Future Generation Computer Systems 27 (2011) 182–194

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Efficient data consolidation in grid networks and performance analysis

P. Kokkinos *, K. Christodoulopoulos, E. Varvarigos

Department of Computer Engineering and Informatics, University of Patras, Greece
Research Academic Computer Technology Institute, Patras, Greece

ARTICLE INFO

Article history:
Received 4 December 2009
Received in revised form
27 July 2010
Accepted 8 August 2010
Available online 27 August 2010

Keywords:
Grids
Task scheduling
Data migration
Routing

ABSTRACT

We examine a task scheduling and data migration problem for grid networks, which we refer to as the Data Consolidation (DC) problem. DC arises when a task concurrently requests multiple pieces of data, possibly scattered throughout the grid network, that have to be present at a selected site before the task's execution starts. In such a case, the scheduler and the data manager must select (i) the data replicas to be used, (ii) the site where these data will be gathered for the task to be executed, and (iii) the routing paths to be followed; this is assuming that the selected datasets are transferred concurrently to the execution site. The algorithms or policies for selecting the data replicas, the data consolidating site and the corresponding paths comprise a Data Consolidation scheme. We propose and experimentally evaluate several DC schemes of polynomial number of operations that attempt to estimate the cost of the concurrent data transfers, to avoid congestion that may appear due to these transfers and to provide fault tolerance. Our simulation results strengthen our belief that DC is an important problem that needs to be addressed in the design of data grids, and can lead, if performed efficiently, to significant benefits in terms of task delay, network load and other performance parameters.

1. Introduction

Grids consist of geographically distributed and heterogeneous computational and storage resources that may belong to different administrative domains, but are shared among users by establishing a global resource management architecture. A variety of applications can benefit from grid computing; some applications, called CPU-intensive applications, involve computationally intensive problems on small pieces of data (or datasets), while others, called data-intensive applications, perform computations on large-sized datasets, stored at geographically distributed resources. In the latter case, the grid is usually referred to as a data grid. Examples of data-intensive applications appear in life sciences, high-energy physics and astrophysics, where large amounts of data are created, processed and stored in a distributed manner. It is evident that, in these applications, communication delays play a key role and considerably affect the completion times of the tasks. In such an environment, the collaboration of task scheduling and data management is essential for boosting grid performance.

Generally, the scheduling of tasks to the available resources is a difficult problem. This is because grids are quite dynamic, with resource availability and load varying rapidly with time, while at the same time tasks have diverse characteristics and requirements. Moreover, data management involves various data handling issues related to the sites at which the datasets are stored, the way datasets are replicated, and the time instances when these are replaced–reshuffled or moved across the network.

In this work, we examine a task scheduling and data migration problem, called *Data Consolidation* (Fig. 1). Data Consolidation (DC) applies to data-intensive applications that need more than one piece of data to be transferred to an appropriate site, before the application can start its execution at that site. The DC problem consists of three interrelated subproblems: (i) the selection of the replica of each dataset (i.e., the data repository site from which to obtain the dataset) that will be used by the task, (ii) the selection of the site where these pieces of data will be gathered and the task will be executed, and (iii) the selection of the paths the datasets will follow in order to be concurrently transferred to the data consolidating site. Furthermore, the delay required for transferring the output data files to the originating user (or to a site specified by the user) should also be accounted for. In most cases, the task's required datasets will not be located into a single site, and a Data Consolidation operation is therefore required. Generally, a number of algorithms or policies can be used for solving these three subproblems either separately or jointly. Moreover, the order in which these subproblems are handled may be different from the order in which they are presented here, while the performance optimization criteria used may also vary. The algorithms or policies for solving these subproblems comprise a DC scheme.

The DC problem is considered as a continuous-time problem (as opposed to a one-time problem), where the decisions ((i)–(iii))

* Corresponding author at: Department of Computer Engineering and Informatics, University of Patras, Greece.
E-mail address: kokkinop@ceid.upatras.gr (P. Kokkinos).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.08.005

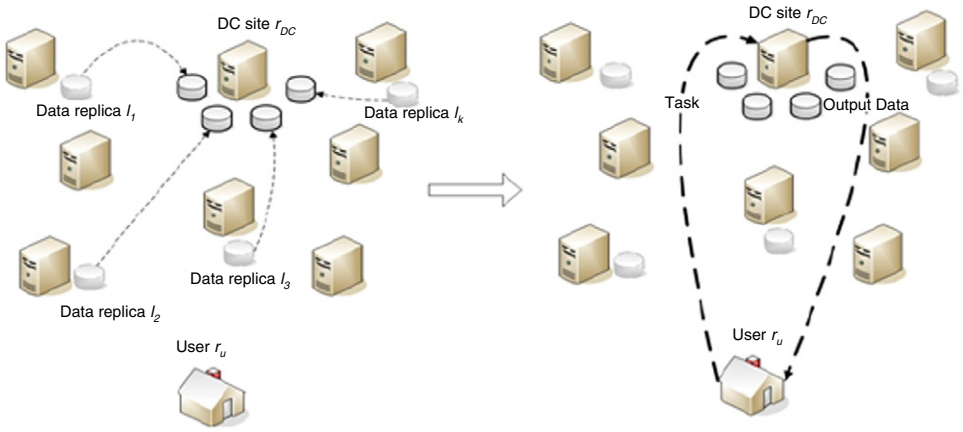


Fig. 1. A Data Consolidation scenario. Initially, the datasets a task requires are transferred to a single site r_{DC}. After all data transfers have been completed, the task itself is also transferred to the site, where it is executed. Finally, the task's output data are transferred back to the task's originating user r_u.

above taken for one task affect the decisions taken for future tasks and are affected by the decisions taken earlier for previous tasks. For example, after a task has been scheduled and DC is performed for it, replicas will move at different locations, and some computation and communication resources may still be occupied by it at the time the next task is considered. Therefore, the performance criteria that we use for evaluating the schemes are not the DC completion time of a single task, but the average task delay (at steady state, over an infinite time horizon), the average network load induced by a task, and the task success ratio. In Section 4.3, we evaluate the (polynomial in all cases) number of operations required by the proposed DC schemes.

We propose and experimentally evaluate a number of DC schemes. Some consider only the computational or only the communication requirements of the tasks, while others consider both kinds of requirement. Our proposed schemes calculate, for each candidate site, the maximum (worst) cost of accessing any of the best replicas of the task's requested datasets, selecting the site that gives the minimum worst transfer cost value. We also propose DC schemes, which are based on Minimum Spanning Trees (MSTs) that concurrently route the datasets so as to reduce the congestion that may appear in the future, due to these transfers. We also investigate the required number of operations of the proposed schemes and show that it is polynomial in the number of sites comprising the grid network. A number of interesting issues are also investigated, including ways the proposed schemes can be extended so as to provide fault tolerance against failures in the storage resources. We evaluate the proposed schemes by performing a large number of simulation experiments for different values of the parameters involved, such as the number of datasets a task requires, the number of sites, the task computation and communication requirements, and the task arrival rate. In this way we investigate the effects of the grid application and network characteristics on the DC problem, and show that important performance benefits can be obtained if the DC operation is performed efficiently. Finally, we compare different approaches for estimating the access cost of a number of datasets, when these are requested concurrently.

In our work, we study the DC problem mainly from a theoretical viewpoint, making a number of simplifying assumptions. We assume that a central scheduler is responsible for task scheduling and data management, and that it has complete knowledge of the static (such as the computation and storage capacity) and the dynamic (such as the number of running and queued tasks and the location of each data replica) characteristics of the computation resources and of the network connecting them. We also do not take into account the communication delays of transferring messages between the user and the scheduler and between the scheduler and the resources, since we assume that these are negligible compared to the total execution time of the tasks, at least for the data-intensive scenarios that are considered in this work. We believe that these assumptions are necessary so as to make the DC problem manageable, considering the large number of parameters it depends on. Moreover, we argue that these assumptions do not depart from reality, and they are similar to those used by other works in the literature. Today, grid networks and the corresponding middleware (e.g., gLite [1]) follow a two-level scheduling approach. A central scheduler decides the pool of resources (e.g., belonging to a particular administrative authority) where a task will be assigned, while a local scheduler decides upon the exact resource where the task will be executed. In addition, there are mechanisms (for example in the gLite middleware, these are the File & Replica Catalog Service, the Workload Management System, the Monitoring and Discovery Service, the Information Super Market, and others) that provide to the scheduler information regarding the location of data replicas and the values of several static and dynamic parameters such as the number of tasks in each resource. An important issue with the dynamic information, which changes over time, is its validity. That is, the information that the scheduler receives from such mechanisms may not be accurate since it changes over time, while the scheduler is updated at periodic intervals. In a previous work of ours [2] we have investigated such issues for the case of the gLite middleware.

Although these assumptions restrict the direct applicability of the proposed schemes, there are examples of real-world applications, related to the problem that we examine, that can benefit from our investigation. Montage [3] is a toolkit for constructing custom, science-grade mosaics by composing multiple astronomical images. In most cases, the astronomical datasets are massive, and they are stored in distributed archives that are remote with respect to the available computational resources. KoDaVis [4] deals with the visualization of large data sets from atmosphere research. In KoDaVis, instead of storing data at one central site, where the simulation is performed, data are distributed at various sites. Fragments of the dataset are sent to the visualization system. This way, the processing of the data can be achieved without storing the whole datasets locally. It is true, though, that an application/task may not need all the datasets at the time it starts executing, but, as we will show, it is usually beneficial for the application to perform the dataset transfers concurrently and before the task's execution. If a task requires only a small part of each dataset, then the DC schemes could be coupled with remote I/O methods, so as to reduce the amount of data that is transferred to the selected site. In addition, in MapReduce [6], the data communication between the Map and Reduce phases could very well benefit from optimized

communication paths, or from optimized Reduce task placement as performed by our proposed DC schemes.

The remainder of the paper is organized as follows. In Section 2, we report on previous work. In Section 3, we formulate the Data Consolidation problem. In Section 4, we analyze the DC problem, propose a number of DC schemes, and examine the number of operations required for their execution. In Section 5, we present the simulation environment and the performance results obtained for the proposed schemes. Finally, conclusions are presented in Section 6.

2. Previous work

The Data Consolidation (DC) problem involves task scheduling, data management and routing issues. Usually these issues are handled separately in the corresponding research papers. There are several studies that propose algorithms for assigning tasks to the available resources in a grid network. Data replication mechanisms are used so as to reduce the time needed for accessing particular datasets. Similarly, various policies have also been proposed for selecting the data's replicas for the execution of a task. Also, a large number of research papers address routing issues in grid networks.

A survey of the existing grid scheduling policies is presented in [7]. The scheduling of tasks to resources has been considered in [8–10], among others works, where several centralized, hierarchical or distributed scheduling schemes are presented. Other works incorporate economic models in grid scheduling, as in [11], which proposes scheduling algorithms that take into account deadline and budget constraints. Fair scheduling in grid networks has also been addressed in [12–14].

A usual data management operation in grids is data migration, that is, the movement of data between resources. Mechanisms for accessing the whole or part of the data remotely (remote I/O) also exist. The effects of data migration in grids have been considered in [15]. The most common data migration technique is data replication [16–18], which is the process of distributing replicas of data across sites. When different sites hold replicas of a particular dataset, significant benefits can be realized by selecting the best replica among them, that is, the one that optimizes a desired performance criterion such as access latency, cost, and security [19,20]. Furthermore, the problem of parallel downloading different parts of a dataset from various replica holding resources, as a means to decrease the download time of that dataset, has been investigated for peer-to-peer networks and the Internet [21,22], and also for grid networks [23]. A number of works consider both task scheduling and data replication issues. The authors in [24] suggest that it is better to perform data replication and task scheduling separately, instead of trying to jointly optimize these two operations. In [25], the authors propose a data management service that proactively replicates the datasets at selected sites, while an intelligent Tabu-search scheduler dispatches tasks to resources so as to optimize the execution time and system utilization metrics. The authors in [26–28] present the OptorSim simulator and jointly consider task scheduling and data replication for the case where a task requests a number of datasets sequentially. In this case, data replication of a specific dataset to a selected site is performed when a task requires this dataset for its execution. Also, the authors in [29] propose the Integrated Replication and Scheduling Strategy (IRS) scheduler, which combines scheduling and replication strategies.

The effective use of the communication/network resources is an important consideration, especially in data grid networks. Network resource provisioning and management protocols are developed in [30] as a way to provide end-to-end Quality of Service (QoS) in grids. However, routing is not usually taken into account in the task scheduling and data management related works, which rely mainly on the routing capabilities of the underlying network (or simulator). In [31], communication resources are explicitly taken into account in a bandwidth reservation system within the GARA framework [32]. In [33], a multicost algorithm for the joint time scheduling of the communication and computation resources to be used by a task is proposed. The algorithm selects the computation resource to execute the task, determines the path to be used for routing the single input data, and finds the starting times for the data transmission and the task execution, performing advance reservations.

The Data Consolidation (DC) problem addressed in the present work arises when a task requires, before the start of its execution, multiple datasets stored at different sites. In this case, we believe that it is beneficial for the application to organize the transfers of the datasets concurrently so as to decrease the task's total execution time. Even though this seems like a logical scenario, especially for data-intensive applications, most of the related works seem to ignore it, assuming either that each task needs for its execution only one large piece of data [24,33], or that it requires a number of datasets sequentially [16,17,26–28]. Our work does not consider any specific dynamic data replication strategy; instead, we assume that a dynamic data replication strategy is in place that distributes replicas in the grid, while data consolidation is performed when a task actually requests a number of datasets before its execution. In most cases the task's required datasets will not be located into a single site, and their consolidation to the selected site will be required before task execution. Furthermore, most of the related works do not explicitly examine important issues like the joint replica selection (and estimation of the cost of the transfers) and the joint routing of these replicas (so as to avoid congestion delays that each of the corresponding transfers may cause to each other). Note that the Data Consolidation problem applies also to the case where data are not moved before task execution, but are accessed remotely (remote I/O) when needed. In this case, our Data Consolidation schemes could select the data replicas, the site where the task will be executed, and the paths that will be reserved for transferring the datasets (or part of the datasets) to the site concurrently, when the task actually needs them (remote file I/O). This way, we could also dynamically re-estimate our choices regarding the data replicas selected and the paths established, during the task's execution. The evaluation of our Data Consolidation schemes when combined with remote I/O access is left for future work.

Finally, parts of this work have been presented in [34]. In the present paper, we provide a more complete analysis of the DC problem, significantly extending parts of the original paper. In particular, we introduce new schemes, namely, the TotalCost-Q, TotalCost-MST, and MST-Cost DC policies, and perform a more extensive study of the performance of the new and previously presented algorithms. We also evaluate the number of operations required for the proposed schemes. In addition, we compare the sum versus max operation used for defining site access costs and we consider DC in conjunction with resource resiliency schemes.

3. Problem formulation

We consider a grid network, consisting of a set R of $N = |R|$ sites (resources) that are connected through a wide-area network (WAN). Each site $r \in R$ contains at least one of the following entities: a computational resource that processes submitted tasks, a storage resource where datasets are stored, and a network resource that performs routing operations. There are also a number of simple routers in the network. The path between two sites r_i and r_j has maximum usable capacity equal to $C_{i,j,}$, defined as the minimum of the path's links capacities and propagation delay equal to $d_{i,j}$.

The computation resource of site r_i has total computation capacity P_i, measured in computation units per second (e.g., million instructions per second — MIPS). Each resource also has a local scheduler and a queue. Tasks arriving at the resource are stored in its queue, until they are assigned by the local scheduler to an available CPU. For the sake of being specific, we assume that the local scheduler uses the first-come first-served (FCFS) policy, but other policies can also be used. We should note that the local schedulers, (e.g., Torque scheduler) utilized in the computing elements (CEs) of a gLite [1] powered grid network, use the FCFS policy as their default task queuing policy. At a given time, a number of tasks are in the queue of resource r_i or are being executed in its CPU(s) using a space-sharing policy. The storage resource of site r_i has storage capacity S_i, measured in data units (e.g., bytes). Users located somewhere in the network generate atomic (undivisible and non-preemptable) tasks with varying characteristics.

A task needs for its execution L pieces of data (datasets) I_k of sizes V_{I_k}, $k = 1, 2, \ldots, L$. A dataset I_k has a number of replicas distributed across various storage resources. The total computation workload of the task is equal to W, and the final results produced have size equal to Δ. W and Δ may depend on the number and size of datasets the task requires. The datasets consolidate to a single site, which we will call the Data Consolidation (DC) site r_{DC}. This site may already contain some datasets, so no transferring is needed for them. The total communication delay that dataset I_k experiences consists of the propagation, the transmission, and the queuing delays. The propagation delay of path (r_i, r_{DC}) is denoted by $d_{i,\mathrm{DC}}$ and its usable capacity by $C_{i,\mathrm{DC}}$ (minimum capacity available at all intermediate links). A piece of data I_k transmitted over a path (r_i, r_{DC}) experiences total communication queuing delay $Q_{i,\mathrm{DC}}^{\mathrm{Comm}}$, because of other pieces of data utilizing the links of the path. In general, the type of transport media used (opaque packet switching, transparent networks such as a wavelength routed optical WDM network or OBS), determines whether the queuing delay is counted once at the source (transparent networks) or is accumulated over all intermediate nodes (opaque networks). Finally, before starting execution at the DC site, a task experiences a processing queuing delay $Q_{\mathrm{DC}}^{\mathrm{Proc}}$, due to other tasks utilizing the resource's computational capacity or already queued.

We assume that a central scheduler is responsible for the task scheduling and data management. The scheduler has complete knowledge of the static (computation and storage capacity, etc.) and the dynamic (number of running and queued tasks, data stored, etc.) characteristics of the sites. We do not take into account the communication delay of transferring messages between the user and the scheduler and between the scheduler and the resources, since we assume that they are negligible compared to the total execution time of the task, at least for the data-intensive scenarios that we consider in this study.

A task created by a user at site r_u asks the central scheduler for the site where the task will execute. Upon receiving the user's request, the scheduler examines the computation and data related characteristics of the task, such as its workload, the number, the type, and the size of datasets needed, and the sites that hold the corresponding datasets. The scheduler based on the used Data Consolidation scheme (Section 4.2) selects (i) the sites that hold the replicas of the datasets the task needs, (ii) the site where these datasets will consolidate and the task will be executed, and (iii) the routes over which to transfer these datasets. The decisions concerning (i)–(iii) can be made jointly or separately. Note that the free capacity of the storage resource r_{DC} must be larger than the total size of the datasets that will consolidate:

$$S_{r_{\mathrm{DC}}} \geq \sum_{k=1}^{L} V_{I_k}. \tag{1}$$

The free storage capacity of a resource includes not only the actual free space but also the space occupied by datasets that are not used and can be deleted. If needed, the oldest unused datasets are deleted from the DC site (other policies can also be applied, however these are not the focus of this work). If no site is found that fulfils Eq. (1), the corresponding task fails. Otherwise, if Eq. (1) is fulfilled by at least one site, then the scheduler orders the data holding sites to transfer the datasets to this DC site. The scheduler, also, transfer this task to the DC site (Fig. 1). The task's execution starts only when both the task and all of its needed datasets have arrived at the DC site. After the task finishes its execution, the results return back to the task's originating user.

Finally, we assume that no dynamic replication strategies operate in the network. A dynamic replication strategy, such as the ones presented in [18], permits the dynamic movement of data between the storage resources, independently from the various task requests. For example, a popular dataset can be copied to more than one resources so as to be easily accessible when needed. Such strategies are expected to reduce the data transfers required for a DC operation, reducing at the same time the task's execution delay.

A table with all the notations used, follows.

R	The set of resources
r_i	A resource i
r_u	The resource where the user that created the task is located
N	The number of resources/sites in the grid network
$C_{i,j}$	The maximum usable capacity between two sites r_i and r_j
$d_{i,j}$	The propagation delay between two sites r_i and r_j
P_i	The computation capacity of site r_i
S_i	The storage capacity of site r_i
L	The number of datasets a tasks requires for its execution
I_k	A dataset
V_{I_k}	The size of the dataset I_k
W	The workload of a task
Δ	The size of the output results produced
r_{DC}	The data consolidation (DC) site
$Q_{i,\mathrm{DC}}^{\mathrm{comm}}$	The total communication queuing delay a piece of data experiences, when transmitted over a path (r_i, r_{DC})
$Q_{\mathrm{DC}}^{\mathrm{Proc}}$	The processing queuing delay a task experiences before starting execution at the DC site

4. Data consolidation

In what follows, we present several Data Consolidation (DC) schemes.

4.1. Data consolidation delays

We assume that the scheduler has selected the data holding sites (replicas), $r_k \in R$, for all datasets I_k, $k = 1, \ldots, L$, and the DC site r_{DC}. Note that the DC site may already have some pieces of data, and thus no transferring is required for these pieces (i.e., $r_k = r_{\mathrm{DC}}$ for some k). In general, such a data-intensive task experiences both communication (D_{comm}) and processing (D_{proc}) delays. The communication delay D_{comm} of a task, considering also the delay for transferring the final results from the DC site r_{DC} to the originating user's site r_u, is

$$\begin{aligned} D_{\mathrm{comm}} &= D_{\mathrm{cons}} + D_{\mathrm{output}} \\ &= \max_{k=1,\ldots,L} \left(\frac{V_{I_k}}{C_{k,\mathrm{DC}}} + Q_{k,\mathrm{DC}}^{\mathrm{Comm}} + d_{k,\mathrm{DC}} \right) \end{aligned}$$

$$+\left(\frac{\Delta}{C_{DC,u}}+Q_{DC,u}^{Comm}+d_{DC,u}\right), \quad (2)$$

where D_{cons} is the time needed for the task's data to consolidate to the DC site r_{DC} and D_{output} is the delay of the output data to be transferred to the originating user's site r_u. The computational delay is given by

$$D_{proc}=Q_{DC}^{Proc}+\frac{W}{P_{DC}}. \quad (3)$$

The total delay suffered by a task is

$$D_{DC}=D_{comm}+D_{proc}. \quad (4)$$

Note that $Q_{k,DC}^{Comm}$ and Q_{DC}^{Proc} are difficult to estimate since the former depends on the utilization of the network and the latter depends on the utilization of the computation resource. For this reason, we propose a variety of algorithms, some of which assume that this information is known, while others do not make this assumption, and we compare their performance.

4.2. Proposed schemes

As stated before, the DC problem consists of three subproblems: (i) the selection of the repository sites r_k from which the dataset I_k, $k=1,2,\ldots,L$, will be transferred to the DC site, (ii) the selection of the DC site r_{DC} where the datasets will accumulate and the task will be executed, and (iii) the selection of the paths (r_k, r_{DC}) the datasets will follow. In general, DC schemes can make these decision based on various criteria such as the computation and storage capacity of the resources, their load, the location and the sizes of the datasets, the bandwidth availability and the expected delay, the user and application behaviours, and the price a user is willing to pay for using the storage and computation resources.

In what follows, we propose a number of DC schemes that consider only the data consolidation (*ConsCost*) or only the computation (*ExecCost*) or both kinds (*TotalCost, TotalCost-Q*) of task requirements. Algorithms with similar considerations have also been proposed in [26–28] that use, however, a different model (sequential access). In the proposed algorithms and in the simulation results that follow, we assume that no output data is returned back to the user, and as a result D_{output} is equal to zero. Even though this parameter may be important in some cases, we decided to concentrate our description and our simulation results on the three more important and complex subproblems that comprise the DC problem in data grids, as described above.

(i) *Random–Random (Rand) scheme*: In this scheme the data replicas used by a task and the DC site are randomly chosen. The paths are selected using a simple Dijkstra algorithm. This scheme was employed for comparison purposes.

(ii) *Consolidation-Cost (ConsCost) scheme*: We select the replicas and the Data Consolidation site that minimize the data consolidation time (D_{cons}), assuming that the communication queuing delays ($Q_{k,DC}^{Comm}$) are negligible.

Given a candidate DC site r_j, we select for each dataset I_k the corresponding data holding site r_i ($I_k \in r_i$) that minimizes the transfer time:

$$\min_{r_i\in R, I_k\in r_i}\left(\frac{V_{I_k}}{C_{i,j}}+Q_{i,j}^{Comm}+d_{i,j}\right), \quad (5)$$

where R is the set of all resources and $d_{i,j}$ the propagation delay between site r_i and r_j. Note that in this algorithm we consider the communication queuing delays to be negligible, and thus $Q_{i,j}^{Comm}=0$. The data consolidation time D_{cons} of candidate DC site r_j is equal to the maximum transfer time of any dataset:

$$D_{cons}(r_j)=\max_{k=1,\ldots,L}\left(\min_{r_i\in R, I_k\in r}\left(\frac{V_{I_k}}{C_{i,j}}+Q_{i,j}^{Comm}+d_{i,j}\right)\right). \quad (6)$$

In the ConsCost scheme we select the DC site (r_{DC}) that minimizes the data consolidation time:

$$r_{DC}=\arg\min_{r_j\in R}\left(D_{cons}(r_j)\right). \quad (7)$$

The paths are constructed using the Dijkstra algorithm.

(iii) *Execution-Cost (ExecCost) scheme*: We select the DC site that minimizes the task's execution time:

$$r_{DC}=\arg\min_{r_j\in R}\left(Q_j^{Proc}+\frac{W}{P_j}\right), \quad (8)$$

while the data replicas are randomly chosen. Since our focus is more on the communication overhead of the DC problem combined with the execution times of the tasks, we consider the processing queuing delay Q_j^{Proc} of a resource r_j as negligible and that the task's workload is known a priori. In general, it is possible to estimate this delay based on the tasks already assigned to a resource and on the average delay tasks previously executed on it have experienced. Also, regarding the a priori knowledge of the task's workload, there are algorithms that can be used to provide such estimates [35,36]. On the other hand, if the computation workload of a task is not known a priori, we can simply choose the resource with the largest computation capacity P_j. Finally, in the ExecCost scheme the paths are constructed using the Dijkstra algorithm.

(iv) *Total-Cost (TotalCost) scheme*: We select the replicas and the DC site that minimize the total task delay. This delay includes the time needed for transferring the datasets to the DC site and the task's execution time. This scheme is the combination of the two above schemes. The paths are constructed using the Dijkstra algorithm.

The pseudocode of the TotalCost scheme is presented below.

Algorithm 1 (*Total-Cost (TotalCost) Scheme*)**.**

We consider a task that needs for its execution L pieces of data (datasets) I_k, of sizes V_{I_k}, $k=1,\ldots,L$.
The scheduler performs the following actions in order to select the DC and the data holding sites

// Find the Data Consolidation site r_{DC}, by looping though each candidate site:

1 **for** each candidate site r_j **do**

// Find the data holding site for each dataset:

2 **for** each dataset I_k **do**

// For each data holding site calculate the time in order to transfer I_k from this site to r_j:

3 **for** each data holding site r_i, where $I_k \in r_i$ **do**

$$transferTime_{I_k,i,j}=\frac{V_{I_k}}{C_{i,j}}+Q_{i,j}^{Comm}+d_{i,j}$$

end for

// Select as data holding site for dataset I_k the one that leads to the minimum time in order
// to transfer I_k to r_j:

4 Select r_i where $transferTime_{I_k,j}$ $=\min_{r_i\in R}(transferTime_{I_k,i,j})$

end for

// The data consolidation time of the candidate DC site r_j is equal to the maximum transfer time of
// any dataset I_k, from the selected data holding site to r_j:

5 $D_{\text{cons}}(r_j) = \max_{k=1,\ldots,L} \left(transferTime_{l_{k,j}}\right)$

end for

6 // Select the DC site (r_{DC}) that minimizes the data consolidation and task execution time:

$$r_{\text{DC}} = \arg\min_{r_j \in R} \left(D_{\text{cons}}(r_j) + Q_j^{\text{Proc}} + \frac{W}{P_j} \right)$$

(v) *Total-Cost Queuing (TotalCost-Q) scheme*: This scheme is similar to the TotalCost scheme, but with the addition that we take into account the communication queuing delays at the links. Generally, scheduling and routing can be improved if link queue length information is available. However, in practice, it is quite difficult to keep such information up to date at all the sites. In grid networks, there are appropriate mechanisms for gathering and communicating queue length information, such as the Network Weather Service [5] or the gLite's middleware Monitoring and Discovery Service [1]. In our simulations, for the Total-Cost Queuing scheme we assume that the central scheduler has instantly valid and updated queue length information for every network link. One of our goals is to investigate the benefits induced to the DC problem when such information is used, and whether the additional effort and costs (of any kind) to acquire it are justified by the performance improvements that can be obtained.

Regarding the third subproblem of the DC, that is, the routing of the datasets, we investigated a number of tree-based DC schemes, for selecting the paths to be followed by the datasets. Intuitively, this seems like the right thing to do, since in DC we have many repository sites (the leaves, or intermediate nodes of a tree) whose datasets are transferred to a DC site (the root). Since we want these transfers to occur concurrently, we try to select the tree using as a criterion either the time for transferring the data over a link or the load of a link, as measured by the size of data queued or under transmission at it, minimizing the delay due to congestion effects that may arise by the concurrent data transfers. In this way, we execute an offline optimization, where we have a number of concurrent data transfers to perform, and try to find the best way to actually perform them, so as to reduce the effects (e.g., delay on network links) that one transfer may induce to the other. This optimization is very important, and several other interesting approaches can be also considered (find the optimum time instance at which these transfers should begin, select the path and wavelength in an optical network, and others). Other works consider only the current network congestion that is caused by the existing traffic in the network due to other data transmissions under way related to previous tasks' datasets.

In the algorithms proposed previously, we used the tree constructed by Dijkstra's shortest path algorithm for routing (Shortest Path Tree — SPT). In the following algorithms, we use Minimum Spanning Trees (MSTs) obtained by Kruskal's algorithm. Other tree algorithms can also be used, such as the Essau–Williams MST algorithm [37], or algorithms for solving the Steiner tree problem.

(vi) *Total-Cost Minimum Spanning Tree (TotalCost-MST) scheme*: In this algorithm, we first select the data replica holding sites and the DC sites using the TotalCost algorithm. Next, we assign to each link a weight, which is equal to the size of the queued data plus the size of the data that will pass through this link based on the decisions made using the TotalCost algorithm. Finally, we apply a Minimum Spanning Tree algorithm (Kruskal) to construct the paths the datasets will follow.

The TotalCost-MST algorithm tries to improve the routing paths selected by the TotalCost algorithm between the data holding sites and the Data Consolidation site. Fig. 2(a) shows an example of the paths initially selected by the TotalCost algorithm, for the datasets *A*, *B* and *C*, where many datasets cross the same network link(s). By applying the MST approach, the paths selected are improved by spreading the network traffic more evenly across the network (Fig. 2(b)).

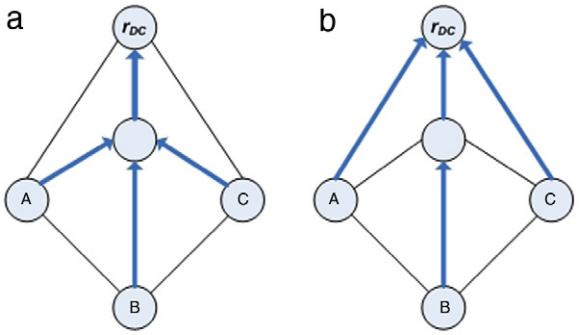


Fig. 2. The paths selected for transferring datasets A, B and C to the r_{DC} site: (a) using the original TotalCost algorithm, (b) after the application of the MST approach in the TotalCost-MST algorithm.

(vii) *Minimim Spanning Tree-Cost algorithm (MST-Cost) scheme*: In this algorithm, we alter the order in which the three subproblems of DC are handled. Specifically, when a new task requests service, we assign to each link (i, j) a weight based on the delay required for transmitting over this link, which includes both the queuing and the propagation delay:

$$Q_{i,j}^{\text{Comm}} + d_{i,j}. \tag{9}$$

Next, we apply an MST algorithm (Kruskal) to construct the paths the datasets should follow, independently of the data replica holding and DC sites that will be chosen in the next phases. Following that, we apply the TotalCost algorithm in order to find these sites.

4.3. Number of operations for serving a task

Data consolidation is viewed in this paper as a continuous-time problem, where the decisions taken for one task affect the decisions taken for the tasks that will arrive in the future and are affected by the decisions taken earlier for previous tasks. In this context, we are interested in the number of operations (complexity) required by the proposed DC schemes. For the Rand algorithm this is polynomial, as it randomly chooses the $L + 1$ sites used in the DC operation. Similarly, the ExecCost algorithm complexity is polynomial, since it randomly selects the L data holding sites, while the r_{DC} site is chosen among the N sites of the grid network based on the task execution time criterion. All the other proposed algorithms are based on the ConsCost algorithm. The complexity of these algorithms is determined by the complexity of the ConsCost algorithm, since any additional operations performed by these algorithms, for example the construction of the Minimum Spanning Tree, require polynomial time. The complexity of Kruskal's MST algorithm is $O(E \cdot \log N)$, where E is the number of links of the grid network. In the ConsCost algorithm, for each candidate DC site, we choose for each dataset the replica site that minimizes its transferring delay. That is, for each dataset, at most all the N sites (assuming that all the sites hold a replica of this dataset) are evaluated as candidates for participating in the DC operation. Then, based on the decisions made for all L datasets, we calculate the data consolidation time for this particular replica selection and candidate DC site. So, the execution of a shortest path algorithm, with polynomial complexity, is required for each candidate DC site r_{DC}. The complexity of Dijkstra's algorithm is $O(N^2)$ in order to find the shortest paths from a single source (the candidate DC site) to all other nodes in a graph. Next, this operation is performed for all the candidates

DC sites, that is, for all the N grid sites. At the end of the ConsCost algorithm, the r_{DC} site and the corresponding L sites with the minimum data consolidation time (Eq. (7)) are selected. Consequently, the total complexity of the ConsCost algorithm, for a single task, is polynomial and equal to $O(N^3)$.

Of course, the polynomial complexity of the ConsCost algorithm is the result of its suboptimal decisions. The ConsCost algorithm does not optimize the overall consolidation time over all datasets and over all candidate replica sites, whose combinations increase exponentially with the number of sites in the grid network. This can be seen from Eq. (5), where for each dataset the replica site with the minimum transferring delay is chosen, without considering the effect of one choice (one replica's selection) to the other.

In addition, we should note that, in this complexity analyses, we do not consider the complexity of the information gathering mechanisms, such as the communication and computation queuing delays in the network.

5. Performance evaluation

We implemented the proposed DC schemes and the required data grid network entities (e.g., computation and storage resources) in the Network Simulator (ns-2) [38]. Ns-2 provides a manageable environment for simulating the network resources of the grid, which is particularly important for the evaluation of DC schemes. The Boost library [39] provided us with the implementations of the Minimum Spanning Tree algorithms. *In all the experiments performed, the algorithms' running time is a few seconds, for a single task scheduling operation.*

5.1. Simulation environment

Generally, in most data-intensive grid applications, there are few advanced laboratories and research centers around the world where large amounts of data (in the scale of TB and GB) are produced. These data consolidate, when needed, at a central site for processing. Furthermore, a data grid usually has a hierarchical structure, as is the case, for example, with the European DataGrid Testbed [40] (currently know as Enabling Grids for E-sciencE — EGEE). The hierarchical structure usually consists of multiple "tiers", where each tier has its own storage capacity. Tier 0 holds all the master files/datasets.

Our simulation environment was based on these facts. Specifically, we used the topology presented in Fig. 3, which is derived from the EGEE [41] topology. Our network consists of 11 nodes and 16 links, of capacities equal to 10 Gbps. In our experiments we assume a P2P (opaque) network; the delay for transmitting between two nodes includes the propagation, queuing and transmission delays at intermediate nodes. Only one transmission is possible at a time over a link, so a queue exists at every node to hold the data waiting for transmission.

We assume that five sites are equipped with computation and storage resources, while the others act as simple routers. We also assume that there is a Tier 0 site in the network, which holds all the datasets, but does not have any computational capability. Each experimental scenario was run five times, using an independent random seed. In every repetition, the placement in the network of the five sites and the Tier 0 was random. Experiments with more sites equipped with computation and storage resources were also performed.

The size of each dataset is given by an exponential distribution with average V_l (GB). At the beginning of the simulation a given number of datasets are generated and two copies of each dataset are distributed in the network; the first is distributed among the sites and the second is placed at the Tier 0 site. The storage capacity

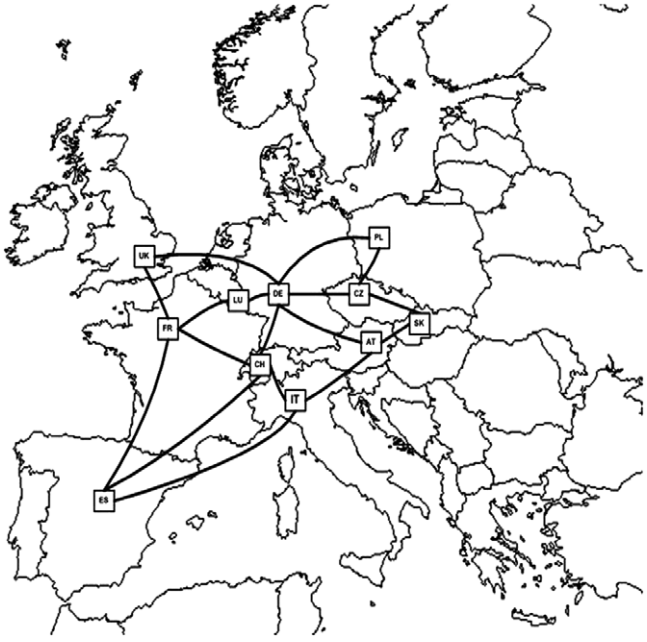


Fig. 3. The topology used in our simulations.

Table 1
The average size V_l and the number L of datasets each task requests. The total number of available datasets in the grid network and their total size is also illustrated.

L	V_l (GB)	Total number of datasets	Total size (TB)
2	300	50	15
3	200	75	15
4	150	100	15
6	100	150	15
8	75	200	15
10	60	250	15
2	400	50	20
3	266	75	20
4	200	100	20
6	133	150	20
8	100	200	20
10	80	250	20

of each storage resource is 50% of the total size of all the datasets. Since the storage capacity is bounded, there are cases where a site does not have the free storage capacity required to store a needed dataset. In such a case, one or more of the oldest and unused datasets are deleted until the new dataset can be stored at the resource.

In each experiment, users generate a total of 10 000 tasks, with exponential arrival rates of average value λ. Unless stated otherwise, we assume that $\lambda = 75$ tasks/s (but we also examine other task arrival rates: $\lambda = 50, 75, 100, 125, 150$ and 200 tasks/s). In all our experiments we keep constant the average total data size S that each task requires:

$$S = L \cdot V_l, \tag{10}$$

where L is the number of datasets a task requests and V_l is the average size of each dataset. We use average total data size S equal to 600 and 800 GB and examine the (L, V_l) pair values presented in Table 1. In each experiment, the total number of available datasets changes in order for their total size to remain the same: 15 TB and 20 TB, respectively (Table 1).

The task workload W correlates with the average total data size S, through a parameter a, as

$$W = a \cdot S. \quad (11)$$

In our simulations, we use parameter a as follows: given the total data size S of a task (different for each task) and a, we use Eq. (11) to calculate the workload of this task. The parameter a measures whether a task is a computation-intensive or data-intensive one. As a increases, the tasks become more CPU-intensive, while as a decreases the tasks have fewer computation demands. We alter the parameter a and examine the performance of our DC strategies. Unless stated otherwise, in our experiments we create data-intensive tasks by setting $a = 0.01$. We also examine CPU-intensive tasks (a takes values up to 11). Also, when a task completes its execution we assume that there is no output data returned to the originating user.

5.2. Performance metrics

We use the following metrics to measure the performance of the algorithms examined.

- The average task delay, which is the time that elapses between the creation of a task and the time its execution is completed at a site.
- The average load per task imposed to the network, which is the product of the size of datasets transferred and the number of hops these datasets traverse.
- The task success ratio, which is the ratio of the number of tasks that were successfully scheduled to the total number of tasks generated. When a large number of tasks are queued or under execution, it may be impossible for the scheduler to find a resource with sufficient free storage space, where a new task's datasets can consolidate (see Eq. (1)). In this case the task cannot be scheduled, and it fails.
- The Data Consolidation (DC) probability, which is the probability that the selected DC site will not have all the datasets required by a task and as a result data consolidation will be necessary.

The first metric characterizes the performance of the DC strategy in executing a single task, while the second and the third metrics express the overhead the DC strategy induces to the network. The fourth metric gives information on the way the DC site is selected, with respect to the datasets that are located (or not) at this DC site.

5.3. Simulation results

5.3.1. Basic schemes

Fig. 4 shows the DC probability for the Rand, ExecCost, ConsCost and TotalCost schemes, when tasks request different number of datasets L for their execution. The higher the number L of datasets a task requests, the higher is the probability that these datasets will not be located at the DC site, given that the storage capacity of a site is limited. The ConsCost and TotalCost algorithms exhibit smaller DC probability than the Rand and ExecCost algorithms, since the former algorithms select the DC site by taking into account the consolidation delay, which is small for sites holding many or all of the datasets needed by a task. On the other hand, the Rand and ExecCost algorithms select the DC site randomly or almost randomly (as is the case for ExecCost, given that the tasks have negligible computation complexity). As L increases, the probability of not finding all the data at a site increases and converges to 1 for all examined algorithms.

Fig. 5 shows the average task delay for the DC algorithms examined. We observe that the algorithms that take the data consolidation delay into account (namely, the ConsCost and TotalCost algorithms) behave better than the algorithms that do not consider this parameter (that is, the Rand and ExecCost algorithms), in terms of the task delay. As the number L of datasets a task requires increases, the average task delays of all the algorithms converge. Specifically, for the ConsCost and TotalCost algorithms, the average task delay increases as the number of datasets a task requires increases, since the probability that a DC site will not hold all the data a task needs (i.e., the DC probability) also increases (Fig. 4), resulting in more data transfer. In contrast, in the Rand and ExecCost algorithms, the average task delay decreases as L increases, because of the decrease in the size of the concurrent transferred datasets V_l (Eq. (10)). Thus, for the Rand and ExecCost algorithms that (almost) randomly select the DC site, the data consolidation time and its impact on the average task delay decrease as L increases.

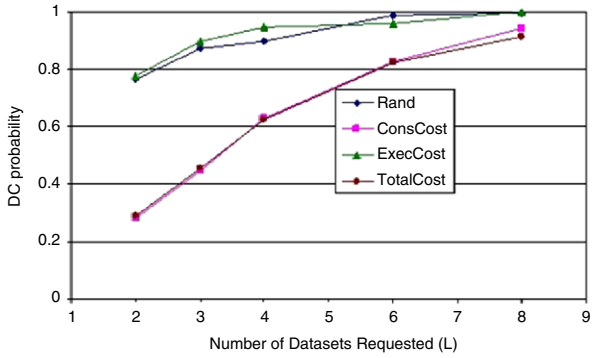


Fig. 4. The DC probability for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks request a different number of datasets L for their execution. The average total data size per task is $S = 600$ GB.

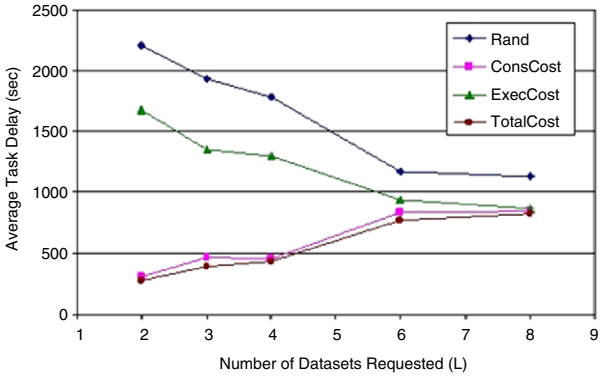


Fig. 5. The average task delay (in s) for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks request a different number of datasets L for their execution. The average total data size per task is $S = 600$ GB.

Fig. 6 shows the average network load per task for the various DC algorithms, when tasks request different number of datasets L for their execution. We observe that the algorithms that do not take into account the data consolidation delay (that is, the Rand and ExecCost algorithms) induce, on average, a larger load on the network than the algorithms that do take this into account (ConsCost and TotalCost algorithms). This is because the former algorithms transfer on average more data, over longer paths. Moreover, the decisions made by these algorithms are not affected by the dataset sizes V_l or their number L, and as a result they induce on average the same network load. By analyzing our results, we observed that these algorithms transfer on average the same number of bytes over paths of equal on average length, irrespectively of L and V_l. The superior performance of ExecCost over that of Rand is because ExecCost assigns tasks to resources in a more balanced way, based on the task execution times. That is, it first assigns tasks to the most powerful resource, where tasks

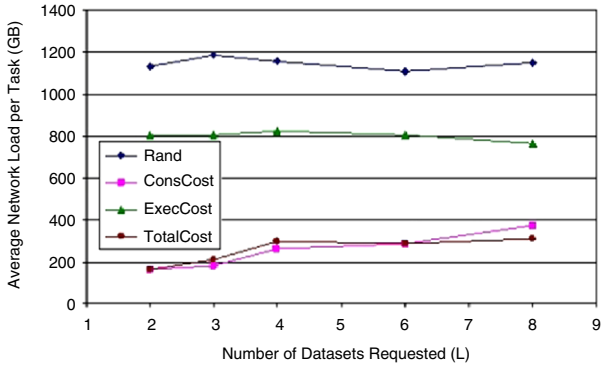


Fig. 6. The average network load per task (in GB) for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks request a different number of datasets L for their execution. The average total data size per task is $S = 600$ GB.

and their datasets are stored until they are executed. When this resource does not have sufficient storage capacity to store the dataset of the following tasks, the ExecCost algorithm chooses the second most powerful resource, and so on. At some point this cycle starts (more or less) all over again. In this way, there is a high probability that consecutive tasks will find some of their required datasets in the resource where they are assigned by the ExecCost algorithm. This reduces the network load in comparison to the Rand algorithm. Of course, this property does not appear in the DC metric, since the resource selected by the algorithm changes when it does not have any free space left. The algorithms that take into account the data consolidation delay (namely, the ConsCost and TotalCost algorithm) induce a smaller load on the network. This load increases as the number of datasets L increases, as can be explained by the increasing probability that a DC site will not hold all the required data (Fig. 4), and will thus have to transfer more datasets. In addition, these algorithms are affected mainly by the tasks' data dependencies, since their computational load is small (see step 6 in Algorithm 1). Also, the TotalCost and the ConsCost use the same policies for selecting the data replicas and the paths; as a result, both algorithms perform similarly. We should note that in all the experiments performed very few tasks failed (of the order of 4–6 tasks per experiment).

5.3.2. Dimensioning task and resource related parameters

Fig. 7 illustrates the average delay and the average network load induced per task for the proposed DC schemes, when tasks become more CPU-intensive rather than data-intensive. In order to examine this effect, we increased the parameter a used in Eq. (11). We observe that the TotalCost algorithm performs better in all cases. When tasks are data-intensive, it achieves small task delay and network load, and behaves similarly to the ConsCost algorithm. As tasks become more CPU-intensive (higher values of parameter α), the TotalCost algorithm continues to achieve small task delay and behaves similarly to the ExecCost algorithm, while the average task delay achieved by the ConsCost algorithm becomes very large. Finally, the network load induced by the TotalCost algorithm increases as tasks become more CPU-intensive, although it remains smaller than that induced by the ExecCost and Rand algorithms.

Fig. 8illustrates the average delay and the average network load per task for the proposed DC algorithms, as a function of the task arrival rate λ. As expected, the average task delay increases for all the algorithms examined when the task arrival rate increases. Again, the ConsCost and TotalCost algorithms produced better results than the Rand and ExecCost algorithms. We observe that the ConsCost and TotalCost algorithms induce the same network load per task, without being influenced by the increase in the task arrival rate. In contrast, the network load induced by the Rand and ExecCost algorithms decreases as the task arrival rate increases, since their corresponding DC probability decreases rapidly. This means that more tasks find a DC site that holds all the needed datasets, and as a result the load induced to the network by the various data transfers decreases.

We also performed a number of experiments assuming different values for the number of sites that are equipped with computational and storage resources. In all our previous experiments, we assumed that five such sites existed, while all the other nodes were acting as simple routers. Fig. 9 presents the results obtained when varying the number of sites between two and ten, while keeping the total storage capacity of the grid network the same as in the case where we had only five sites. We observe that, as the number of sites increases, the average task delay and the load induced to the network also increase. This is because having a larger number of sites increases the chance that the data will not be located at the DC site or at sites close to that. As a result, more data transfers are needed, increasing the task delay and network load.

5.3.3. Sum versus max operation used for defining the site access cost

In the ConsCost and TotalCost algorithms, we calculate the access cost of each DC site, for a given task, as the maximum transfer cost of any of the datasets (best replica of each dataset) the task requires, and select the site with the minimum access cost. This seems like the right thing to do, since the maximum cost of a dataset is the one that influences the duration of the data

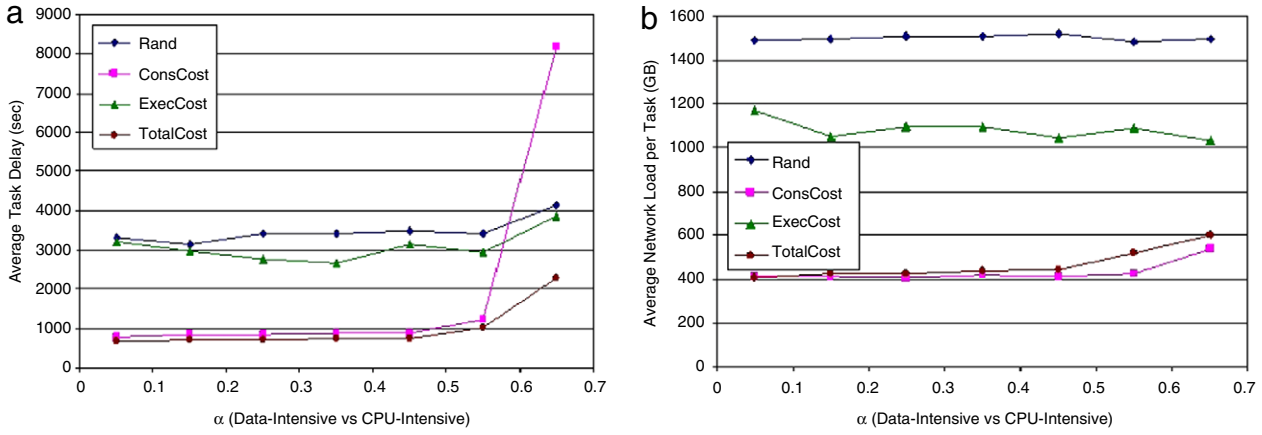


Fig. 7. (a) The average task delay (in s) and (b) the average network load per task (in GB) for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks become more CPU-intensive rather than data-intensive (increasing values of parameter α) for average total data size per task $S = 800$ GB.

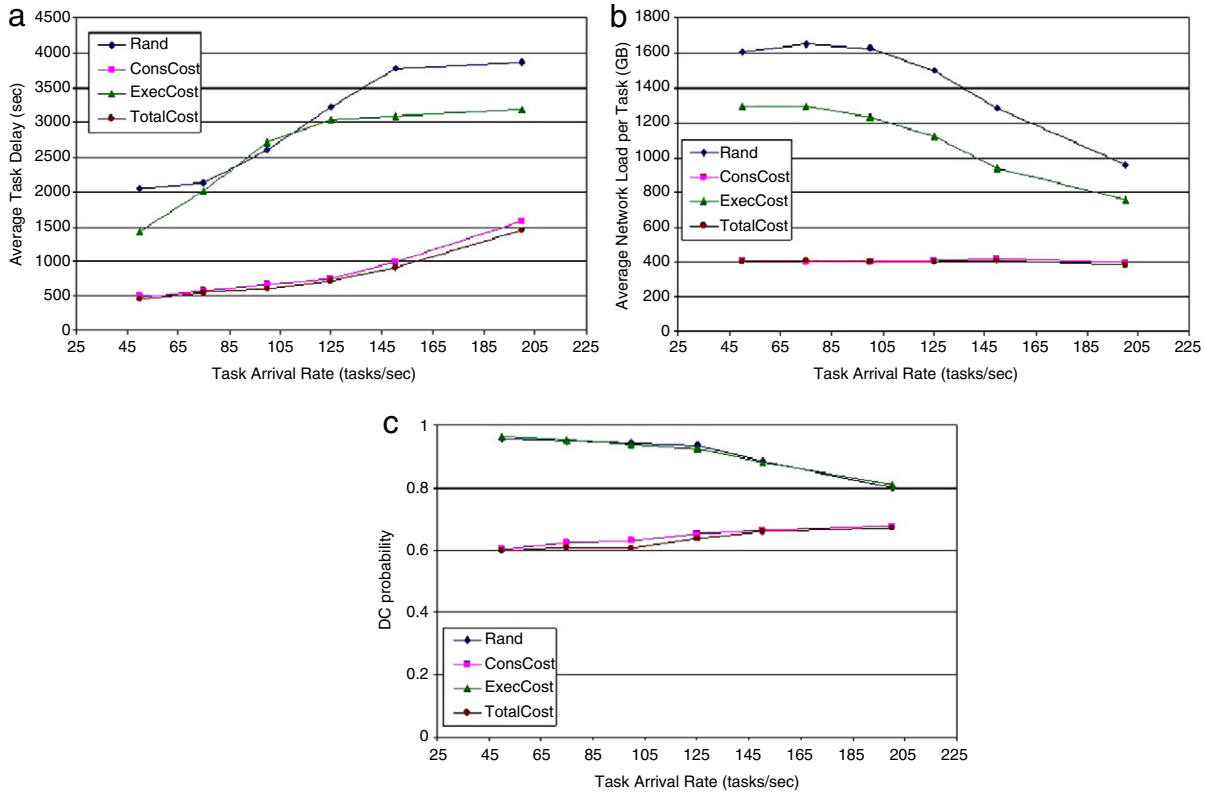


Fig. 8. (a) The average task delay (in s) and (b) the average network load per task (in GB) and (c) the DC probability for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, as a function of the average task arrival rate γ. The average total data size per task is $S = 800$ GB.

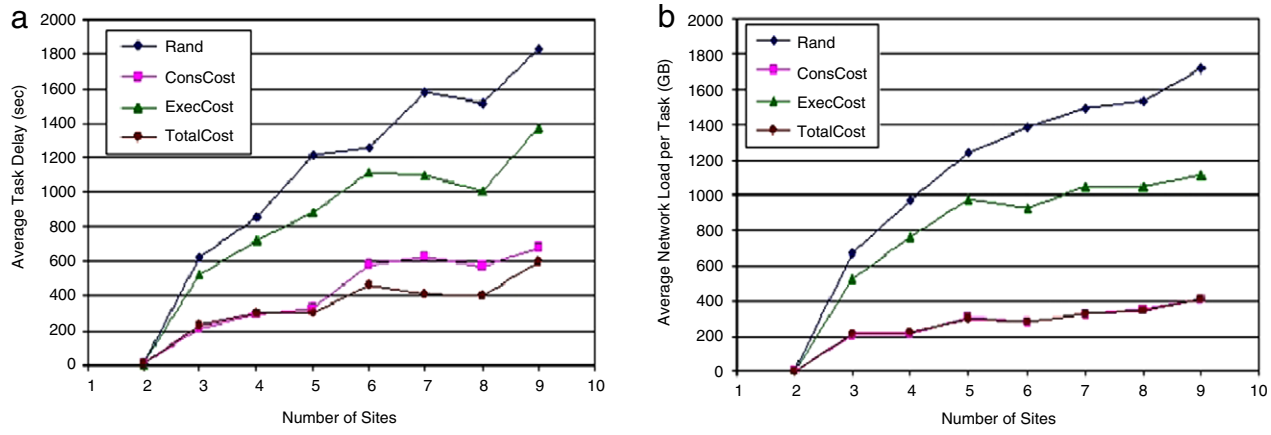


Fig. 9. (a) The average task delay (in s) and (b) the average network load per task (in GB) for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when the number of sites increases. The average total data size per task is $S = 800$ GB.

consolidation operation. However, we also performed experiments in which we calculated the access cost of each site by summing the times required to access the datasets (best replica of each dataset). This approach was followed in [26–28] for the sequential (and not concurrent) datasets access scenario.

For our experiments, we changed the TotalCost algorithm in order to calculate the access cost of each site as the sum of the access costs for each dataset required by the corresponding task. We named this new algorithm TotalCostSUM. In Fig. 10 we observe that the TotalCost algorithm achieves smaller task delay than the TotalCostSUM algorithm. We note that, when a task requests only a small number of datasets (<4), the two algorithms perform similarly, since the sum and max operations result in most cases in the selection of the same site for task execution. However, as the number of datasets increases, the TotalCost algorithm outperforms the TotalCostSUM, showing the importance of considering the transfers of all the datasets jointly. On the other hand, the TotalCostSUM algorithm results in a slightly smaller network load than the TotalCost algorithm.

We performed a number of additional experiments under a variety of scenarios and assumptions on the traffic load. Our results indicated that in a number of cases, especially for very large task loads, the TotalCostSUM algorithm outperformed the TotalCost with respect to the task delay metric. In these heavy load cases it seems that considering the sum of the transfer access costs of the datasets, as a metric for selecting the consolidation site, is a better optimization criterion than considering the maximum dataset's access cost. This is probably expected for heavy loads, since the sum

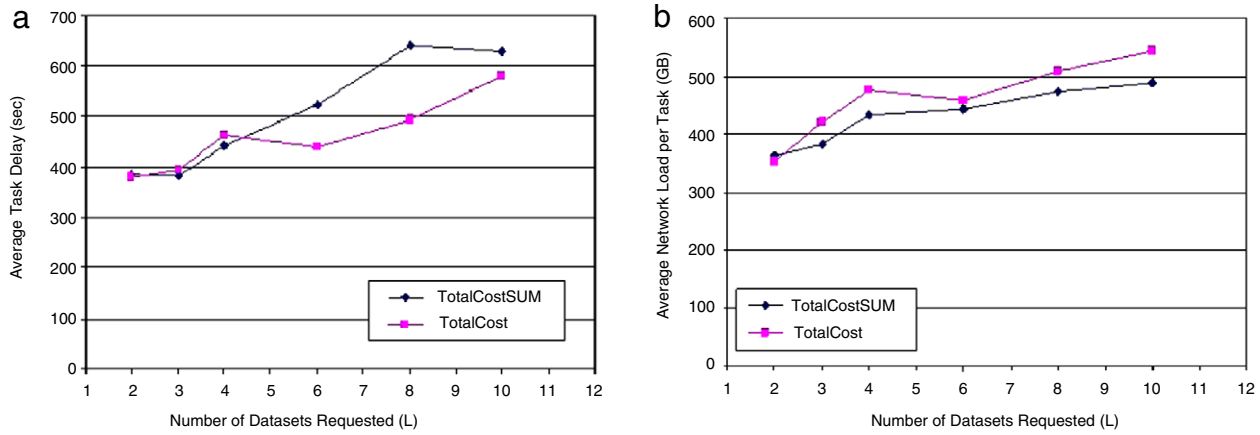


Fig. 10. (a) The average task delay (in s) and (b) the average network load per task (in GB), for the TotalCost and the TotalCostSUM DC algorithms, when tasks request a different number of datasets L for their execution. The average total data size per task is $S = 800$ GB.

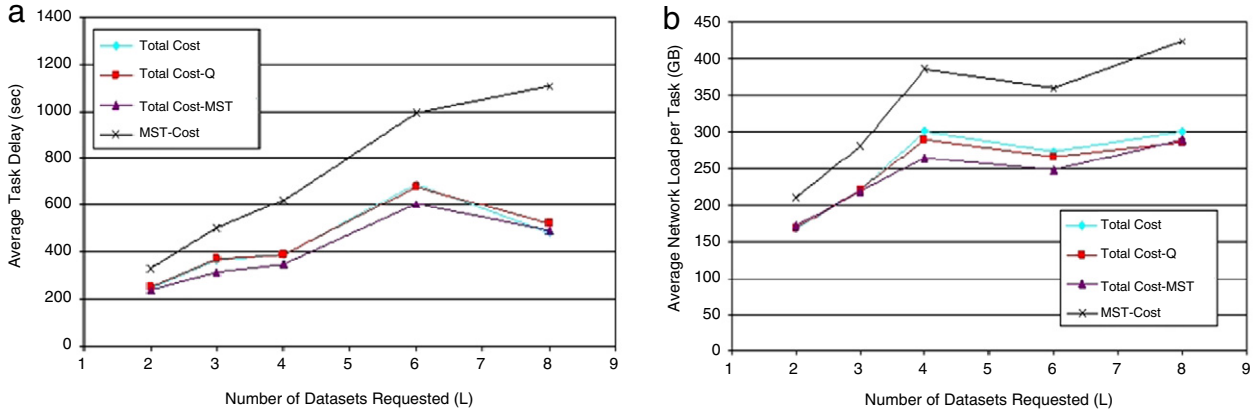


Fig. 11. (a) The average task delay (in s) and (b) the average network load per task (in GB), for the TotalCost, TotalCost-Q, TotalCost-MST and MST-Cost DC algorithms, when tasks request a different number of datasets L for their execution. The average total data size per task is $S = 800$ GB.

of the transfer access costs is related to the load induced in the network (which at heavy loads has a great effect on the delay), while for light load the maximum of the access costs is a better measure of the data consolidation delay. These results also indicate the complexity of the DC operation and its dependency on a number of different parameters, such as the network congestion investigated next.

5.3.4. Tree and queuing delay based schemes

We performed a number of simulations with the TotalCost-Q, TotalCost-MST and MST-Cost DC algorithms, and compared them with the TotalCost algorithm. Our results show that tree-based algorithms and especially Minimum Spanning Tree algorithms, fit quite well to the DC problem and can considerably improve performance. Moreover, we show that taking into account the queuing delays is not so beneficial, and does not justify the communication and processing overhead required for communicating such information across the network. Finally, it seems that the order in which the DC operations are performed, and in particular the order in which the corresponding DC subproblems are solved, significantly affects the efficiency of the DC schemes.

In Fig. 11, we observe that the TotalCost-MST algorithm performs better than the TotalCost and the TotalCost-Q algorithms. The TotalCost-MST algorithm makes, in most case, similar decisions regarding the selection of the data holding sites and the DC site with the other algorithms. This is confirmed by the DC probability graph, which is identical for all the algorithms. However, the TotalCost-MST algorithm builds more efficient paths than the Dijkstra algorithm, resulting in smaller network load and average task delay. Furthermore, we observe that the TotalCost-Q algorithm produces only slightly better results than the TotalCost algorithm, though the former takes queuing delays into account. Generally, the sizes of the queue lengths must be large (corresponding to many and/or large entries), in order for the queuing delays to have a noticeable effect. So, in many practical cases, the knowledge of the links queuing delay is not necessary for making efficient DC-related decisions. Finally, we observe that the MST-Cost algorithm produces worse results than the TotalCost, TotalCost-Q and TotalCost-MST algorithms. This occurs because, by applying a MST algorithm first, we limit the number of links, and as a result the number of possible decisions one can make regarding the data repository and the DC sites. This has a significant negative effect on the final results.

5.3.5. Data consolidation and resiliency

An important issue for Data Consolidation (DC) schemes is the resiliency they can provide against failures of storage or network resources. In this section, we combine the proposed DC schemes with two simple resiliency techniques and examine their performance. In the first technique, called Double Site, we select two Data Consolidation sites, namely, the ones that are best and second best according to the corresponding DC scheme used, and we transfer the required datasets to both sites. The task is transferred only to the first site, while the other is used as a backup in case the first site fails. In the second technique, called Half Data, we again select in the same way two DC sites, but in the second-best site we

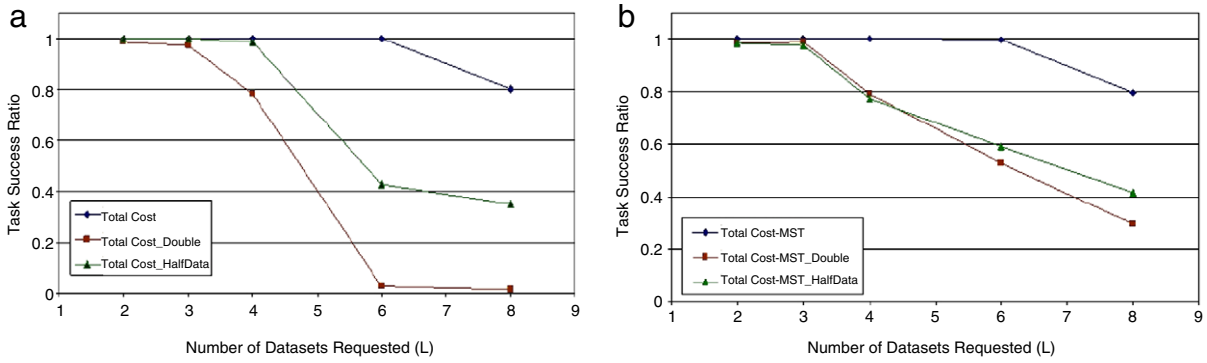


Fig. 12. The task success ratio (a) without and (b) with MST-based routing algorithms for the routing of the datasets, when tasks request a different number of datasets L for their execution. The average total data size per task is $S = 800$ GB.

transfer only half (in size) of the datasets needed by the task. This way we reduce the load induced to the network, but we also reduce the resiliency efficiency, since in a case of failure at the first DC site, the rest of the data will have to be transferred to the second DC site.

When one of the proposed resiliency techniques is applied then the network load increases and the storage resources occupied are also increased, reducing the grid's ability to serve future task requests and increasing task delay. This results in longer reservation times of the storage resources and, consequently, it may be impossible for the scheduler to find a resource with sufficient free storage space where a new task's datasets can consolidate. In this case, the task cannot be scheduled and it fails. An efficient DC scheme that better handles network congestion can achieve smaller task delay and smaller storage resource reservation time per task, and as a result larger task success ratios. This is indicated in Fig. 12, where the TotalCostMST_Double algorithm achieves resiliency, while reducing network congestion and achieving larger task success ratios than the TotalCost_Double algorithm. We should note that in the Double Site technique more datasets are transferred, and the success ratio is generally smaller than in the Half Data technique. However, we observe that when the MST approach is used (Fig. 12(b)) the TotalCostMST_Double and the TotalCostMST_HalfData schemes achieve similar task success ratios. Generally, these results indicate the importance of concurrently transferring datasets and carefully choosing the routing paths followed (e.g., tree-based Data Consolidation scheme) so as to decrease network congestion. Finally, we should note that the incorporation of the described resiliency techniques into the Data Consolidation schemes does not increase their complexity appreciably.

6. Conclusions

In this work, we have examined the Data Consolidation (DC) problem in grid networks. The DC problem arises when a task needs for its execution, concurrently, two or more pieces of data, possibly scattered throughout the grid network, and consists of three subproblems: (i) the selection of the repository site from which to obtain the replica of each dataset to be used for task execution, (ii) the selection of the site where these datasets will accumulate and the task will be executed, and (iii) the selection of the paths the datasets will follow as they are transferred over the network. These subproblems can be solved jointly or independently. To the best of our knowledge, this is the first time that these issues have been jointly considered. We have proposed a number of DC schemes, of polynomial number of required operations, that consider data consolidation, task execution and/or queuing delays, and have evaluated them under a variety of scenarios and choices for the parameters involved. Our simulation results indicated that the DC schemes that consider both the computation and the communication requirements of the tasks behave better than those that consider only one of them. We also showed the importance of optimizing the concurrent transfers of the datasets required for a task's execution. In particular, by using Minimum Spanning Trees instead of Shortest Paths for routing the datasets concurrently, we reduced the network congestion that may appear in the future due to these transfers. Our results also indicated that applying graph theory concepts, and specifically Minimum Spanning Tree algorithms, in the DC problem can be quite beneficial when considering resiliency issues. Moreover, we showed that in many cases considering the link queue lengths is not necessary for performing the DC operation efficiently. Finally, our simulation results showed that for light load the maximum of the access costs is a better measure of the data consolidation delay than the sum, while the opposite is true for heavy loads.

Acknowledgement

This work has been supported by the European Commission through the IP Phosphorus project.

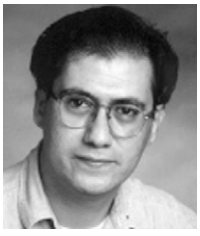
References

[1] E. Laure, S.M. Fisher, Á. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, A. Edlund, Programming the grid with gLite, Computational Methods in Science and Technology 12 (2006) 33–45.
[2] A. Kretsis, P. Kokkinos, E. Varvarigos, Developing scheduling policies in gLite middleware, Cluster Computing Grid (2009) 20–27.
[3] D.S. Katz, et al., Astronomical image mosaicking on a grid: initial experiences, in: Engineering the Grid—Status and Perspective, American Scientific Publishers, 2006.
[4] KoDaVis: http://www.viola-testbed.de/index.php?id=kodavis.
[5] R. Wolski, N. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Future Generation Computer Systems 15 (1999) 757–768.
[6] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: Symposium on Operating System Design and Implementation, 2004.
[7] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience 32 (2) (2002) 135–164.
[8] T. Braun, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837.
[9] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: HPDC, USA, 2002.
[10] Y. Cardinale, H. Casanova, An evaluation of job scheduling strategies for divisible loads on grid platforms, in: HPC&S, 2006.
[11] R. Buyya, M. Murshed, D. Abramson, S. Venugopal, Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimisation algorithm, International Journal of Software: Practice and Experience (SPE) 35 (5) (2005) 491–512.

[12] K.H. Kim, R. Buyya, Fair resource sharing in hierarchical virtual organizations for global grids, in: Intl. Conf. on Grid Computing, 2007.
[13] N. Doulamis, E. Varvarigos, T. Varvarigou, Fair scheduling algorithms in grids, Transactions on Parallel and Distributed Systems 18 (11) (2007) 1630–1648.
[14] P. Kokkinos, E. Varvarigos, A framework for providing hard delay guarantees and user fairness in grid computing, Future Generation Computer Systems 25 (6) (2009) 674–686.
[15] H. Shan, L. Oliker, W. Smith, R. Biswas, Scheduling in heterogeneous grid environments: the effects of data migration, in: Intl. Conference on Advanced Computing and Communication, 2004.
[16] R. Rahman, K. Barker, R. Alhajj, Study of different replica placement and maintenance strategies in data grid, in: IEEE/ACM International Symposium on Cluster Computing and Grid, 2007, pp. 171–178.
[17] R. Rahman, K. Barker, R. Alhajj, Replica placement strategies in data grid, Journal of Grid Computing 6 (1) (2008) 103–123.
[18] A. Dogan, A study on performance of dynamic file replication algorithms for real-time file access in data grids, Future Generation Computer Systems 25 (8) (2009) 829–839.
[19] R.M. Rahman, K. Barker, R. Barker, A predictive technique for replica selection in grid environment, in: Intl. Symposium on Cluster Computing and the Grid, 2007, pp. 163–170.
[20] S. Vazhkudai, S. Tuecke, I. Foster, Replica selection in the globus data grid, in: Intl. Symp. on Cluster Computing and the Grid, 2001.
[21] J. Byers, M. Luby, M. Mitzenmacher, Accessing multiple mirror sites in parallel: using tornado codes to speed up downloads, in: IEEE INFOCOM, 1999, pp. 275–283.
[22] P. Rodriguez, E. Biersack, Dynamic parallel access to replicated content in the internet, IEEE/ACM Transactions on Networking 10 (4) (2002) 455–465.
[23] R. Chang, M. Guo, H. Lin, A multiple parallel download scheme with server throughput and client bandwidth considerations for data grids, Future Generation Computer Systems 24 (8) (2008) 798–805.
[24] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: Intl. High Performance Distributed Computing Syumposium, 2002, pp. 352–358.
[25] A. Elghirani, R. Subrata, A. Zomaya, Intelligent scheduling and replication in datagrids: a synergistic approach, in: Symposium on Cluster Computing and the Grid, 2007, pp. 179–182.
[26] W.H Bell, D.G Cameron, L. Capozza, A.P. Millar, K. Stockinger, F. Zini, Simulation of dynamic grid replication strategies, optorsim, in: LNCS, vol. 2536, 2002, pp. 46–57.
[27] W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger, F. Zini, Optorsim: a grid simulator for studying dynamic data replication strategies, International Journal of High Performance Computing Applications 17 (4) (2003) 403–416.
[28] D. Cameron, A. Millar, C. Nicholson, R. Carvajal-Schiaffino, F. Zini, K. Stockinger, Optorsim: a simulation tool for scheduling and replica optimisation in data grids, in: Computing in High Energy and Nuclear Physics, 2004.
[29] A Chakrabarti, R. Dheepak, S Sengupta, Integration of scheduling and replication in data grids, in: LNCS, vol. 3296, 2004, pp. 375–385.
[30] Phosphorus: http://www.ist-phosphorus.eu/.
[31] G. Hoo, W. Johnston, I. Foster, A. Roy, QoS as middleware: bandwidth reservation system design, in: Intl. Symposium on High-Performance Distributed Computing, HPDC, 1999, pp. 345–346.
[32] I. Foster, et al., A distributed resource management architecture that supports advance reservations and co-allocation, in: Intl. Workshop on Quality of Service, IWQoS, 1999.
[33] T. Stevens, M. De Leenheer, C. Develder, B. Dhoedt, K. Christodoulopoulos, P. Kokkinos, E. Varvarigos, Multi-cost job routing and scheduling in grid networks, Future Generation Computer Systems 25 (8) (2008) 912–925.
[34] P. Kokkinos, K. Christodoulopoulos, A. Kretsis, E. Varvarigos, Data consolidation: a task scheduling and data migration technique for grid networks, Cluster Computing Grid (2008) 722–727.
[35] Peter A. Dinda, Online prediction of the running time of tasks, Cluster Computing Grid 5 (3) (2002) 225–236.
[36] N. Doulamis, A. Doulamis, A. Litke, A. Panagakis, T. Varvarigou, E. Varvarigos, Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing, Computer Communications 30 (2007) 499–515.
[37] L. Esau, K. Williams, On teleprocessing system design, IBM Systems Journal 5 (1966) 142–147.
[38] Ns—network simulator. http://www.isi.edu/nsnam/.
[39] Boost: http://www.boost.org/.
[40] W. Hoschek, F. Jaén-Martínez, A. Samar, H. Stockinger, K. Stockinger, Data management in an international data grid project, in: Intl. Workshop on Grid Computing, 2000.
[41] EGEE: http://www.eu-egee.org/.

Panagiotis Kokkinos received his Diploma in Computer Engineering and Informatics in 2003, and his M.S. degree in Integrated Software and Hardware Systems in 2006, both from the University of Patras, Greece. He has worked in the private sector. He is currently a Ph.D. student in the Department of Computer Engineering and Informatics of the University of Patras. His research activities are in the areas of ad hoc networks and grid computing.

Konstantinos Christodoulopoulos received his Diploma of Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 2002, and his M.Sc. degree in Advanced Computing from Imperial College London, UK, in 2004. He received his Ph.D. degree from the Computer Engineering and Informatics Department of the University of Patras, Greece. His research interests are in the areas of protocols and algorithms for optical networks and grid computing.



Emmanouel (Manos) Varvarigos received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens in 1988, and his M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 1990 and 1992, respectively. He has held faculty positions at the University of California, Santa Barbara (1992–1998, as an Assistant and later an Associate Professor) and Delft University of Technology, the Netherlands (1998–2000, as an Associate Professor). In 2000 he became a Professor in the Department of Computer Engineering and Informatics at the University of Patras, Greece, where he heads the Communication Networks Laboratory. He is also the Director of the Network Technologies Sector (NTS) at the Research Academic Computer Technology Institute (RA-CTI), which, through its involvement in pioneering research and development projects, has a major role in the development of network technologies and telematic services in Greece. Professor Varvarigos has served on the organizing and program committees of several international conferences, primarily in the networking area, and on national committees. He has also worked as a researcher at Bell Communications Research, and has consulted with several companies in the US and in Europe. His research activities are in the areas of protocols for high-speed networks, ad hoc networks, network services, parallel and distributed computation and grid computing.