END-TO-END CACHE SYSTEM FOR GRID COMPUTING: DESIGN AND EFFICIENCY ANALYSIS OF A HIGH-THROUGHPUT BIOINFORMATIC DOCKING APPLICATION

Jose Ignacio Garzon[1]
Eduardo Huedo[2]
Ruben Santiago Montero[2]
Ignacio Martin Llorente[2]
Pablo Chacon[1]

Abstract

Cache techniques are an efficient tool to reduce latency times in transfer operations through Grid systems. Although different approximations to introduce cache facilities into Grid computing have already been studied, they require intrusive modifications of Grid software and hardware. Here, we propose an end-to-end cache system that is implemented over scheduling services. This cache system requires neither changes in the Grid software nor introduction of new software in the Grid resources. Parallel Grid adaptation of many high-throughput computing applications that use the same data intensively could enjoy great benefits from our cache system. The maintenance of cacheable data in the resources of already-executed tasks allows faster executions of future tasks assigned to the same resources. To analyze the performance of our end-to-end cache system, we tested it with a new protein–protein docking application. The obtained results confirm our cache system's robustness and efficiency gain for this kind of high-throughput application.

Key words: grid computing, cache system, high-throughput application, scheduler, protein-protein docking

1 Introduction

The phenomenal growth in computer networks and communication over the Internet has engendered new paradigms to meet the needs of hard computational applications. Grid computing exploits high computational power and gives access to large numbers of distributed data using a geographically distributed global and heterogeneous resource pool (Foster and Kesselman, 1999; Baker et al. 2002; De Roure et al. 2003). Using these delocalized heterogeneous resources controlled by different administrations requires solving a wide range of problems related to their coordination and security. To face such problems, different approximations have provided middleware, such as Globus (Foster and Kesselman, 1997), GRIA (Surridge et al. 2005) or UNICORE (Romberg, 2002), to supply services to manage Grid environments. Among the services provided by middleware, efficient data communication through the Grid environment is critical due to its distributed nature. Caching techniques can improve data communications. These techniques are an efficient way to manage temporary data and reduce latency times when common data are accessed frequently.

Different approximations to introduce cache mechanisms into Grid computing have already been studied; however, they have mainly been oriented towards the use of collaborative cache systems working at different levels in the Grid middleware. Therefore, middleware modifications and the introduction of specific hardware into the environment are necessary. Here, we propose an end-to-end cache system that can be implemented over the procedure that schedules tasks to resources. It requires neither modification of the Grid middleware nor installation of new services in the Grid resources. This novel cache system is inspired by Web caches. Briefly, Grid resources use scheduler facilities to maintain cache buffers. These cache buffers contain the input files of previous executions, which can be reused for future tasks. This strategy lowers the transfer time for task settling.

This cache system reduces the execution response time of high-throughput computing applications that operate on the same data repeatedly. A wide range of applications in different scientific areas, such as bioinformatics, physics or astrophysics, can make good use of this cache system. Grid adaptation of these applications usually implies parallel execution of tasks that use the same data. A cache system will facilitate the execution of tasks by reusing data from previous tasks. Here, we study the performance of a new protein–protein docking application. Protein–protein docking predicts the three-dimensional structure of a

The International Journal of High Performance Computing Applications, Volume 00, No. 0, xxxxxx 2009, pp. 000–000
DOI: 10.1177/1094342009350469

Figures 3–7 appear in color online: http://hpc.sagepub.com

[1] CIB-CSIC. RAMIRO DE MAEZTU 9, MADRID 28040, SPAIN (EGARZON@CIB.CSIC.ES)

[2] FACULTAD DE INFORMATICA, UCM. PROF. JOSE GARCIA SANTESMASES S/N, MADRID 28040, SPAIN

macromolecular complex from the unbound structures of its components. It requires a large and costly search of all the possible relative docking positions. The search uses information about the components repeatedly to do such an exhaustive search. The use of our cache system will reduce the transfer time needed in a Grid environment, where parallel tasks explore different regions of the docking search space.

The paper is organized as follows: Section 2 surveys the different cache systems proposed for Grid computing in the scientific community; Section 3 describes our novel end-to-end cache system, showing alternatives for different possible scenarios; Section 4 adapts this cache system over the GridWay meta-scheduler; Section 5 describes the challenging bioinformatic application (FRODOCK) used to test the cache performance; Section 6 compares the performance achieved by FRODOCK over a Grid environment with and without the new end-to-end cache system; and Section 7 presents conclusions and future work.

2 Grid Cache Systems Overview

A Grid environment is a distributed system, usually over the Internet, where large data entities must be moved frequently between different sites. This requires intensive and efficient use of network and storage resources to minimize transmission latency times. Although services such as Grid file transfer protocol (GridFTP) (Bresnahan et al. 2007) and reliable file transfer (RFT) (Madduri et al. 2002) in Globus allow data transmission through Grid environments, they do not provide policies to organize transmissions efficiently. In this sense, data movement in a Grid environment is similar to intensive hyperText transfer protocol (HTTP) traffic over the World Wide Web (WWW). Although the characteristics of data movement in both environments are not exactly the same (more intensive communication in the WWW, but with lower data load and a different communication model, etc.) the mechanics to improve one environment can be adapted to the other. By storing, replicating, and managing data in temporal and distributed repositories, WWW caching (Barish and Obraczka, 2000) provides a set of techniques to reduce traffic latency. An obvious alternative is the adaptation of Web caching mechanics to Grid environments to boost traffic efficiency. Different variations of Web caching appear as a function of different configurations: the location of the cache repository (in the client machine that requests the information, in the provider machine, in a middle node through the communication path), the level of transparency of the cache to the users, the use of communication between different cache systems in different machines, the refreshment and replacement policies of the cache repository, and the mechanics to secure data consistency. All these features, when correctly adapted, provide mechanics to create suitable cache functionalities for a Grid environment. Taking into account the available middleware services and the nature of the applications to be supported, different approximations can be implemented.

Initial versions of the Globus toolkit middleware included a data movement and access service called Global Access to Secondary Storage (GASS). This service provided applications with access to remote files (Bester et al. 1999). Among the features incorporated into the service to obtain high performance, GASS implemented movement strategies where cache space was kept over resources, reducing data movement through the Grid. In this way, GASS provided a distributed, multi-user system cache. However, this cache system scaled poorly, due to data replica management. The GASS service was finally considered obsolete and removed from version 4 of the Globus Toolkit. Yonny et al. (2007) have proposed a basic infrastructure for the management of collaborative caches to operate and control different cache mechanics dynamically in a Grid environment. In this cooperative system, different caches exchange both data and metadata. Therefore, it is mandatory to implement cache system services in all the resources, since each cache system depends on local and Grid facilities. As this infrastructure is for scientific applications, they dismiss issues such as coherency and consistency of duplicate data because of the read-only nature of the data in use. Other authors have studied the use of a remote shared cache for Grid resources (Tierney et al. 2000). This system is a hardware-based solution, as physical elements must be introduced into the Grid environment to act as cache servers. Here, the concept of "cache" means fast storage with a temporal nature. The shared cache for all the resources in the Grid cache is implemented with a Distributed-Parallel Storage System (DPSS). Briefly, a DPSS consists of a set of low-cost workstations, each one of which has several parallel disks. A DPSS distributes data over these workstations and allows concurrent data access. The workstations keep a cache window of the data, while the parallel disks act as tertiary storage that maintains the full data repository. Each data source deposits data in the cache, and data consumers read from and eventually modify the information. The aim of this system is to provide fast access to a large number of data produced in high-speed data streams resulting from many scientific applications. This cache system does not require a consistency policy, as data are not replicated. Other hardware-based cache systems have been developed. The concept of a Storage Resource Manager (SRM) is introduced as a Grid component to support storage management of large distributed datasets (Shoshani et al. 2004). Although this is not a cache service, it includes cache techniques to make data access more efficient. Cache replacement policies over SRMs (Liu and Song, 2008; Otoo and Shoshani, 2003) are proposed with an evaluation model adapted to

the data transfer in Grids. Other approximations also use some of the concepts of caching techniques to provide different Grid functionalities, such as a high-level data replication service proposed in Chervenak et al. (2005). In this approach, services in different resources store replicated data, keeping information about their locations and transferring it through the Grid environment as necessary. Although the system uses a replication policy similar to policies usually present in cache systems, the aim of the service is fast dissemination of information over the Grid environment, without taking into account the reduction of Grid traffic or temporal storage.

3 End-to-End Cache System

All the approximations to a Grid cache system described above require the introduction of new elements in the Grid environment. They introduce either new software services on the Grid resources or new hardware to provide cache services. This intrusive implementation entails a complex process. Tasks, such as the dissemination of the required software, introduction of hardware, and modification of previous services and protocols, could be very costly and laborious. Moreover, the intrusive implementation must be accepted by the Grid community, which must collaborate in the implementation process. Although these drawbacks are not intractable and are part of the creation and evolution of Grid environments, it is possible with existing middleware services to introduce cache facilities without intrusive strategies such as those mentioned above. We present a Grid cache system with cache buffers implemented over resources with no new hardware or software requirements. We designed this cache system to facilitate the execution of scientific high-throughput applications in which parallel tasks operate on shared data, providing similar GASS functionality of earlier versions of Globus.

This new end-to-end cache system takes advantage of the necessity of scheduling. Grid scheduling (Foster and Kesselman, 1999; Schopf, 2004) is the task of discovering, evaluating, and allocating resources in the Grid to perform tasks. Before the Grid executes a task, a staging process takes place to choose a suitable resource and to create an application environment on it for the input data. To this end, the scheduler transfers the files to the resource and executes a batch scheduling routine. Similarly, when the task completes the execution, the scheduler must recollect output data and move the data to the user's location. All these stages are done by scheduling services that interact with local services in the resources and with middleware services to get information about resources and data transfer. A scheduler service acts as a proxy service, connecting users with the Grid's computational potential while hiding the complexity of middleware services in a heterogeneous environment. A cache system can be implemented easily by modifying the scheduler to run cache repository management operations in addition to its regular staging and recollection actions. In this cache system, the cache repositories are kept in the resources and management is realized from the user's machine. All of the communication to control the cache repositories takes place over the scheduler services, avoiding intrusive modifications to resources. This cache system works over the services in the middleware layer and is completely transparent to the user.

3.1 End-to-End Cache Architecture

The target data that our system aims to cache are mainly application input files. To do a task, the user calls a scheduler service from a Grid resource (the user's node), which distributes executions to computational elements (cluster-worker nodes inside Grid resources). The cache system manages cache directories in all of the worker nodes that execute tasks. In such directories, cacheable input files must be copied. When a cacheable file is needed, the cache system checks whether it can be found in the cache directory (because a previous task has copied it) or whether it must be transferred from the user.

Note that this system, although distributed through the resources of the Grid, works as a local cache for a single user. We have not implemented a multi-user cache system because of the impossibility of staging a multi-user common cache directory with user permissions in worker nodes. As all of the cache management operations are realized by the scheduler, they cannot exceed the user's capabilities. Therefore, the cache directory can only be established over a user-owned storage directory provided by the local administrator, which cannot be accessed by other users. Although the use of replica management techniques can solve this lack of common storage, it has been settled that their introduction would produce scalability problems in the cache implementation, similarly to what happened in the GASS service. This cache system is mainly designed to support high-throughput applications, allowing data reusability for parallel tasks of the same application. The specific nature of this kind of application makes common data for different users unlikely. Moreover, security considerations preclude common storage since it could allow access to users' private data. Coherence mechanisms are also unnecessary because cacheable data are not modified. There is no need to implement a cache replacement policy. Since the disk storage policy is under local administration, it is not possible to determine the maximum available cache size or when a cache file is deleted. Finally, because of the high-throughput orientation of the cache system, data reuse between different applications is not expected. Therefore, instead of a cache replacement policy, it is more suitable to introduce a

mechanism to delete an application's data after all of its tasks have been completed.

As we will show below, this type of cache system significantly improves the performance of applications that use input data intensively or in adaptations of high-throughput applications where parallel processes operate on shared data.

The implementation of transfers is a key design feature of the cache service. Usually, a cluster front-end provides access from the Grid environment to different organizational resources. This cluster front-end receives Grid external requests and, depending on their characteristics, determines the suitable worker node to be employed. All of the communication produced by scheduler services to employ resources must be addressed to cluster front-ends. In some organizations, the secondary storage is shared by all of the computational resources. In other words, copying a file to the front-end makes it accessible to all of the cluster-worker nodes. In this situation, the scheduler can launch the transmission of the input data to the cluster front-end, knowing that these data will be accessible to any worker node of this resource. In this situation, all of the transfer processes can be managed by the scheduler from the user's node. A more complex scenario arises if the secondary storage is not shared. In this case, copying data in the cluster front-end does not guarantee its accessibility by the selected worker node. Thus, the scheduler cannot initialize the transmission process without knowing the final destination. To prevent this situation, the process must be initialized from the worker node after it is chosen.

Because of the existence of these two possible scenarios, the scheduler must implement different transfer schemes in order to manage remote executions. When shared secondary storage is available in the resources, the scheduler maintains local control in the transfer processes by a direct transfer scheme (the transmission is launched from the user's node to the cluster front-end); otherwise, the transfer processes must be initiated remotely from the cluster-worker nodes. In this case, the scheduler must implement a reverse transfer scheme (the transmission is required from the cluster-worker node to the user's node). The employed transfer scheme affects the cache service implementation directly. This implementation must be adapted by establishing two different cache policies: the centralized cache policy, to work over the direct transfer scheme, and the remote cache policy, to work over the reverse transfer scheme. The next sections describe these two operational modes.

3.1.1 Centralized Cache Policy (Shared Secondary Storage) In this case, the data can be directly pre-loaded from the user, since the scheduler service can reach the cluster-worker node storage. The scheduler must assess the availability of the cacheable input files in the target resources. To do that, this service maintains a cache database containing information about which files have been copied to the different cluster front-ends. With this database, the scheduler can control data contents in the different Grid resources, determining when it is necessary to transfer a data file to a resource and when this transfer operation can be avoided because a previous instance of the file exists in the resource. The scheduler must interact with the database when staging a task in a resource to determine the convenience of the transfer operations and to actualize the contents in the database, indicating that new cached data are accessible in a resource. In addition, the cluster-worker node's local scheduler routine must be modified to operate with the cache directory, extracting cacheable data from it. Finally, since disk storage is under each resource's local administration, the maintenance of cache directories is compromised. Depending on the administration policy, cache directories can be kept all the time, periodically erased or even deleted just after the execution of a task. For this reason, mechanics to guarantee the consistency of the database must be provided.

3.1.2 Remote Cache Policy (Non-shared Secondary Storage) When direct access to the secondary storage of the cluster-worker nodes cannot be established from the user, the communication will be initiated from the recipient (the cluster-worker node). In this situation, the cluster-worker node determines the convenience of transferring cacheable files, checking the cache directory instead of consulting a database in the user's node. Consequently, all of the cache transfer management is delegated to routines executed on the cluster-worker node. These routines also have to carry out access operations over the cache directory, extracting cacheable data as in the centralized policy. Finally, because no database in the user's node is required in this case, there is no need for consistency operations as in the centralized policy case.

3.2 End-to-End Cache Implementation

To manage the cache replicas and avoid duplications in the cache directories, the input files are labeled with message-digest algorithm (MD5) identifiers to ensure univocal identification. Briefly, the steps followed by the scheduler to manage the cache when a task is launched are:

1. identify the MD5 labels of the cacheable input files;
2. in the selected cluster-worker node, check whether files with the MD5 labels can be found;
3. if they are present, create links to the cache files in the task-work directory;
4. otherwise, transfer the input files from the user to the cache directory in the cluster-worker node and link them from the work directory.

The implementation of these operations depends on the policy established.

3.2.1 Centralized Cache Policy The scheduler interacts dynamically with the database when content in the cluster front-end cache directories is modified. The basic operations of this interaction are:

- insert entry: include record entries for new cacheable files in a resource;
- delete entry: remove records of files that no longer exist in a resource;
- check entry: check the status of a file in a resource;
- recover all entries: give access to the full collection of records.

Note that all of these operations require the resource names and file MD5 identifiers. These operations are accomplished during the process of task launching in the Grid environment. During the staging phase in a resource, together with the regular operations, the scheduler checks in the cache database which cacheable files must be copied to the resource cache directory (see Figure 1).

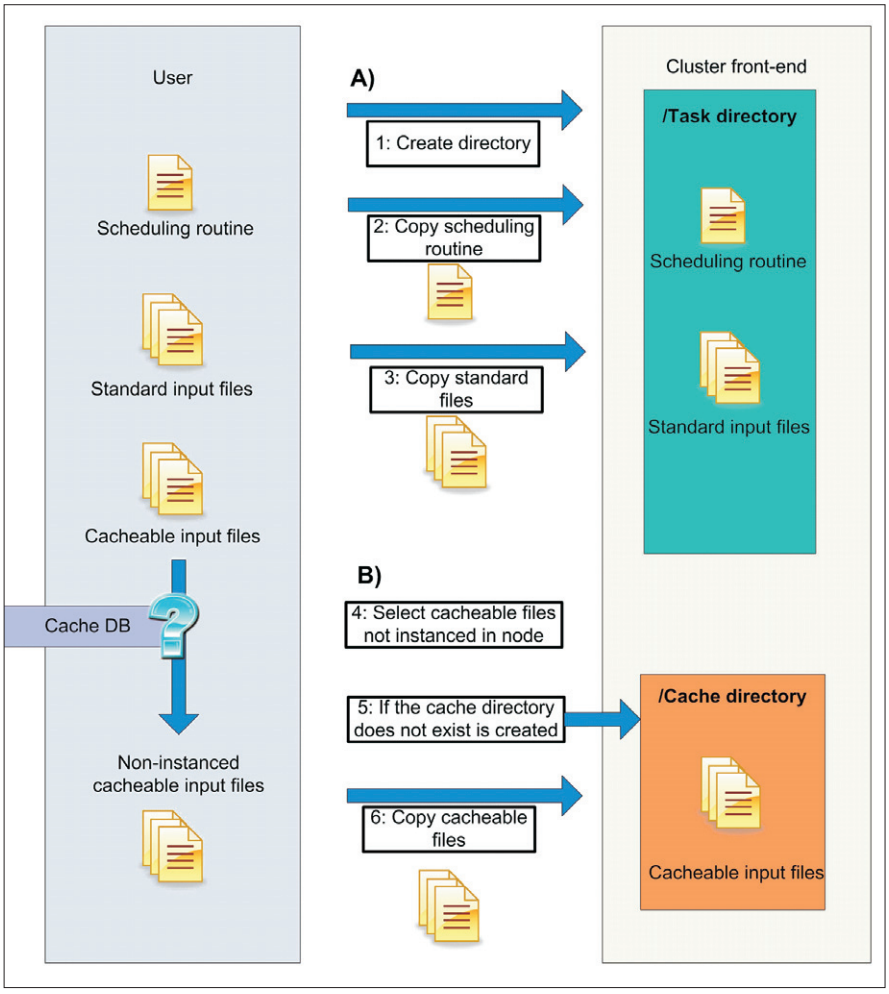

Fig. 1 Differences in the staging process without and with a cache system. Without a cache system (A): a suitable environment directory for the execution of the task is created (step 1) and scheduling routine and non-cacheable files are copied over it (steps 2 and 3). With a cache system (B): in addition to steps 1, 2 and, 3, the cache database is consulted about the existence of the cacheable files over the cluster front-end (4), those that are not in the resource are copied to the cache directory (6). Note that the cache system requires the cache directory creation when it is not found in the cluster front-end (5).

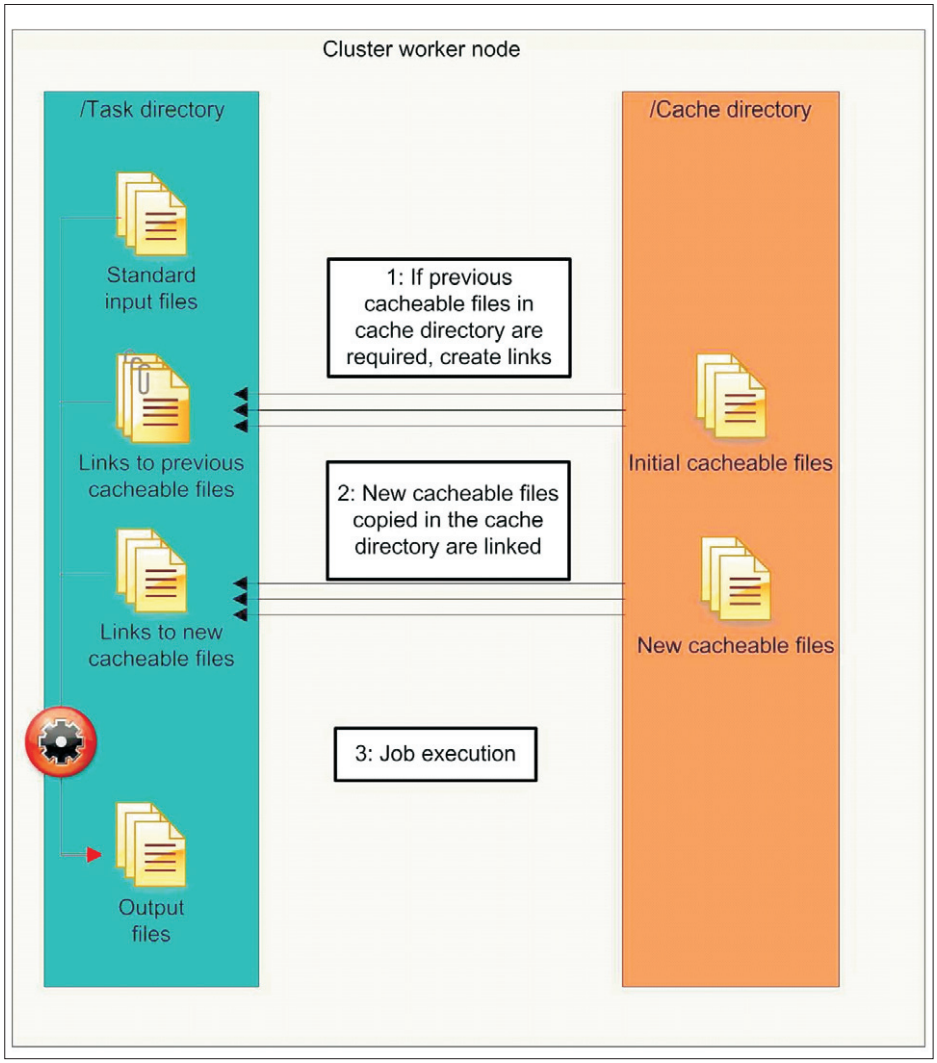

Fig. 2 Comparison between staging local operations in a cluster-worker node without and with a cache system. Without a cache system it is only necessary to launch the task's executable with the corresponding input files (3). To give support to the cache system, links to both previously existing cacheable files in the cache directory (1) and cacheable files copied by the scheduler (2) must be established before launching the executable. Finally, the task executes (3).

Once the scheduler has set the execution environment, the cluster front-end selects a cluster-worker node to do the task. Over this cluster-worker node, a routine carries out local operations for preparing the execution of the task and launches it. The local operations must include both linking previous existing cacheable files in the task directory and new files copied by the scheduler (see Figure 2).

When new files are copied in the cache directory, the scheduler modifies the cache database to include the new instances of the cacheable files in the resource. The modification of the cache database must be done only after obtaining feedback about cache staging. Therefore, it is generally necessary to wait for the end of the task execution (see Figure 3).

Finally, due to lack of control over disk storage policy in the resources, the scheduler periodically checks whether the entries in the cache database are up to date and accurate. At a fixed interval, the scheduler recovers

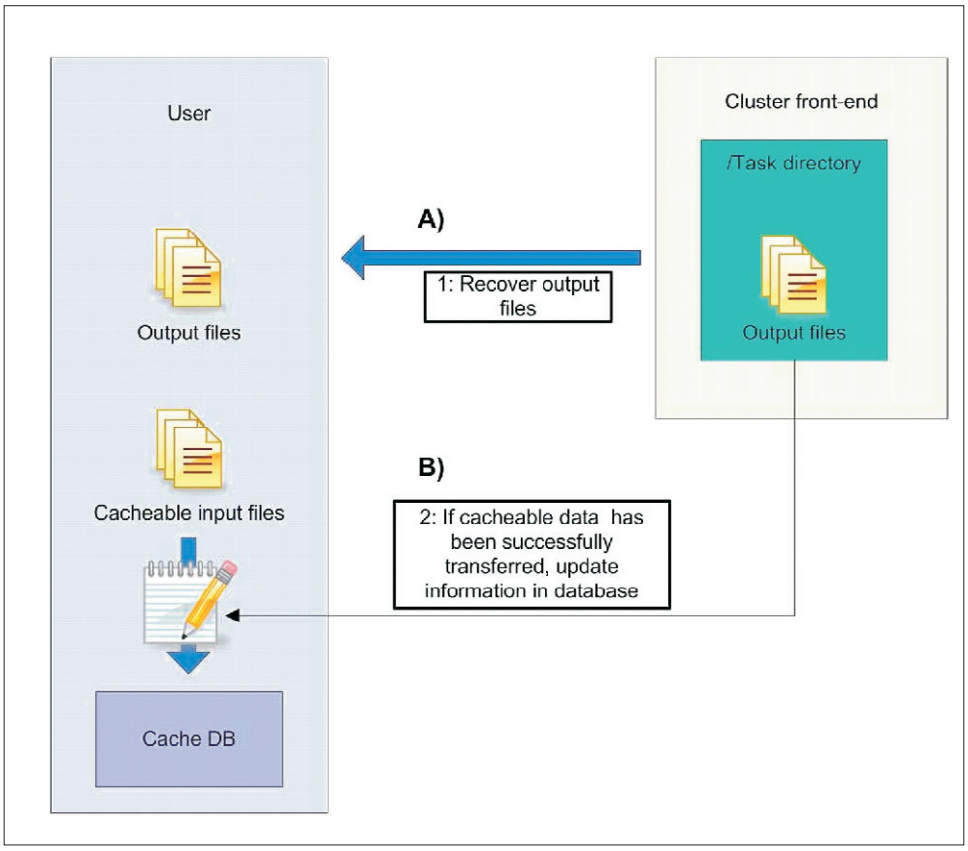

Fig. 3 Recovering process without and with a cache system. Without a cache system (A) the output files obtained in the task's execution must be recovered (1). With a cache system (B), if feedback confirming success in cacheable files transfer is returned, entries for the cacheable files are also inserted into the cache database (2).

all of the records from the cache database and, for each instance, queries the corresponding cluster front-end about the existence of a cacheable file. If the answer is negative, that is, the cache file is not present, the scheduler deletes the instance in the cache database; otherwise, the cache entry is updated to keep track of the last access (see Figure 4). Accomplishing these check operations in a suitable interval guarantees database consistency. In the unlikely case that a task launches while the cache database is inconsistent with data content in the resource, the task will fail due to lack of input information, forcing the scheduler to reschedule the task execution. A suitable interval in the checking process will avoid a second consistency miss. The reduction in the execution productivity due to this situation is acceptable because of its low probability. Finally, note that in the extreme case where the local administration policy erases all data after the execution of a task, the cache system is useless.

3.2.2 Remote Cache Policy As explained in the architecture description, a routine executed by the scheduler over a cluster-worker node will carry out all cache management. During the staging process, the scheduler only creates the task's environment, establishing regular communication with the cluster front-end. This cluster front-end chooses a suitable cluster-worker node where the task directory will be created and a scheduling routine will be transferred. Then, the scheduling routine prepares the execution of the task. In addition to the regular actions, the scheduling routine stages input file processing: that is, it checks whether the cache directory exists, creating it if not, links the available cacheable files of the cache directory to the task directory, and copies the missing ones from the user. After all of these operations, the task can be safely executed (see Figure 5).

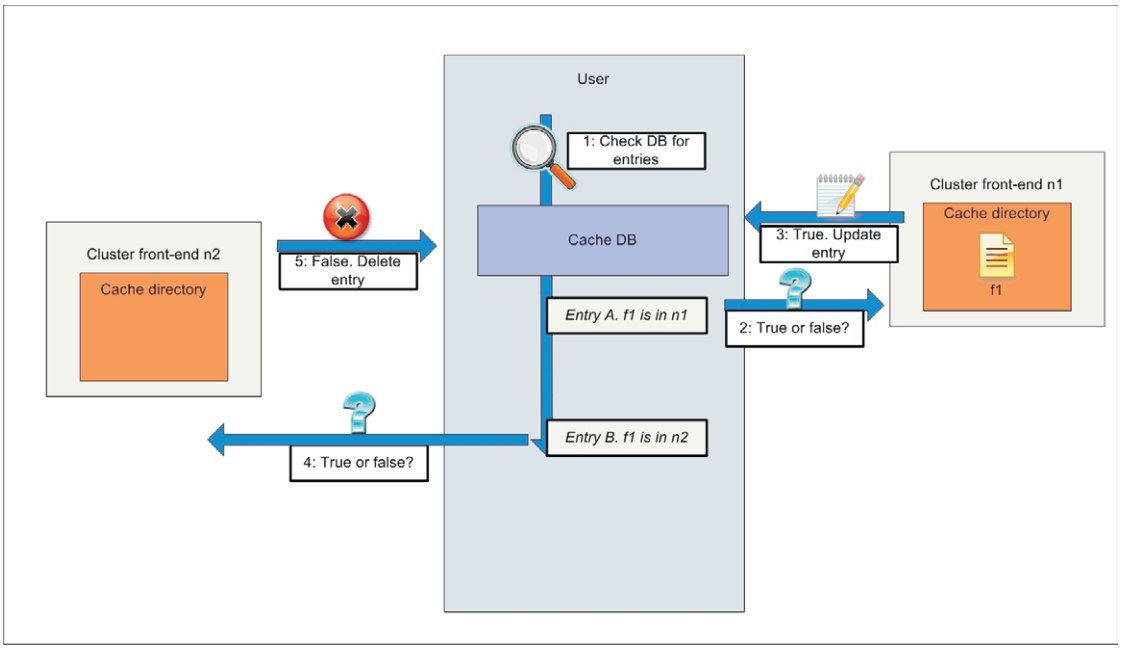

Fig. 4 Cache database update scheme. The scheduler checks the cache database searching entries (1). With Grid middleware services, the scheduler asks whether cacheable file f1 can be found in the cache directory in resource n1 (2). If the answer is positive, then the entry is updated (3). The scheduler asks whether f2 can be found in the cache directory in n2 (4). If the answer is negative, then the scheduler deletes the entry in the cache database (5).

3.3 Policy Comparison

Both policies' performances are similar. Only the centralized policy produces additional Grid communication transit between the user and the cluster front-ends for maintaining consistency in the cache database. However, this transit only implies signals without data load. Moreover, the centralized policy allows the scheduler service to know where cacheable files have been staged. The scheduler can use this information to decide how to stage tasks more efficiently, giving priority access to resources with specific required files. This is a clear advantage of the centralized policy over the remote one. Nevertheless, the centralized policy can be set only when the scheduler transfer scheme is direct and the cluster front-end and cluster-worker nodes share secondary storage in all of the resources in use. On the contrary, the remote policy is more versatile and can be used in both cases, with or without shared secondary storage, and in environments where a combination of both types is present.

4 Cache System Adaptation over the GridWay Meta-scheduler

Grid scheduling or super scheduling has been defined in the literature as the process of scheduling resources over multiple administrative domains. Currently, a great variety of schedulers have been implemented. Some examples are AppLeS (Su et al. 1999), GridLab (Allen et al. 2003), Condor-G (Frey et al. 2004), and GANESH (Bhatt et al. 2007). Although they have different characteristics, all schedulers must follow a general process. This process includes the following phases: system selection and preparation; task submission, monitoring, migration, and termination (Schopf, 2004). To coexist with any Grid-scheduling system, the cache system management must consider the different phases of the scheduling process. As we described above, the cache system will interact mainly with the task submission and termination phases.

In our study, we used the GridWay meta-scheduler (Huedo et al. 2004). This meta scheduler is a service implemented over the Globus middleware. GridWay manages task execution and brokers resources, allowing unattended, reliable and efficient execution of complex tasks and collection of tasks on Grid heterogeneous environ-

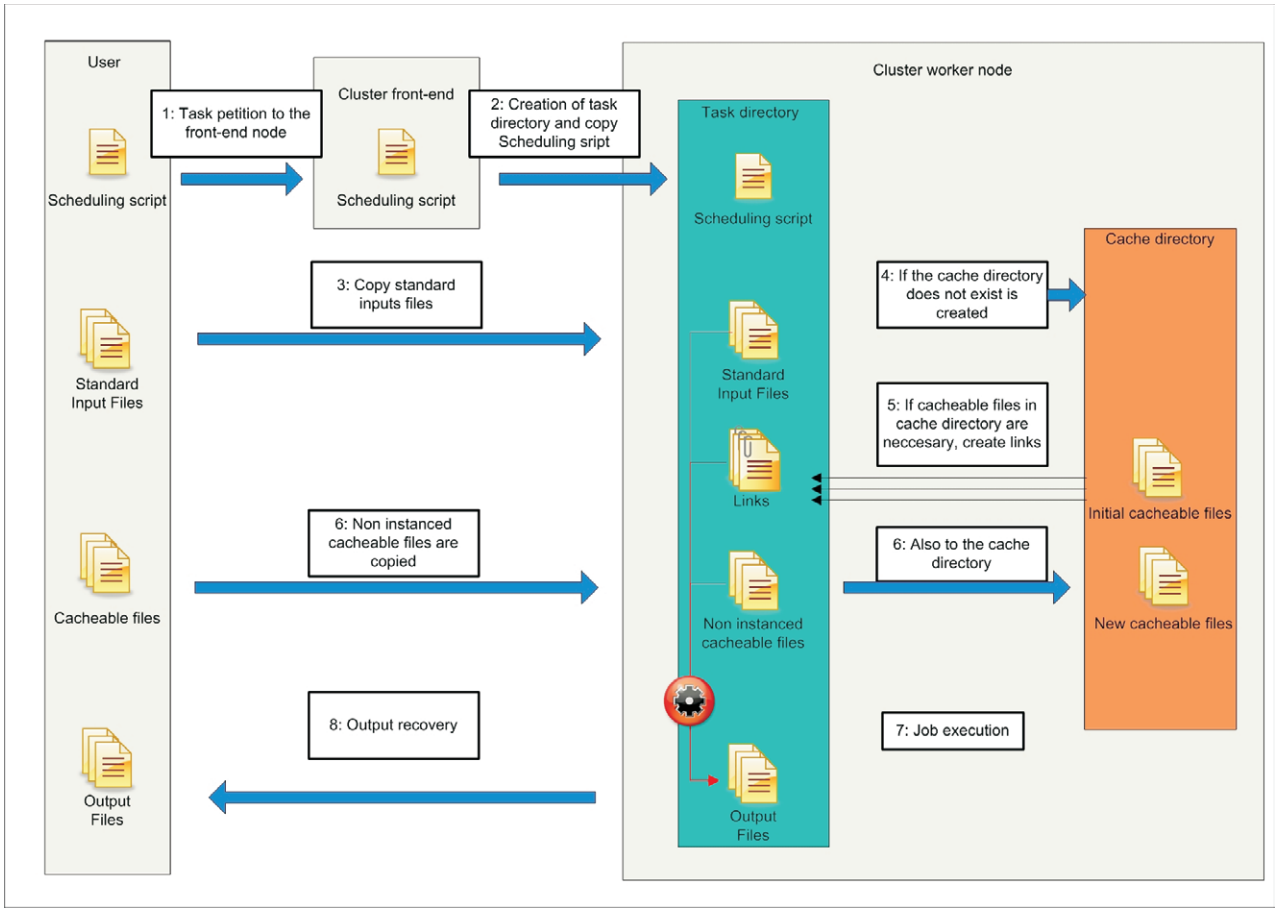

Fig. 5 Scheme to carry out a task with a remote cache policy. The scheduler in the user's resource makes a task petition to the cluster front-end (1). The cluster front-end selects a suitable cluster-worker node, creates a directory, and transfers the scheduling routine (2). Scheduling routine execution begins by copying the non-cacheable files from the user (3). If the cache directory does not exist, then it is created (4), otherwise the cacheable files in the cache directory are linked in the task work directory (5). Cacheable files that cannot be found in the cache directory are copied from the user to the cache directory and linked in the work directory (6). Finally, the task is launched (7) and the output files are transferred to the user (8).

ments. The GridWay daemon (GWD) interacts with different Middleware Access Drivers (MADS) to access Grid services. The GWD is connected with the different MADS through Managers, which provide the following (see Figure 6):

- Request Manager: receives task petitions from the user and introduces them in a task queue. It also allows deleting and holding operations over the tasks.
- Information Manager: recollects information about available resources in the Grid, using middleware services to create a resource queue. It also recollects information about the performance of the resource.
- Dispatch Manager: using the queue information, it allocates tasks from the request queue in resources. It uses resource performance information to make assignations and eventually increase efficiency. It also migrates tasks between different resources.
- Transfer Manager: allows file transmission between resources through middleware services.
- Execution Manager: manages task execution over resources, checking execution time and performance by interacting with GridFTP, grid resource management system (GRAM), and local resource management system (LRMS) services. It also labels tasks as candidates to be migrated when they are scheduled in resources for too

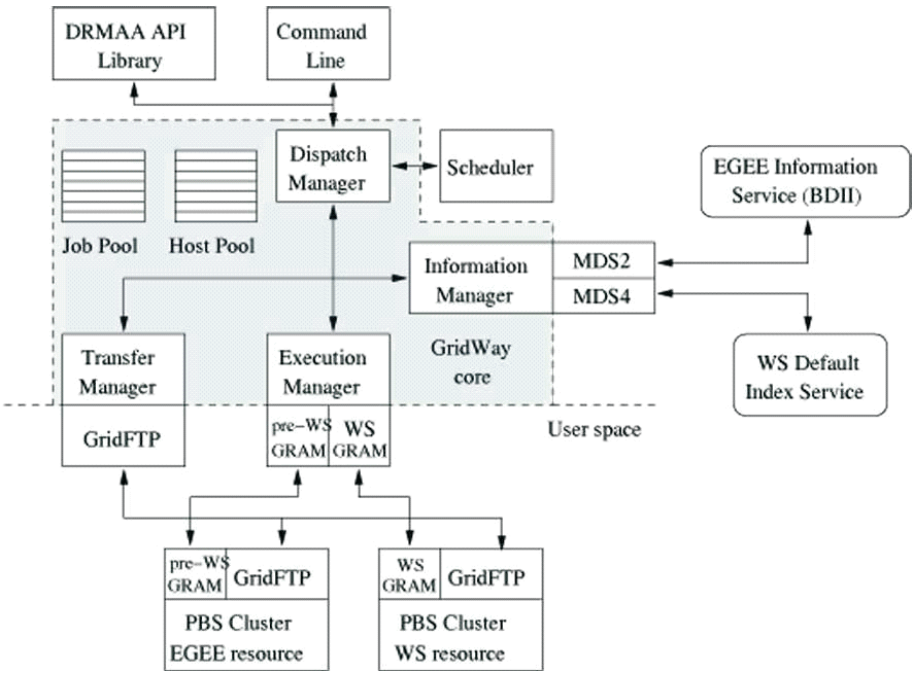

Fig. 6 GridWay architecture and its interaction with Grid services.

long. A Wrapper Routine, which is executed over the resources in a batch environment, provides management and control of the remote executions. This routine does all the initial and final adaptations necessary for the correct execution of a task.

These functionalities allow adapting executions of tasks over changing Grid conditions, providing failure recovery mechanisms and maintaining transparency for users.

Changes for introducing a cache system in this scheduler require modifications to the Request, Transfer, Execution, and Information managers. GridWay provides different transfer schemes depending on the storage nature of the Grid environment. A direct transfer scheme is established when the Grid resources guarantee shared storage. In this case, the Transfer Manager interacts with the GridFTP service to do transfers. When shared storage is not guaranteed, the transfer scheme is reversed. In this case, the transfer operations are launched from the cluster-worker nodes by the Wrapper Routine through the GridFTP service. Depending on the transfer scheme, different cache policies must be implemented, introducing different changes in the GridWay modules:

- Centralized policy: used with the direct transfer scheme. Interaction with the centralized cache database requires modifications in the following:
- Request Manager: the user's interface is modified to allow the user to specify cacheable files. In short, a template file specifies the interaction between the user and the Request Manager. The user fills up different fields in this template describing the task to run, specifying the executable file needed, the input files, and resource requirements. New template fields have been included in order to let the user specify whether the executable file must be kept in the cache and which input files must be treated as cacheable. Letting the user determine the cacheable files allows avoiding useless caching of files that are used in a single task. If a single task that will never be repeated executes, it is of no use to cache the executable file attached to the task, as it will never be needed again. In any case, it is possible to force cache functionality for all files in use. The Request Manager, reading the template file, marks up all the cacheable files appropriately and computes their MD5 identifiers for future use.
- Transfer Manager: every time the Transfer Manager receives a request to transfer a set of files to a specific cluster front-end, it checks whether each files has been marked up as cacheable. While those that have not been are directly transferred, for those so marked, the transfer manager does a database query in order to determine whether the files have been already set in the cache directory of the resource. If the answer is

positive, the file transfer operation is cancelled, completing only when the answer is negative.

- Wrapper Routine: the batch script routine executed by GridWay in the selected cluster-worker node is modified to manage the cache directory. It links both previously cached files needed by the task and new cacheable files transferred by the Transfer Manager.
- Execution Manager: after the establishment of new cacheable files in a cache directory, the local database must be updated with new entries. In order to keep the database consistent, it is necessary to verify the correct completion of the cache insertions. This verification can be obtained only if the task is successfully performed. As the Wrapper Routine is executed remotely in the cluster-worker node, GridWay cannot determine the reason for an incorrect execution of a task. Possible reasons for task failure can be an error in the access of the cache directory or an incorrect transfer operation for one of the cacheable files. In case of task failure, the grid cannot guarantee that these operations have completed correctly since the cache modifications were compromised. For these reasons, the Execution Manager only modifies the database when it detects that tasks have finalized correctly.
- Information Manager: this manager periodically carries out the checking operations needed to maintain database consistency due to the lack of control over storage capabilities in remote resources. The Information Manager triggers these operations in an appropriately short interval in order to guarantee consistency when tasks are launched.

- Remote policy: employed with the reverse transfer scheme. As all of the cache operations rely on the resources, modifications of the scheduler are slight. In short, the Request manager must carry out new functions as described for the centralized policy. Contrary to the centralized policy, all of the cache functionality resides in the Wrapper Routine. Thus, together with the linking operations, now the Wrapper Routine communicates with the user to transfer input files.

GridWay design follows a "loosely coupled" philosophy, meaning that its functionality is kept in a user-level layer, running only in the user's resource, not introducing lower-level functionalities and relying on middleware-level services to operate. This conception of GridWay as a client tool allows an easy integration of the cache system. It is only necessary to modify software for the scheduler daemon that must be only installed on the user's resource. Thus, the introduction of new features over the Grid environment is not necessary, facilitating the use of the modified scheduler.

5 FRODOCK: a Grid-adapted High-throughput Application

The protein–protein docking problem constitutes a major challenge in the area of computational structural biology (Deremble and Lavery, 2005; Bonvin, 2006; Ritchie et al. 2008). The goal of traditional protein–protein docking algorithms is to take the three-dimensional coordinates of two proteins that are known to interact, called the ligand and the receptor, and to derive a model for their complex structure. Such predictions and their integration with experimental data can yield new insights into the basic principles of molecular recognition and the mechanisms of protein association. Protein–protein algorithms generally consist of an initial stage during which the unbound components are combined rigidly. This stage generates large numbers of potential predictions, which are then assessed in a second refinement stage. Here we focus on the first stage, which does a rigid-body orientational sampling of a ligand protein molecule relative to a fixed protein receptor molecule while maximizing a docking scoring function. The 6D sampling space of relative orientations between the ligand and the receptor is huge and hence computationally demanding. To tackle this search efficiently, many current approaches follow a classical Fast Fourier Transform (FFT)-based algorithm described by (Katchalski-Katzir et al. 1992). Although the FFT techniques reduce drastically the time needed to carry out a docking search, this time can still be very large, depending on the size of the receptor and ligand components and on the number of interaction potentials used. A single docking search in a standard computer can take many hours or even several days.

Retaining the use of FFT techniques, our new protein–protein docking application improves efficiency with a spherical harmonics (SHs) methodology, which has been successfully used to fit atomic structures into electron microscopy (EM) density maps (Garzon et al. 2007b), and which has also been adapted to Grid computation (Garzon et al. 2007a). The details of this application, called FRODOCK (Fast ROtational DOCKing), can be found elsewhere (Garzon et al. 2009). Briefly, FRODOCK accelerates the rotational part of the search with an effective expansion in the SHs of the grid potentials. These potentials are represented by concentric layers of SHs functions. To complete the full exploration, FRODOCK samples the translational space around the receptor uniformly with a fixed step size. FRODOCK uses three potentials in function scoring: the Van der Waals potential, electrostatic potential, and desolvation potential (Chen and Weng 2003).

Although this method reduces drastically the time needed for the docking search, the use of three different potentials still implies a hard-computing demand. More-

over, uniformly translational sampling can eventually explore more than 10^5 points, depending on the geometry and size of the ligand and receptor structures. Since the rotational exploration in each translational point can be done independently, FRODOCK can be easily adapted to parallel execution. Parallel adaptation follows these steps:

- A preliminary phase pre-computes data of both proteins. These data are related to the different potentials. This phase can run locally on the user's resource.
- In the parallel phase different tasks are launched through the Grid system to explore different sets of positions of the ligand with respect to the receptor. All of the input files are common to all of the tasks except the set of translational search positions, which are specific for each task. The number of tasks launched will define the granularity of the parallel phase. If few tasks are launched, each task will do the rotational search for many translational positions, with a coarse granularity. If many tasks are launched, each task will search for a few translational positions, making the granularity more refined.
- A final phase combines results for all of the tasks over the user's resource, selecting the solutions that provide a higher scoring function.

This parallel strategy can reduce large docking searches to only a few minutes. Considering that classical FFT approaches take hours or days to do a docking computation, this is a clear improvement.

Compared with other alternatives, such as cluster environments or multi-processor computers, a Grid is a suitable environment in which to run this parallel docking application, as it provides access to large amounts of computational resources typically not available in a biology laboratory. Similar applications already use distributed resources to accomplish their tasks (Chang et al. 2007). Finally, all of the parallel FRODOCK tasks require input files, including pre-computed three-dimensional grid potential maps, ligand SHs representations, and a list of ligand translational positions. All of these input files, except for the list of translational positions, are common in all of the tasks. The application behaves optimally if these files are cached efficiently. For this reason, FRODOCK is a perfect candidate for studying the performance of the cache system.

6 Performance Analysis of the Cache System

6.1 Experimental Conditions

To check the efficiency of the cache system adapted to the GridWay meta scheduler, the FRODOCK application was launched with and without cache system support. We have chosen the largest example (hence, the most computationally demanding test) from a standard protein–protein benchmark (Mintseris et al. 2005). In this example, called the 1N2C test, the unbound structures of the Nitrogenase Mo-Fe and Nitrogenase Fe proteins are used as input to test whether FRODOCK can correctly predict their known complex structure. The structures, that is, the three-dimensional atomic coordinates, have been retrieved from the Protein Data Bank (http://www.rcsb.org/) with entry accession identifiers 3MIN and 2NIP for the unbound components and 1N2C for the complex structure. The last was only used to cross validate the predictions obtained. As an illustrative example, an acceptable complex structure obtained with FRODOCK is shown in Figure 7. In a standard computer, the resolution of this case by FRODOCK takes around 16–17 hours.

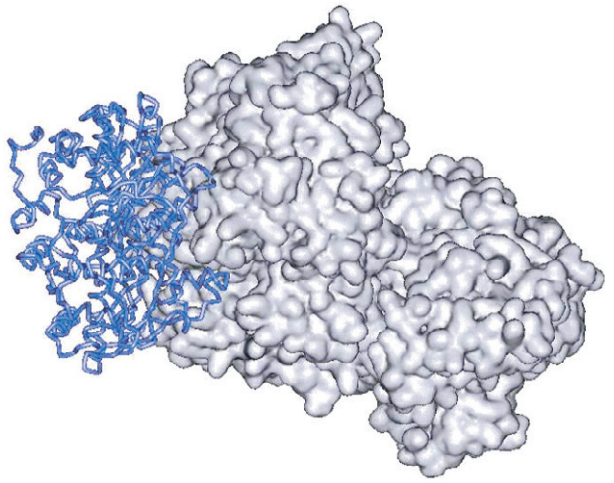

Fig. 7 The best solution found with FRODOCK in illustrative test case 1N2C. The surface map represents the receptor. The tube structure corresponds to the first good ligand position found. This prediction has a root mean-squared deviation (RMSD) of 8 Å with respect to the correct position, which is close enough for an acceptable solution. This prediction is found among the 100 first solutions returned by FRODOCK.

To run FRODOCK in the Grid, the following input files are required:

- Executable application: all of the tasks run the same executable. This is a cacheable file (3 MB).
- Precalculated Potential grids: information about the receptor is summarized in four files containing the Van der Waals potential (11.5 MB), the electrostatic poten-

Table 1
Characteristics of EGEE and Local Grid Resources.

Hosts	Domain	SO	Arch.	Mhz	Nodes	DRMS	Location	Shared storage
ce2	.egee.cesga.es	ScientificSLBer	i686	3000	112	lcgsge	Spain	No
Ce	.gina.sara.nl	ScientificSLBer	i686	2670	792	pbs	The Netherlands	Yes
clrlcgce03	.in2p3.fr	ScientificSLBer	i686	3200	240	lcgpbs	France	No
cox01	.grid.metu.edu.tr	ScientificSLBer	i686	1600	248	lcgpbs	Turkey	No
iut03auvergridce01	.univ.bpclermont.fr	ScientificSLBer	i686	3600	144	lcgpbs	France	No
paugrid1	.pamukkale	ScientificSLBer	i686	1600	44	lcgpbs	Polony	No
polgrid1	.in2p3.fr	ScientificSLBer	i686	2200	370	lcgpbs	France	No
trekker	.nikhef.nl	CentOSFinal	i686	2000	1150	pbs	The Netherlands	Yes
lcg38	.sinp.msu.ru	ScientificSLSL	i686	2700	104	lcgpbs	Rumany	No
aquila	local	Linux	i686	1995	4	pbs	Local	Yes

tial (11.5 MB), the desolvation potential (7.4 MB), and the atom accessible surface area (5.4 MB). All tasks need these files.

- Spherical harmonic representation of the ligand: the ligand SHs representation is precomputed in two files. Both are cacheable as they are needed by all the tasks (1.2 MB).
- Translational search space: each task receives a collection of translational positions where the ligand must be moved. These collections of positions are different for each task and are not cacheable. The size of these files varies depending on the number of tasks into which the application is split. As the number of total tasks to perform (*NT*) increases, the number of positions assigned to each task decreases, and consequently the files become smaller. For example, if 214,188 is the number of positions to explore in the 1N2C test, then the number of points per task ranges from 2,142 when *NT* = 100 to 612 when *NT* = 350. Since the task execution time decreases when *NT* increases, the *NT* variation allows control of the application's granularity.

In summary, note that without a cache system, each task execution requires transferring eight input files to the resources, while with it, only the transfer of one file (collection of points) will be necessary in the most favorable case.

The Grid infrastructure used for the experiments corresponds to the BIOMED virtual organization of the EGEE project (http://www.eu-egee.org). As a testbed, we selected nine sites or resources distributed through different European countries. To prevent resource saturation, the number of tasks concurrently executed in each resource is limited to 10. Thus, no more than 90 tasks can be executed concurrently. The scheduler was configured with default values to concurrently launch 15 tasks in periods of 30 seconds. At any rate, if the launching ratio is kept fast enough, then modifications of these parameters slightly affect the performance obtained by the cache system. Finally, since some of the resources in the EGEE infrastructure have no shared secondary storage, GridWay interacts with such an infrastructure using a reverse transfer scheme. In this case, it is mandatory to establish the remote policy in the cache system. In order to provide resources to test the centralized policy, we have also used a local resource. The EGEE and local resources are detailed in Table 1.

6.2 Cache Performance Analysis

The main advantage of our cache system is that it reduces transfer times during parallel task execution. To analyze the effect of the cache on transfer time, we have carried out several experiments over single resources. To this end, we ran the docking application test over a single resource, varying its granularity by splitting the search into a different number of parallel tasks (*NT* = 100, 200, and 300). Granularity changes affect both the task transfer time and the execution time and hence affect their relative weight over the overall execution time. Since we must also analyze resource characteristics and cache policy, we repeated the experiments in three different scenarios. We ran the experiments over a resource with shared storage

(*ce*) and without shared storage (*paugrid1*). As mentioned before, GridWay establishes a reverse transfer scheme with the EGEE resources, avoiding the centralized cache. To test this policy, we have also repeated the experiments over a local resource (*aquila*).

To characterize the cache efficiency, we calculate the total time spent transferring input files with no cache support by:

$$T_{\text{nocache}} = nt(t_{nc} + t_c), \tag{1}$$

where nt is the number of tasks performed ($nt = NT$ when only one resource is available), t_c is the transfer time of the cacheable input data (seven files for the FRODOCK application) and t_{nc} is the transfer time for the non-cacheable input data (one file). When cache support is available, the transfer time is now calculated by:

$$T_{\text{cache}} = nt \cdot t_{nc} + nf \cdot t_c, \tag{2}$$

where nf is the number of tasks that failed to find their required cacheable files in the cache directory. Taking into account that in the best case the cacheable files will be transferred only once, the lower bound on transfer time can be defined as:

$$T_{\text{min}} = nt \cdot t_{nc} + t_c. \tag{3}$$

Using these equations, the transfer efficiency can be defined as the cache reduction rate of the transfer time:

$$E_{\text{cache}} = 1 - \frac{nt \cdot t_{nc} + nf \cdot t_c}{nt(t_{nc} + t_c)}, \tag{4}$$

$$E_{\text{max}} = 1 - \frac{nt \cdot t_{nc} + t_c}{nt(t_{nc} + t_c)}. \tag{5}$$

The transfer efficiency ranges from 0 when there is no reduction to values close to 1 when a high reduction is obtained. E_{max} characterizes the maximum efficiency that can be obtained.

To calculate the cache transfer efficiency, in the experiments with cache available, t_{nc} is taken from the average time needed by all of the tasks and the t_c value is the sum of the average times to transfer all the cacheable files. The addition of both values corresponds to the total transfer time per task when no file is found in the cache. Instead of using the total number of missed cacheable files, we estimate a more realistic nf value to provide a size-weighted measure of the missed data. Because of this, the miss of a large file will increase nf more than the miss of a small one. In the case of experiments with cache, we obtain the total transfer time per task directly from the average time for transferring all of the tasks. The final parameters obtained in the three scenarios are listed in Table 2.

Figure 8 shows the high efficiencies obtained in all of the resources. In the case of *ce,* the maximum expected efficiency is attained for all of the *NT* values. This maximum efficiency can be explained by the existence of a shared secondary storage. This allows access to the cacheable data after a single transfer operation to all of the cluster-worker nodes. However, the *nf* values for *ce* are slightly larger than 1, indicating that few files are transferred at least twice. The concurrent launching of tasks can explain this observation. As tasks are launched in groups of 15, some tasks will start the transfer operation at the same time for some files, increasing the *nf* val-

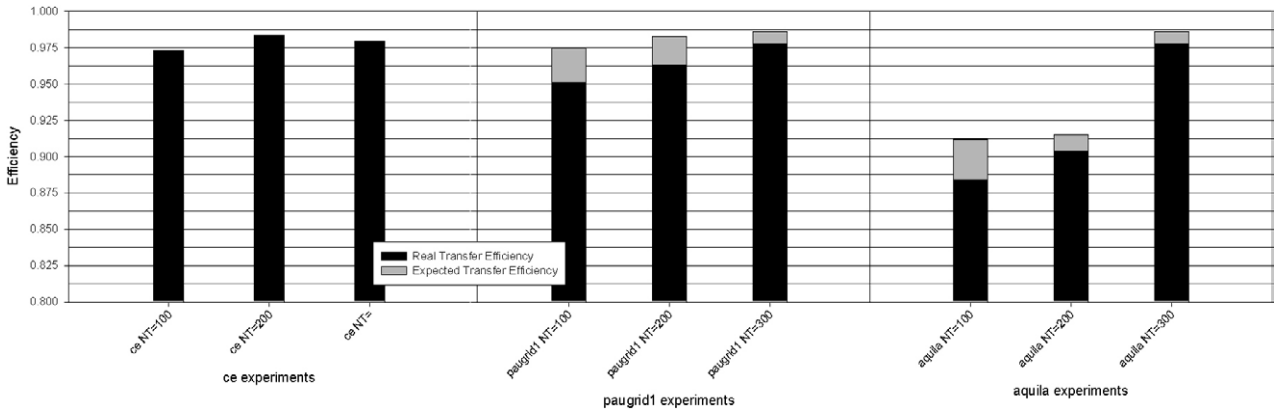

Fig. 8 Transfer efficiency obtained with the cache over different resources. Experiments over *ce* and *paugrid1* use remote cache policy; centralized policy is used over *aquila*.

Table 2
Experimental Data Obtained for the 1N2C Docking Application in Different Resources and with Different *NT*. Resources Indicates the Grid Resource Used in the Experiment; *NT* the Number of Parallelized Tasks; Cache Indicates whether Cache Support is Available; T_c And T_{nc} are the Average Transfer Times to Transfer Cacheable Data and Non-cacheable Data (Only for Experiments with Cache); Ttransfer is the Average Transfer Time Necessary to Copy all of the Input Data for One Task, This Value is Computed by Adding T_c And T_{nc} for Experiments with the Cache; Total Transfer Time is the Total Time to Transfer Operations for all of the Tasks; *nf* is the Average Size-weighted Number of Files Missed in the Cache (only for Experiments with Cache); Total Exec. Time is the Total Time to Execute all of the Tasks.

Resources	*NT*	Cache	T_c	T_{nc}	Ttransfer	Total Transfer Time	*nf*	Total Exec. Time
Ce	100	Yes	17.30	0.30	17.6	48	**1.04**	44733
Ce	100	No	–	–	21.9	2196	–	45181
paugrid1	100	Yes	64.40	1.00	65.4	321	**3.43**	71529
paugrid1	100	No	–	–	65.1	6511	–	69212
aquila	100	Yes	29.80	2.56	31.6	376	**3.89**	58565
aquila	100	No	–	–	16.0	1603	–	58681
Ce	200	Yes	17.40	0,20	17.6	58	**1.04**	44611
Ce	200	No	–	–	22.3	4481	–	45171
paugrid1	200	Yes	64.80	0.80	65.6	479	**4.92**	70530
paugrid1	200	No	–	–	73.2	14654	–	70339
aquila	200	Yes	31.50	2.75	34.2	660	**3.49**	58810
aquila	200	No	–	–	16.6	3334	–	58798
Ce	300	Yes	17.20	0.30	17.5	109	**1.10**	45422
Ce	300	No	–	–	21.4	6462	–	45437
paugrid1	300	Yes	64.70	0.70	65.4	435	**3.48**	74024
paugrid1	300	No	–	–	69.7	20889	–	69871
aquila	300	Yes	31.50	3.25	34.7	1093	**3.74**	58885
aquila	300	No	–	–	1.5	4658	–	58928

ues. At any rate, this transfer concurrency does not have a sensible effect on transfer efficiency. On the contrary, the expected efficiency is not reached in *aquila*, where *nf* ranges between 3 and 4. The higher values of *nf* and t_{nc} produce a sharper reduction in the efficiency. The high values of *nf* are a consequence of the centralized policy. In this policy, the database update is delayed until tasks end, enabling more transfer concurrency events. In the case of *paugrid1*, *nf* values also affect the transfer efficiency, preventing the attainment of maximum efficiency. However, the obtained efficiencies are very high (over 0.95) provided that no common secondary storage is available for all of the cluster-worker nodes. We can conclude that although no shared storage exists for all of the nodes, they must be grouped, forming several clusters where storage is shared. In addition, the same nodes must be repeatedly selected for executing many tasks. Together, these two reasons explain how only a few full transfer operations are needed. In any case, the improvement in transfer efficiency resulting from maintaining shared storage in resources with the cache system is evident.

We would expect efficiency to increase when granularity increases (that is, when *NT* is augmented). As the number of tasks increases, the number of translational positions to explore per task decreases, and the corresponding non-cacheable files (sets of positions per task) are smaller. This reduction of non-cacheable data modifies the balance in total transfer time per tasks. The t_{nc} reduction increases the t_c contribution. Consequently, avoiding transfer operations of cacheable information by means of the cache system will reduce total transfer time more and hence increase efficiency. However, due to the

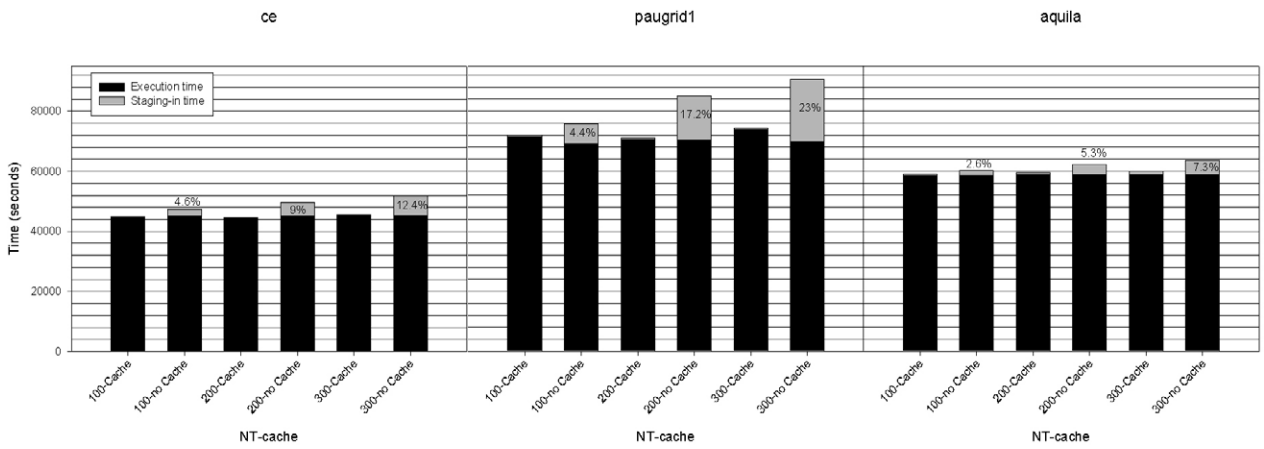

Fig. 9 Operational time (transfer + executions) for all of the tasks carried out in each experiment. Percentage values indicate the transfer rate in the operational time. The tasks' pending time for execution is not considered in operational time.

small size of the set of translations, its variation has a slight effect on the maximum expected efficiency. In the case of *ce*, where t_{nc} is very small, the efficiency variations could be explained by dynamic changes in the transfer Grid conditions. In *paugrid1* and *aquila*, the granularity reduction increases efficiency due to larger t_{nc} values. In these resources, the difference between the obtained and maximum expected efficiency is reduced as *NT* increases. Because *nf* remains almost constant despite the increase in the number of tasks, the cache-miss ratio decreases inversely with the success ratio. In the extreme case, if an infinite number of tasks are performed, the obtained efficiency would equal the maximum expected efficiency. Since the granularity reduction implies more tasks to perform, the total number of transfer operations necessary to realize the full experiment increases when no cache support is available. On the contrary, when the cache system is used, there is no increase. Figure 9 makes this clear, where the operational time for each experiment is computed as the combination of transfer time and execution time for all the tasks. When the cache system is in use, the total transfer time is not significant compared to the execution time. Thus, the cache system provides a reduction of the total operational time that, in *paugrid* where t_c is very high, reaches around 20% for *NT* = 300. This rate will increase as granularity decreases (as *NT* increases). Because of this, when the cache system is available, we can study the granularity for an application's execution in a Grid system without considering transfer overloads when this granularity is finer. Determining efficient application granularity can be difficult due to the necessity of a suitable balance between concurrent execution degree, resource saturation, and transfer time overload. Therefore, removing one of these factors will permit an easier determination of the optimal granularity.

Finally, we observe no significant difference between centralized and local policies' behaviors, as the variation in performance relies mainly on resource characteristics. The only important difference, as mentioned above, relies on a slight increase in *nf* in the centralized policy due to a wider time window in transfer concurrency. However, for large numbers of parallel tasks, this increase will not be perceptible in the final efficiency.

6.3 Application Performance Analysis

To analyze the effect of the cache system on the FRODOCK's overall execution in experimental conditions, we have used all nine EGEE resources specified in Table 1 to solve the 1N2C problem with a remote cache policy.

We have checked different granularity degrees by varying the number of tasks (*NT* = 100, 150, 200, 250, 300, and 350). We repeated each experiment five times to avoid the effects produced by variations in behavior in the Grid resources. We did the experiments with and without cache support. The executions with and without the cache have been interspersed to reduce the effect of temporal variations in the Grid resources. The average total time needed for all of the parallel tasks with and without the cache as a function of *NT* is shown in Figure 10. One can observe that, except for *NT* = 100 where the performances are comparable, the executions with cache support are faster. The comparable times in *NT* = 100 are

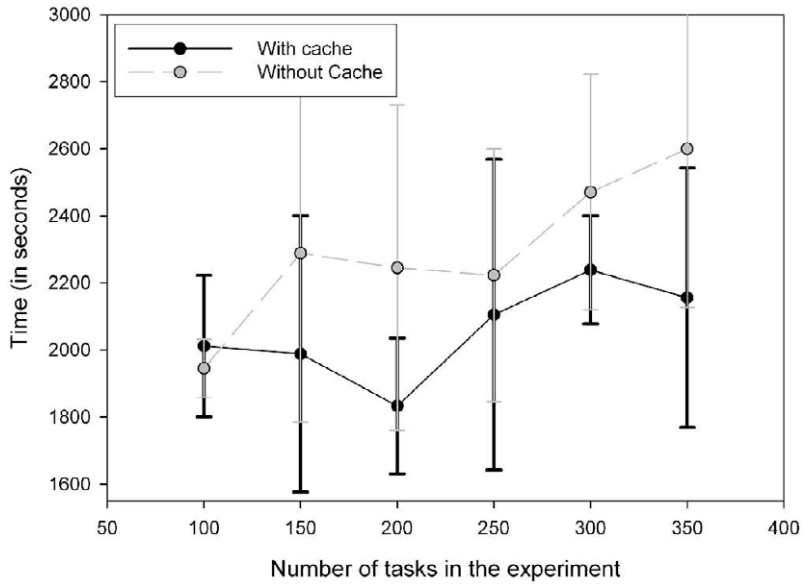

Fig. 10 Average and standard deviation of total times needed in the Grid environment to solve the 1N2C problem. The *X* axis shows the number of tasks (*NT*) used to parallelize the full execution. The dashed line corresponds to non-cache support executions, while the solid line corresponds to cache support executions.

a consequence of the low number of tasks, which does not permit getting enough benefit from the cache facilities. Moreover, with $NT = 100$, the task's computational weight is higher and the transfer time rate over the full task's lifetime is reduced. $NT = 200$ has the best relative cache performance, corresponding to the optimal granularity degree for both the EGEE infrastructure and the cache system used. The values of *NT* over 350 are not expected to reduce execution time as they will heavily saturate the Grid environment. Therefore, we will focus the performance analysis on tests with the optimal value of $NT = 200$.

Figure 11 shows the average experimental performance obtained with $NT = 200$ with and without cache support. The experimental performance is defined as the number of tasks completed per second and gives a measure of the productivity of a Grid system. As it can be seen, with cache support, all 200 tasks run with an experimental performance of 0.103 tasks per second, whereas without cache, performance decreases to 0.087 tasks per second. These values show the better behavior of the cache supported system, as it finishes 15.53% more tasks in the same period of time. Table 3 shows the average experimental gain provided by the cache system for different granularity degrees. As expected, we improve performance for all *NT* except $NT = 100$.

Table 3
Comparative Performance Obtained for Different *NT*.

NT	Cache performance	Non-cache performance	% gain
100	0.0431	0.0439	–2
150	0.0658	0.0586	10.95
200	0.1034	0.0874	15.53
250	0.0933	0.0889	4.626
300	0.1229	0.1134	7.726
350	0.1492	0.1266	15.10

Figure 12 shows average times for staging-in (input files transfer) and execution for all of the tasks correctly executed on different resources, depending on cache support. While the execution does not differ significantly from the times obtained without cache support, the staging-in time is reduced in the resources when the cache system is available. This reduction is provided by cache access to input files for some of the tasks. This fact limits the number of transfer operations needed for the tasks. As the transfer operations are reduced, the average stag-

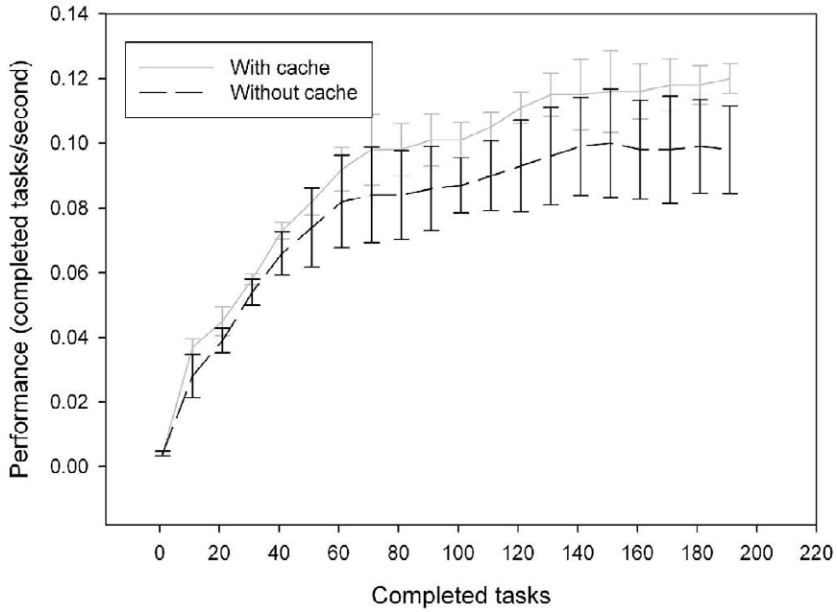

Fig. 11 Average experimental performance (completed tasks per second) with (solid line) and without (dashed line) cache when application is parallelized into 200 tasks (*NT* = 200). Error bars show standard deviation.

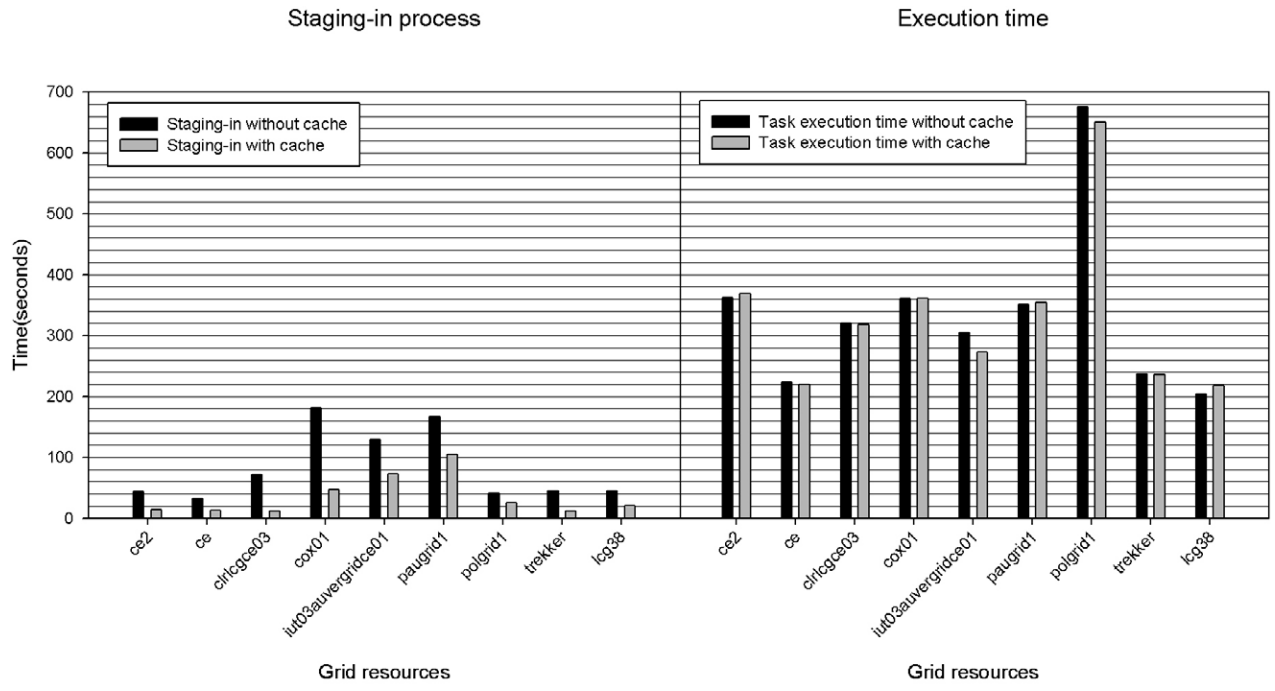

Fig. 12 Average times for staging-in file (left) and executing (right) tasks in each resource with *NT* = 200 and with (grey columns) and without (black columns) the cache system.

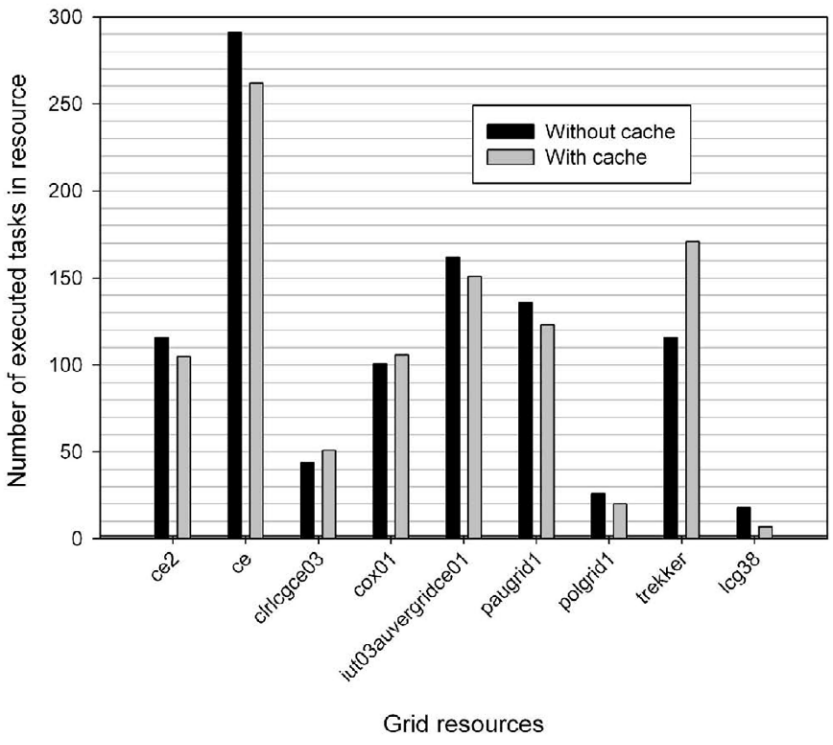

Fig. 13 Total number of executed tasks in each resource for the five experiments with (grey column) and without (black column) the cache system for *NT* = 200.

ing-in process is exposed less to the variation in the Grid transfer latencies derived both from differences between resources' behavior and from changing dynamic conditions. Consequently, the variation in the staging-in process between resources is lower with cache support (32.7 of standard deviation) than without (59.0 of standard deviation).

The homogenization of the staging-in process has a direct effect on overall task performance on different resources. Figure 13 shows that with cache support, the task load (executed tasks in each resource) is slightly more balanced (79.7 of standard deviation) than with no cache support (83.7). Homogeneity increases because fewer productive resources (*trekker*, *clrlcgce03)* do more tasks and there are fewer tasks waiting for execution over the most productive resources (*ce*). As a consequence, the task production variation between low-production resources and high-production resources decreases. The use of the cache system allows better exploitation of the Grid capabilities for high-throughput applications, as the parallel tasks are better distributed over the resources.

Finally, Figure 14 shows both the maximum expected and the obtained transfer efficiency (Equations (5) and (4)) provided by the cache system in all resources. As it can be observed, the maximum expected efficiency varies from the different resources. There are two issues affecting maximum efficiency: Input/Output (I/O) performance and the number of tasks executed by the resource. Resource I/O performance reflects how fast data are transferred to or from the resource. Avoiding transfer operations will increase transfer efficiency more when I/O performance is slow. Similarly, the cache miss rate will decrease when the number of tasks processed by a given resource increases. Although all of the resources except *lcg38* present efficiencies over 50%, the real transfer efficiencies are far from the maximum due to cache misses. The lost efficiency rate depends on the proportion of cache misses produced for all of the tasks executed in the resource. If a resource usually selects cluster-worker nodes that have already processed tasks, there will be few cache misses, and the efficiency will be slightly affected. Resources where shared secondary storage exists (such as *ce* and *trekker*) will have a very low cache miss rate.

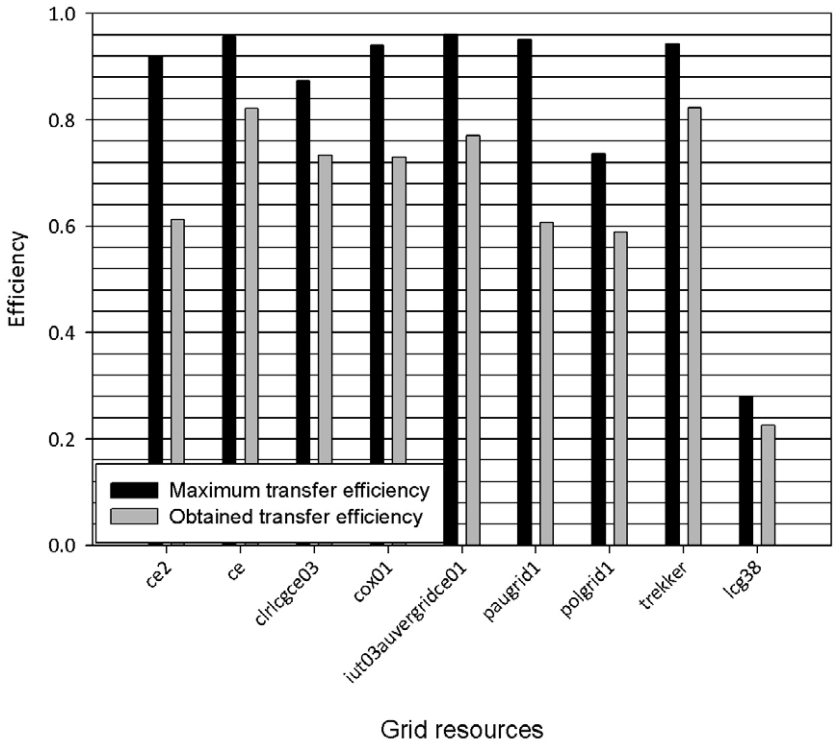

Fig. 14 Maximum expected and obtained transfer efficiencies for each host with *NT* = 200.

7 Conclusions and Future Work

We have presented a new cache system to improve the performance of high-throughput applications that access shared data repeatedly. Unlike other approximations to provide cache facilities in Grid computing, the end-to-end cache system does not need a hard intrusive modification of the Grid environment since it works over the scheduler functionality layer. To implement our cache system, only the scheduler service needs to be modified. Scheduler functionalities can maintain cache directories over the cluster-worker nodes. Cacheable files are stored in these directories when they are transferred from the user and future requests of the files are redirected to the cache directories. The cache system has been designed with two different policies for adapting its functionality to different Grid scenarios and transfer schemes. While the remote policy can be used both with common and independent secondary storage in the nodes of a resource, the centralized policy is only adequate when common secondary storage in the Grid resources is available. Although the first policy is more general, the latter provides information about the cache availability of the files in the environment. The scheduler could use this information to improve its task scheduling.

The cache system has been adapted over the GridWay meta scheduler. In order to test its efficiency, we have used a bioinformatic application called FRODOCK, which carries out protein–protein docking. We have carried out experiments on resources with different characteristics to measure the effect of the cache on transfer time. Common shared storage in resources allows for efficiencies close to the maximum expected. Moreover, the maximum expected efficiency depends on the resource I/O performance. When a resource has a low I/O performance, the expected efficiency will be higher, and the transfer time reduction will be more significant. In addition, increasing the number of tasks running on a resource decreases the probability of cache misses and hence increases the transfer efficiency. Therefore, the cache system eliminates communication overload produced for fine application granularities. Consequently, to consider the granularity degree for a given application, only concurrency and saturation over the Grid resources must be studied. Finally, the performance of both cache policies (centralized and remote) has proven to be very similar;

performance variations are affected much more by the resources' characteristics.

To measure the effect on overall application execution time in real conditions, we ran the application over a heterogeneous Grid environment using the remote policy cache system. The results with the cache system show better performance, with a maximum of 15% reduction in the execution time. The transfer time was also reduced below 50% in the majority of resources. Moreover, the Grid resources behave more homogeneously when the cache functionality is set. These results are very representative of the improvement provided by the cache system. The cache system has also been tested with other applications. This includes a three-dimensional multiresolution fitting bioinformatic application (Garzon et al. 2007b) and a standard Grid Benchmark (Frumkin and Van der Wijngaart, 2002) that has been artificially adapted to require input data. In both cases similar performance has been achieved (10–15%).

The cache system has been designed from a user's perspective and works only with information provided by one single user. Since we have focused our work on high-throughput applications with a high degree of data reusability, we have not considered the case of data shared between different users. In future work, we will study alternatives for data management in the cache, mainly oriented to the data deletion mechanism after the application executions. Finally, the information maintained by the centralized policy can be of great value for the scheduler when assigning tasks to resources. The scheduler can make better assignments by using this dynamic information, improving performance significantly. Thus, future work will focus on task scheduling improvements provided by the centralized policy.

Authors' Biographies

José Ignacio Garzón received his M.E. in Computer Science (2001) from the Universidad de Granada (UGR). He works on his Ph.D. in the Structural Bioinformatics Group at the Centro de Investigaciones Biológicas, Spanish National Research Council (CSIC), in Madrid. His research interests lie mainly in Structural Biology fitting algorithms and Grid Technology.

Eduardo Huedo received his M.E. in Computer Science (1999) and Ph.D. in Computer Architecture (2004) from Universidad Complutense de Madrid (UCM). He has been an Assistant Professor of Computer Architecture and Technology in the Department of Computer Architecture and System Engineering at UCM since 2006. Previously, he was Postdoctoral Researcher in the Advanced Computing Laboratory at Centro de Astrobiología (CSIC-INTA), associated to the NASA Astrobiology Institute. His research interests include Performance Management and Tuning, Parallel and Distributed Computing and Grid Technology.

Rubén S. Montero received his B.S. in Physics (1996), M.S. in Computer Science (1998), and Ph.D. in Computer Architecture (2002) from the Universidad Complutense de Madrid (UCM). He has been an Associate Professor of Computer Architecture and Technology in the Department of Computer Architecture and System Engineering at UCM since 2006. He has held several research appointments at ICASE (NASA Langley Research Center) where he worked on computational fluid dynamics, parallel multigrid algorithms, and Cluster computing. Nowadays, his research interests lie mainly in Grid Technology, in particular in adaptive scheduling, adaptive execution, and distributed algorithms.

Ignacio M. Llorente received his B.S. in Physics (1990), M.S. in Computer Science (1992), and Ph.D. in Computer Architecture (1995) from the Universidad Complutense de Madrid (UCM). He has been the Executive M.B.A. of the Instituto de Empresa since 2003. He is Full Professor of Computer Architecture and Technology in the Department of Computer Architecture and System Engineering at UCM and Senior Scientist at Centro de Astrobiología (CSIC-INTA), associated to the NASA Astrobiology Institute. He has held several appointments since 1997 as a Consultant in High-performance Computing and Applied Mathematics at ICASE (NASA Langley Research Center). His research areas are Information Security, High-performance Computing, and Grid Technology.

Pablo Chacón received his Ph.D. degree in Biochemistry from the Universidad Complutense de Madrid, Spain, in 1999. From 2000 to 2003 he was research associate at The Scripps Research Institute (TSRI) in La Jolla, CA where he developed software for image processing and biomolecular docking. He is currently staff scientist at the Centro de Investigaciones Biológicas, Spanish National Research Council (CSIC), in Madrid. His structural bioinformatics group is interested in the development of efficient algorithms to solve biophysical problems.

References

Allen, G., Goodale, T., Radke, R., Russell, M., Seidel, E., Davis, K., Dolkas, K. N., Doulamis, N. D., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Shalf, J. and Taylor, I. (2003). Enabling applications on the Grid: a Gridlab overview. *Int. J. High Perform.* **17**(4): 449–466.

Baker, M., Buyya, B. and Laforenza, D. (2002). Grids and Grid technologies for wide-area distributed computing. *Softw.: Pract. Experien.* **32**: 1437–1466.

Barish, G. and Obraczka, K. (2000). World Wide Web caching: trends and techniques. *IEEE Commun. Mag.* **38**: 178–184.

Bester, J., Foster, I., Kesselman, C., Tedesco, J. and Tuecke, S. (1999). GASS: a data movement and access service for wide area computing systems. In 6th Workshop on I/O in Parallel and Distributed Systems, Atlanta, Georgia, EEUU.

Bhatt, H. S., Patel, R. M., Kotecha, H. J., Patel, V. H. and Dasgupta, A. (2007). GANESH: Grid application management and enhanced scheduling. *Int. J. High Perform.* **21**(4): 419–428.

Bonvin, A. M. (2006). Flexible protein–protein docking. *Curr Opin Struct Biol.* **16**(2): 194–200.

Bresnahan, J., Link, M., Khanna, G., Imani, Z., Kettimuthu, R. and Foster, I. (2007). Globus GridFTP: what's new in 2007. In Proceedings of the First International Conference on Networks for Grid Applications, Lyon, France.

Chang, M. W., Lindstrom, W., Olson, A. J. and Belew, R. K. (2007). Analysis of HIV wild-type and mutant structures via in silico docking against diverse ligand libraries. *J. Chem. Inf. Model.* **47**(3): 1258–1262.

Chen, R., Li, L. and Weng, Z. (2003). ZDOCK: an initial-stage protein-docking algorithm. *Proteins* **52**(1): 80–87.

Chervenak, A., Schuler, R., Kesselman, C., Korada, S. and Moe, B. (2005). Wide area data replication for scientific collaborations. In Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, Washington, DC, USA IEEE Computer Society.

De Roure, D., Baker, M. and Jennings, N. (2003). The Evolution of the Grid in Grid Computing: Making the Global Infrastructure a Reality. New York: John Wiley & Sons, pp. 65–100.

Deremble, C. and Lavery, R. (2005). Macromolecular recognition. *Curr Opin Struct Biol.* **15**(2): 171–175.

Foster, I. and Kesselman, C. (1997). Globus: a metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl.* **11**: 115–128.

Foster, I. and Kesselman, C. (1999). The Grid: Blueprint for a New Computing Infrastructure. San Francisco: Morgan-Kaufman.

Frey, J., Tannenbaum, T., Livny, M., Foster, I. and Tuecke, S. (2004). Condor-G: a computation management agent for multi-institutional Grids. *Clust. Comput.* **5**(3): 237–246.

Frumkin, M. and Van der Wijngaart, R. F. (2002). NAS Grid benchmarks: a tool for Grid space exploration. *Clust. Comput.* **5**(3): 247–255.

Garzon, J. I., Huedo, E., Montero, R. S., Llorente, I. M. and Chacon, P. (2007a). Adaptation of a multi-resolution docking bioinformatics application to the Grid. *J. Softw.* **2**(2): 1–10.

Garzon, J. I., Kovacs, J., Abagyan, R. and Chacon, P. (2007b). ADP_EM: fast exhaustive multi-resolution docking for high-throughput coverage. *Bioinformatics* **23**(4): 427–433.

Garzon, J. I., Kovacs, J., Abagyan, R. and Chacon, P. (2009). FRODOCK: a new approach for fast rotational protein–protein docking. *Bioinformatics* **25**: 2544–2551.

Huedo, E., Montero, R. S. and Llorente, I. M. (2004). A framework for adaptative executions in grids. *Softw.: Pract. and Experien.* **34**: 631–651.

Katchalski-Katzir, E., Shariv, I., Eisenstein, M., Friesem, A. A., Aflalo, C. and Vakser, I. A. (1992). Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. *Proc. Natl. Acad. Sci. U S A.* **89**(6): 2195–2199.

Liu, F. and Song, F. (2008). An optimization of resource replication access in Grid cache. *Advan. Grid Pervas. Comput.* **5036**: 83–92.

Madduri, R. K., Hood, C. S. and Allcock, W. E. (2002). Reliable file transfer in Grid environments. In Proceedings of the 27th Annual IEEE Conference on Local Computer Networks, Zürich, Switzerland, pp. 737–738.

Mintseris, J., Wiehe, K., Pierce, B., Anderson, R., Chen, R., Janin, J. and Weng, Z. (2005). Protein–protein docking benchmark 2.0: an update. *Proteins* **60**(2): 214–216.

Otoo, E. and Shoshani, A. (2003). Accurate modeling of cache replacement policies in a data Grid. In the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), Washington, DC, USA, IEEE Computer Society, p. 10.

Ritchie, D. W., Kozakov, D. and Vajda, S. (2008). Accelerating and focusing protein–protein docking correlations using multi-dimensional rotational FFT generating functions. *Bioinformatics* **24**(17): 1865–1873.

Romberg, M. (2002). The unicore grid infrastructure. *Sci. Program.* **10**(2): 149–157.

Schopf, J. M. (2004). Ten actions when superscheduling. In Grid Resource Management, edited by S. A. W. Nabrzyski. Norwell, MA: Kluwer Academic Publishers.

Shoshani, A., Sim, A. and Gu, J. (2004). Storage resource managers: essential components for the Grid. In Grid Resource Management: State of the Art and Future Trends, edited by J. Nabrzyski, J. M. Schopf and J. Węglarz, Norwell: Kluwer Academic Publishers, pp. 321–340.

Su, A., Berman, F., Wolski, R. and Strout, M. M. (1999). Using Apples to schedule simple SARA on the computational Grid. *Int. J. High Perform.* **13**(3): 253–262.

Surridge, S., Taylor, S., De Roure, D. and In, Z. E. (2005). Experiences with GRIA; industrial applications on a Web services Grid. In Proceedings of the First International Conference on e-Science and Grid Computing (E-SCIENCE '05), Washington, DC, USA, pp. 98–105.

Tierney, B., Johnston, W. and Lee, J. (2000). A cache-based data intensive distributed computing architecture for "Grid" applications. In CERN School of Computing, Marathon, Greece.

Yonny, C., Pierson, J. M. and Brunie, L. (2007). Management of a cooperative cache in Grids with Grid cache services. *Concurr. Comput.: Pract. Experien.* **19**(16): 2141–2155.