The Maximal Utilization of Processor Co-Allocation in Multicluster Systems

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

Abstract

In systems consisting of multiple clusters of processors which employ space sharing for scheduling jobs, such as our Distributed ASCI[1] Supercomputer (DAS), co-allocation, i.e., the simultaneous allocation of processors to single jobs in multiple clusters, may be required. In studies of scheduling in single clusters it has been shown that the achievable (maximal) utilization may be much less than 100%, a problem that may be aggravated in multicluster systems. In this paper we study the maximal utilization when co-allocating jobs in multicluster systems, both with analytic means (we derive exact and approximate formulas when the service-time distribution is exponential), and with simulations with synthetic workloads and with workloads derived from the logs of actual systems.

1 Introduction

Over the last decade, clusters and distributed-memory multiprocessors consisting of hundreds or thousands of standard CPUs have become very popular. In addition, recent work in computational and data grids [3, 15] enables applications to access resources in different and possibly widely dispersed locations simultaneously—that is, to employ processor *co-allocation* [12]—to accomplish their goals, effectively creating single multicluster systems. Invariably, when scheduling parallel jobs in a single cluster, the utilization is well below 100% [17] even when enough load is offered to the system, because the jobs in execution may not leave enough processors idle for any eligible waiting job. This problem is of course also present when using co-allocation in multicluster systems. In this paper we study the maximal achievable utilization when using co-allocation in multicluster systems with both analytic means and with simulations.

Scheduling parallel jobs in single-cluster systems has received very much attention (see, e.g., [14]). In the most simple model, rigid jobs (which have predefined, fixed sizes) are scheduled according to the FCFS policy with space sharing, in which jobs run to completion on exclusively allocated processors. In order to improve the performance, techniques such as malleable jobs (which may vary in size over their lifetimes), gang scheduling (using time sharing across multiple processors), and different forms of backfilling (allowing certain jobs not at the head of the queue to start execution) have been devised. Because of its simplicity, we restrict ourselves in this paper in the context of multicluster systems to rigid jobs and space sharing, but we do consider a backfilling policy.

When considering co-allocation in multicluster systems, it has to be decided whether and how to spread jobs across the separate clusters. We distinguish three job request types. Ordered requests need specific numbers of processors in each of the clusters, while unorderd requests specify only these numbers and are indifferent as to the clusters in which these numbers of processors are allocated. For comparison with the single-cluster case, we introduce total requests, which only specify the total number of processors needed, equal to the number required by (un)ordered requests, in a single cluster. For the job-component sizes and the job service times, we use synthetic distributions and logs of actual systems.

An important system-oriented performance metric of parallel systems is the maximal utilization. From a practical point of view, a high utilization of real systems is often seen as a measure of successful (and profitable) operation. In [17] it is shown for real systems that FCFS yields $40 - 60\%$ utilization, that backfilling increases it by 15 percentage points, and that reducing the maximum allowable job size further increases utilization. From a theoretical perspective, a very important problem in mathematical models of parallel job scheduling is to find the values of such pa-

[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.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

rameters as the arrival rates, job sizes, and service times for which the system is stable. In this paper we are interested in both points of view.

One aspect we do not explicitly consider in this paper is the additional communication delays introduced by the relatively slow intercluster connections when jobs are spread across multiple clusters. As we define the (maximal) utilization based on the total time processors are allocated to jobs, these delays have hardly any effect on the utilization: If all jobs experience the same slowdown due to wide-area communication, then reducing the arrival rate by the same factor entails the same utilization (although, of course, a lower throughput).

An important user-oriented metric is the (mean) response time. In previous papers [8, 9, 10], we have assessed the influence on the mean response time of the job structure and size, the sizes of the clusters in the system, the ratio of the speeds of local and wide-area communications, and of the presence of a single or of multiple queues in the system. Also in [13], co-allocation (called multi-site computing there) is studied, with as performance metric the (average weighted) response time. There, jobs only specify a total number of processors, and are split up across the clusters. The slow wide-area communication is accounted for by a factor r by which the total execution times are multiplied. Co-allocation is compared to keeping jobs local and to only sharing load among the clusters, assuming that all jobs fit in a single cluster. One of the most important findings in [13] is that for r less than or equal to 1.25, it pays to use co-allocation.

Our five-cluster second-generation Distributed ASCI Supercomputer (DAS) [1, 16] (and its predecessor), which was an important motivation for this work, was designed to assess the feasibility of running parallel applications across wide-area systems [6, 18, 20]. In the most general setting, grid resources are very heterogeneous; in this paper we restrict ourselves to homogeneous multicluster systems such as the DAS. Showing the viability of co-allocation in such systems may be regarded as a first step in assessing the benefit of co-allocation in more general grid environments.

2 The Model

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

2.1 The DAS System

The DAS (in fact the DAS2, the second-generation system which was installed at the end of 2001 when the first-generation DAS1 system was discontinued) [1, 16] is a wide-area computer system consisting of five clusters of dual-processor nodes, one with 72, the other four with 32 nodes each. The clusters are interconnected by the Dutch university backbone for wide-area communications, while for local communications inside the clusters Myrinet LANs are used. The system was designed for research on parallel and distributed computing. On single DAS clusters the PBS scheduler [5] is used, while jobs spanning multiple clusters can be submitted with Globus [4].

2.2 The Structure of the System

We model a multicluster system consisting of C clusters of processors, cluster i having N_i processors, $i = 1, \ldots, C$. We assume that all processors have the same service rate. By a job we understand a parallel application requiring some number of processors, possibly in multiple clusters (*co-allocation*). Jobs are rigid, so the numbers of processors requested by and allocated to a job are fixed. We call a task the part of a job that runs on a single processor. We assume that jobs only request processors and we do not include in the model other types of resources.

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. A job is represented by a tuple of C values, each of which is generated from a synthetic distribution or from a log (see Section 2.5), or is of size zero. We will consider three cases for the structure of job requests:

1. In an *ordered request* the positions of the request components in the tuple specify the clusters from which the processors must be allocated.

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 for the components.

3. For *total requests*, there is a single cluster and a request specifies the single number of processors needed, which is obtained as the sum of the values in the tuple.

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. Unordered requests model applications like FFT, which needs few data, and in which tasks in the same job component share data and need intensive communication, while tasks from different components exchange little or no information.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

2.4 Placement and Scheduling Policies

For ordered and total requests it is clear when a job fits or not. To determine whether an unordered request fits, we try to schedule its components in decreasing order of their sizes on distinct clusters. If placing them in this order does not succeed, no other order will. Possible ways of placement include First Fit (FF; fix, once and for all, 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).

We assume that in the system there is a single global queue. As the scheduling policy we use First Come First Served (FCFS), and Fit Processors First Served (FPFS). In FPFS, when the job at the head of the queue does not fit, the queue is scanned from head to tail for any jobs that may fit. To avoid starvation, we introduce a paramater MaxJumps specifying the maximal number of times a job can be overtaken, and maintain a counter indicating this number for each job. Obviously, this counter is non-increasing in the queue from head to tail. FPFS is a variation on backfilling [19], for which it is usually assumed that (estimates of) the service times are available before jobs start, and in which the job at the head of the queue may not be delayed by jobs overtaking it.

2.5 The Workload

For the workload, we have to specify the arrival process, the sizes of the (components of the) jobs, and the service times of the jobs. In our analysis in Section 4 and in some of our simulations, we don't have to specify the arrival process; when we do have to specify it, we assume it to be Poisson. For both job sizes and service times, we use either synthetic distributions, or logs from actual systems, amongst which the DAS.

Job Sizes

The basic synthetic distribution of the sizes of the job components is either the uniform distribution $U[n_1, n_2]$ on some interval $[n_1, n_2]$ with $0 < n_1 \leq n_2$, or the distribution $D(q)$ defined as follows: $D(q)$ takes values on some interval $[n_1, n_2]$ with $0 < n_1 \leq n_2$, and the probability of having job-component size i is $p_i = q^i/Q$ if i is not a power of 2 and $p_i = 3q^i/Q$ if i is a power of 2, with Q such that the p_i sum to 1. This distribution favours small sizes, and sizes that are powers of two, which has been found to be a realistic choice [11]. In some cases we use the sum of multiple copies of the distributions defined above for job-component sizes. For ordered jobs, we always assume that the non-zero job-component sizes are independent. For unordered jobs and equal cluster sizes (say N) we also consider dependent job-component sizes; in that case, for each job we first generate its total size, say J, and then split into the smallest possible number of components ($\lceil J/N \rceil$) of equal size (plus or minus one).

For the job sizes of actual systems, we use the log of the Cornell Theory Center (CTC) of Feitelson's parallel workload archive [2], and a log of a three-month period of the largest cluster (with 128 processors) of the first-generation DAS1 system. When considering multicluster systems of total size 128 in Section 7, we leave out all jobs of larger size from the CTC log. Although co-allocation is (was) possible on the DAS2 (DAS1), so far it has not been used enough to let us obtain statistics on the sizes of the jobs' components. As both logs are for single-cluster systems, they only contain the total sizes of jobs. Statistics of the two logs are presented in Table 1 (cv stands for coefficient of variation), and the density of the DAS1 sizes, which also has a preference for small numbers and powers of two, is presented in Figure 1. When using these logs, we only consider unordered jobs, clusters of equal size, and generate the (dependent) job-component sizes as described above.

job-size statistics			
	no. of jobs	mean	cv
CTC	79,302	10.72	2.26
CTC-128	78,865	9.53	1.83
DAS1	30,558	23.34	1.11

Table 1. Statistics of the job-size distributions derived from the logs (in CTC-128, only jobs of size at most 128 are included).

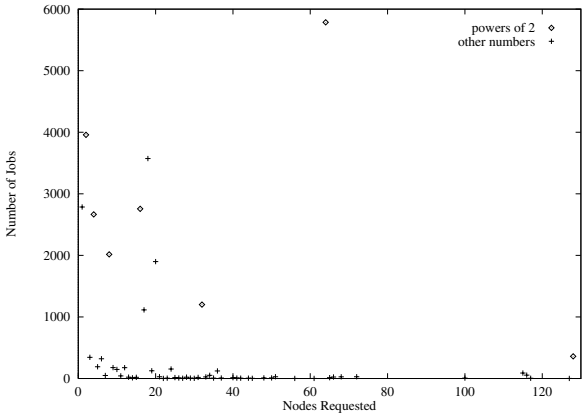

Figure 1. The density of the job-request sizes for the largest DAS1 cluster (128 processors).

Job Service Times

The synthetic job service-time distributions we use are the deterministic, the exponential, and the hyperexponential distributions. For data from actual systems, we again use the CTC and DAS1 logs. In the DAS1 log, $28,426$ jobs were recorded with both their starting and ending times, and so we could compute their service times. Table 2 gives statistics of the service times in the logs, and Figure 2 presents the density of the DAS1 service times up to 900 seconds, which is the maximal runtime during the day.

service-time statistics			
	no. of jobs	mean	cv
CTC	$79,302$	$10,983.42$	1.65
CTC-128	$78,865$	$11,006.09$	1.65
DAS1-st	$28,426$	356.45	5.37
DAS1-900	$26,850$	62.66	2.05

Table 2. Statistics of the service-time distributions derived from the logs (in DAS1-st, only the jobs whose service times can be computed are included, and DAS1-900 only contains jobs with service times at most equal to 900 seconds).

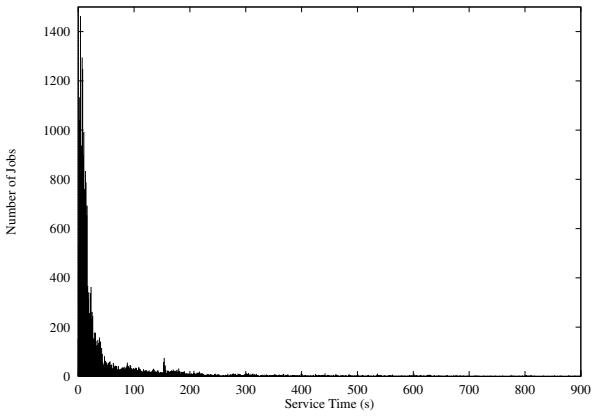

Figure 2. The density of the service times for the largest DAS1 cluster.

3 Reasons for Capacity Loss

In the model described in Section 2, processors may be idle even while there are waiting jobs because the job at the head of the queue does not fit (FCFS), or because no job further down the queue eligible for scheduling fits (FPFS). 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 $l = 1-\rho_m$. In this section we discuss the reasons for capacity loss; at least four such reasons can be distinguished in multicluster systems with space sharing and rigid jobs.

First, it may be due to the structure of job requests. Ordered requests will obviously entail a higher capacity loss than unordered ones, which in turn yield a higher capacity loss than total requests.

Second, it may be due to the distribution of the job-component sizes. With ordered requests, the capacity loss l can be very high, to the point that it approaches 1 for large cluster sizes and large numbers of clusters, as can be seen in the following somewhat pathological case. When in a multicluster with C clusters of size N, all jobs have $\lceil (N+1)/2 \rceil$ tasks in cluster 1 and only one task in all the other clusters, l is close to $(C-0.5)/C$ for large N, which is arbitrarily close to 1 for large C.

Third, the scheduling policy employed may cause capacity loss. The job at the head of the queue may not fit, while some job further down the queue does fit, and so, a policy that deviates from the arrival order may have a lower capacity loss than FCFS.

A fourth reason for having $\rho_m < 1$ is that we are considering an *on-line* problem, taking scheduling decisions without knowledge of future job arrivals or service times. Such knowledge might be exploited by a policy that deviates from the arrival order.

4 Formulas for FCFS

In this section we first present an expression for the average maximal Multi-Programming Level (MPL) in multicluster systems with the FCFS policy and 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 FCFS, unordered requests, and WF component placement, which we validate with simulations in Section 6. Finally, we consider the maximal utilization for large numbers of clusters and ordered requests. In this section we assume that the service-time distribution is exponential.

4.1 Ordered requests

In this section we assume that requests are ordered. Let F be the (multidimensional) job-size distribution, which

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

(allowing components of size zero) is defined on the set

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

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

In our treatment of multiclusters with ordered requests below we follow [7]. 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 $(z_1, \ldots, z_B)$, 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 $\overline{Z} = (\overline{z}_1, \overline{z}_2, \ldots, \overline{z}_B)$ 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 $\overline{Z}$ 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 [7] explained above can be extended in a straightforward way to our situation—the important 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

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

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:

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

which can be written as

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

For single clusters this expression coincides with the formula in [7], p. 468. Denoting by s be the average total job size, the maximal utilization is given by

$$\rho_m = \frac{M \cdot s}{\sum_i N_i}. \tag{3}$$

There is one other case of some interest in which the maximal utilization can be derived easily, namely when the service time is deterministic and the system has a slotted behaviour, which means that all jobs being served simultaneously start and finish at exactly the same time. Of course, if the latter happens once, it will happen forever. This behaviour can come about when there are two possible job sizes that exclude each other, that is, the sum of their sizes in some dimension i exceeds N_i. Then there is a positive probability that two consecutive jobs have these sizes, so the second (and perhaps some other jobs) can only start when the first completes. From then onwards, in each slot the system is filled with random jobs until the next one does not fit. So the distribution of the sizes of the jobs in service corresponds to the general job-size distribution, and we find for the maximum time-average MPL M:

$$M = \sum_{i=1}^{B}(F^{(i)}(\overline{N}) - F^{(i+1)}(\overline{N})) \cdot i. \tag{4}$$

Of course, Eq. (3) also holds here.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

4.2 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. The job-size distribution F is now defined on

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

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 $\overline{s} = (s_1, s_2, \ldots, s_C)$ the reversed vector $rev(\overline{s})$ be defined as $rev(\overline{s}) = (s_C, s_{C-1}, \ldots, s_1)$, and the ordered vector $ord(\overline{s})$ as the vector with the elements of $\overline{s}$ 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 h defined by:

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

Then, putting $F^{[2]} = F \star F$, we define inductively $F^{[i]} = F^{[i-1]} \star F$ for $i = 3, 4, \ldots$. 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)):

$$M = \frac{1}{1 - \sum_{i=2}^{B} F^{[i]}(\overline{N})/(i(i-1))}, \qquad (5)$$

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).

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.

5 Simulation Methods

Unfortunately, when the job-component sizes are dependent, the computation of Eq. (2) for more than four clusters is very time-consuming; the same always holds for Eq. (5), whether job-component sizes are independent or not. In addition, we would like to validate our approximation of the maximal utilization for unordered jobs with the FCFS policy. Finally, in some cases, such as for non-exponential service times and for the FF placement policy of unordered jobs, our formulas and approximations do not apply. For these three reasons, we resort to (two types of) simulations.

In the first type of simulations, for FCFS with synthetic service-time distributions, we simulate the complete queueing model with Poisson arrivals, and we take the utilization in these simulations that yields an average response time of at least $1,500$ time units (the average service time is 1 time unit). When simulating a queueing model close to its maximal utilization, it is very difficult to find out whether the simulation is still in its transient phase, and programming difficulties like running out of space for datastructures such as job queues may arise. However, 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) (see Section 6).

In the second type of simulations, when the scheduling policy is FCFS, we simulate the system in heavy traffic in that we suppose that the queue is always long enough when a job departs that we can schedule jobs until the next one does not fit. As we then only generate new jobs at departure times, and only one more than fits on the system, we don't encounter the difficulties mentioned above when simulating a queueing system close to its maximal utilization. Of course, we cannot use this method with FPFS because then we want to search the queue for jobs that fit. However, in an adaptation for FPFS of this method, we do model Poisson arrivals, and simulate the system for $100,000$ arrivals. We then take the arrival rate (and so the utilization) for which the queue length doesn't hit either the value of 0 or of 1000 for the largest number of arrivals.

6 The Accuracy of the Approximation and the Simulations

In this section we assess the accuracy of the approximation of Eqs. (5) and (3) of the capacity loss for unordered

Table 3. 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 total requests, with job-component-size distribution $U[n_1, n_2]$.

job comp. size distr.		capacity loss						
		$C = 1$		$C = 4$				
				ordered		unordered		total
n_1	n_2	exact	simulation	exact	simulation	approximation	simulation	exact
1	4	0.032	0.033	0.149	0.150	0.050	0.053	0.038
1	5	0.043	0.044	0.176	0.176	0.065	0.067	0.047
1	13	0.139	0.139	0.345	0.344	0.187	0.192	0.120
1	16	0.169	0.169	0.380	0.379	0.233	0.239	0.148
4	5	0.051	0.052	0.111	0.111	0.043	0.048	0.043
4	13	0.145	0.145	0.302	0.301	0.186	0.188	0.149
4	16	0.174	0.175	0.337	0.337	0.250	0.255	0.167
5	13	0.149	0.150	0.292	0.292	0.170	0.175	0.146
5	16	0.177	0.178	0.321	0.321	0.260	0.260	0.186
13	16	0.094	0.095	0.094	0.094	0.094	0.094	0.094

jobs, and of the simulation methods presented in Section 5. In this section, all clusters are of size of 32, all job-component sizes are non-zero and mutually independent, the scheduling policy is FCFS, for unordered jobs the placement policy is WF, and the service-time distribution is exponential.

In Tables 3 and 4 we show both exact (i.e., derived from Eq. (2)), approximation (i.e., derived from Eq. (5)), and simulation results for a single cluster and for multicluster systems with ordered, unordered, and total requests for different distributions of the job-component sizes. The simulation results for a single cluster in both tables, and those for unordered jobs in the multicluster in Table 3, have been obtained with simulations of type one; the remainder have been obtained with simulations of the second type. The exact and simulation results for the single cluster and for the multicluster with ordered jobs agree extremely well; the match of the approximation and simulations for unordered jobs is quite reasonable.

As an aside, the results for multiclusters show the reduction in capacity loss when going from ordered to unordered to total requests.

7 Results

In this section we present results for the capacity loss as it depends on many parameters. The results in this section for ordered and total requests when the service time is exponential and the scheduling policy is FCFS, are obtained with the formulas of Section 4; all remaining results have been obtained with simulations of the second type type as described in Section 5. Unless otherwise specified, all our results are for a multicluster system consisting of 4 clusters of 32 processors each (or, when considering total requests, for a single cluster of size 128), for independent non-zero job-component sizes, for exponential service times, for the FCFS policy, and for the WF placement policy for unordered jobs.

7.1 The Influence of the Job-Size and Service-Time Distribution

We consider the five distributions for the sizes of the job components described in Table 5. Note that the first three and the latter two have almost identical means.

distribution	mean	cv
$U[1, 7]$	4.000	0.500
$D(0.9)$ on $[1, 8]$	3.996	0.569
$D(0.768)$ on $[1, 32]$	3.996	0.829
$U[1, 14]$	7.500	0.537
$D(0.894)$ on $[1, 32]$	7.476	0.884

Table 5. The means and the coefficients of variation of the distributions of the job-component sizes.

As expected, in Figure 3 we find that the capacity loss decreases when going from ordered to unordered (except for $D(0.894)$, roughly speaking, the capacity loss is cut in half), and from unordered to total requests. In addition, when the mean, the coefficient of variation, or the maximum of the job-component-size distribution is larger (these are not independent), the performance is poorer.

In Figure 4 we compare the performance for unordered jobs with independent and dependent job-component sizes;

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

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

job comp. size distr.	capacity loss						
	$C = 1$		$C = 2$				
			ordered		unordered		total
q	exact	simulation	exact	simulation	approximation	simulation	exact
0.95	0.293	0.295	0.457	0.455	0.397	0.397	0.261
0.90	0.249	0.251	0.426	0.425	0.339	0.340	0.190
0.85	0.188	0.188	0.359	0.356	0.254	0.258	0.135
0.80	0.134	0.135	0.289	0.287	0.181	0.187	0.098
0.75	0.097	0.097	0.234	0.231	0.133	0.139	0.076
0.70	0.073	0.074	0.194	0.192	0.102	0.109	0.061
0.65	0.057	0.058	0.166	0.163	0.081	0.088	0.051
0.60	0.046	0.047	0.144	0.141	0.067	0.073	0.043
0.55	0.038	0.039	0.127	0.124	0.056	0.062	0.038
0.50	0.032	0.032	0.113	0.111	0.048	0.054	0.033

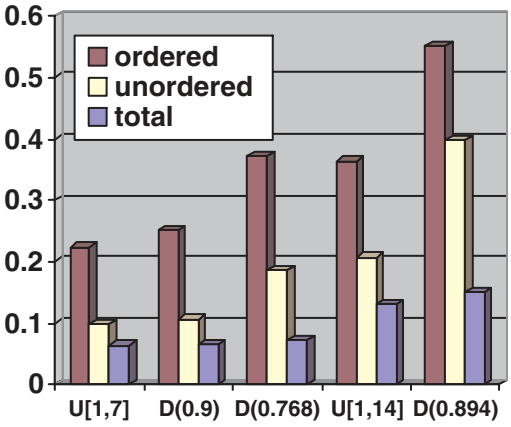

Figure 3. The capacity loss depending on the job-component-size distribution.

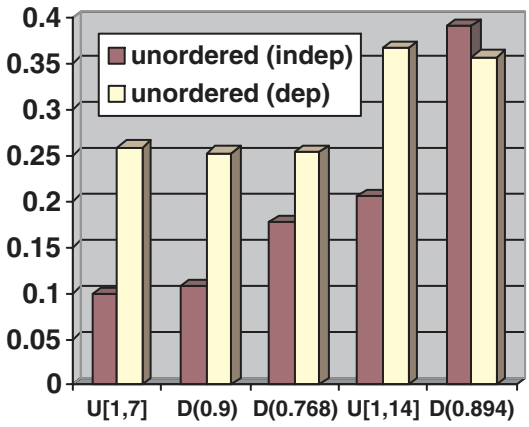

Figure 4. The capacity loss for unordered jobs with (in)dependent job-component sizes.

in the latter case we first generate the total job size as the sum of four copies of the component-size distribution, and then spread a job across the system as explained in Section 2.5. We find that when the maximum of the basic distribution used is low ($U[1, 7]$, $D(0.9)$, and $U[1, 14]$) or when large values hardly ever occur ($D(0.768)$), dependent component sizes lead to poorer performance. The reason for this is that then the job components are much larger than when they are independent. When the maximum of the basic distribution is higher and larger values are more common, as is the case for $D(0.894)$ on $[1, 32]$, this behaviour is reversed. In addition, for dependent component sizes, only the mean, and not the distribution, matters.

We varied the service-time distribution from deterministic through exponential to hyperexponential with a cv of 3 up to 10 (for the capacity loss, the mean is immaterial). We found that this distribution doesn't have a large impact on the capacity loss. For ordered (unordered) jobs it went from 0.239 (0.103) for deterministic to 0.262 (0.117) for hyperexponential with a cv of 10, which means about a 10% (14%) increase. The job-component-size distribution has an impact on the widths of holes in a schedule (i.e., the number of idle processors), while the service-time distribution has consequences for the lengths of these holes (i.e., the time processors are left idle). Apparently, the former is much more important than the latter.

7.2 The Influence of the Scheduling and Placement Policies

In this section we first assess to what extent FPFS increases the maximal utilization compared to FCFS, see Figure 5. We consider unordered jobs and the two service-time distributions from Section 7.1 for which there is the most room for improvement. We find that a small value for MaxJumps (say 5) does not improve performance by much, but that a larger value (here 50) yields a decrease of capacity loss over FCFS of about 10 percentage points, an improvement similar to that found in practice from backfilling [17].

The results of comparing the placement policies FF and WF for unordered jobs are presented in Figure 6. The difference in performance is not very large; apparently, whereas the explicit aim of WF is to keep balanced loads in the clusters, FF achieves the same goal.

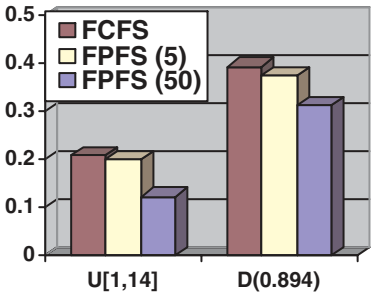

Figure 5. The capacity loss when using FCFS or FPFS with different values for MaxJumps for unordered jobs.

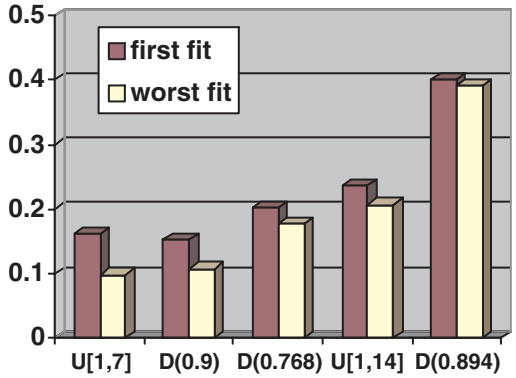

Figure 6. The capacity loss depending on the placement policy for job components for unordered jobs.

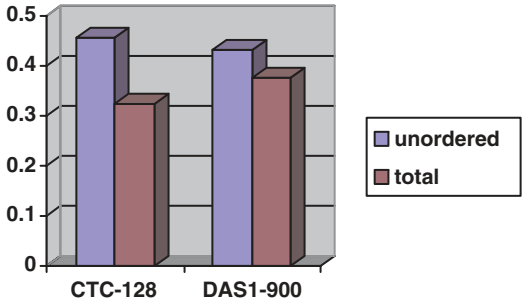

Figure 7. The capacity loss for the two logs.

7.3 Using Data from the Logs

In Figure 7 we show the capacity loss for unordered and total requests when using both the sizes and the service times as they appear in the logs of the CTC (using only the jobs of sizes at most equal to 128) and the DAS1 (see Section 2.5). From the CTC log, we use the pairs consisting of the sizes and service times of the jobs as they appear in the log, while for the DAS we sample the distributions of the sizes and the service times as they appear in the log independently. For the total jobs, we simply use a single cluster of size 128 and the job sizes from the logs; for unordered requests, we split up jobs (with dependent component sizes) as explained in Section 2.5. The capacity losses for the two logs are similar, even though the statistics of the logs are different (see Tables 1 and 2).

7.4 Co-allocation versus no Co-allocation

In this section we discuss the benefit of co-allocation over no co-allocation when there is a choice, that is, when all jobs can be scheduled without co-allocation on a single cluster. We consider a multicluster with two cases of unordered jobs (and WF placement). In either case, the (total) request sizes are obtained as the sum of 4 components generated from $D(0.9)$ on $[1, 8]$, but in one case, there are four job components, while in the other, the numbers are added to have a single job component. As the results in Figure 8 indicate, rather than waiting for enough idle processors in one cluster, spreading requests over multiple clusters brings better performance. However, we do point out that with co-allocation, jobs will experience a slowdown due to wide-area communications, which is not accounted for here.

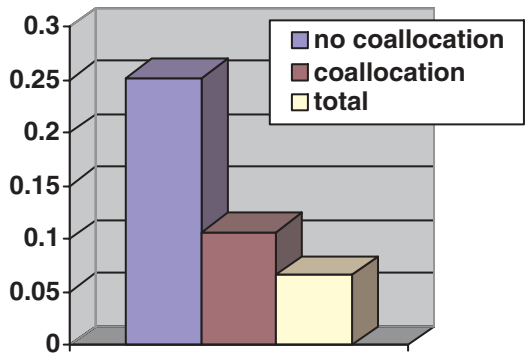

Figure 8. A comparison of the capacity loss when co-allocation is or is not employed; with co-allocation, jobs requests are unordered.

8 Conclusions

We have studied co-allocation in multicluster systems with both analytic means and with simulations for a wide range of parameters. We have derived an exact (for ordered requests) and an approximate (for unordered requests) formula for the maximal utilization when the service times are exponential, and we have shown with simulations that the latter is quite accurate. Our main conclusions are 1) that the capacity loss of co-allocation can be quite large, but 2) that for unordered jobs with independent job-component sizes co-allocation may be an attractive option, even when all jobs fit in a single cluster, and 3) that the job-size distribution is more critical for the capacity loss than the service-time distribution. We remark that while scheduling in single clusters has received an enormous amount of attention, hardly any work on co-allocation has been performed so far, and that a much more detailed study is called for.

References

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

[2] *Feitelson's Parallel Workload Archive*. `www.cs.huji.ac.il/labs/parallel/workload`.

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

[4] *Globus*. `www.globus.org`.

[5] *The Portable Batch System*. `www.openpbs.org`.

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

[7] R. Bryant. Maximum Processing Rates of Memory Bound Systems. *J. of the ACM*, 29:461–477, 1982.

[8] A. Bucur and D. Epema. The Influence of the Structure and Sizes of Jobs on the Performance of Co-Allocation. In D. Feitelson and L. Rudolph, editors, *6th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *LNCS*, pages 154–173. Springer-Verlag, 2000.

[9] A. Bucur and D. Epema. The Influence of Communication on the Performance of Co-Allocation. In D. Feitelson and L. Rudolph, editors, *7th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *LNCS*, pages 66–86. Springer-Verlag, 2001.

[10] A. Bucur and D. Epema. Local versus Global Queues with Processor Co-Allocation in Multicluster Systems. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *8th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2537 of *LNCS*, pages 184–204. Springer-Verlag, 2002.

[11] 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.

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

[13] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On Advantages of Grid Computing for Parallel Job Scheduling. In *2nd IEEE/ACM Int'l Symposium on Cluster Computing and the GRID (CCGrid2002)*, pages 39–46, 2002.

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

[15] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

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

[17] J. P. Jones and B. Nitzberg. Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization. In D. Feitelson and L. Rudolph, editors, *5th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1659 of *LNCS*, pages 1–16. Springer-Verlag, 1999.

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

[19] D. Lifka. The ANL/IBM SP Scheduling Systems. In D. Feitelson and L. Rudolph, editors, *1st Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *LNCS*, pages 295–303. Springer-Verlag, 1995.

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