Eighth IEEE International Symposium on Cluster Computing and the Grid

# Data Consolidation: A Task Scheduling and Data Migration Technique for Grid Networks

P. Kokkinos, K. Christodoulopoulos, A. Kretsis, and E. Varvarigos
*Department of Computer Engineering and Informatics, University of Patras, Greece and Research Academic Computer Technology Institute, Patras, Greece.*

## Abstract

*In this work 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 needs for its execution two or more pieces of data, possibly scattered throughout the Grid Network. In such a case, the scheduler and the data manager must select the data replicas to be used and the site where these will accumulate for the task to be executed. The policies for selecting the data replicas and the data consolidating site comprise the Data Consolidation problem. We propose and experimentally evaluate a number of DC techniques. Our simulation results brace our belief that DC is an important technique for Data Grids since it can substantially improve task delay, network load and other performance related parameters.*

## 1. Introduction

The continuing deployment of high speed networks is making the vision of Grid Networks a reality. 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.

There is a variety of applications that can benefit from Grid computing; some involve computationally intensive problems on small datasets, while others, called data-intensive applications, perform computations on large sized data, 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 such applications data network communication delays play a key role and considerably affect task completion times. In such an environment the collaboration of task scheduling and data management is essential for boosting Grid performance.

Generally, the scheduling of tasks to resources is a difficult problem, since Grids are quite dynamic, with resource availability and load varying rapidly with time, while at the same time tasks have very diverse characteristics and requirements. The extent to which scheduling is able to cope with such an unpredictable environment significantly affects the utilization of the resources and the Quality of Service (QoS) provided to the users. On the other hand, data management involves various data handling issues related to the sites data are stored, the way data are replicated (migrated), and the times they are replaced-reshuffled or moved over the network. Data replication is regarded as one of the main optimization techniques available for providing fast data access and reliability. In data replication, data are replicated in a number of sites throughout the Grid Network so that the tasks can achieve efficient and fast access to the huge and widely distributed data.

In this work we evaluate a task scheduling and data migration problem, called Data Consolidation (DC). DC applies to data-intensive applications that need for their execution more than one pieces of data. In such a case the scheduler must select both the replica of each dataset (meaning the data repository site from which to obtain the dataset) that will be used by the task, and the site where these pieces of data will accumulate and the task will be executed. The delay required for transferring the output data files to the originating user (or to a site specified by him) should also be accounted for. The policies for selecting the data replicas and choosing the data consolidating site, compromise the DC problem. We propose and experimentally evaluate a number of DC techniques. Our simulation results show that if DC is performed efficiently, important benefits can be obtained in terms of task delay, network load and other performance parameters of interest. Note that DC can be considered as the dual of the data replication problem, where from one site the data are scattered to many repository sites. Through this dual relationship we will show that DC can also provide, by reversing the procedure, data replication services. More complex scenarios can also be considered. An example is the case where the task's execution produces intermediate results that are transferred to another site in order to be used for the execution of a second task, defining in this way a complex workflow. Such complex workflows are outside the scope of the current report, and are left for future work.

DC applies to the case where a task needs for its execution multiple pieces of data (datasets). An example application of DC is Montage [19], which is a toolkit for constructing custom, science-grade mosaics by composing multiple astronomical images. The mosaic to be constructed is specified by the user in terms of a set of parameters, including dataset, location and size on the sky, coordinate system and projection, and spatial sampling rate. In most cases, the astronomical datasets are massive, and are stored in distributed archives that are, in most cases, remote with respect to the available computational resources. The Montage toolkit is used by a number of NASA projects and is usually run on both single- and multi-processor computers, including clusters and Grids.

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 and propose a number of DC techniques. In Section 5 we present the simulation environment and the performance results obtained for the proposed techniques. Finally, conclusions are presented in Section 6.

## 2. Previous Work

Grid Networks can be distinguished into Computational and Data Grids based on whether they serve

978-0-7695-3156-4/08 $25.00 © 2008 IEEE
DOI 10.1109/CCGRID.2008.18

computationally intensive (cpu-intensive) or data-intensive applications. A number of task scheduling and data management algorithms have been proposed so far in the literature. The scheduling of tasks to resources has been considered, among others works, in [1][2][3], where several centralized, hierarchical or distributed scheduling schemes are presented. Most scheduling algorithms try to minimize the total average task delay [3] and maximize resource utilization. Other performance metrics used are the average task slowdown [2], defined as the ratio of the task's total delay to its actual run time, the probability to miss the deadline, and several other metrics. Other works incorporate economic models in Grid scheduling. For example, [4] proposes scheduling algorithms that take into account deadline and budget constraints. Fair scheduling in Grid Networks has also been addressed in [5][6].

Data management in Grids deals mainly with data migration, that is, with the decisions regarding the place and the time application-related datasets should be moved. The data migration cost depends on the storage resources, data transfer protocols, network topology and network conditions. A number of works have considered the effects of data migration in Grids [11]. The most common data migration technique is data replication, which is the process of distributing replicas of data across sites. Data replication reduces access latency and bandwidth consumption and improves the reliability and the resilience of the Grids by increasing data availability. When different sites hold replicas of a particular dataset, there is a significant benefit realized by selecting the best replica among them. The best replica is the one that optimizes a desired performance criterion such as the access latency, the cost and the security [9][10][13]. To better select a replica, a predictive approach can be used. Under this approach the history of data transfer times helps in predicting the current data transfer throughput [9]. In [19] alternative dynamic replication strategies were evaluated, and it was found that significant savings in latency and bandwidth can be obtained if the access patterns are characterized by some degree of geographical locality.

Techniques that jointly address the task scheduling and data replication problems are proposed in [10][11][12][15][17]. Specifically, in [10] the task scheduling and the data replication problems are decoupled, and algorithms for both problems are proposed and evaluated. In [12] the data management service proactively replicates the datasets at selected sites, while an intelligent Tabu-search scheduler dispatches tasks to resources so as to optimize execution time and system utilization metrics. The authors in [15] deal with the problem of integrating scheduling and replication strategies. An Integrated Replication and Scheduling Strategy (IRS) scheduler is proposed, aiming at improving performance based on the coupling between the scheduling and replication strategies. In [17] the authors develop a suite of task scheduling and data replication algorithms. Their simulation results show that scheduling tasks to sites that contain the needed datasets and asynchronously replicating popular datasets to remote sites work well.

Our proposed Data Consolidation (DC) scheme is performed when a task needs for its execution multiple pieces of data stored at different sites. Though this seems like a very obvious scenario, especially for data-intensive application, most of the related works seems to ignore it. Specifically, most related works assume that each task needs for its execution only one large piece of data [10][11]. There is, however, a small number of works that assume that a task needs for its execution more than one pieces of data [12][15]. In [15] the authors focus on data replication and integrate scheduling and replication strategies. In this context, they propose a number of policies for selecting the data consolidating site, considering this procedure as a scheduling problem. Contrary to the aforementioned approach, in this work we focus on the selection of the Data Consolidating (DC) site, considering it as problem consisting of two parts. In the first part, called data management, the data replicas that the task will use are selected, as in [9][14]. In the second part, called task scheduling, the site where these datasets will consolidate (accumulate) for the task's execution is decided. The policies for selecting the data replicas and the DC site comprise the DC problem. In our work we propose a number of solutions for the DC problem and perform a large number of simulation experiments to evaluate their applicability and performance. Furthermore, although we do not consider in this work data replication strategies (re-shuffling the data in the network at specific instances), DC can be used as an indirect data replication technique.

In practice, a Data Grid usually has a hierarchical structure, as is the case, for example, with the European DataGrid Testbed [20]. Therefore, a Data Grid usually consists of multiple "tiers", where each tier has its own storage capacity. Tier 0 holds all of the master files/datasets, Tier 1 consists of national centers and below that there are regional centers. Datasets can be placed at each tier to increase data availability among different sites.

The present work is, to the best of our knowledge, the first time Data Consolidation (DC) is used in the context of Grids. DC is usually encountered in IT environments, where it is used to overcome server, storage and application sprawling by consolidating the data centers into fewer, centralized locations. Typically, these data reside in legacy systems and must be migrated into new systems, a process which is complicated, time-consuming, resource-intensive and high-risked. A number of companies are offering DC solutions for the enterprises. The IT related DC has offline characteristics, where data are consolidated before the applications' execution. In contrast, in a Grid environment, DC is performed online, in the sense that it takes place when an application tasks needs to be executed.

## 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 the submitted tasks, a storage resource where data 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 $C_{i,j}$ 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. The local scheduler uses the First Come First Served (FCFS) policy. Other policies can also be used. At any 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) of sizes $I_k$, $k$ =1,…,$L$. 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 $\Delta$. $W$ and $\Delta$ 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 $r_{DC}$. The DC site may already contain some datasets so that no transferring is needed for them. A piece of data $I_k$ transmitted over a path ($r_i$, $r_{DC}$) experiences total communication queuing delay $Q_{i,DC}^{Comm}$, because of other pieces of data utilizing the links of the path. The propagation delay of this path is denoted by $d_{i,DC}$ and its usable capacity by $C_{i,DC}$ (maximum capacity available at intermediate links). In general the type of transport media used (opaque packet switching, transparent networks such as optical WDM network, OBS etc), determines whether the queuing delay is counted once (transparent networks) or is present at every intermediate site (opaque networks). For the rest of the paper we will denote by $Q_{i,DC}^{Comm}$ the total communication queuing delay irrespective of the underlying network. Finally, a task before executed at the DC site experiences a processing queuing delay $Q_{DC}^{Proc}$, because of other tasks utilizing the resource's computational capacity.

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 transfering messages between the user and the scheduler and between the scheduler and the resources, since 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).

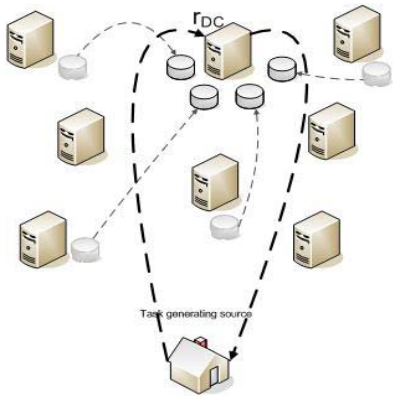

Figure 1: A Data Consolidation senario

A task created by user located at a 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, the size of data needed, the sites that hold the corresponding data etc. The scheduler based on the used Data Consolidation algorithm (Section 4.2), selects (i) the sites that hold replicas of the datasets the task needs and (ii) the site where these datasets will consolidate and the task will be executed. The decisions concerning (i) and (ii) can be made jointly or separately. Note that the capacity of the storage resource $r_{DC}$ must be larger than the total size of the datasets that will consolidate:

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

Next, the scheduler orders the data holding sites to transfer the datasets to the DC site. The scheduler, also, orders the user to transfer his task to the DC site (Figure 1). The tasks execution in the DC site starts only when the task and all of its needed datasets have arrived at the site. After the task finishes execution, the results return back to the originating user.

## 4. Data Consolidation

Data Consolidation (DC) is performed when a task needs for its execution two or more pieces of data. In this case the scheduler must select: (i) the sites that hold replicas of the datasets the task needs and (ii) the site where the data will consolidate and the task will be executed. The policies for selecting the data holding sites and the DC site, compromise the DC problem. In what follows, we propose and experimentally evaluate a number of DC techniques.

### 4.1 Theoretical Analysis

We assume that the scheduler has selected the data holding sites (replicas), $r_k \in R$, for all datasets $I_k$, $k$ =1,…,$L$, and the DC site $r_{DC}$. Note that the DC site may already have some pieces of data and thus for these pieces no transferring is required (i.e., $r_k$=$r_{DC}$ 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:

$$D_{comm} = D_{cons} + D_{output} =$$

$$\max_{k=1\ldots L}\left(\frac{I_k}{C_{k,DC}} + Q_{k,DC}^{Comm} + d_{k,DC}\right) + \left(\frac{\Delta}{C_{DC,u}} + Q_{DC,u}^{Comm} + d_{DC,u}\right)$$

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:

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

The total delay suffered by a task is:

$$D_{DC} = D_{comm} + D_{proc} .$$

Note that $Q_{k,DC}^{Comm}$ and $Q_{DC}^{Proc}$ are difficult to be estimated since the former depends on the utilization of the network and the latter depends on the utilization of the computation resource.

### 4.2 Proposed Techniques

As stated before the DC problem consists of two sub-problems: (i) find the repository sites $r_k$ from which the dataset $I_k$, $k$=1,2,…,$L$, will be transferred to DC site and (ii) find the DC site $r_{DC}$ where the task will be executed. In

general, DC techniques will 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 pieces of data, the bandwidth availability and the expected latency, the user and application behaviors, the price a user is willing to pay for using storage and computation resources, etc.

In this work we propose a number of categories of DC algorithms and describe for each category some basic and easy to implement algorithms.

- **Random:**
  - Random-Random (Rand) algorithm: The data replicas used by the task and the DC site are randomly chosen.
  - Random-Origin (RandOrig) algorithm: The data replicas used by the task are randomly chosen and the DC site is the one that created the task.
- **Time:**
  - Consolidation-Cost (ConsCost) algorithm: We select the replicas and the Data Consolidation site that minimize the data consolidation time ($D_{cons}$).

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

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

    where $R$ is the set of all resources and $d_{i,j}$ the propagation delay between site $r_i$ and $r_j$. The data consolidation time ($D_{cons}$) of candidate DC site $r_j$, is given by:

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

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

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

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

$$r_{DC} = \arg\min_{r_i \in R} \left( Q_i^{Proc} + \frac{W}{P_i} \right),$$

    while the data replicas are randomly chosen.

    Although we cannot always calculate the processing delay $Q_i^{Proc}$ of a resource $r_i$, it is possible to estimate it based on the tasks already assigned to it or based on the average delay the tasks executed on it have experienced, etc. Moreover, if the computation workload $W$ of a task is not a-priori known, we can simply choose the resource with the largest computation capacity $P_i$.
  - Total-Cost (TotalCost) algorithm: 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, the task's execution time, and the time needed for the output data to be transferred to the task's originating user. This algorithm is the combination of the two above algorithms, and is similar to the Cost based Job Scheduling algorithm presented in [15].
- **Traffic:**
  - Smallest-Data Transfer (SmallTrans) algorithm: We select the DC site for which the smallest number of datasets (or the datasets with the smallest total size) need to be consolidated for the task's execution.

# 5. Simulation

We implemented a Data Grid Network in the Network Simulator (ns-2) [1]. Ns-2 provides a manageable environment for simulating the network resources of the Grid, which is important for the evaluation of DC techniques. We evaluated the performance of the Random-Random (Rand), Consolidation-Cost (ConsCost), Execution-Cost (ExecCost) and the Total-Cost (TotalCost) algorithms.

## 5.1 Simulation Environment

In our simulations we used the NSFNET topology [21], which consists of 14 nodes and 21 links. All link capacities are equal to 1Gbps. We assume a P2P (opaque) network; the delay for transmitting between two sites includes the propagation delays of the links, and the 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. Note that although in the simulation experiments the communication queuing delay $Q_{i,j}^{Comm}$ exists, it is not taken into account in the ConsCost and TotalCost algorithms, since it is hard to estimate.

We assume that only a number of nodes are equipped with a computational and a storage resource (such nodes are called *sites*), while the other nodes act as simple routers. In our simulations we used 5 sites of equal storage and computational capacities. In future work, we plan to use more sites and evaluate the scalability of the proposed algorithms. Also, a node exists in the network, which acts as a Tier 0 site. The Tier 0 site holds all the datasets but does not have any computational capability.

Initially, 50 datasets exist in the network and the size of each dataset is given by an exponential distribution with average $I$. At the beginning two copies of each dataset exist; the first is distributed among the 5 sites and the second is placed at Tier 0 site. The storage capacity of each storage resource is 50% of the total size of all the datasets. Thus, on average a site can hold 25 datasets. When a storage resource does not have the necessary capacity to store a needed dataset, then a randomly chosen, unused dataset is deleted. It is possible that more than one unused datasets are deleted until the new dataset can be stored in the resource.

In each experiment, users generate a total of 50.000 tasks, with exponential interarrival times of average value $1/\lambda$. Unless explicitly stated, we assume that $1/\lambda$=0.01 sec. In all our experiments we keep constant the average total data size $S$ that the tasks require:

$$S = L \cdot I, \qquad (1)$$

where $L$ is the number of datasets a task requests and $I$ is the average size of each dataset. More specifically, in our experiments we use average total data size $S$ equal to 15000 MB and we examine the following ($L$, $I$) pair values: (2, 7500), (3, 5000), (4, 3750), (6, 2500), (8, 1875), (10, 1500).

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

$$W = a \cdot S. \qquad (2)$$

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. (2) to calculate the workload of this task. The parameter $a$ defines a tradeoff between computation

and data-intensive tasks. As *a* increases the tasks become more cpu-intensive, while as *a* decreases the tasks have less computation demands. We alter the parameter *a* (0.01, 0.1, 0.5, 1, 2, 4, 8, 11) and examine how our DC strategies behave. Unless explicitly stated, in our experiments we create data-intensive tasks by setting $a = 0.01$. Also, when a task completes its execution we assume that there is no output data returned to the originating user.

Each experimental scenario was run 5 times, using an independent random seed. In every repetition, the placement in the network of the 5 sites and the Tier 0 was random. Finally, to account for the transient regime at the beginning and at the end of simulations, measurements start when we have reached a steady load and stop when the last task has been submitted.

### 5.2 Simulation Metrics

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

- The average task delay. A task's delay is defined as the time that elapses between its creation and the time its execution is completed at a site.
- The average load per task imposed in the network. This network load depends on the size of datasets transferred and on the number of hops these datasets traverse.
- The Data Consolidation (DC) probability. The probability that the selected DC site will not have all the datasets required by a task and as a results DC will be necessary.

The first metric characterizes the efficiency of the DC strategy with respect to the execution of a single task, while the second expresses the overhead the DC strategy induces to the network. The third 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

Figure 2, 3 and 4 present results when tasks request different number of datasets *L* for their execution. In these experiments the average total data size per task is $S = 15000$ MB, and *L* and *I* take values as described previously.

Figure 2 shows the DC probability for the various DC algorithms examined, that is, the probability that the datasets a task requests are not found in the chosen DC site. The higher the number *L* of datasets a task requests, the higher is the probability (for all algorithms) that these datasets will not be located at the DC site, given that the size of datasets a site can hold is limited. The ConsCost and TotalCost algorithms exhibit smaller DC probability than the Rand and ExecCost algorithms. This is because the ConsCost and TotalCost algorithms select a 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 at random or almost at random (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.

Figure 3 shows the average task delay for the various 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. This is because the probability that a DC site will not hold all the data a task needs (i.e., the DC probability) increases as the number of datasets a task requires increases (Figure 2), resulting in more data transfer, and an increase in the average task delay. On the contrary, for the Rand and ExecCost algorithms the average task delay decreases. This happens because the size of the transferred datasets *I* decrease as *L* increases (Eq. (1)). Thus, for the Rand and ExecCost algorithms that (almost) randomly select the DC site, as *L* increases the overhead of the data consolidation time and its impact on the average task delay decreases.

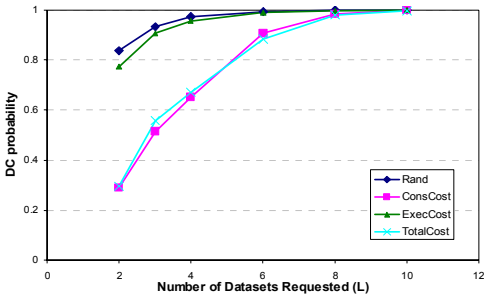


Figure 2: The DC probability for the proposed DC algorithms, when tasks request different number of datasets, *L*, for their execution. The average total data size per task is *S*=15000 MB.

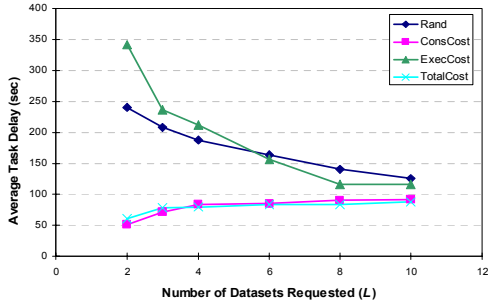


Figure 3: The average task delay (sec) for the proposed DC algorithms, when tasks requests different number of datasets, *L*, for their execution. The average total data size per task is *S*=15000 MB.

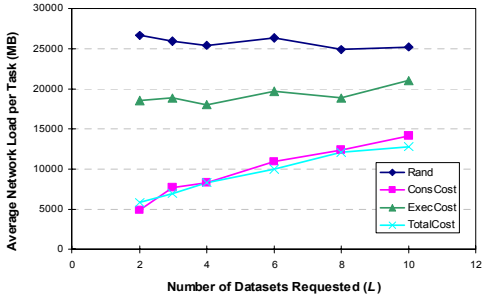


Figure 4: The average network load per task (MBytes) for the proposed DC algorithms, when tasks request different number of datasets, *L*, for their execution. The average total data size per task is *S*=15000 MB.

Figure 4 shows the average network load per task (in Mbytes) 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, largest load to the network than the algorithms that do take the data consolidation delay 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 $I$ 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 $I$. The superior performance of ExecCost over that of Rand is because ExecCost assigns task to resources in a more uniform way, based on the task execution times. On the other hand, the algorithms that take into account the data consolidation delay (namely, the ConsCost and TotalCost algorithm), induce a smaller load in the network. This load increases as the number of datasets $L$ increase. This can be explained by the increasing probability that a DC site will not hold all the data a task needs (Figure 2), and thus having to transfer more datasets as $L$ increases.

Figure 5 and 6 illustrate the average delay and the average network load per task for the proposed DC algorithms, when tasks become more cpu- than data-intensive. In order to examine this effect we increased the parameter $a$. We observe that the TotalCost algorithm behaves better in all cases. When tasks are data-intensive, it achieves small task delay and network load and behaves similar to the ConsCost algorithm. As tasks become more cpu-intensive 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.

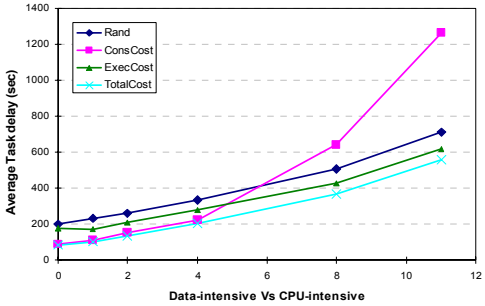


Figure 5: The average task delay (sec) for the proposed DC algorithms, when tasks become more cpu- than data-intensive. The average total data size per task is $S$=15000 MB.

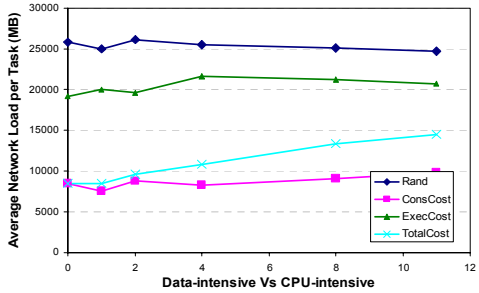


Figure 6: The average network load per task (Mbytes) for the proposed DC algorithms, when tasks become more cpu than data-intensive. The average total data size per task is $S$=15000 MB.

## 6. Conclusions

In this work we examined Data Consolidation (DC) in Grid Networks, which is a task scheduling and data migration technique that is performed when a task needs for its execution two or more pieces of data, possibly scattered throughout the Grid Network. We proposed a number of DC policies that consider data consolidation or/and task execution delay. We showed that if DC is performed efficiently, important benefits can be obtained in terms of task delay, network load and other performance parameters of interest.

### Acknowledgment

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

## References

[1] T. Braun et al, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, J. of Parallel and Distributed Computing, Vol. 61, No. 6, pp. 810-837, 2001.

[2] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, HPDC, USA, 2002.

[3] Y. Cardinale, H. Casanova, An evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms, HPC&S, 2006.

[4] R. Buyya, M. Murshed, D. Abramson, S. Venugopal, Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm, Intl J. of Software: Practice and Experience (SPE), Vol. 35, No. 5, pp. 491-512, 2005.

[5] K. H. Kim, R. Buyya, Fair Resource Sharing in Hierarchical Virtual Organizations for Global Grids, Intl Conf. on Grid Computing, 2007.

[6] N. Doulamis, E. Varvarigos, T. Varvarigou, Fair Scheduling Algorithms in Grids, Tran. on Parallel and Distributed Systems, Vol. 18, No. 11, pp. 1630-1648, 2007.

[7] G. Shao, R. Wolski, F. Berman, Modeling the Cost of Redistribution in Scheduling, SIAM Conference on Parallel Processing for Scientific Computing, 1997.

[8] S. Frechette, D. R. Avresky, Method for Task Migration in Grid Environments, Intl Symposium on Network Computing and Applications.

[9] R. M. Rahman, K. Barker, R. Barker, A Predictive Technique for Replica Selection in Grid Environment, Intl Symposium on Cluster Computing and the Grid , pp 163-170, 2007.

[10] K. Ranganathan, I. Foster, Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications, Intl High Performance Distributed Computing Sumposium, pp 352-358, 2002.

[11] H. Shan, L. Oliker, W. Smith, R. Biswas, Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration, Intl Conference on Advanced Computing and Communication, 2004.

[12] A. Elghirani, R. Subrata, A. Zomaya, Intelligent Scheduling and Replication in Datagrids: a Synergistic Approach, Intl Symposium on Cluster Computing and the Grid , pp 179-182, 2007.

[13] W.H Bell , D.G Cameron, L. Capozza, A. P. Millar, K. Stockinger, F. Zini, Simulation of Dynamic Grid Replication Strategies, OptorSim, LNCS, Vol 2536 , pp 46-57, 2002.

[14] Vazhkudai, S., Tuecke, S., and Foster, I., Replica Selection in the Globus Data Grid, Intl Symp. on Cluster Computing and the Grid, 2001.

[15] A Chakrabarti, R. Dheepak, S Sengupta, Integration of Scheduling and Replication in Data Grids, LNCS, Vol 3296 , pp 375-385, 2004.

[16] K. Ranganathan, I. Foster, Identifying Dynamic Replication Strategies for a High – Performance Data Grid, LNCS Vol 2242, pp 75-86, 2001.

[17] K. Ranganathan, I. Foster, Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids, J. of Grid Computing Vol 1, pp 53-62, 2001.

[18] Ns - network simulator, http://www.isi.edu/nsnam/.

[19] D. S. Katz at al, Astronomical Image Mosaicking on a Grid: Initial Experiences, Engineering the Grid - Status and Perspective, (Editors: B. Di Martino, J. Dongarra, A. Hoisie, L. Yang, and H. Zima), American Scientific Publishers, 2006.

[20] W. Hoschek, F. Jaén-Martínez, A. Samar, H. Stockinger, and K. Stockinger, Data Management in an International Data Grid Project, Intl Workshop on Grid Computing, 2000.

[21] NSFNET: http://www.nsfnet-legacy.org/