An evaluation of the close-to-files processor and data co-allocation policy in multiclusters

Citation for published version (APA):

Mohamed, H. H., & Epema, D. H. J. (2004). An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *Proceedings of the IEEE International Conference on Cluster Computing (ICCC'04, San Diego CA, USA, September 20-23, 2004)* (pp. 287-298). Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/CLUSTER.2004.1392626>

DOI:

[10.1109/CLUSTER.2004.1392626](https://doi.org/10.1109/CLUSTER.2004.1392626)

Document status and date:

Published: 01/01/2004

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters

H.H. Mohamed and D.H.J. Epema

Faculty of Electrical Engineering, Mathematics, and Computer Science

Delft University of Technology

P.O. Box 5031, 2600 GA Delft, The Netherlands

e-mail: H.H.Mohamed, D.H.J.Epema@ewi.tudelft.nl

Abstract

In multicluster systems, and more generally, in grids, jobs may require co-allocation, i.e., the simultaneous allocation of resources such as processors and input files in multiple clusters. While such jobs may have reduced runtimes because they have access to more resources, waiting for processors in multiple clusters and for the input files to become available in the right locations may introduce inefficiencies. In previous work, we have studied through simulations only processor co-allocation. Here, we extend this work with an analysis of the performance in a real testbed of our prototype Processor and Data Co-Allocator with the Close-to-Files (CF) job-placement algorithm. CF tries to place job components on clusters with enough idle processors which are close to the sites where the input files reside. We present a comparison of the performance of CF and the Worst-Fit job-placement algorithm, with and without file replication, achieved with our prototype. Our most important findings are that CF with replication works best, and that the utilization in our testbed can be driven to about 80%.

1 Introduction

Grids offer the promise of transparent access to large collections of resources for applications demanding many processors and access to huge data sets. In fact, the needs of a single application may exceed the capacity available in each of the subsystems making up a grid, and so *co-allocation*, i.e., the simultaneous access to resources of possibly multiple types in multiple locations, managed by different resource managers [11], may be required.

Even though multiclusters and grids offer very large amounts of resources, to date most applications submitted

to such systems run in single subsystems managed by a single scheduler. With this approach, grids are in fact used as big load balancing devices, and the function of a grid scheduler amounts to choosing a suitable subsystem for every application. The real challenge in resource management in grids lies in co-allocation. Indeed, the feasibility of running parallel applications in multicluster systems by employing processor co-allocation has been amply demonstrated [23, 6].

In previous work, we have studied co-allocation of only processors by means of simulations [8, 9, 6]. In this paper, we extend this work by adding data as a resource to be co-allocated, and by carrying out a performance study in a real testbed. In addition to requiring processors in different clusters, we assume now that jobs have a (large) input file that has to be transferred to all locations where job components will run prior to the execution of the job. For this purpose, we introduce the Close-to-Files (CF) policy which tries to place job components on or close to sites where the input file is located. We have built CF into our Processor and Data Co-Allocator (PDCA), which implements a co-allocation service in our five-cluster Distributed ASCI Supercomputer (DAS) (see Section 2.1).

There are two important problems to be solved when using processor and data co-allocation and when doing performance experiments in a real system like the DAS. The first is to achieve the simultaneous availability of resources managed by different local schedulers. In our setting, we would like to transfer the input file first to the right locations before actually starting a job rather than occupying the processors while waiting. However, most local schedulers lack a processor-reservation mechanism, so we cannot make a reservation for the time when we expect the input file to be present. We present a workaround method for this reservation problem in this paper. The second problem is that we cannot exclusively claim a system like the DAS for

doing experiments. However, the background load from the regular users of the system may vary between experiments, making their results difficult to compare. One solution to this problem that we employ is to inject jobs of our own to try to maintain the same background load in different experiments.

The main contributions of this paper are 1) the design and implementation of the CF job-placement policy that tries to put job components close to their input file; 2) a performance comparison of CF with the Worst-Fit job-placement policy, with and without replication, and 3) a performance analysis of our workaround method for processor reservation. The results of the experiments with our prototype show that the combination of the CF policy and file replication is very beneficial. We also find that our reservation mechanism works without wasting much processor time, and that with co-allocation we can drive the utilization in our testbed to about 80%.

2 A Model for Co-allocation

In this section, we first describe our multicluster testbed, the DAS. Then we present our model for co-allocation in multiclusters, and more generally, in grids. Finally, we discuss our Processor and Data Co-Allocator, which implements processor and data co-allocation in the DAS.

2.1 The Distributed ASCI Supercomputer

The DAS [2] is a wide-area computer system consisting of five clusters (one at each of five universities in the Netherlands, amongst which Delft) of dual-processor Pentium-based nodes, one with 72, the other four with 32 nodes each. The clusters are interconnected by the Dutch university backbone (100 Mbit/s), while for local communications inside the clusters Myrinet LANs are used (1200 Mbit/s). The system was designed for research on parallel and distributed computing. On single DAS clusters the scheduler is PBS [4]. Before the PDCA was implemented, jobs spanning multiple clusters could only be submitted with Globus's DUROC component [3]. Each DAS cluster has its own separate file system, and therefore, in principle, files have to be moved explicitly between users' working spaces in different clusters.

2.2 The structure of the system and of jobs

We assume a multicluster environment with sites that each contain computational resources (processors) and a file server. By a job we mean a *parallel application* requiring files and processors that can be split up into several *job components* which can be scheduled to execute on multiple sites simultaneously (*co-allocation*) [7, 5, 11]. This allows the execution of large parallel applications requiring more processors than available on a single site. We assume that the input of a whole job is a single data file that is needed by all components of the job. These input files are stored, and possibly replicated, at different sites, and they have unique logical names. We assume that there is a replica manager that maps the logical file names specified by jobs onto their physical location(s).

Job requests are supposed to be *unordered*, meaning that a job only specifies the numbers of processors needed by its components, but not the sites where these components should run. It is the task of the grid scheduler to determine in which cluster each job component should run, to move the input file to those clusters before the job starts, and to start the job components simultaneously. We currently assume that the executables of the jobs are already present at the execution sites, although our mechanisms can easily be extended to the case when this is not true. The sites where a job's components run are called its *execution sites*, and the site(s) where its input file resides are its *file sites*. In this paper we assume that there is a single central grid scheduler, and the site where it runs is called the *submission site*. Of course we are aware of drawbacks of single central instances and currently we are working on extending our design to several distributed grid schedulers.

2.3 The Processor and Data Co-Allocator

The basic mechanisms for co-allocation in real systems have existed for several years, for instance, in the form of the DUROC component of the Globus Toolkit [3]. However, they are rudimentary, their use is still rather an exception, and the accompanying resource allocation policies are lacking. Another important drawback of DUROC is that it requires jobs to specify exactly the sites where their components should run. Therefore, we have developed our own Processor and Data Co-Allocator (PDCA) prototype of a co-allocation service in our DAS system. It employs Grid services such as job execution, file transfer, replica management and security and authentication. The components of the PDCA have been implemented using the Java CoG

Kit [1], which provides a subset of the Globus Toolkit Grid services from a higher-level framework.

The PDCA accepts jobs requests and uses a placement algorithm (see Section 3.1) to try to place jobs. We use the Globus Resource Specification Language (RSL) [3] as our job description language with the RSL "+" construct to aggregate job components to form multi-requests. On success of job placement, the PDCA first initiates the third-party file transfers from the selected file sites to the execution sites of the job components, and then it tries to claim the processors allocated to the components (see Section 3.3). Once the claiming is successful, the components are sent for execution to their respective execution sites. Synchronization of the start of the job components is achieved through a piece of code which delays the execution of the job components until the job actually starts (see Section 3.3).

3 Scheduling jobs

In this section we discuss the three elements of how the PDCA schedules jobs that require processor and data co-allocation in multiclusters. First, we present the Close-to-Files algorithm for placing single jobs in the system. Second, we describe the placement queue, which holds jobs that cannot be placed immediately when they are submitted. The third element is the mechanism used for actually claiming processors when a job has been successfully placed.

3.1 The Close-to-Files algorithm

Placing a job in a multicluster means finding a suitable set of execution sites for all of its components and suitable file sites for the input file. (Different components may get the input file from different locations.) The most important consideration here is of course finding execution sites with enough processors. However, when there is a choice among execution sites for a job component, we choose the site such that the (estimated) delay of transferring the input file to the execution sites is minimal. We call the placement algorithm doing just this the Close-to-Files (CF) algorithm. It uses the following parameters in its decisions:

- **The numbers of idle processors in the sites of a grid:** A job component can only be placed on an execution site which will have enough idle processors at the job start time.
- **The file size:** The size of the input file, which enters in the estimates of the file transfer times.

```

1. order job components according to decreasing size
2. for each (job component  $j$ ) do
3.    $S_j$  = set of potential execution sites
4.   if ( $S_j \neq \emptyset$ ) then
5.     select an  $E \in S_j$ 
     /* we select the first, see text */
6.   else
7.      $P_j$  = set of potential pairs of execution site, file
       site
8.     if ( $P_j \neq \emptyset$ ) then
9.       for each ( $(E, F) \in P_j$ ) do
10.        estimate the file transfer time  $T_{(E,F)}$ 
11.        select the pair  $(E, F) \in P_j$  with minimal
           $T_{(E,F)}$ 
12.       for each (file site  $F'$  of the job) do
13.        insert  $(E, F')$  into the history table  $H$ 
14.     else job submission fails

```

Figure 1. Pseudo-code of the Close-to-Files job-placement algorithm.

- **The network bandwidths:** The bandwidth between a file site and an execution site gives the opportunity to estimate the transfer time of a file given its size. Therefore, we need to measure and forecast the network bandwidth when selecting the execution sites in order to minimize the file transfer times.

When given a job to place, CF operates as follows (the line numbers mentioned below refer to Figure 3.1). It first orders the components of a job according to decreasing size (line 1), and then tries to place job components in that order (loop starting on line 2). The decreasing order is used to increase the chance of success for large components.

For a single job component j , CF first determines the set S_j of *potential execution sites* (line 3); these are the file sites of the job that have enough idle processors to accommodate the job component. If S_j is not empty, we pick an element from it as the execution site of the component (line 5). (We currently have a function that returns the names of the file sites in alphabetical order, and we pick the first.)

If the set S_j of potential execution sites is empty (line 6), we might consider all pairs of execution sites with sufficient idle processors and files sites of the job, and try to find the pair with the minimal file transfer time. In large grids, there may be so many sites that is not very efficient, and so CF maintains a *history table* H with a subset of pairs of

execution sites and file sites to consider. From H , we first select all *potential pairs* (E, F) of execution site, file site, with E having a sufficient number of idle processors for the job component and F being a file site of the job (line 7). If no such pair exists in H , the job component, and hence the whole job, currently cannot be placed (line 14). Otherwise, CF estimates for each selected pair the file transfer time from the file site to the execution site (line 10), and picks the pair with the lowest estimate (line 11). If (E, F) is the pair selected, CF inserts into H all pairs (E, F') with F' a file site of the job (lines 12, 13). Note that if the history table is initially empty, it will remain empty. Therefore, it has to be initialized with some set of suitable pairs of execution and file sites.

For comparison with the CF policy, we also consider a version of the Worst-Fit (WF) algorithm. WF places the job components in decreasing order of their sizes on the execution sites with the largest (remaining) number of idle processors. In case the files are replicated, we select for each component the replica with the minimum estimated file transfer time to that component's execution site. Again, if any of a job's components cannot be placed, the whole job currently cannot be placed.

Note that both CF and WF may place multiple job components on the same cluster. We also remark that both CF and WF make perfect sense in the absence of co-allocation.

3.2 The placement queue

When a job is submitted to the system, the PDCA tries to place it immediately according to either the CF or the WF policy. However, this placement may fail. Therefore, the PDCA maintains a so-called *placement queue* that holds all jobs that have been submitted but that have not yet been successfully placed. A job that upon its submission fails to be placed, is appended to the tail of this queue.

The PDCA regularly scans the placement queue from head to tail to see whether any job in it can be placed. For each job in the queue we maintain its number of *placement tries*, which is the number of times the PDCA has (unsuccessfully) tried to place it, including the initial try at the job's submission. When the number of placement tries of a job exceeds a threshold, the whole job submission fails and the job is removed from the queue. (This threshold can be set at ∞ , never removing any job.) The time interval between successive scans of the placement queue is adaptive; it is computed as the product of the average number of placement tries of the jobs in the placement queue and a fixed time interval (which is a parameter of the PDCA).

Figure 2 shows the timeline of a job's submission from its Job Submission Time until its Job Finish Time (most of it will be explained in Section 3.3). The time instant of the successful placement of a job is called its Job Placement Time (JPT, point B in Figure 2), and the time elapsed between a job's Job Submission Time and its JPT is its *placement time*.

Figure 2. The timeline of a job submission.

3.3 Claiming processors

When a job is successfully placed by CF or WF, it cannot be started immediately because its input file will in general not be present at its execution sites. Of course, we can add a small piece of code to the executable of the job with the sole purpose of delaying the application and start the job rightaway, but this is wasteful of processor time. Another possibility is to estimate the job's start time based on the file transfers that have to take place, and reserve the processors in the job's execution sites. However, our testbed, in which openPBS [4] is used (see Section 2.1), and many resources in grids in general, do not support processor reservations. Therefore, we employ a work-around method to achieve co-allocation in which we postpone *claiming* the processors until a later time, running the risk that they are then not available anymore.

In this method, first of all, after the successful placement of a job, its File Transfer Time (FTT) is calculated as the maximum of all the estimated transfer times. The Job Start Time (JST) is then estimated as the sum of its JPT and its FTT (see Figure 2). We then try to claim the processors at the Job Claiming Time (JCT, point C in Figure 2), which is set at $L \times \text{FTT}$ after the JPT (L is a parameter of the PDCA, where $0 < L < 1$). If the claiming is successful, the job's components are started to which a small piece of code has been added with the sole purpose of delaying the execution of the job barrier until the job start time. Synchronization is achieved by making each component wait on the barrier until it hears from all the other components.

If the claiming of processors for any of the job components fails, then the *claiming try* for the whole job fails. In that case, we perform successive claiming tries. For each such try we recalculate the JCT by adding to the current JCT the product of L and the time remaining until the JST. When no claiming try succeeds before we reach the JST, we initialize the JST to an artificial value by adding a fixed interval to the current JST, and we repeat the same claiming procedure. When these additional claiming tries still fail and the number of times we have performed this whole claiming procedure exceeds some threshold which may again be ∞ , the job submission fails.

We call the time between a job's JPT and the successful claiming of the processors for it the Processor Gained Time (PGT) of the job, and the time between the successful claiming and the actual job start time its Processor Wasted Time (PWT) (see Figure 2). During the PGT, jobs submitted through other schedulers than our grid scheduler can use the processors. The time from the submission of the job until its actual start time is called the Total Waiting Time (TWT) of the job.

Variations on our placing and claiming scheme are of course possible. For instance, when some but not all components of a job can be placed, we may claim the processors for those components immediately and hold them for some amount of time, and try to find resources for the rest later. Or, instead of sticking to the original choices of execution and file sites, we may try to find others if claiming is not successful.

4 Experimental Setup

In this section we describe the setup of the experiments we have conducted to assess the performance of CF and WF. We describe the test application that we use in our experiments and we detail the workload we impose on the system. In our experiments, we did not impose limits on the number of placement and claiming tries. The fixed interval that is the basis of the time between two successive scans of the placement queue (Section 3.2) and the fixed interval after which we start claiming again (Section 3.3) are set to 1 and 4 minutes, respectively. The parameter L determining when to start claiming is set at 0.75. As there are only five clusters in our testbed, we initialize the history table H to contain all possible pairs of execution site, file site.

4.1 The test application

In our experiments, we use an artificial application, which consists of a real parallel application to which, because it uses little input data itself, we have added large input files. These files are only transferred, but not actually used, and the application simply deletes them when it starts after having verified their presence. We have previously adapted this application, which implements an iterative algorithm to solve the two-dimensional Poisson equation on the unit square, to co-allocation on the DAS [6]. The unit square is split up into a two-dimensional pattern of rectangles of equal size among the participating processors. When we execute this application on multiple clusters, this pattern is split up into adjacent vertical strips of equal width, with each cluster using an equal number of processors.

4.2 The workload

In our experiments, we put a workload of jobs to be co-allocated on the DAS that all run the application of Section 4.1, in addition to the regular workload of the ordinary users. Jobs have 1, 2, or 4 components of equal size, which can be 8 or 16. Each job component requires the same single file of either 2, 4, or 6 GByte. For the input files we consider two cases, one without file replication and another with each file replicated in three different sites. In either case, the file(s) are randomly distributed across the file servers. For a single job, its number and size of components, and the size of its input file are picked at random and uniformly. The execution time of our test application ranges between 90.0 and 192.0 seconds.

We assume the arrival process of our jobs at the submission site to be Poisson. Based on the above description and the total number of available processors in the DAS, we generate two workloads of 200 jobs, W_{30} which utilizes 30% of the system and W_{50} which utilizes 50% of the system. With W_{30} , the last job arrives around 6500 seconds after the arrival of the first job, and with the W_{50} , the last job arrives after 3900 seconds.

4.3 Background load

One of the problems we have to deal with is that we do not have control over the background load imposed on the DAS by other users.

During the experiments, we monitor this background load and we try to maintain it at 30%–40%. When this utilization

(a) CF without replication

(b) WF without replication

(c) CF with replication

(d) WF with replication

Figure 3. The utilizations for CF and WF with workload W_{30} .

falls below 30% in some cluster, we inject dummy jobs just to keep the processors busy. If the utilization rises above 40% because of jobs from other users and there are dummy jobs, we kill the dummy jobs to lower the utilization to the required range. In case the background load rises above 40% and stays there for more than a minute without any of our dummy jobs, the experiment is declared void and is repeated.

5 Performance Results

In this section, we present the results of the experiments described in Section 4.

5.1 Presentation of the results

We will present the results of each experiment with three graphs. The first of these shows the different utilizations in the system. Here, the background load is the utilization due to the jobs of the regular users. The *actual co-allocation load* is the utilization due to our own co-allocation workload *without* the load incurred between the Job Claiming Time and the Job Start Time. We also show the PWT utilization, which is the utilization due to the Processor Wasted Time of the co-allocated jobs, and the PGT utilization, which is the fraction of the system capacity gained because we do not claim processors immediately when job placement is successful. As the PDCA keeps track of the processors on which it has placed jobs, the PGT utilization can only be used by local jobs (or jobs submitted by other schedulers), but not by other jobs submitted through the PDCA. The *real total utilization* in the system is the sum of the background load, the actual co-allocation load, and the PWT load.

The second graph shows the average job Placement Time and job FTT (the sum of these two is the Total Waiting Time (TWT)). The third graph presents the average numbers of placement and claiming tries. It should be noted here that the results for the different workloads are presented with different scales to make them visible.

5.2 Results for the workload of 30%

Figure 3 shows the utilizations of our experiments with workload W_{30} for CF and WF with and without replication. As can be seen, in all four cases the real total utilization is about 70%. Because the actual co-allocation load is about 30%, because the experiments finished shortly after the submission of the last job, and because the maximum length of the placement queue is only 10, we conclude that the system is stable with workload W_{30} .

Figure 4 shows that the average job FTT with the CF policy is smaller than with the WF policy, both with and without replication. Furthermore, with replication, CF is more successful in finding execution sites "closer" to the file sites, which results in a smaller average job FTT. As a result, the TWT of the jobs is also reduced. The decrease in the average job FTT when the files are replicated for CF is expected because the number of potential execution sites increases.

For both placement policies, the average TWT increases as the number or the size of the job components increases, both with and without replication. The explanation for this is first that more time is likely to be spent waiting for clusters to have enough processors available simultaneously. Second, the job FTT goes up because more files are likely to be moved with more job components, causing more placement tries (see Figure 5). The increase in the number of placement tries causes a rise in job Placement Time since the PDCA waits for an interval between successive placement tries (see Section 4).

The average number of claiming tries with this workload are quite low as shown in Figure 5. Note that an average number of claiming tries equal or close to 1 means that we start claiming after $0.75 \times \text{FTT}$ (see Section 3.3), and that the PGT utilization is three times the PWT utilization. We conclude that the combination of CF and replication performs best.

5.3 Results for the workload of 50%

Figure 6 shows the utilizations of our experiments with workload W_{50} for CF and WF with and without replica-

tion. Our main purpose with this workload is to see to what utilization we can drive the system. From the figure we see that the real total utilization during our experiments is 70–80%. However, the actual co-allocation load is well below 40%, the experiments are only finished long after the last job arrival, and the length of the placement queue length goes up to 30, which shows that the system is saturated. So we conclude that we can drive the real total utilization not higher than what we achieve here.

With this workload and with the CF policy, clusters "close" to files will mostly be occupied forcing more long file transfers. As a result, the average FTT for CF and WF are relatively close to each other both with and without replication as shown in Figure 7.

Since the system is saturated with this workload, more time is spent waiting for clusters to have enough processors simultaneously. This explains the increase in the number of placement tries (see Figure 8) and job Placement Times (Figure 7) as the number or size of the job components increases. However, similarly as with workload W_{30} , the numbers of claiming tries (Figure 8) are still quite low.

5.4 CF with high background loads

It may be expected that the success of our workaround method for processor reservation depends on the size and variation of the background load. In our previous experiments we observed that the number of claiming tries is quite low, indicating the success of our workaround method for processor reservation with 30–40% background load. However, this background load is fairly low. Therefore, to further test this method we performed experiments with workload W_{30} with CF and with replication while trying to maintain a background load of 50% or 60% (again employing dummy jobs as described in Section 4.3).

Figure 9 shows the results of these experiments. For both background loads, the actual co-allocation load is much lower than 30%, and the experiments take (much) more time than expected (the last job arrives at about time 6500), and so the system is saturated. The real total utilization is 75–80%. Figure 10 shows that the numbers of claiming tries are still quite low even with these much higher background loads, indicating the success of our workaround method for reservation. It should be noted that the increase in the number of claiming tries has the positive effect of reducing the PWT in favor of the PGT.

Figure 4. The average Placement Time and File Transfer Time for CF (left bars) and WF (right bars) with workload W_{30} .

6 Related Work

To the authors' knowledge, this is the first performance study of processor and data co-allocation in multicluster systems, and in particular, the first such study that involves a real implementation in a real testbed.

In our previous work [8, 9, 6] we have studied with simulations processor co-allocation in multiclusters with space sharing of rigid jobs for a wide range of such parameters as the number and sizes of the job components, the number of clusters, the service-time distributions, and the number of queues in the system. In [13, 14], co-allocation (called multi-site computing there) is studied also with simulations, 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 is that for r less than or equal to 1.25, it pays to use co-allocation. In [20] an architecture for a grid superscheduler is proposed, and three job migration algorithms are simulated. However, there is no real implementation of this scheduler, and jobs are confined to run within a single subsystem of a grid, reducing the problem studied to a traditional load-balancing problem.

In [18], the Condor class-ad matchmaking mechanism for matching single jobs with single machines is extended to "gangmatching" for co-allocation. The running example in [18] is the inclusion of a software license in a match of a job

and a machine, but it seems that the gangmatching mechanism might be extended to the co-allocation of processors and data.

Thain et al. [22] describe a system that links jobs and data by binding execution and storage sites into I/O communities that reflect the physical reality. A job requesting particular data may be moved to a community where the data are already staged, or data may be staged to the community in which a job has already been placed. Other research on data access has focused on the mechanisms for automating the transfer of and the access to data in grids, e.g., in Globus [3] and in Kangaroo [21], although there less emphasis is placed on the importance of the timely arrival of data.

In [19], the scheduling of sequential jobs that need a single input file is studied in grid environments with simulations of synthetic workloads. Every site has a Local Scheduler, an External Scheduler (ES) that determines where to send locally submitted jobs, and a Data Scheduler (DS) that asynchronously, i.e., independently of the jobs being scheduled, replicates the most popular files stored locally. All combinations of four ES and three DS algorithms are studied, and it turns out that sending jobs to the sites where their input files are already present, and actively replicating popular files, performs best.

In [15], the creation of abstract workflows consisting of application components, their translation into concrete workflows, and the mapping of the latter onto grid resources is considered. These operations have been implemented using the Pegasus [12] planning tool and the Chimera [16] data definition tool. The workflows are represented by DAGs, which are actually assigned to resources using the Con-

(a) Without replication

(b) With replication

Figure 5. The average number of placement and claiming tries with workload W_{30} .

(a) CF without replication

(b) WF without replication

(c) CF with replication

(d) WF with replication

Figure 6. The utilizations for CF and WF with workload W_{50} .

(a) Without replication

(b) With replication

Figure 7. The average Placement Time and File Transfer Time for CF (left bars) and WF (right bars) with workload W_{50} .

(a) Without replication

(b) With replication

Figure 8. The average number of placement and claiming tries with workload W_{50} .

dor DAGMan and Condor-G [17]. As DAGs are involved, no simultaneous resource possession implemented by a co-allocation mechanism is needed.

In the AppLes project [10], each grid application is scheduled according to its own performance model. The general strategy of AppLes is to take into account resource performance estimates to generate a plan for assigning file transfers to network links and tasks (sequential jobs) to hosts.

7 Conclusions

We have addressed the problem of scheduling multi-component jobs that require both processor and data co-allocation in multicluster systems such as our DAS. We have proposed the Close-to-Files (CF) policy that tries to

place job components in clusters that already contain the input file, and a method to deal with the lack of a capability of the local schedulers to do processor reservations. Our results show that the combination of the CF policy and file replication is very beneficial, that the system utilization with a mixed workload of co-allocated and local jobs can be driven to about 80%, and that workaround mechanism for processor reservation can be used without wasting much processor time.

As future work, we are planning to create better processor-reservation mechanisms, to remove the bottleneck of a single global queue for jobs that need co-allocation, to add better fault-tolerance mechanisms (e.g., tolerating submission site crashes and file-transfer failures), and to consider more heterogeneous systems. We are also planning to incorporate the segmentation of a large input file and then scheduling the transfer of only the required chunks to the job com-

(a) 50% background load

(b) 60% background load

Figure 9. The utilizations for the CF algorithm with workload W_{30} with different background loads.

(a) 50% background load (left bars) and 60% background load (right bars)

(b) The average number of placement and claiming tries

Figure 10. The average Placement Time and File Transfer Time (left), and the number of placement and claiming tries (right) of the CF algorithm with workload W_{30} with different background loads.

ponents as a means to optimize the transfer times and the required space. In addition, we want to allow flexible jobs that only require total numbers of processors (although the way of dividing the input files across the job components is then not obvious),

References

- [1] Java CoG Kit. <http://www-unix.globus.org/cog/>.
- [2] The Distributed ASCI Supercomputer (DAS). <http://www.cs.vu.nl/das2>.
- [3] The Globus Toolkit. <http://www.globus.org/>.
- [4] The Portable Batch System. www.openpbs.org.
- [5] S. Ananad, S. Yoginath, G. von Laszewski, and B. Alunkal. Flow-based Multistage Co-allocation Service. In B. J. d'Auriol, editor, *Proc. of the International Conference on Communications in Computing*, pages 24–30, Las Vegas, 2003. CSREA Press.
- [6] S. Banen, A. Bucur, and D. Epema. A Measurement-Based Simulation Study of Processor Co-Allocation in Multiclustur Systems. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *9th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *LNCS*, pages 105–128. Springer-Verlag, 2003.
- [7] A. Bucur and D. Epema. Local versus Global Queues with Processor Co-Allocation in Multiclustur Systems. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *8th Workshop on Job Scheduling Strategies for Parallel Pro-*

- cessing, volume 2537 of *LNCS*, pages 184–204. Springer-Verlag, 2002.
- [8] A. Bucur and D. Epema. The Performance of Processor Co-Allocation in Multiclusted Systems. In *3rd IEEE/ACM Int'l Symposium on Cluster Computing and the GRID (CC-Grid2003)*, pages 302–309. IEEE Computer Society Press, 2003.
 - [9] A. Bucur and D. Epema. Trace-Based Simulations of Processor Co-Allocation Policies in Multiclusted Systems. In *12th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-12)*, pages 70–79. IEEE Computer Society Press, 2003.
 - [10] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middle-ware for the Grid. pages 75–76, 2000.
 - [11] K. Czajkowski, I. T. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. In *Proc. of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 219–228, 1999.
 - [12] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. M. K. Vahi, S. Koranda, A. Lazzarini, and M. A. Papa. From Metadata to Execution on the Grid Pegasus and the Pulsar Search. Technical report, 2003.
 - [13] C. Ernemann, V. Hamscher, U. Schwiigelshohn, 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] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. Enhanced Algorithms for Multi-Site Scheduling. In *3rd Int'l Workshop on Grid Computing*, pages 219–231, 2002.
 - [15] E. D. et al. Mapping Abstract Complex Workflows onto Grid Environments. *J. of Grid Computing*, 1:25–39, 2003.
 - [16] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *14th Int'l Conf. on Scientific and Statistical Database Management (SSDBM 2002)*, 2002.
 - [17] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 7–9, San Francisco, California, August 2001.
 - [18] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *12th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-12)*, pages 80–89. IEEE Computer Society Press, 2003.
 - [19] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 Edinburgh, Scotland*, July 2002.
 - [20] H. Shan, L. Olike, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *Supercomputing '03*, 2003.
 - [21] D. Thain, J. Basney, S. son, and M. Livny. The Kangaroo Approach to Data movement on the Grid. In *proc. of the Tenth IEEE Symposium on High Performance Distributed Computing, San Francisco, California*, August 2001.
 - [22] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the Well: Creating Communities for Grid I/O. In *Proc. of Supercomputing, Denver, Colorado*, November 2001. <http://www.cs.wisc.edu/condor/nest/papers/community.pdf>.
 - [23] R. van Nieuwpoort, J. Maassen, H. Bal, T. Kielmann, and R. Veldema. Wide-Area Parallel Programming Using the Remote Method Invocation Method. *Concurrency: Practice and Experience*, 12(8):643–666, 2000.