottom).

mance of ordered requests by the fact that they look for the minimum number of clusters where such a job fits. The maximal utilization and the response time could be improved by trying to spread the flexible jobs over all the clusters. However, that change would cause a deterioration of the performance for flexible requests by increasing the cost of communication. For smaller jobs using load-balancing can improve the performance of the system by leaving more room for ordered jobs and allowing them to obtain the needed resources sooner. Figure 6 depicts the variation of the response time for a job mix of 50% ordered requests and 50% flexible requests, where the components of the job requests are obtained from a uniform distribution on the interval $[1, 4]$. In this case the performance is visibly better when load-balancing is used for flexible requests.

These results indicate that the scheduling decisions must be a compromise convenient for all types of jobs present in the system.

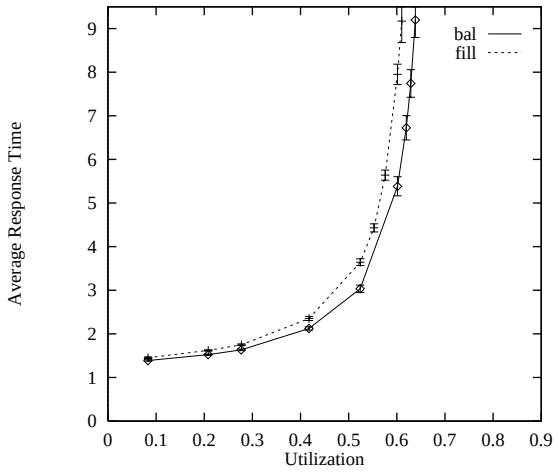


Figure 6: The performance of a system with mixed requests for equal percentages of flexible and ordered request, with components obtained from an uniform distribution on $[1, 4]$.

4 Simulations based on a real application

We extracted a more complex communication pattern from an application implementing a parallel iterative algorithm to solve the Poisson equation with a red-black Gauss-Seidel scheme. We ran the application on the DAS system and measured the durations of the different steps of the algorithm, subsequently placing these values in our simulation model for ordered and total requests, and assessing the performance of the system in terms of average response time as a function of the utilization, for this specific application.

4.1 Description of the application

The application searches for a solution in two dimensions; the computational domain is the unit square, split into a grid of points with a constant step. The number of points in the grid constitute the size of the grid. At each iteration (computation step) each grid point has its value updated as a function of its previous value and the values of its neighbours. The grid is split into "black" and "red" points, and first all points of one colour are visited followed by the ones of the other colour. With such an approach the values of the grid points are updated only twice during a sweep over all points.

The domain of the problem is split into rectangles among the participating processes. Each process gets a contiguous part of the domain containing a

number of grid points and does its own computation. However, processes must communicate in order to exchange the values of the grid points on the borders and to compute a global stopping criterion. The amount of communication, so also the communication time, is influenced by the way processes split up the domain, the number of participating processors, the size of the grid, and the initial data. In our example we chose to maintain constant the grid size and the initial data. When running the application on a multicluster system, the way processes are distributed over clusters influences the communication time, since sending an intercluster message takes significantly longer than sending an intracluster one. For the same grid size and the same initial data, we ran the application on the DAS, for different numbers of processes and for different divisions on lines and columns.

8 processes - $4x2$, $8x1$

16 processes - $2x8$, $4x4$

All the rules for rigid jobs and pure space sharing are obeyed: each process is scheduled on a distinct processor where it runs until completion, and all the processors granted to the application are simultaneously acquired and released. Since the domain is evenly split among the participating processes, the amount of computation, communication, and the number of iterations are dependent on the number of processors used to perform the algorithm. The application is implemented in MPI and has the following structure:

```
if (proc_index == 0)
{
  read the initial data;
 /* number of points in the data
 domain, number of nodes, sources */
 broadcast data to all the processes;
}
/* start computation */
do
   {
     update half of the points
          (e.g. black points);
     update the other half of the
          points (e.g. red points);
     exchange borders with the
          neighbours;
     compute local-error;
     distribute local-error;
     compute global-error;
   }
    until global-error <= limit;
  write results;
```

We ran the application on a single cluster for all the cases mentioned above, and determined the duration of the job and the number of iterations needed to reach convergence. The results are presented in Table 2.

For the cases $4x2$ and $4x4$, we also ran the application on four clusters of the DAS, scheduling an equal number of processes on each cluster. More detailed measurements for the different steps of the algorithm were done, to be used in simulations: the duration of the computation steps, the time needed for exchanging borders in both single cluster and multicluster cases, and the time cost of diffusing local errors and computing the global error. The two *update* steps were measured together and considered as a single computation step. Being a single comparison instruction, the *compute local-error* step was ignored. The last two steps were also measured together, being performed as a single MPI routine. Comparing the results from the multicluster with the single cluster case we can notice as expected that the communication time is larger when messages are sent over the intercluster links. Since the duration of the communication in general is influenced also by the way the job is spread over the clusters, and there are many other ways of spreading the tasks, we do not assume that the data extracted concerning the communication steps is valid also in other configurations. However, the duration of the computation steps does not depend on the way tasks are distributed over nodes, but only on the size of the problem domain, the number of participating nodes and the way they divide the problem domain. Since the exchange of borders produces an important part of the communication amount, and intercluster communication takes longer, we can expect that the extra-time due to communication increases with the number of borders shared by processors from different clusters.

4.2 Simulation results

The way of co-allocation provided on the DAS corresponds to the case of ordered requests from our simulations. We used this application structure and the data collected from the DAS in the simulations, replacing the distribution D of job component sizes chosen before with the two job sizes considered above: (2,2,2,2) and (4,4,4,4) and assessed the performance of the system (response time as a function of the system utilization) for ordered requests. Because the job components are equal, identical performance results would be obtained for unordered requests. Simulations were also performed for the single cluster, using the data collected for job sizes 8 and 16 (cases $4x2$ and $4x4$ respectively). The results of the simulations are presented in Figure 7. Since for the multicluster simulations the components of the requests are equal and they sum up to the two values of requests used for the single cluster case, and the total size of the multicluster is equal to the size of the single cluster, the

Config	Nr Iterations	Elapsed Time Update (ms)	Exchange Borders Single-Cluster (ms)	Exchange Borders Multicluster (ms)
4x2	2436	0.953-0.972	0.408-0.450	5.9-7.3
8x1	2418	0.970-0.994	0.260-0.315	–
4x4	2132	0.480-0.515	0.350-0.425	6.3-7.7
8x2	2466	0.470-0.525	0.337-0.487	–

Table 2: Results of the measurements

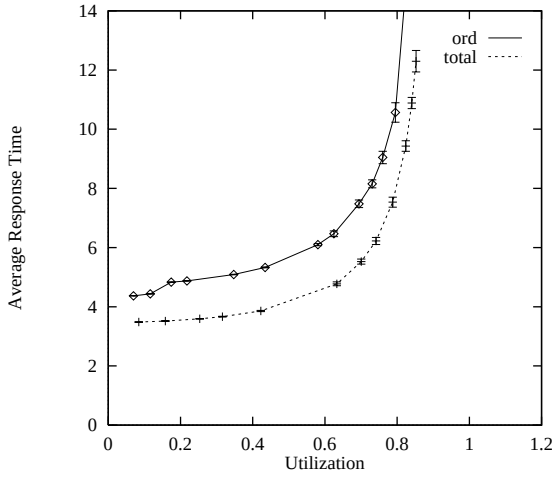


Figure 7: The average response time as a function of the utilization for ordered requests compared to total requests, with data obtained from an application implementing a parallel iterative algorithm

two cases would have identical performance in the absence of communication. This implies that the worse performance displayed by the ordered requests compared to total requests is caused by the slow intercluster communication and not by the rigidity of the ordered requests.

Only the main loop of the algorithm was simulated, the initialization and the I/O part of the algorithm being ignored. It can be noticed that the chosen algorithm has a very regular and localized communication structure (each process communicates with all his neighbours and just with them) and requires strong synchronization. Its structure is typical for parallel iterative algorithms. It is very suited to be submitted as a flexible request being advantageous to schedule such a job on the smallest number of clusters possible and minimizing the number of intercluster borders (borders shared by processors from different clusters). Because of its synchrony, the performance of a job performing such an algorithm is determined by its slowest task.

5 Related work

The problem of scheduling rigid jobs by pure space sharing in a multiprocessor system has been studied extensively; for instance, see [1]. Whereas we approach the problem of the maximal utilization from a more theoretical perspective, in [14] 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.

In [4], the influence of splitting the processors of multiprogrammed, shared-memory NUMA multiprocessors into groups on the performance of the system is studied. It is assumed that a job starts as a single process, and that it may grow by creating additional processes. The best strategy for initial placement was found to be Worst Fit, because it leaves the largest room for the growth of jobs inside a pool. In [18], a general discussion of some problems occurring in designing meta-schedulers is presented, along with a performance comparison of two such designs. In the first, parts of the systems making up the metacomputing system are dedicated to metajobs, i.e., jobs that need resources under the control of different schedulers. In the second, no resources are dedicated to metajobs, and reservations for such jobs are only made when they are submitted. Overall, the latter design yields better results, for instance, in terms of utilization. In [2], an algorithm for co-allocating a fixed set of applications is simulated. All applications together are represented by a single DAG that includes the precedence constraints and resource conflicts among all of their tasks. The aim is to find a co-allocation yielding the minimum makespan.

In [16], two multidimensional bin-packing algorithms, in which both bins and items are represented by d-dimensional vectors, are studied using simulations. This problem resembles the scheduling problem studied in this paper for ordered jobs without communication. An important element in the algorithms is the extensive search of the list of items for a

suitable candidate to place next, which is not realistic in our setting as we don't want to deviate too much from FCFS.

Finally, let us briefly mention some of the other research that is being performed in the context of the DAS project, a general overview of which can be found in [7]. 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 [15] and of the applications [3]. In [15], a library is presented with 'wide-area optimal' versions of the collective communication primitives of MPICH, a widely used version of MPI. It was shown that substantial performance improvements over MPICH are obtained for communication in wide-area systems. In [3], several techniques for optimizing algorithms on a multilevel communication structure (LAN clusters connected by a WAN, such as the DAS) were implemented and analyzed. The optimizations either reduced intercluster traffic or masked the effect of intercluster communications and caused a significant performance improvement. The authors concluded that many medium-grain parallel applications can be optimized to run well on a multilevel, wide-area system.

6 Conclusions

We modeled multicluster systems such as our DAS system, where rigid jobs are scheduled by pure space sharing, and studied its performance in the presence of communication in terms of the average response time as a function of the utilization. We distinguished four types of requests: total, flexible, unordered and ordered, and simulated workloads consisting of only a single type of requests, and of mixes of flexible and ordered requests. For flexible requests we compared the results for two ways of placement on the clusters for the case when only flexible requests are present in the system, and also for the case when they are combined with ordered requests.

Our results show that the performance of multicluster systems deteriorates with the increase of the ratio of the speeds of intercluster and intracluster communication. In addition, it turns out that the performance for flexible requests is much closer to that for total requests than to that of either ordered or unordered requests.

In both single-cluster and multicluster systems, communication introduces some extra capacity loss. This capacity loss is larger for multicluster systems and also grows with the increase of the communication speed ratio. It is also dependent on the total amount of communication, being larger for ordered and unordered requests where more intercluster messages are sent, than for flexible requests. Reducing the amount of intercluster communication may help provided that the number of intercluster messages is evenly spread among jobs. If the algorithm is modified in such a way as to have a lower average for the communication time, but some tasks of the job communicate more than before, the performance is decreased instead of being improved. Under a workload consisting of a mix of requests types, the performance is determined not only by the separate behaviours of the different types of jobs, but also by the way in which they interact. The scheduling decisions for each of them must be taken considering also the effects they have on other job types, and are a compromise among the requirements of the distinct request types.

Future work will include simulations and measurements using traces from the DAS instead of theoretical distributions, different numbers of clusters, clusters with distinct sizes, and requests with variable number of components. We also intend to extend our model of a multicluster allowing the simultaneous presence in the system of jobs requiring co-allocation and jobs asking for nodes from just one cluster. We would like to look at different communication patterns from real applications and compare their performance in the presence of co-allocation. The model with mixed requests can also be detailed in the hope to find scheduling decisions which can better satisfy the different request types.

References

[1] K.Aida, H.Kasahara and S.Narita. Job Scheduling Scheme for Pure Space Sharing Among Rigid Jobs. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1459, pages 98-121. Springer-Verlag, 1998.

[2] A.H. Alhusaini, V.K. Prasanna, and C.S. Raghavendra, A Framework for Mapping with Resource Co-Allocation in Heterogeneous Computing Systems, Proc. 9th Heterogeneous Computing Workshop (HCW2000), C.S. Raghavendra (ed.), pp. 273-286, 2000.

[3] H.E.Bal, A.Plaat, M.G.Bakker, P.Dozy and R.F.H.Hofman. Optimizing Parallel Applications for Wide-Area Clusters. In Proceedings of the 12th International Parallel Processing Symposium (IPPS'98), pages 784-790, Orlando, Fl., April 1998

[4] T.B.Brecht. An Experimental Evaluation of Processor Pool-Based Scheduling for Shared-Memory NUMA multiprocessors. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1291, pages 139-165. Springer-Verlag, 1997.

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

[6] The Distributed ASCI Supercomputer; http://www.cs.vu.nl/das/.

[7] H.E. Bal et al. The Distributed ASCI Supercomputer Project. ACM Operating Systems Review, Vol. 34(4), pages 76-96, 2000.

[8] D.G.Feitelson and L.Rudolph. Toward Convergence in Job Schedulers for Parallel Supercomputers. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1162, pages 1-26. Springer-Verlag, 1996.

[9] D.G.Feitelson and L.Rudolph. Theory and Practice in Parallel Job Scheduling. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1291, pages 1-34. Springer-Verlag, 1997

[10] D.G.Feitelson and M.A.Jette. Improved Utilization and Responsiveness with Gang Scheduling. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1291, pages 238-261. Springer-Verlag, 1997.

[11] D.G.Feitelson. Packing Schemes for Gang Scheduling. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1162, pages 89-110. Springer-Verlag, 1996.

[12] The Global Grid Forum; http://www.gridforum.org.

[13] Globus; http://www.globus.org.

[14] J.Patton Jones and B.Nitzberg. Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1659, pages 1-16. Springer-Verlag, 1999.

[15] T.Kielmann, R.F.H.Hofman, H.E.Bal, A.Plaat and R.A.F.Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99), pages 131-140, Atlanta, Ga., May 1999.

[16] W. Leinberger, G. Karypis, and V. Kumar, Milti-Capacity Bin Packing Algorithms with Applications to Job Scheduling under Multiple Constraints, Proc. 1999 Int'l Conference on Parallel Processing, D. Panda and N. Shiratori (eds), pp. 404-412, 1999.

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

[18] Q. Snell, M. Clement, D. Jackson, and C. Gregory, The Performance Impact of Advance Reservation Meta-Scheduling, In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1911, pages 137-153. Springer-Verlag, 2000.

[19] A.I.D. Bucur and D.H.J. Epema, The Influence of the Structure and Sizes of Jobs on the Performance of Co-Allocation. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1911, pages 154-173. Springer-Verlag, 2000.