A Measurement-Based Simulation Study of Processor Co-Allocation in Multicluster Systems

S. Banen, 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 interconnected by relatively slow connections such as our Distributed ASCI Supercomputer (DAS), applications may benefit from the availability of processors in multiple clusters. However, the performance of single-application multicluster execution may be degraded due to the slow wide-area links. In addition, scheduling policies for such systems have to deal with more restrictions than schedulers for single clusters in that every component of a job has to fit in separate clusters. In this paper we present a measurement study of the total runtime of two applications, and of the communication time of one of them, both on single clusters and on multicluster systems. In addition, we perform simulations of several multicluster scheduling policies based on our measurement results. Our results show that in spite of the fact that inter-cluster communication is much slower then intra-cluster communication, the performance of multicluster operation can be very reasonable compared to single-cluster execution.

1 Introduction

Over the last decade, clusters and distributed-memory multiprocessors consisting of hundreds or thousands of standard CPUs have become very popular. Compared to single-cluster systems, multicluster systems consisting of multiple, geographically distributed clusters interconnected by a relatively slow wide-area network can provide a larger computational power. Instead of smaller groups of users with exclusive access to their single clusters, larger groups of users can share the multicluster, potentially leading to lower turn-around times and a higher utilization, and making larger job sizes possible. One such multicluster system is the Distributed ASCI Supercomputer (DAS) [1], which was designed and deployed by the Dutch Advanced School for Computing and Imaging (ASCI) in the Netherlands. The possibility of creating multiclusters fits with the recent interest in computational and data GRIDs [2, 12], 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 goals.

There are two potential problems when employing multicluster systems. First, applications may not be suitable for multicluster execution because they can not deal very well with the slow wide-area links. Second, scheduling a multicomponent application across a multicluster system (i.e., performing *co-allocation*) meets with more restrictions than scheduling a job in a single cluster because now each of the components has to fit in a separate cluster. In this paper we first investigate and compare the total runtimes of single-cluster and multicluster execution of two parallel applications modeling physical phenomena by performing measurements on the DAS. Our main conclusion is that both applications, with appropriate parameter settings in one of them, are very well suited for multicluster operation. Subsequently, we assess the performance of several scheduling policies for co-allocation in multiclusters with simulations using the runtime measurements. We have also performed detailed measurements of the time spent in communication of one of the two applicatons. Because the results of these measurements are not used in our simulations, they are relegated to an appendix.

In previous papers [6, 7, 9], 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 [11], 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 [11] is that for r less than or equal to 1.25, it pays to use co-allocation. In [10], we consider the maximal utilization, i.e., the utilization at which the system becomes saturated, as a performance metric.

Our five-cluster second-generation Distributed ASCI Supercomputer (DAS) [1, 13] (and its predecessor), which was an important motivation for this work, was designed to assess the feasibility of running parallel applications across wide-area systems [5, 14, 17]. 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 system model

In this section we describe our model of multicluster systems and the scheduling policies we will evaluate.

2.1 The Distributed ASCI Supercomputer

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) is a wide-area computer system consisting of five clusters (one at each of five universities in The Netherlands, amongst which Delft University of Technology) of dual-processor nodes, one with 72, the other four with 32 nodes each. Each node contains two 1-Ghz Pentium-IIIs and at least 1GB RAM. The clusters are interconnected by the Dutch university backbone for wide-area communications (100 Mbit/s), while for local communications inside the clusters Myrinet LANs are used (1,200 Mbit/s). The system was designed for research on parallel and distributed computing. On single DAS clusters the PBS [4] scheduler is used, while jobs spanning multiple clusters can be submitted with Globus [3]. The current version of Globus is unable to use the fast local DAS interconnect (Myrinet); all Globus communication goes over TCP/IP sockets (this problem will be solved in the near future).

2.2 The structure of the system

We model a multicluster system consisting of C clusters of processors, of possibly different sizes. 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 and the placement policies

Jobs that require co-allocation have to specify the number and the sizes of their components, i.e., of the sets of tasks that have to go to the separate clusters. A job is represented by a tuple of C values, at least one of which is strictly positive. We consider *unordered requests*, for which the components of the tuple specify the numbers of processors the job requires in the separate clusters, allowing the scheduler to choose the clusters for the components. Such requests model applications like FFT, where tasks in the same job component share data and need intensive communication, while tasks from different components exchange little or no information. To determine whether an unordered request fits, we try to schedule its components in decreasing order of their sizes on distinct clusters. We use Worst Fit (WF) to place the components on clusters.

2.4 The scheduling policies

In a multicluster system where co-allocation is used, jobs can be either single-component or multi-component, and in a general case both types are simultaneously present in the system. A scheduler dealing with the first type of jobs can be local to a cluster and does not need any knowledge about the rest of the system. For multi-component jobs, the scheduler needs global information for its decisions.

Treating both types of jobs equally or keeping single-component jobs local and scheduling only multi-component jobs globally over the entire system, having a single global scheduler or schedulers local to each cluster, all these are decisions that influence the performance of the system. In [9] we have studied several policies, some of which with multiple variations; in this paper we consider the following approaches:

1. **[GS]** The system has one *global scheduler* with one global queue, for both single- and multi-component jobs. All jobs are submitted to the global queue. The global scheduler knows at any moment the number of idle processors in each cluster and based on this information chooses the clusters for each job.

2. **[LS]** Each cluster has its own *local scheduler* with a local queue. All queues receive both single- and multi-component jobs and each local scheduler has global knowledge about the numbers of idle processors. However, single-component jobs are scheduled only on the local cluster. The multi-component jobs are co-allocated over the entire system. When scheduling is performed all enabled queues are repeatedly visited, and in each round at most one job from each queue is started. When the job at the head of a queue does not fit, the queue is disabled until the next job departs from the system. At each job departure the queues are enabled in the same order in which they were disabled.

3. **[LP]** The system has both a global scheduler with a global queue, and local schedulers with local queues. Multi-component jobs go to the global queue and are scheduled by the global scheduler using co-allocation over the entire system. Single-component jobs are placed in one of the local queues and are scheduled by the local scheduler only on its corresponding cluster. The *local* schedulers have *priority*: the global scheduler can schedule jobs only when at least one local queue is empty. When a job departs, if one or more of the local queues are empty both the global queue and the local queues are enabled. If no local queue is empty only the local queues are enabled and repeatedly visited; the global queue is enabled and added to the list of queues which are visited when at least one of the local queues gets empty. When both the global queue and the local queues are enabled at job departures, they are always enabled starting with the global queue. The order in which the local queues are enabled does not matter since the jobs in them are only started on the local clusters.

In all the cases considered, both the local and the global schedulers use the FCFS policy to choose the next job to run.

3 The Applications

In this section we describe the two applications for which we will perform measurements on the DAS.

3.1 The Ensflow Application

The Ensflow application [18] uses the data-assimilation technique to understand the evolution of streams and eddies in the ocean near the southern tip of Africa. In this technique, information from observations of the system is combined with information on the evolution of the system obtained from an implementation of the laws of physics. This can be done by using ensemble models that do not calculate the evolution of a single state but rather of a large number (an ensemble, typically 50-500) of different states (ensemble members). In our case there are 60 ensemble members that evolve for a period of 20 days with a time step of 24 hours. Every 240 hourly time steps, an analysis and an update of the ensemble members are done to obtain the optimal estimate for the past period. Each of the ensemble members evolves independently of the others during the time between analysis and update. The sequence of ensemble averages over time describes the development of the ocean's currents best fitting the observations. The application has the following structure:

```
/*--------initialisation--------*/
initiate 60 ensembles;

/*--------start main loop-------*/
while time < stop_time
   /* computation */
   evolve the 60 ensembles;
   if (time = time_to_analyse)
       /* computation + communication */
       analyse and update ensembles;
   endif
endwhile
/*---------end main loop--------*/
```

The main loop is executed 20 times, with two data adjustments. Only during the data adjustment phase (analysis and update ensembles) data are exchanged (using MPI). The data of the ensemble members are local to the processors, and the ensemble members are distributed evenly over the processors. To avoid processors from being unnecessarily idle, we choose the number of processors such that the number of ensemble members is an exact multiple of it. In [18], the Ensflow application is described in more detail, and measurements of the total runtime on two multiprocessors are presented.

3.2 The Poisson Application

Our Poisson application implements a parallel iterative algorithm to find a discrete approximation to the solution of the two-dimensional Poisson equation (a second-order differential equation governing steady-state heat flow in a two-dimensional domain) on the unit square. For the discretization, a uniform grid of points in the unit square with a constant step in both directions is considered. The application uses a red-black Gauss-Seidel scheme (see for instance [15], pp. 429–433), for which the grid is split up into "black" and "red" points, with every red point having only black neighbours and vice versa. In every iteration, each

grid point has its value updated as a function of its previous value and the values of its neighbours, and all points of one colour are visited first followed by the ones of the other colour. The application, which is implemented in MPI, has the following structure:

```
/*--------initialisation--------*/
if  proc_index = 0
   read the initial data;
   /* communication */
   broadcast data to all the processes;
endif

/*--------start main loop-------*/
while global-error => limit
   /* computation */
   update black points;
   update red points;
   /* communication */
   exchange borders with neighbours;
   /* communication + synchronization */
   collect/distribute global error;
endwhile
/*---------end main loop--------*/
```

The domain of the problem is split up into a two-dimensional pattern of rectangles of equal size among the participating processes. In our experiments, we assign only one process to a processor. A way of splitting up the domain is called a process(or) configuration, and is indicated by $h \times v$, with h, v the numbers of processes in the horizontal and vertical directions, respectively. In Section 4 we will consider the numbers of processors and the processor configurations as shown in Table 1.

Table 1. The processor configurations used in our measurements.

total number of processors	processor configuration
8	4x2
16	4x4
32	8x4
64	8x8

Every process communicates with each of its neighbours in order to exchange the values of the grid points on the borders and to compute a global stopping criterion. Exchanging borders takes place in four consecutive steps; first all communication in the direction top is performed, and then in the directions bottom, left and right. The amount of communication depends on the size of the grid, the number of participating processes, and the initial data. When we execute the Poisson application on multiple clusters, the process grid is split up into adjacent vertical strips of equal width, with each cluster running an equal consecutive number of processes (we assume processes to be numbered in column-major order). For instance, for process configuration 4x4 and two clusters, the processes are split up as depicted in Figure 1. Here, processors 4–11 have to exchange border information with processors in the other cluster.

3	7	11	15
2	6	10	14
1	5	9	13
0	4	8	12

Figure 1. The process grid for the Poisson application for process configuration 4x4 divided over two clusters (left–right).

4 Runtime Measurements

In this section we present the results of the measurements of our two applications on the DAS. We use Globus for submitting multicomponent jobs to the DAS. In all of our experiments, the jobs always have components of equal size. Since Globus is currently unable to use the fast local DAS interconnect (Myrinet) but uses the slower local Ethernet instead, we employ both PBS and Globus for running the applications in a single DAS cluster. The PBS measurements yield the best performance of single-cluster operation, but the single-cluster Globus measurements make for a fairer comparison with the multicluster results. Measurements with Globus on a system with C clusters are labeled with Globus-C.

4.1 Total runtime of the Ensflow application

For an investigation of the total runtime we ran the Ensflow application once for different numbers of processors and clusters. The results of the measurements are presented in Fig. 2.

The gaps for 15 processors and Globus-2 and Globus-4, for 20 processors and Globus-3, and for 30 processors and Globus-4 are due to the fact that then we cannot have equal-size job components. The gap for Globus-1 with 60 processors is caused by the limitation of 32 processors in a single cluster when using Globus. We find that the performance of multicluster execution for all numbers of clusters considered compared to single-cluster execution is very good for this application. In addition, the speedup is quite reasonable. Relative to the 12-processor case, the efficiency

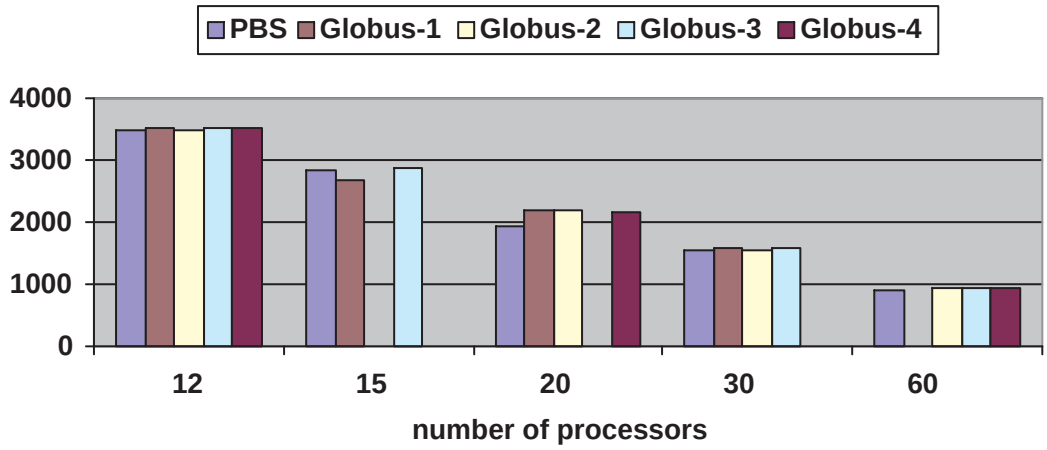

Figure 2. The total runtime of the Ensflow application (in seconds) for different numbers of processors and clusters. (No data when the number of processors is not a multiple of the number of clusters, and for 60 processors with Globus-1.)

slowly decreases to about 0.7 for 60 processors. The explanation of the good performance of multicluster execution is that this application has a relatively small communication component.

4.2 Total runtime of the Poisson application

For a first investigation of the total runtime of the Poisson application, we ran the application once varying the grid size, the total number of processors (see Table 1 for the corresponding processor configurations), and the number of clusters. In addition to the total runtime, we also record the number of iterations needed to reach convergence. The results of the measurements are presented in Table 2, and graphically in Fig. 3. (Because of the numbers of processors we consider, we cannot use three clusters.) Again there are gaps for Globus-1, this time for 64 processors in a single cluster, for the same reason as above.

We find that for a very small grid size, the runtime may increase considerably when using more clusters. However, for a large grid size, the performance of multicluster execution compared to single-cluster execution is quite reasonable. Since the processor configuration influences the number of iterations needed to reach convergence (which determines the total runtime), it is difficult to make a general statement about the speedup. In particular for grid sizes 1000x1000 and 2000x2000, the number of iterations is very variable. However, for grid size 4000x4000 the number of iterations is almost constant, and the speedup when going from 8 to 64 processors for PBS, Globus-2, and Globus-4 is 6.5, 6.0, and 5.8, respectively.

For a further investigation of the total runtime we now fix the processor configuration to 4x4, and we add a few grid sizes. For every set-up (grid size and number of clusters) we ran the application ten times. The results of the measurements (minimum, average, and maximum) are presented in Table 3. For a better comparison, we depict in Fig. 4 for every grid size the (average) runtimes relative to the (average) single-cluster PBS runtimes (which are normalized to 1).

It is clear that for large grid sizes, this application is well suited for multicluster execution. The explanation is that the two major components of the total runtime, the time for updating all grid points (computation) and the time for exchanging border grid points (communication), increase in a different way when the grid size increases. When the total number of grid points increases with a factor g, the number of grid points to be exchanged increases with a factor $\sqrt{g}$. Since communication is the component that causes the poor performance of multicluster execution, it is to be expected that for larger grid sizes (with relatively smaller communication components), multicluster execution performs relatively better.

5 Performance Evaluation of the Scheduling Policies

In this section we assess the performance of the multicluster scheduling policies introduced in Sect. 2.4 with simulations for several workloads differentiated by the numbers of components into which jobs are split and by the percentages of jobs running each of the two applications introduced in Sect. 3. The simulations are for a multicluster with 4 clusters of 32 processors each. The simulation programs were implemented using the CSIM simulation pack-

Table 2. The number of iterations and the total runtime (in seconds) of the Poisson application for different grid sizes and numbers of processors and clusters.

grid size	total number of processors	number of iterations	PBS	Globus-1	Globus-2	Globus-4
100 x 100	8	2436	0.74	3.23	11.5	15.0
	16	2132	0.74	3.59	12.1	11.8
	32	2158	0.93	4.54	17.4	17.4
	64	2429	1.21	—	24.2	21.1
1000 x 1000	8	2630	70.9	86.6	109	114
	16	4347	60.2	78.6	119	125
	32	4356	34.3	46.7	68.8	67.4
	64	2650	8.1	—	30.7	31.7
2000 x 2000	8	2630	291	335	358	365
	16	4387	265	292	339	332
	32	4387	134	161	193	191
	64	2650	46.8	—	80.4	85.1
4000 x 4000	8	2630	1230	1277	1390	1463
	16	2644	649	725	766	767
	32	2651	357	371	402	440
	64	2650	188	—	231	251

age [16].

We will present our simulation results in terms of response time as a function of the utilization. We define the *gross utilization* as the utilization computed from the actual service times experienced by jobs, which for multicomponent jobs includes the time spent in the slow wide-area communication. The *net utilization* is defined as the utilization computed from the single-cluster service times of jobs of the same total size, which gives a measure of the throughput of the system. When there is no co-allocation, there is no wide-area communication and the net utilization is equal to the gross utilization. In this section we only look at the gross utilization and depict the response time as a function of this utilization, because that is a fair basis for comparing the policies.

In Sect. 5.1 we present the workloads in the simulations. Section 5.2 discusses the influence of the numbers and sizes of the job components on the performance, while in Sect. 5.3 the benefits and disadvantages of co-allocation are discussed, compared to a system without co-allocation. In Sect. 5.4 we make a general comparison of the policies. Section 6 compares for all the policies and workloads the gross and the net utilization, which shows how efficient the global applications use the gross utilization offered.

5.1 The workloads

Each of the jobs in the simulated workload is supposed to run one of our two applications; in the case of the Poisson application, we assume the grid size to be 4000x4000. We assess three cases: 100% of the jobs in the system run the Poisson application, 100% of the jobs run the Ensflow application, and each of the two applications is represented by 50% of the jobs in the system. Tables 4 and 5 display the execution times measured on the DAS for the two applications in the several configurations that we are using in the simulations. These values are the same as the ones depicted in Fig. 2, and in Fig. 3 for grid size 4000x4000; for a single cluster we use the PBS runtimes. We assume the interarrival times to be exponentially distributed.

Table 4. The execution times (in seconds) for the Poisson application, depending on the total job size and the number of components, used in the simulations.

Total job size	Number of job components		
	1	2	4
8	1230.0	1390.0	—
16	649.0	766.0	767.0
32	357.0	402.0	440.0

Jobs are split up in different ways, but their components are always of equal size, and we also keep the percentages of jobs for each total size always equal. For the same total size, the various splitting choices admitted in the system receive equal probabilities.

We compare a no co-allocation case, when only single-

Table 3. The total runtime (minimum, average, and maximum) of the Poisson application (in seconds) for processor configuration 4x4 for different grid sizes and numbers of clusters.

grid size	PBS			Globus-1			Globus-2			Globus-4		
	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.
50 x 50	0.22	0.23	0.29	1.35	1.60	2.28	5.93	6.29	6.86	6.12	7.62	11.4
100 x 100	0.65	0.72	0.77	3.35	4.12	6.51	14.7	15.3	16.7	14.3	16.7	22.8
200 x 200	1.73	1.83	1.88	6.55	6.87	8.01	26.4	27.6	30.0	24.0	26.5	33.9
400 x 400	4.67	4.95	5.72	12.4	12.8	13.6	32.0	36.5	38.7	28.6	30.8	39.8
1000 x 1000	60.7	63.7	68.3	78.4	78.9	79.4	101	105	108	103	107	118
2000 x 2000	248	257	274	291	296	310	306	309	311	306	323	349
4000 x 4000	701	706	712	720	733	766	743	750	757	728	751	794
10000 x 10000	3734	3841	3948	3878	3960	4078	4012	4081	4160	4215	4235	4285

Table 5. The execution times (in seconds) for the Ensflow application, depending on the total job size and the number of components, used in the simulations.

Total job size	Number of job components			
	1	2	3	4
12	3485.0	3494.0	3504.0	3507.0
15	2836.0	—	2884.0	—
20	1935.0	2207.0	—	2155.0
30	1563.0	1541.0	1584.0	—

component jobs are admitted, to several co-allocation cases. We define the following co-allocation rules:

1. **[no]** There are only single-component jobs, co-allocation is not allowed.

2. **[co]** Both single- and multi-component jobs are allowed, without restrictions on the sizes of job components and the numbers of components.

3. **[rco]** Both single- and multi-component jobs are allowed, but the job-component sizes are restricted to half of the clusters' sizes.

4. **[fco]** Both single- and multi-component jobs. The job-component sizes are restricted to half of the clusters' sizes, and only multi-component jobs with two components are allowed.

In Tables 6, 7, and 8 we show the resulting percentages of jobs for the numbers of components allowed for the Poisson application (here we disallow jobs of size 8 to be split into 4 components), for the Ensflow application, and for an even mix of these, respectively.

5.2 The influence of the numbers and sizes of the job components

In Fig. 5 we show the response time as a function of the (gross) utilization for the three job mixes, the three scheduling policies, and the four co-allocation rules. (In Fig. 5 and in all subsequent figures, the legends are in the right-to-left order of the curves, and the average response time is in seconds.) Because our two applications have very different service times, we assess the performance more in terms of the point where the system saturates (where the reponse-time curves rise very steeply) than in terms of the actual reponse times. The performance is the best for the Poisson application; a reason for this is that in that case all the job sizes are also powers of two, like the clusters' sizes, which makes them fit better in the system. For the Ensflow application the utilization achieved is worse because of the job sizes, which in most combinations add up in a way that leaves more idle processors in the system than in the case of the Poisson application. For all policies and co-allocation rules considered the worst performance is displayed by the mix of the two applications, where the different sizes of jobs are even more difficult to fit in an efficient way.

In all the graphs in Fig. 5 the [co] co-allocation rule yields the poorest performance. This shows that although in general co-allocation provides more flexibility in placing jobs on the system, jobs with conflicting requirements can make the performance worse than that in the absence of co-allocation. The bad performance is due to the simultaneous presence in the system of large single-component jobs, using (almost) entire clusters, and of jobs with many components, even equal to the number of clusters in which case on each of the clusters there has to be enough room to accommodate a job component. Possible improvements are to restrict the maximum size of job components and to limit the number of components of the multi-component jobs. The [rco] co-allocation rule includes the first restric-

Table 6. The percentages of jobs with different numbers of components for the four job compositions for the Poisson application.

Total job size	Number of job components								
	[no]	**[co]**			**[rco]**			**[fco]**	
	1	1	2	4	1	2	4	1	2
8	33.34%	16.67%	16.67%	—	16.67%	16.67%	—	16.67%	16.67%
16	33.33%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	16.665%	16.665%
32	33.33%	11.11%	11.11%	11.11%	0.0%	16.665%	16.665%	0.0%	33.33%

Table 7. The percentages of jobs with different numbers of components for the four co-allocation rules for the Ensflow application.

[no]

Total job size	Number of job components			
	1	2	3	4
12	25.0%	0.0%	0.0%	0.0%
15	25.0%	—	0.0%	—
20	25.0%	0.0%	—	0.0%
30	25.0%	0.0%	0.0%	—

[rco]

Total job size	Number of job components			
	1	2	3	4
12	6.25%	6.25%	6.25%	6.25%
15	12.5%	—	12.5%	—
20	8.34%	8.33%	—	8.33%
30	8.34%	8.33%	8.33%	—

[co]

Total job size	Number of job components			
	1	2	3	4
12	6.25%	6.25%	6.25%	6.25%
15	12.5%	—	12.5%	—
20	0.0%	12.5%	—	12.5%
30	0.0%	12.5%	12.5%	—

[fco]

Total job size	Number of job components			
	1	2	3	4
12	12.5%	12.5%	0.0%	0.0%
15	25.0%	—	0.0%	—
20	0.0%	25.0%	—	0.0%
30	0.0%	25.0%	0.0%	—

tion, while [fco] includes both. The graphs show that in all the cases considered imposing these restrictions significantly improves the performance. For LS, the performance for both the [rco] and [fco] cases proves to be much better than for the no co-allocation case. The same result holds for LP. When there are only single-component jobs LP becomes LS, and that is why for the no co-allocation case with LP the curve for LS is depicted. For GS, co-allocation does not enhance the performance, but maintains or only slightly improves it with the [fco] restrictions—large jobs are always split and only maximum two components are allowed—and deteriorates it in the other cases. For GS, the advantage of more flexibility brought by co-allocation does not compensate the disadvantage of longer service times due to the inter-cluster communication. The GS policy does not restrict single-component jobs to the local clusters, which makes the performance in the absence of communication rather good. Jobs are scheduled in a FCFS manner from the single queue and the more freedom in spreading the jobs on the clusters introduced by co-allocation is not used enough.

5.3 Co-allocation versus no co-allocation

As Fig. 5 shows, in a large number of cases co-allocation can enhance the performance of a multicluster system, but it is necessary to avoid the simultaneous presence in the system of jobs with conflicting requirements. In [8] we have shown that large single-component jobs and jobs with many components deteriorate the performance. Moreover, combining such jobs makes it even worse, which is also confirmed by Fig. 5. For LS and LP it is enough to avoid large single-cluster jobs to make co-allocation worthwhile. Since LS stores multi-component jobs in all local queues, it provides (compared to the other policies) more flexibility and a larger choice—any of the jobs at the top of the queues—at each moment when a scheduling decision has to be taken. This is why avoiding jobs with many components does not influence the performance much. LP keeps all multi-component jobs in the global queue and the jobs with many components, which are more difficult to fit, impact the performance more. This can be concluded from

Table 8. The percentages of jobs with different numbers of components for the four co-allocation rules and a mix of the Poisson and Ensflow applications in equal proportions.

[no]

Total job size	Number of job components			
	1	2	3	4
8	16.67%	0.0%	—	—
16	16.67%	0.0%	—	0.0%
32	16.66%	0.0%	—	0.0%
12	12.5%	0.0%	0.0%	0.0%
15	12.5%	—	0.0%	—
20	12.5%	0.0%	—	0.0%
30	12.5%	0.0%	0.0%	—

[rco]

Total job size	Number of job components			
	1	2	3	4
8	8.335%	8.335%	—	—
16	5.557%	5.557%	—	5.556%
32	0.0%	8.33%	—	8.33%
12	3.125%	3.125%	3.125%	3.125%
15	6.25%	—	6.25%	—
20	4.167%	4.167%	—	4.166%
30	4.167%	4.167%	4.166%	—

[co]

Total job size	Number of job components			
	1	2	3	4
8	8.335%	8.335%	—	—
16	5.557%	5.557%	—	5.556%
32	5.554%	5.553%	—	5.553%
12	3.125%	3.125%	3.125%	3.125%
15	6.25%	—	6.25%	—
20	0.0%	6.25%	—	6.25%
30	0.0%	6.25%	6.25%	—

[fco]

Total job size	Number of job components			
	1	2	3	4
8	8.335%	8.335%	—	—
16	8.335%	8.335%	—	0.0%
32	0.0%	16.66%	—	0.0%
12	6.25%	6.25%	0.0%	0.0%
15	12.5%	—	0.0%	—
20	0.0%	12.5%	—	0.0%
30	0.0%	12.5%	0.0%	—

the significant improvement brought by the [fco] restrictions compared to the [rco] ones. GS, as mentioned before, has good performance in the absence of communication due to the fact that it can run jobs from the single queue on any of the clusters. However the same single queue makes co-allocation without restrictions ([co]) perform poorly, and only when both the numbers and sizes of the components are restricted ([fco]) is co-allocation an advantage.

5.4 Comparing the policies

In this section we compare the three policies defined for the three application mixes and the different co-allocation rules. From Fig. 5 we conclude that independent of the application mix LS provides the best results for the co-allocation cases. When there are only single-component jobs the performance of GS is better. LP becomes LS when there are just single-component jobs, so the performance of the two policies is the same in the absence of co-allocation.

With the [rco] restrictions LS display much better results than LP, the difference between the two policies being that LP keep all multi-component jobs in a single queue. This relates to our observation for GS that when there is a single queue for multi-component jobs, those with many components are hard to fit and have a strong negative impact on performance. GS is better for the single-component jobs, but once multi-component jobs are allowed, the extra queue for the global jobs in LP and spreading the global jobs among the local queues in the case of LS bring enough benefits as to allow those policies to outperform GS.

Comparing all the cases considered, we concluded that the best results are displayed by LP and LS with the [fco] restrictions. The similar performance of LS and LP in that case shows that for those sizes and numbers for the job components, to have a separate queue for the multi-component jobs is enough and the backfilling effect with a window equal to the number of clusters induced by LS does not bring extra improvements.

6 Gross versus Net Utilization

In Sect. 5 we have studied the average response time as a function of the gross utilization. In this section we discuss the difference between gross and net utilization, and quantify this difference for the cases considered in Sect. 5. We have defined the net and the gross utilization based on the job service times in single clusters with fast local communication, and on the longer service times displayed by multi-component jobs running the same application on multiple clusters (thus using slow inter-cluster communication), respectively. The difference between these utilizations is the capacity lost internally in multi-component jobs due to slow

wide-area links. This internal capacity loss might be reduced by restructuring applications [17] or by having them use (collective-) communication operations optimized for wide-area systems [14].

The performance of a multicluster policy may look good when considering the response time as a function of the gross utilization, but, when there is much internal capacity loss, the performance as a function of the net utilization (or of the throughput) may be poor. This "real" performance of a multicluster policy would improve with more efficient applications or with faster global communication.

In Figs. 6, 8 and 7 we depict the average response time for our three policies, for the three application mixes and for the different ways of co-allocation studied, as a function of both the gross and the net utilization. To assess the difference between the two utilizations at a certain response time, one should compare the graphs in the horizontal direction. Of course, for the same workload (defined by the arrival rate, and so, by the net utilization), the difference between the gross and the net utilization is the same for all scheduling policies and co-allocation rules, albeit at possibly different response times.

The largest difference between the gross and the net utilizations is always displayed for the Poisson application. This is an expected consequence of the fact that this application requires the largest amount of communication. Spreading the jobs running this application on more clusters also results in more wide-area communication than for the Ensflow application, or the equal mix of the two applications.

For all the policies and job mixes, comparing the three co-allocation cases we observe that the largest amount of intercluster communication is shown for the [rco] restrictions, and the least for the [co] restrictions. By limiting the size of the single-component jobs, [rco] and [fco] increase the percentage of multi-component jobs which brings more wide-area communication. Since it limits the number of job components, [fco] yields a lower amount of intercluster communication compared to [rco]. These results are also valid for the Ensflow application, even though the differences are smaller since that application requires very little communication.

Despite the significant difference in performance, for the same application mix and the same restrictions imposed for co-allocation, all the three policies show very similar differences between the graphs for net and gross utilization. In general we could expect that policies with better performance would show more wide-area communication for the same set of jobs.

7 Conclusions

We have performed measurements of two applications on our multicluster DAS system, and we have performed simulations of three multicluster scheduling policies incorporating co-allocation. The performance of multicluster execution of the Ensflow application is very good, which can be explained by its relatively small communication component. Also Poisson application is well suited for multicluster execution, at least for large grid sizes, when the communication component becomes relatively small. The penalty for the slow multicluster communication can be reduced by allowing the computation and communication parts of the processes of a multicluster job to overlap. To be able to make a well-considered decision when to submit an application to a single cluster or across multiple clusters, it would be convenient to have a synthetic application parameterized by the way it is split up across clusters and by its communication pattern, to simulate a range of possible applications.

Our simulations of multicluster scheduling policies show that simply allowing co-allocation without any restrictions is not a very good idea, for none of the policies. In all cases, one should at least limit the job-component sizes, and preferably also the number of job components. Furthermore, we found that the policies with local queues (possibly with a global queue for multicomponent jobs) yield better performance than having only a single global queue.

References

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

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

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

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

[5] 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.

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

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

[8] A. Bucur and D. Epema. An Evaluation of Processor Co-Allocation for Different System Configurations and Job Structures. In *Proc. of the 14th Symp. on Computer Architecture and High Performance Computing*, pages 195–203. IEEE Computer Society Press, 2002.

[9] 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.

[10] A. Bucur and D. Epema. The Maximal Utilization of Processor Co-Allocation in Multicluster Systems. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2003 (to appear; available from www.pds.its.tudelft.nl/ epema).

[11] 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.

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

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

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

[15] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings, 1994.

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

[17] 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.

[18] F. van Hees, A. van der Steen, and P. van Leeuwen. A Parallel Data-Assimilation Model for Oceanographic Observations. *Parallel and Distributed Computing Practices*, 2003 (to appear).

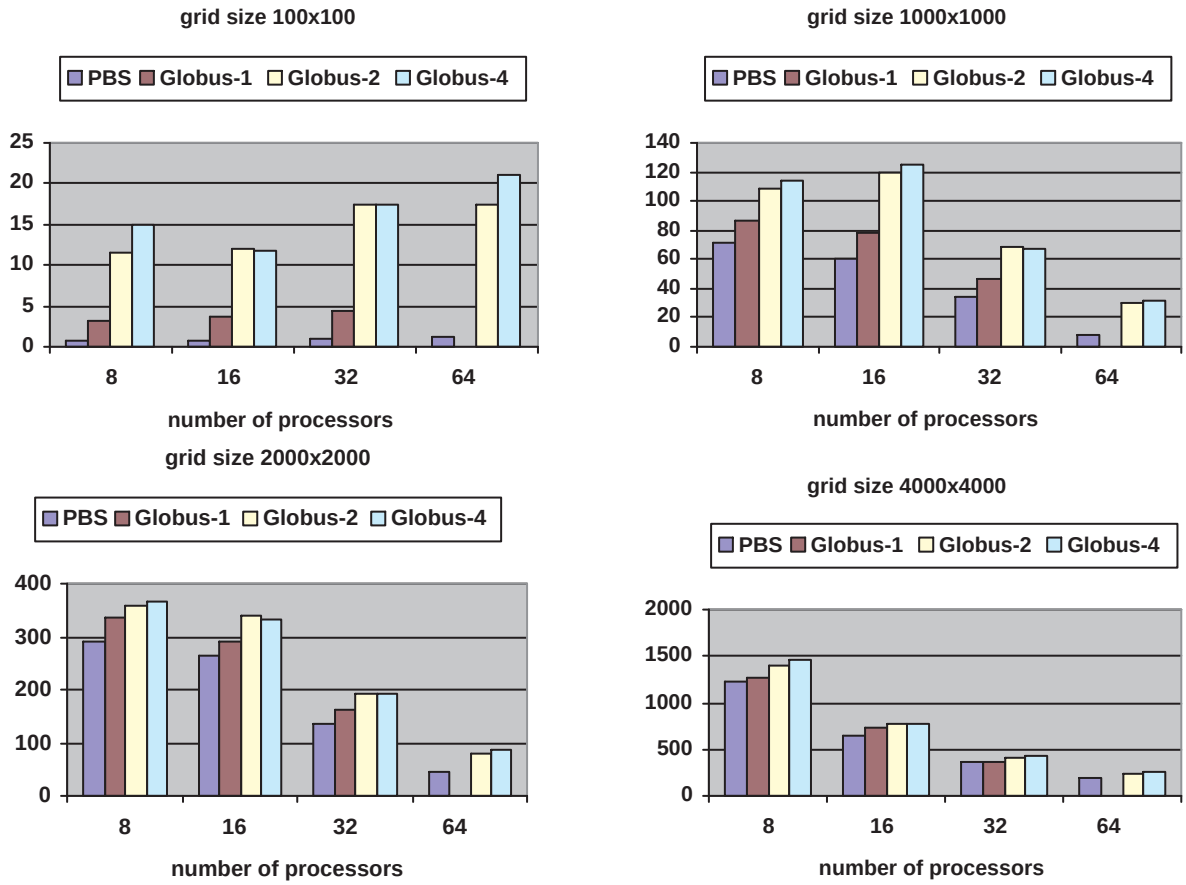

Figure 3. The total runtime of the Poisson application (in seconds) for different grid sizes and numbers of processors and clusters. (No data for 64 processors with Globus-1.)

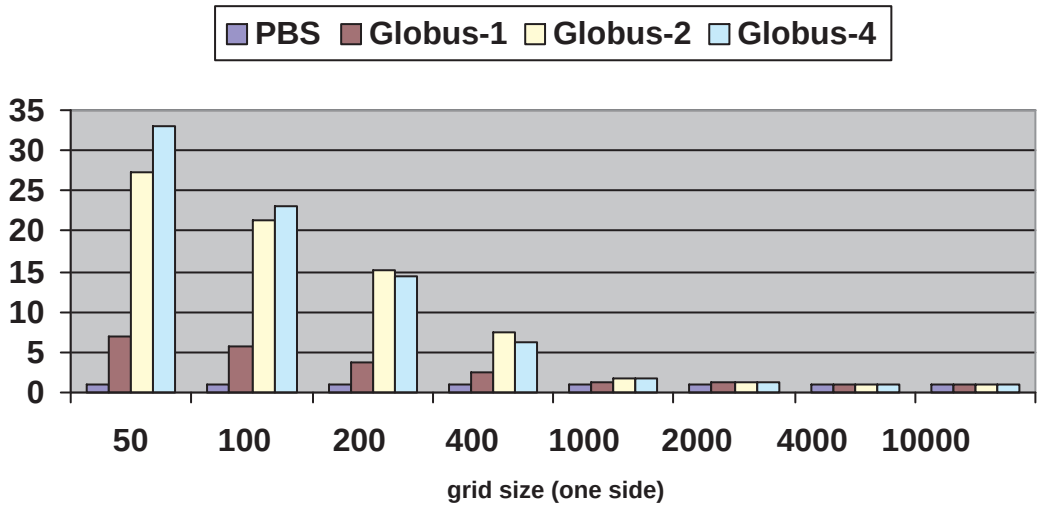

Figure 4. The total runtime of the Poisson application (in seconds) for processor configuration 4x4 for different grid sizes and numbers of clusters, normalized with respect to PBS.

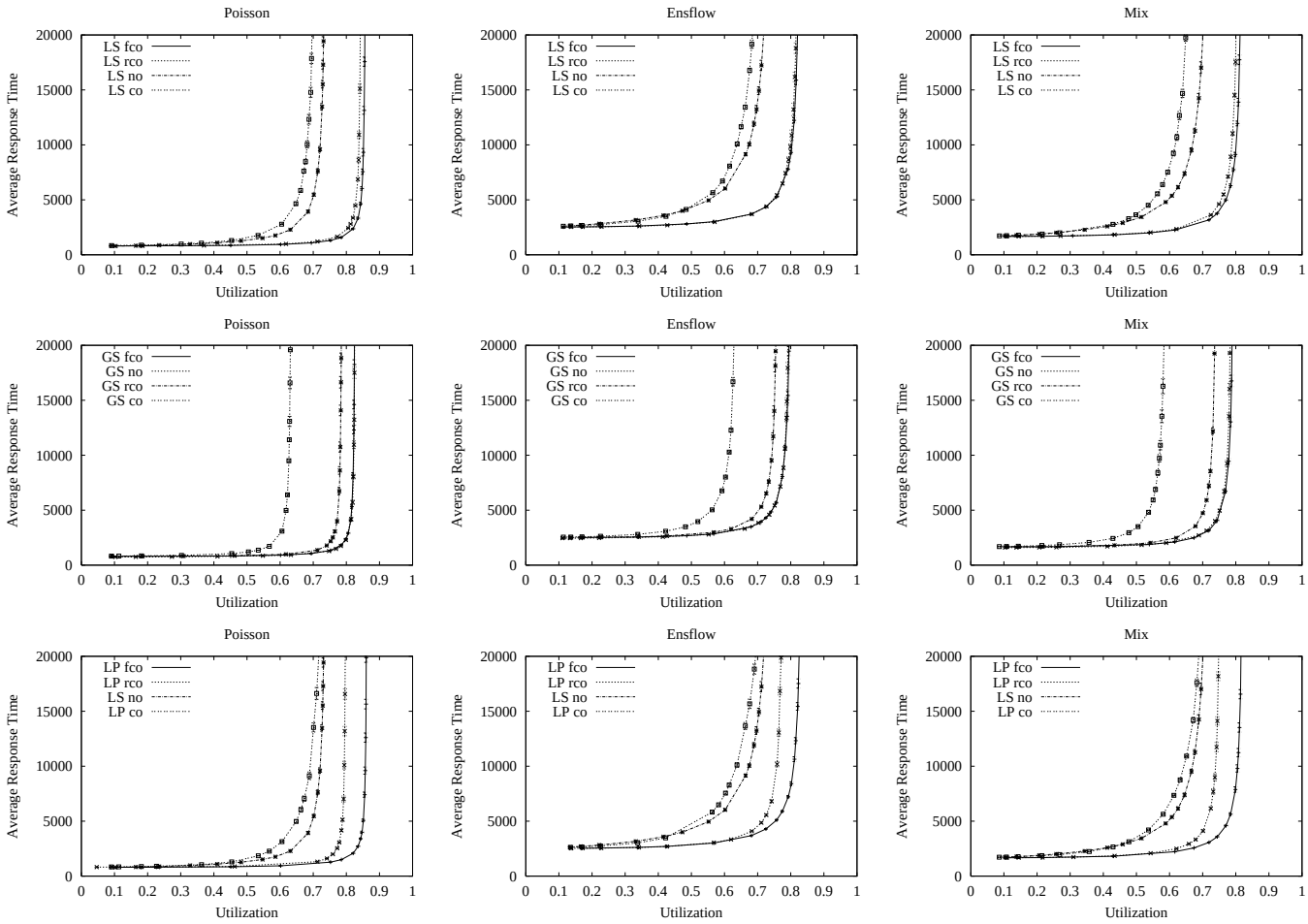

Figure 5. The performance of LS, LP and GS (top-bottom) for the Poisson application, the Ensflow application and a mix of the two in equal proportions (left-right), depending on the numbers of job components allowed in the system.

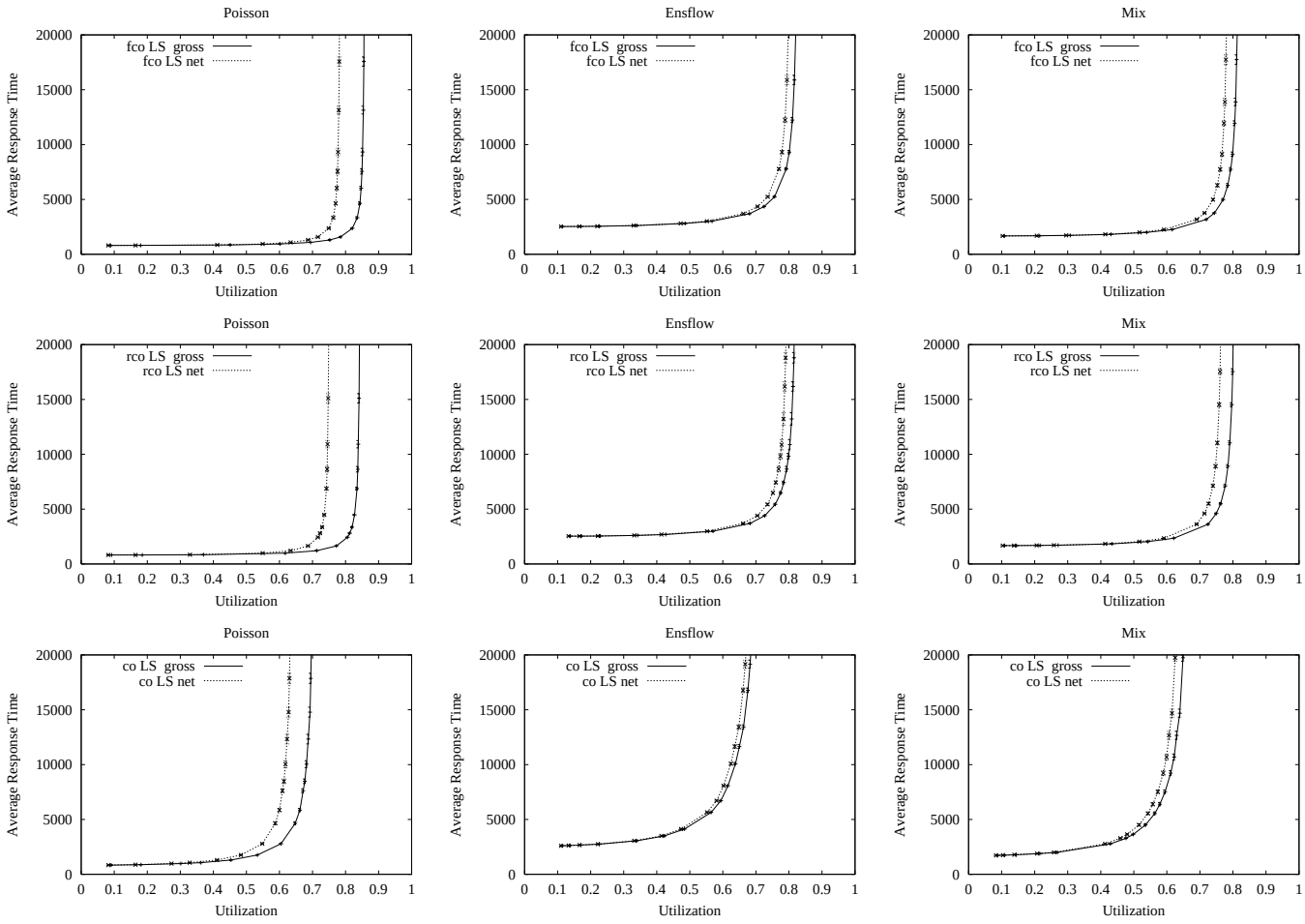

Figure 6. The response time as a function of the gross and the net utilization for the LS policy, the three application mixes and the three co-allocation rules that allow co-allocation.

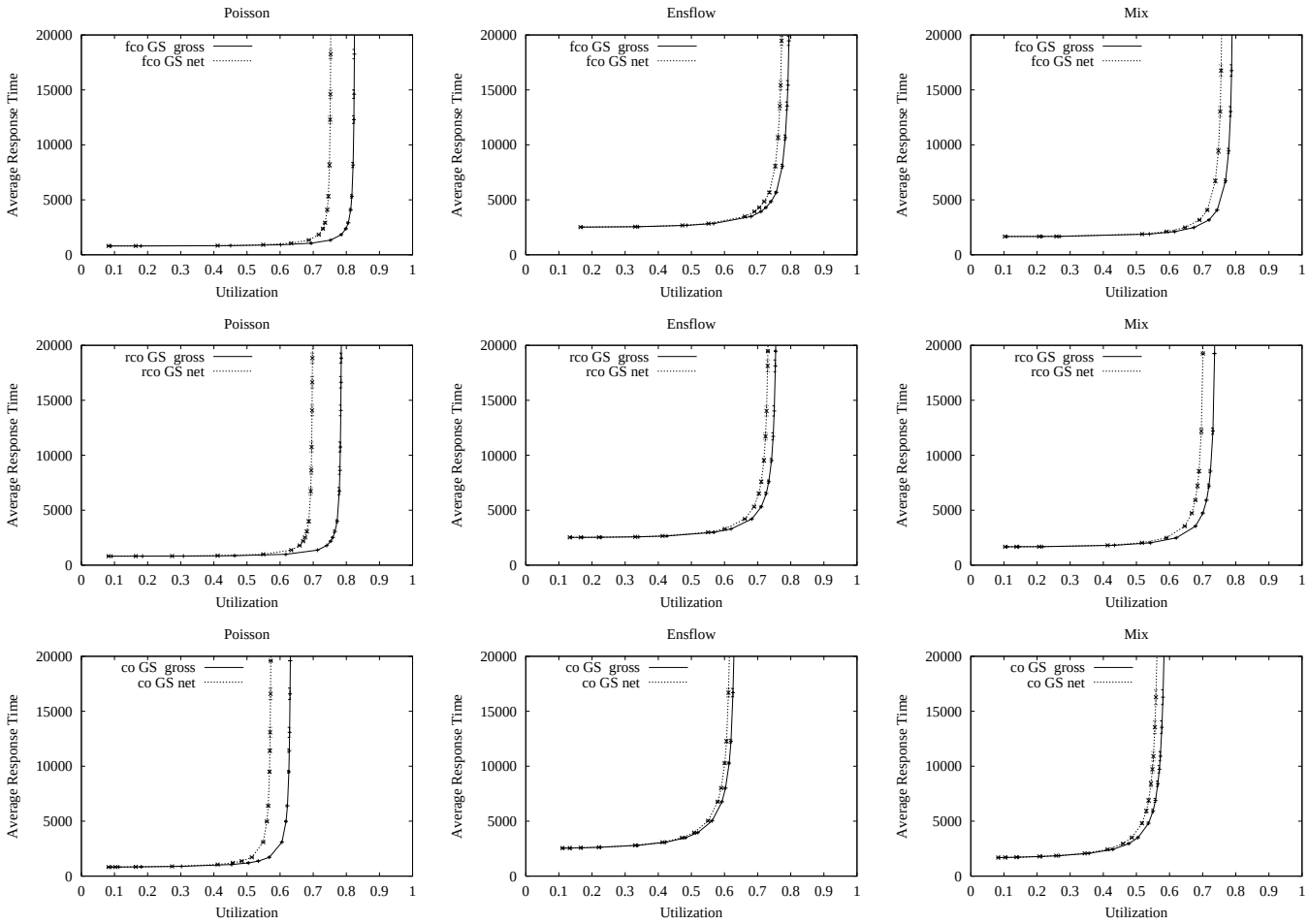

Figure 7. The response time as a function of the gross and the net utilization for the GS policy, the three application mixes and the three co-allocation rules that allow co-allocation.

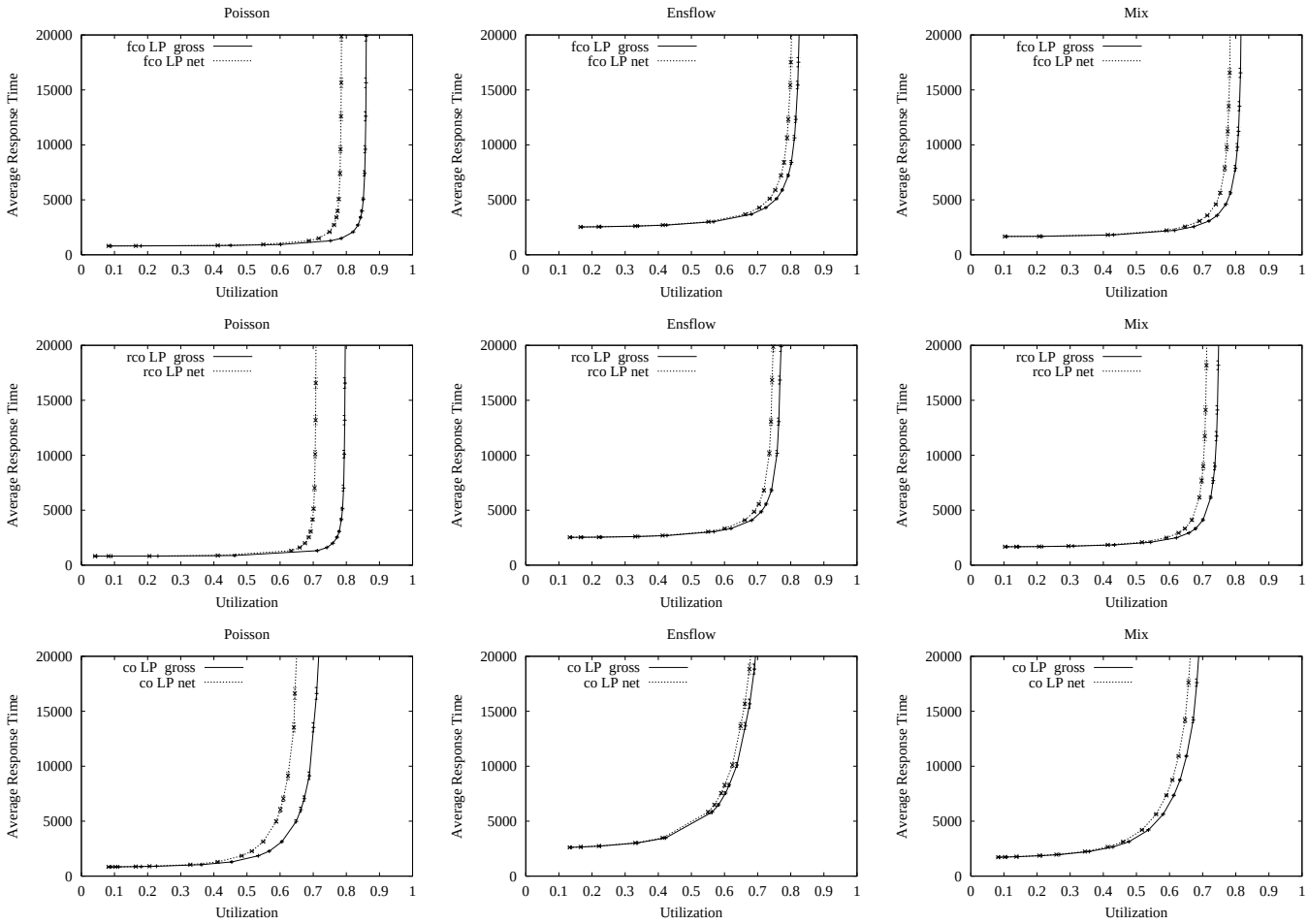

Figure 8. The response time as a function of the gross and the net utilization for the LP policy, the three application mixes and the three co-allocation rules that allow co-allocation.

A Communication-Time Measurements

In this appendix we measure the communication time needed for exchanging the values of border grid points in the Poisson application with processor configuration 4x4. All numbers presented below are averages over ten runs.

A.1 Per-process communication time for exchanging borders

We measure for each individual process(or) the total time (i.e., across all iterations) it spends exchanging borders with other processes. Figure 9 contains the results for different numbers of clusters and grid sizes 100x100 and 4000x4000. As expected, for Globus-2, the processes at the edges of the clusters need more time to communicate, although for some reason also some other processes take much time to communicate. This effect is relatively speaking much larger for grid size 100x100 than for grid size 4000x4000.

For Globus-2 with grid size 100x100 we also measure for each individual process, and in each direction, the total time it spends exchanging borders with other processes; the results are presented in Fig. 10. We see that in the cross-cluster directions left and right, receiving border information takes a relatively large amount of time.

A.2 Synchronized and non-synchronized operation

In our original Poisson application, we do not synchronize processes before they start their communication phases. So then, as soon as a process finishes its computation in an iteration, it starts (trying) to communicate. We added an MPI command to our application in order to enable synchronized operation. Then, all processes synchronize in every iteration before the communication starts, so they all start communicating at (about) the same time. In both synchronized and non-synchronized operation, we measure the communication time in an iteration as the time elapsed between the last process finishing its computation phase and the last process finishing its communication phase. The cause of the difference between these communication times in these two modes of operation lies in the potential parallelism of computation and communication in non-synchronized operation. In general, in this mode of operation, the communication time is smaller, as we will indeed see in Sect. A.3.

A.3 Total communication time for exchanging borders

We define the total communication time for exchanging borders as the sum of the communication times of all iterations, both for synchronized and non-synchronized operation. In Table 9, we show the minimum, average, and maximum (across ten runs) total communication time for synchronized operation; the variation is in general not very large. The difference between single-cluster and multicluster performance (and between PBS and Globus-1) is very large.

Table 10 presents the (average) total communication times for exchanging borders when processes are or are not synchronized before communication. (This table contains the average results of Table 9.) In Fig. 11 we depict the average total communication times from Table 10 after normalization with respect to PBS. We find that with synchronized operation, communication in a single cluster is, depending on the grid size, 10–35 times faster then multicluster communication, while for non-synchronized operation (and realistic grid sizes), this factor is reduced to about 13. In addition, in Table 10 we see that for large grid sizes the performance of multicluster communication strongly improves when the processes are not synchronized, but that this is not the case for a single cluster with PBS.

A.4 Data transfer rate of exchanging borders

We use the results of Table 9 (with synchronized operation) to calculate the data transfer rate when exchanging borders. For this calculation we assume that the slowest communicating process is always an interior process with four borders to exchange. Since the processor configuration is 4x4, the number of grid points to communicate per iteration by an interior process (send and receive) is twice the side of the grid. For a grid point 8 bytes are reserved. In Fig. 12 we see that for PBS the data transfer rate strongly increases when the amount of data to be communicated increases. The highest data transfer rate for PBS is 35 Mbyte/s for grid size 4000x4000, while for multicluster execution (Globus-2 and Globus-4) the highest data transfer rate of over 3 Mbyte/s is reached for a grid size of 2000x2000.

grid size 100x100

0.20	0.28	0.25	0.24
0.26	0.33	0.30	0.30
0.28	0.35	0.31	0.31
0.24	0.29	0.27	0.25

PBS

0.69	1.13	1.11	1.09
1.17	1.53	1.60	1.31
1.60	1.71	1.82	1.67
1.40	1.47	1.50	1.25

Globus-1

5.29	6.44	6.87	0.89
0.68	6.83	6.66	1.12
1.06	12.9	11.5	5.66
0.74	5.72	7.41	5.83

Globus-2

7.13	8.64	7.21	8.37
1.98	5.88	2.20	8.20
3.96	7.23	8.07	9.87
0.64	4.02	2.49	4.95

Globus-4

grid size 4000x4000

3.25	3.71	3.79	3.72
3.69	4.17	4.30	4.18
3.50	3.98	3.92	3.92
3.49	3.93	4.04	3.96

PBS

78	87	99	88
76	86	103	90
79	92	87	77
78	90	86	79

Globus-1

56	107	137	81
54	107	134	83
75	117	128	96
68	118	112	91

Globus-2

96	123	132	111
84	108	108	99
113	130	123	124
108	121	111	120

Globus-4

Figure 9. The total per-process communication times for different grid sizes (in seconds).

5.31	0.11	0.15	0.11
0.23	0.27	0.44	0.29
0.75	5.34	4.83	0.21
0.14	0.11	0.12	0.12

Top

0.13	0.12	0.11	0.13
0.22	0.20	0.17	0.17
0.20	0.14	0.13	0.51
0.33	0.25	0.33	0.25

Bottom

0.02	6.30	0.40	0.09
0.02	6.66	0.27	0.09
0.06	5.98	0.11	0.09
0.07	6.03	0.13	0.09

Left

0.10	0.54	6.12	0.56
0.12	0.53	6.30	0.59
0.10	0.13	6.95	4.51
0.14	0.15	6.98	4.97

Right

Figure 10. The total per-process communication times in each of the four directions for Globus-2 and grid size 100x100 (in seconds).

Table 9. The total communication time for exchanging borders (in seconds) with synchronized operation for processor configuration 4x4.

grid size	number of iterations	PBS			Globus-1			Globus-2			Globus-4		
		min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.
50 x 50	865	0.15	0.16	0.19	0.52	0.62	0.87	3.81	4.67	5.33	4.82	4.96	5.13
100 x 100	2132	0.37	0.39	0.44	1.41	1.51	1.67	9.25	11.8	15.1	12.0	12.2	13.7
200 x 200	3570	0.69	0.72	0.74	2.90	3.04	3.56	15.0	19.8	23.0	20.1	20.3	20.5
400 x 400	3814	0.89	0.91	0.95	4.49	4.68	4.91	18.0	22.5	29.4	17.5	20.8	22.6
1000 x 1000	4347	2.09	2.17	2.31	9.49	10.2	11.8	22.3	28.8	36.5	26.1	29.2	32.1
2000 x 2000	4387	4.21	4.38	4.67	27.1	29.1	32.3	41.3	45.1	49.6	41.2	42.8	45.8
4000 x 4000	2644	4.50	4.88	5.47	127	134	139	154	161	171	167	179	206
10000 x 10000	2644	11.9	12.3	13.1	127	162	202	186	267	398	290	358	427

Table 10. The total communication time for exchanging borders (in seconds) with synchronized and non-synchronized operation for processor configuration 4x4.

grid size	number of iterations	PBS		Globus-1		Globus-2		Globus-4	
		synchronization		synchronization		synchronization		synchronization	
		yes	no	yes	no	yes	no	yes	no
50x50	865	0.16	0.11	0.62	0.54	4.67	2.02	4.96	1.92
100x100	2132	0.39	0.28	1.51	1.31	11.8	4.16	12.2	4.72
200x200	3570	0.72	0.54	3.04	2.71	19.8	7.09	20.3	8.50
400x400	3814	0.91	0.88	4.68	4.20	22.5	9.30	20.8	12.4
1000x1000	4347	2.17	2.00	10.2	11.3	28.8	16.8	29.2	17.5
2000x2000	4387	4.38	3.59	29.1	16.5	45.1	18.5	42.8	31.4
4000x4000	2644	4.88	5.26	134	18.1	161	24.3	179	42.0
10000x10000	2644	12.3	12.7	162	39.8	267	62.5	358	172

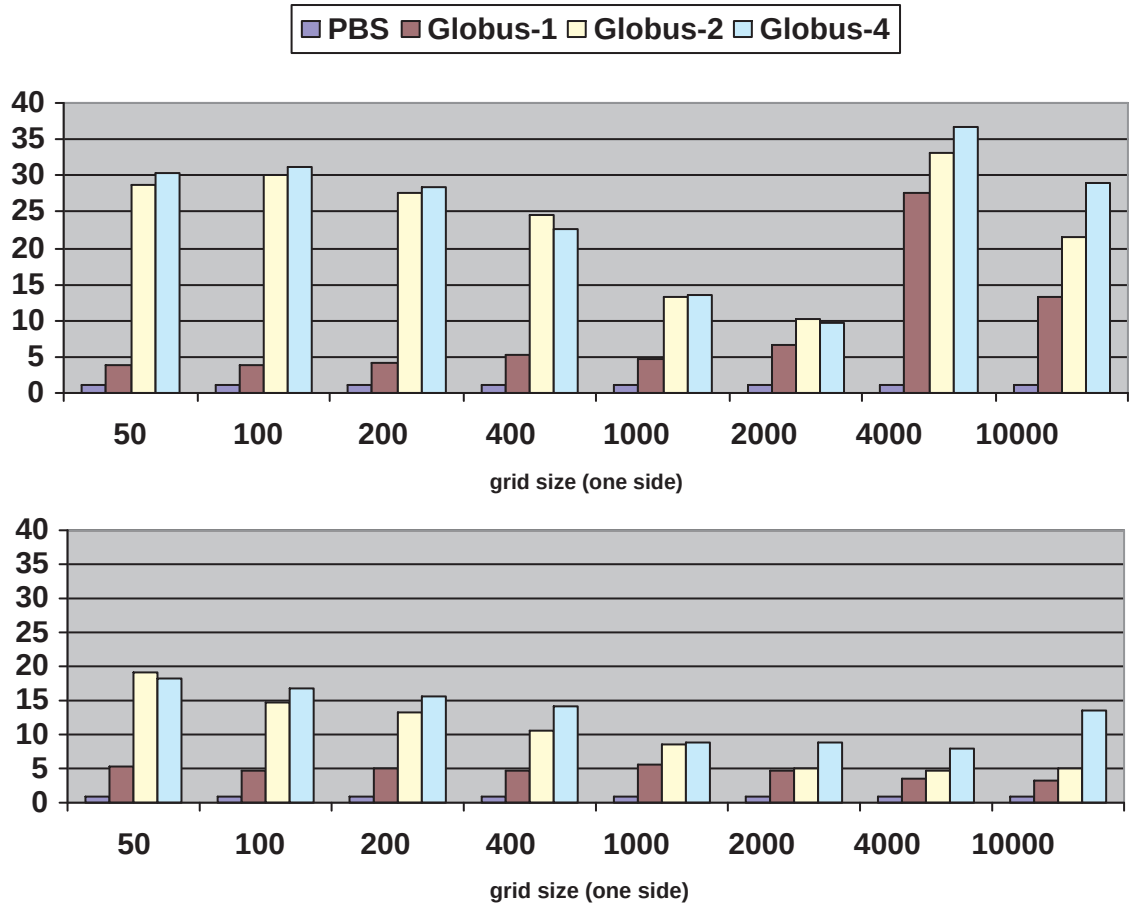

Figure 11. The total communication time for exchanging borders with synchronized (top) and non-synchronized (bottom) operation for processor configuration 4x4, normalized with respect to PBS.

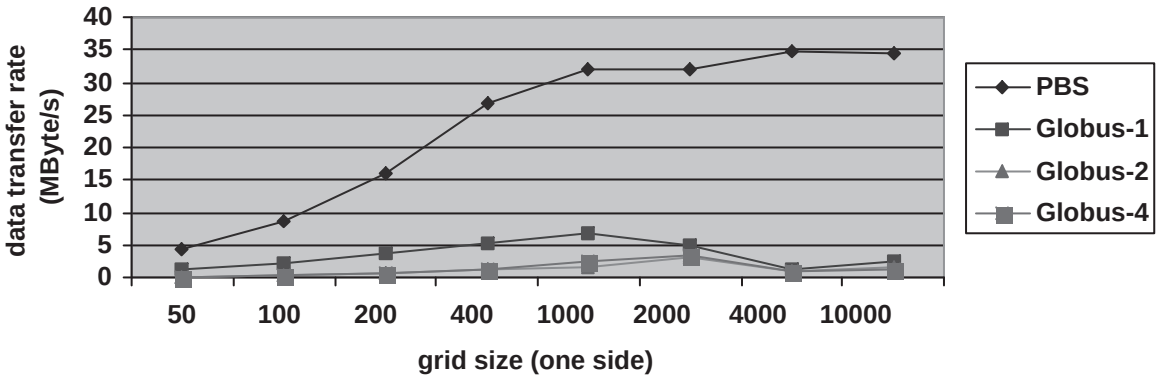

Figure 12. The data-transfer rate when exchanging borders.