6

Data Consolidation and Information Aggregation in Grid Networks

Panagiotis Kokkinos and Emmanouel Varvarigos
University of Patras, Department of Computer Engineering and Informatics Research Academic Computer Technology Institute
Greece

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; among them data-intensive applications that perform computations on large sized datasets, stored at geographically distributed resources. In this context, we identify two important issues: i) data consolidation that relates to the handling of these data-intensive applications and ii) information aggregation, which relates to the summarization of resource information and the provision of information confidentiality among the different administrative domains.

Data consolidation (DC) applies to data-intensive applications that need more than one pieces of data to be transferred to an appropriate site, before the application can start its execution at that site. It is true, though, that an application/task may not need all the datasets at the time it starts executing, but, it is usually beneficial both for the network and for the application to perform the datasets transfers concurrently and before the task's execution. The DC problem consists of three interrelated sub-problems: (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 him) 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 sub-problems either separately or jointly. Moreover, the order in which these sub-problems are handled may be different, while the performance optimization criteria used may also vary. The algorithms or policies for solving these sub-problems compromise a DC scheme. We will present a number of DC schemes. Some consider only the computational or only the communication requirements of the tasks, while others consider both kinds of requirements. We will also describe DC schemes, which are based on Minimum Spanning Trees (MST) that route concurrently the datasets so as to reduce the congestion that may appear in the future, due to these transfers. Our results brace our belief that DC is an important problem that

needs to be addressed in the design of Grids networks, and can lead, if performed efficiently, to significant benefits in terms of task delay, network load and other performance parameters.

Information aggregation relates to the summarization of resource information collected in a Grid Network and provided to the resource manager in order for it to make scheduling decisions. Resource-related information size and dynamicity grows rapidly with the size of the Grid, making the aggregation and use of this massive amount of information a challenge for the resource management system. In addition, as computation and storage tasks are conducted increasingly non-locally and with finer degrees of granularity, the flow of information among different systems and across multiple domains will increase. Information aggregation techniques are important in order to reduce the amount of information exchanged and the frequency of these exchanges, while at the same time maximizing its value to the Grid resource manager or to any other desired consumer of the information. An additional motivation for performing information aggregation is confidentiality and interoperability, in the sense that as more resources or domains of resources participate in the Grid, it is often desirable to keep sensitive and detailed resource information private, while resources are still being publicly available for use. For example, it may soon become necessary for the interoperability of the various cloud computing services (e.g., Amazon EC2 and S3, Microsoft Azure) that the large quantity of resource-related information is efficiently abstracted, before it is provided to the task scheduler. In this way, the task scheduler will be able to use efficiently and transparently the resources, without requiring services to publish in detail their resources characteristics. In any case, the key to information aggregation is the degree to which the summarized information helps the scheduler make efficient use of the resources, while coping with the dynamics of the Grid and the varying requirements of the users. We will describe a number of information aggregation techniques, including single point and intra-domain aggregation and we will define appropriate grid-specific domination relations and operators for aggregating static and dynamic resource information. We measure the quality of an aggregation scheme both by its effects on the efficiency of the scheduler's decisions and also by the reduction it brings on the total of resource information. Our simulation experiments demonstrate that the proposed schemes achieve significant information reduction, either in the amount of information exchanged, or in the frequency of the updates, while at the same time maintaining most of the value of the original information.

The remainder of the paper is organized as follows. In Section 2 we report on previous work. In Section 3 we formulate and analyze the Data Consolidation (DC) problem, proposing a number of DC schemes. In Section 4 we formulate the information aggregation problem and propose several information aggregation techniques. In Section 5 we present the simulation environment and the performance results obtained for the proposed schemes and techniques. 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 (Krauter et al., 2002). A usual data management operation in grids is data migration, that is, the movement of data between resources. The effects of data

migration in grids have been considered in (Shan et al., 2004). The most common data migration technique is data replication (Rahman et al., 2007) (Rahman et al., 2008) (Dogan, 2009), 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, security, etc (Vazhkudai et al., 2001). Furthermore, the problem of parallel downloading different parts of a dataset from various replica holding resources, as a mean to decrease the download time of that dataset, has been investigated for peer-to-peer networks and the Internet (Byers et al., 1999) (Rodriguez et al., 2002), and also for grid networks (Chang et al., 2008). A number of works consider both task scheduling and data replication issues. The authors in (Ranganathan et al., 2002) suggest that it is better to perform data replication and task scheduling separately, instead of trying to jointly optimize these two operations. In (Elghirani et al., 2007) 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 execution time and system utilization metrics. The authors in (Bell et al., 2002) (Bell et al., 2003) (Cameron et al., 2004) present the OptorSim simulator and jointly consider task scheduling and data replication for the case where a task requests sequentially a number of datasets. 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 (Chakrabarti et al., 2004) propose the Integrated Replication and Scheduling Strategy (IRS) scheduler that combines scheduling and replication strategies. The effective use of the communication/network resources is an important consideration, especially in data grid networks. In (Stevens et al., 2008) 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 (Ranganathan et al., 2002) (Stevens et al., 2008), or that it requires sequentially a number of datasets (Rahman et al., 2007)(Rahman et al., 2008) (Bell et al., 2002)(Bell et al., 2003) (Cameron et al., 2004). 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).

Information aggregation has been previously studied mainly in the context of hierarchical data networks (Lee, 1995), where it is performed on network-related parameters in order to

facilitate hierarchical routing. Hierarchical routing is a major issue for data networks, and is important for reducing the memory requirements at the routers (border nodes) for the very large topologies encountered in Internet's infrastructure. A topology is broken down into several layers of hierarchy, thus downsizing the routing tables required, but this comes at the expense of an increase in the average path length. (Kleinrock, Kamoun, 1977) is one of the first works investigating hierarchical routing, where clustering structures are introduced to minimize the routing tables required. Bounds are also derived on the maximum increase in the path length for a given table size. An issue that is central to hierarchical routing is topology information aggregation (Lee, 1995) (Mieghem, 1999). Aggregation techniques in hierarchical topologies try to summarize and compress the topology information advertised at higher levels. In order to perform routing and network resource allocation efficiently, the aggregated information should adequately represent the topology and the characteristics/metrics of the network. Delay and bandwidth are two network-related metrics that are usually aggregated along with the topology. In (Lee, 1995) a number of topology aggregation techniques are presented. An important issue in topology aggregation is the choice of the parameters that are aggregated along with the topology. In (Mieghem, 1999) the parameters that can be aggregated, are distinguished into three classes: additive, min-max and the combination of additive and min-max metrics (multi-criteria). Topology aggregation based on such metrics is investigated in (Bauer et al., 2006), considering only network-related parameters (namely, delay and bandwidth). In the same context (Bauer et al., 2006) presents a topology aggregation scheme that is subject to multi-criteria (delay and bandwidth) constraints. Specifically, a transition matrix is constructed containing the minimum information that describes exactly the traversing characteristics of a domain, that is, the characteristics of the paths connecting border (ingress-egress) node pairs of the domain.

Resource information aggregation in grids has not been studied in detail, despite its practical importance and its impact on the efficiency of the scheduling decisions. Actually, most scheduling algorithms proposed (Krauter et al., 2002) make their decisions using exact resource information. The idea of aggregation has also appeared in sensor networks (Intanagonwiwat et al., 2003), where it is often called data fusion, as a way to reduce the amount of data transferred towards the sink node. Relatively recently, information aggregation appeared as an interesting topic in Peer-to-Peer (P2P) (Renesse et al., 2003) and P2P grid systems (Schulz et al., 2009) (Zhou, Hwang, 2006) (Cai, Hwang, 2007). These works focus on the architecture and on the mechanisms of the system performing the aggregation; on the other hand, in our work we assume that such a mechanism/system is in place and examine the aggregation operations that should be performed for different parameters, the aggregation policies that should be applied and the effects they have on scheduling efficiency. Also, our work applies to all kinds of grids and not specifically to Desktop grids. In addition, resource information aggregation and task scheduling issues have been investigated in a few works, which focus, however, on particular systems and under specific assumptions (Czajkowski et al., 2001) (Muraki et al., 2006) (Rodero et al., 2010). In the Monitoring and Discovery System 2 (MDS2) (Czajkowski et al., 2001) (Muraki et al., 2006) resource management system, information from individual information providers (e.g., resources) is aggregated into collections, where each collection may correspond to a different virtual organization (VO). (Rodero et al., 2010) is a quite recent work where aggregated resource information and scheduling issues are considered. It proposes two aggregation policies (simple and categorized) and two scheduling policies that use the aggregated information. In our work we are interested more in the parameters aggregated,

in the operators used and in the policies applied to summarize them, while we use a simple scheduling policy to evaluate the effects of the aggregation on the scheduling efficiency. We also investigate the tradeoffs between the amount of information exchanged, the frequency of updates required, and the quality of the aggregated information. Most importantly, our work is quite more general, and attempts to fill the gap between the topology information aggregation works presented in (Lee, 1995) (Mieghem, 1999) and grid computing.

3. Data consolidation

3.1 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 $P_{i,j,}$, 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 C_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 (CE) of a gLite (Laure et al., 2006) 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,…,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 that 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 $d_{i,DC}$ and its usable capacity by $P_{i,DC}$ (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 $Q_{i,DC}^{Comm}$, 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 wavelength routed optical WDM network or OBS, etc), determines whether the queuing delay is counted once at the source (transparent networks) or is accumulated over all intermediate nodes (opaque networks). Finally, a task before starting execution at the DC site experiences a processing queuing delay Q_{DC}^{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, the sites that hold the corresponding datasets etc. The scheduler based on the used Data Consolidation scheme, 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), (ii) and (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:

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

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. 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 (Dogan, 2009), 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.

3.2 Data Consolidation (DC) schemes

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

3.2.1 Data Consolidation (DC) delays

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 no transferring is required for these pieces (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:

$$D_{comm} = D_{cons} + D_{output} = \max_{k=1...L}\left(\frac{V_{I_k}}{P_{k,DC}} + Q_{k,DC}^{Comm} + d_{k,DC}\right) + \left(\frac{\Delta}{P_{DC,u}} + Q_{DC,u}^{Comm} + d_{DC,u}\right) \quad (2)$$

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:

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

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 this information is known, while others do not make this assumption, and we compare their performance.

3.2.2 Proposed schemes

As stated before the DC problem consists of three sub-problems: (i) the selection of the repository sites r_k from which the dataset I_k, k=1,2,…,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, the price a user is willing to pay for using the storage and computation resources, etc.

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 (Bell et al., 2002) (Bell et al., 2003) (Cameron et al., 2004) that use however 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 to 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:

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

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 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:

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

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

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

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:

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

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 tasks' 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 tasks' workload, there are algorithms that can be used to provide such estimates. 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 C_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.

3.2.3 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. 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 sub-optimal 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.

4. Information aggregation

4.1 Problem formulation

Our aim is to design efficient and flexible aggregation schemes that can be applied to grids without a significant penalty on the quality of the scheduling (and probably other) decisions taken with the aggregated information. We formulate our problem in a generic way, without assuming grid networks with specific characteristics. The scheduling policies assumed are also relatively simple, since combining our aggregation schemes with a large set of scheduling policies would obscure the focus on the aggregation issues.

We consider a grid consisting of N sites, partitioned according to a hierarchical structure in a total of L domains D_j, j=1,2,…,L. Each site i, i=1,2,…,N, has computational and storage capacity C_i and S_i , respectively, and belongs to one of the L domains. Site i publishes its resource information as a vector V_i that may contain various parameters:

$$V_i = (C_i, S_i, \ldots).$$

These vectors are collected per domain D_j and are published to a higher level of the hierarchy, in the form of an *information matrix* whose rows are the resource site vectors:

$$M_j = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_{|D_j|} \end{bmatrix} = \begin{bmatrix} (C_1, S_1, \ldots) \\ (C_2, S_2, \ldots) \\ \vdots \\ (C_{|D_j|}, S_{|D_j|}, \ldots) \end{bmatrix},$$

where $|\cdot|$ denotes the cardinality of a set and 1,2, …, $|D_j|$ are the sites contained in domain D_j. By performing appropriate operations, to be discussed later, on the information vectors contained in the information matrix, M_j is transformed into the *aggregated information matrix* $\hat{M}_j$.

The grid scheduling problem is usually viewed as a two-level hierarchical problem. At the first level, called *meta-scheduling*, a meta-scheduler allocates tasks to sites, while at the second level, called *local scheduling*, each site schedules the tasks assigned to it on its local computing elements. This is the approach followed by many grid middleware systems, including gLite (Laure et al., 2006), where the Workload Management System (WMS) selects the site where a task will be executed and the local scheduler chooses the Working Node (WN) it will be executed on after reaching that site. Similarly, in our work scheduling is performed at two levels. At the higher level a central scheduler decides the domain D_j a task will be assigned to, and at the lower level a domain scheduler DS_j, decides the exact site in the domain where the task will be executed (Figure 1). Information collection and aggregation is performed, similarly, by a two level monitoring system, consisting of a central monitor *CM* and the domain monitors DM_j, j=1,2,…,*L*.

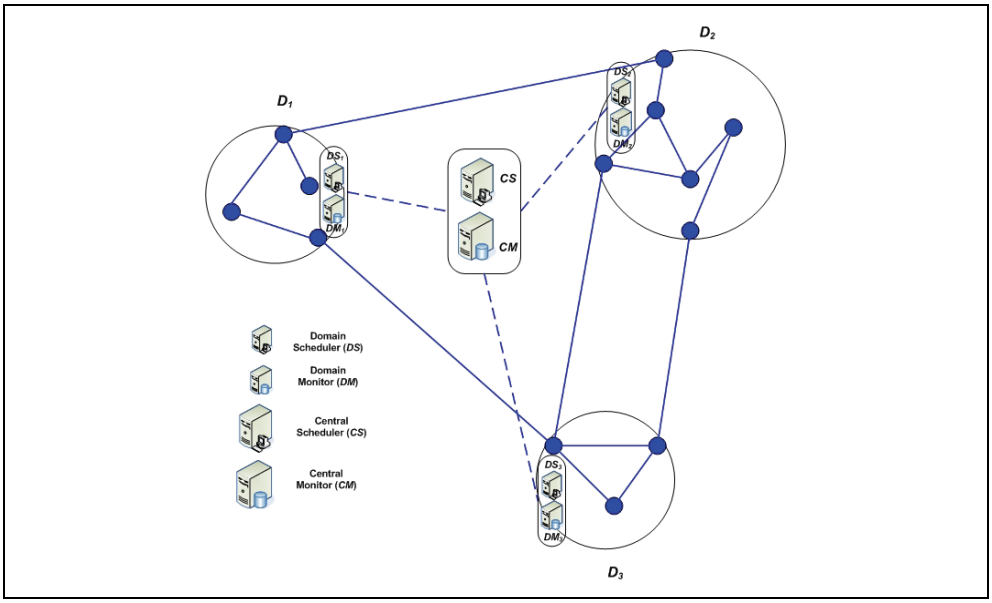

Fig. 1. A two-level hierarchical scheduling and monitoring system. Each domain j has a domain scheduler DS_j and a domain monitor DM_j, while there is also a central scheduler *CS* and a central monitor *CM*.

A user located at some site generates tasks T_m, m=1,2,…, with computational workload W_m, that have to be scheduled and then executed at a resource site.

4.2 Information aggregation

In this section we present the resource information parameters of interest, the operators applied and the proposed aggregation techniques.

4.2.1 Operational framework

In Algorithm 2 we present, in pseudocode, the sequence of operations performed by the information collection mechanism and the proposed aggregation scheme.

Algorithm 2 Resource Information Collection & Aggregation

1. Each site i, i=1,2,…,N, belonging to some domain D_j periodically or reactively (driven by information changes) publishes its information vector V_i to the domain monitor DM_j.
2. Each domain monitor DM_j, j=1,2,…,L, puts together these vectors to form the information matrix M_j.
3. Each domain monitor DM_j, j=1,2,…,L, periodically or reactively (when information changes) computes its aggregated information matrix $\hat{M}_j$ and publishes it to the central monitor CM.
4. The CM collects the aggregated information matrices.

In Algorithm 3 we present the operations performed by a task scheduling scheme that uses the aggregated information.

Algorithm 3 Task Scheduling

1. Upon the arrival of a task T_m, the central scheduler CS looks at the domain matrices provided by the central monitor CM.
2. The central scheduler CS applies an optimization function to the vectors contained in the domain matrices and selects the information vector V that produces the largest value.
3. The CS assigns the task T_m to the domain D_j, where the vector V originated from, and forwards the task to the domain scheduler DS_j.
4. The domain scheduler DS_j receives the task request and selects the exact site the task will be scheduled on, using exact resource information.

Generally, as the number of sites in a domain D_j increases, the amount of information collected by the domain monitors DM_j also increases. Therefore, it is necessary for the information contained in each domain information matrix M_j to be aggregated/ summarized. This is done by performing appropriate associative operations (addition, maximization, etc) on the parameters of the sites' information vectors, in order to transform the information matrix M_j into the *aggregated information matrix* $\hat{M}_j$ which contains a smaller

number of vectors than the original matrix. The operators used for summarizing the information depends on the types of the parameters involved. In what follows, we elaborate on the resource parameters of interest, and the associative operators and aggregation techniques proposed. We also present optimization functions that can be applied by the *CS* to select the optimal information vector.

4.2.2 Information parameters and aggregation operators

The resource information parameters (both static and dynamic) of interest in this work, the operators used for their aggregation and the benefits we get for different choices of the operators are discussed next:

- The computational capacities C_i of the sites, measured in Millions Instructions per Second (MIPS), in a domain D_j can be aggregated by performing a minimum representative operation, an additive operation or by averaging them:

$$\hat{C}_j = \min_{i \in D_j} C_i , \quad \hat{C}_j = \sum_{i \in D_j} C_i \quad \text{or} \quad \hat{C}_j = avg_{i \in D_j} C_i .$$

  Using the minimum representative operator we obtain the minimum capacity of any site in the domain D_j, which would be useful for conservative task scheduling. Using the additive operator we obtain the total computational capacity in the domain, which would be useful for scheduling when a task's workload is divisible, and can be assigned to different resources simultaneously. The minimization, the additive and the average operators (and possibly other operators, such as maximization) could all be used so that a number of capacity related features of the domain are recorded.
- The number of tasks N_i already assigned to the sites can be aggregated over a domain D_j using an additive operation:

$$\hat{N}_j = \sum_{i \in D_j} N_i .$$

  Other operators can also be used, such as the maximum representative or the average number of tasks in the sites of the domain.
- Some tasks require for their execution a number of datasets that may or may not be stored at a given site. If a dataset k exists in site i, we set $I_{ik} = 1$; otherwise, we set $I_{ik} = 0$. This parameter is aggregated over all sites in a domain D_j using a boolean OR operator:

$$\hat{I}_{jk} = \underset{i \in D_j}{OR}\{I_{ik}\} .$$

  Thus, $\hat{I}_{jk} = 1$ means that there is at least one site in domain D_j that holds dataset k.

In any case, the list of parameters and aggregation operators defined above is only indicative of the different options that can be used by the aggregation schemes. Other parameters and operators can also be defined, depending on the needs of the applications and the scheduling algorithms used.

4.2.3 Aggregation schemes

In this subsection we present aggregation techniques for reducing the number of vectors in the information matrix M_j of a given domain D_j.

Single Point Aggregation Scheme

In the *single point aggregation scheme*, the information vectors of the sites in each domain are aggregated into a single information vector. We next show an example of the application of the single point aggregation technique, where the information matrix M_j that has $|D_j|$ rows is reduced to an aggregated information matrix $\hat{M}_j$ that has only one row:

$$M_j = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_8 \end{bmatrix} = \begin{bmatrix} (C_1, S_1, ...) \\ (C_2, S_2, ...) \\ \vdots \\ (C_8, S_8, ...) \end{bmatrix} \Rightarrow \hat{M}_j = \begin{bmatrix} \hat{V} \end{bmatrix} = \begin{bmatrix} (\hat{C}, \hat{S}, ...) \end{bmatrix}.$$

The information transferred to the higher levels is greatly reduced using this aggregation technique, but this happens at the expense of a degradation in the quality of the aggregated information and in the value it has for the resource scheduler.

Intra-Domain Clustering Aggregation Scheme

In the *intra-domain clustering aggregation technique*, each domain D_j, j=1,2,…,L, is partitioned into $h_j \le |D_j|$ intra-domain clusters. For the sites belonging to cluster l, l=1,2,…,h_j, the aggregated vector $\hat{V}_l$ is calculated and sent to domain monitor DM_j. The aggregated information matrix $\hat{M}_j$ containing the aggregated information vectors $\hat{V}_l$, l=1,2,…,h_j, of the clusters, is sent to the higher levels. Various approaches can be used for clustering the sites of a domain. In our work we assume that the sites are clustered randomly.

We next show an example of the application of the intra-domain clustering aggregation technique, where h_j =3 clusters are created in the given domain D_j.

$$M_j = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_7 \\ V_8 \end{bmatrix} = \begin{bmatrix} (C_1, S_1, ...) \\ (C_2, S_2, ...) \\ \vdots \\ (C_7, S_7, ...) \\ (C_8, S_8, ...) \end{bmatrix} \Rightarrow \hat{M}_j = \begin{bmatrix} \hat{V}_1 \\ \hat{V}_2 \\ \hat{V}_3 \end{bmatrix} = \begin{bmatrix} (\hat{C}_1, \hat{S}_1, ...) \\ (\hat{C}_2, \hat{S}_2, ...) \\ (\hat{C}_3, \hat{S}_3, ...) \end{bmatrix}.$$

The number of intra-domain clusters per domain influences the amount of information passed to higher levels and the efficiency of the scheduler's decision.

Reducing Aggregated Information using Domination Relations

In this subsection we introduce the concept of *dominated resources* to further prune the number of information vectors processed by the domain monitors or the number of aggregated information vectors processed by the central monitor. In particular, we say that the information vector V_1 *dominates* information vector V_2, if V_1 is better than V_2 with respect to all the cost parameters. The term "better" is interpreted differently based on the parameters of interest.

For example, consider the information vectors $V_1 = (C_1, S_1, FT_1)$ and $V_2 = (C_2, S_2, FT_2)$. We say that V_1 dominated V_2 if the following conditions hold:

$$C_1 > C_2,\ S_1 > S_2 \text{ and } FT_1 < FT_2$$

The domination relations can either be applied in the vectors of a domain without any further processing, leading to a standalone aggregation technique, or they can be applied along with the single point and intra-domain aggregation techniques presented earlier.

Domain Selection Cost Functions

Upon the arrival of a new task, the central scheduler *CS* selects a proper domain D_j for the execution of the task, and forwards the task request to the corresponding domain scheduler DS_j who assigns it to a specific site in that domain. The *CS* uses aggregated information collected by the central monitor *CM*, while DS_j uses exact resource information in order to make its corresponding scheduling decisions. The *CS* in order to select the appropriate domain for a task's execution (similar are the operations performed by a *DS* so as to select the appropriate site for a task's execution) applies an optimization function to the vectors $\hat{V}$ and the domain giving the optimum value is selected.

5. Performance evaluation

5.1 Data consolidation

5.1.1 Simulation settings

In our simulations we used a topology derived from the EGEE topology, which consists of 11 nodes and 16 links, of capacities equal to 10Gbps. 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.
The size of each dataset is given by an exponential distribution with average V_I (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 Tier 0 site. The storage capacity 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 λ=75 tasks/sec (but we also examine other task arrival rates: λ=50, 75, 100, 125, 150 and 200 tasks/sec). In all our experiments we keep constant the average total data size *S* that each task requires:

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

where *L* is the number of datasets a task requests and V_I is the average size of each dataset. We use average total data size *S* equal to 600 GB and examined various (L, V_I) pair values. In each experiment the total number of available datasets changes in order for their total size to remain the same: 15 TB.
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 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 results Data Consolidation will be necessary.

5.1.2 Results

Figure 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.

Figure 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 (Figure 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_I (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 decreases as L increases.

Figure 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_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 V_I. 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 and their datasets are stored until they are executed. When this resource does not have

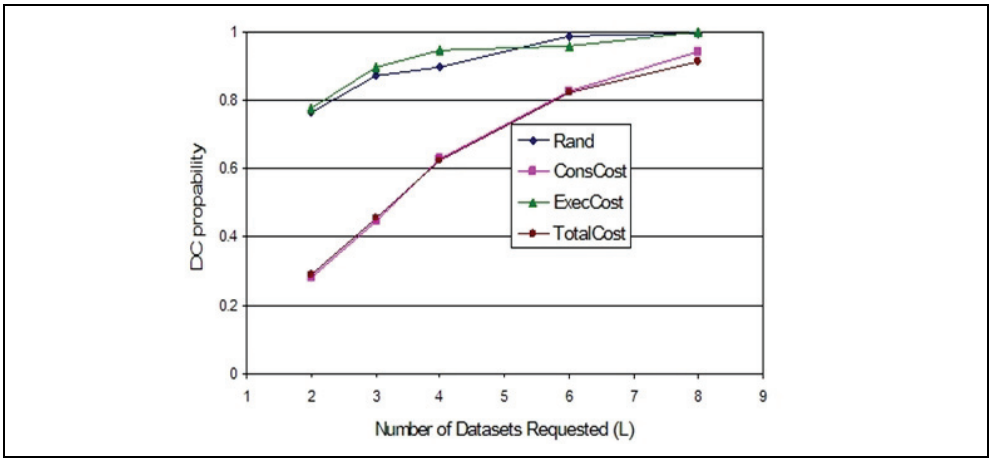

Fig. 4. The DC probability for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks request a different number of datasets L for their execution. The average total data size per task is S=600 GB.

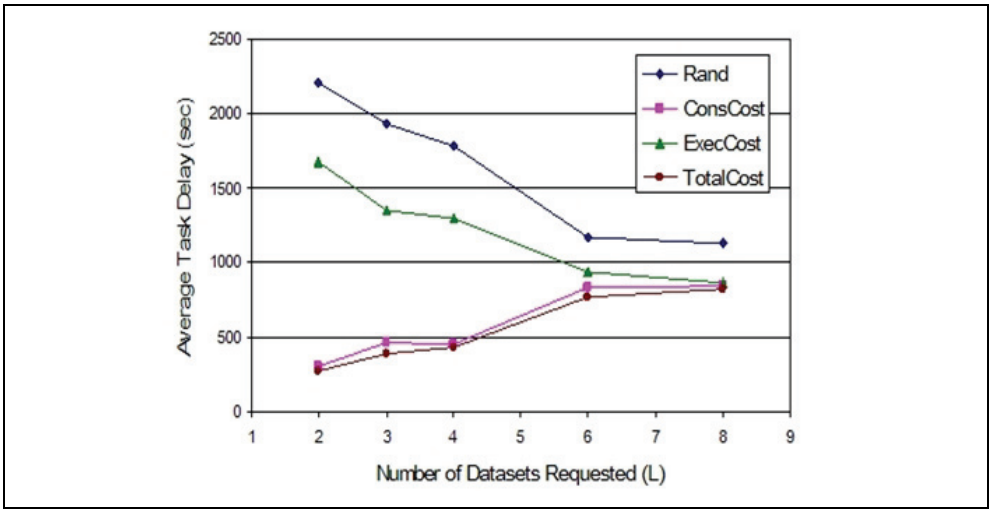

Fig. 5. The average task delay (in sec) for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks requests a different number of datasets L for their execution. The average total data size per task is S=600 GB.

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 (Figure 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).

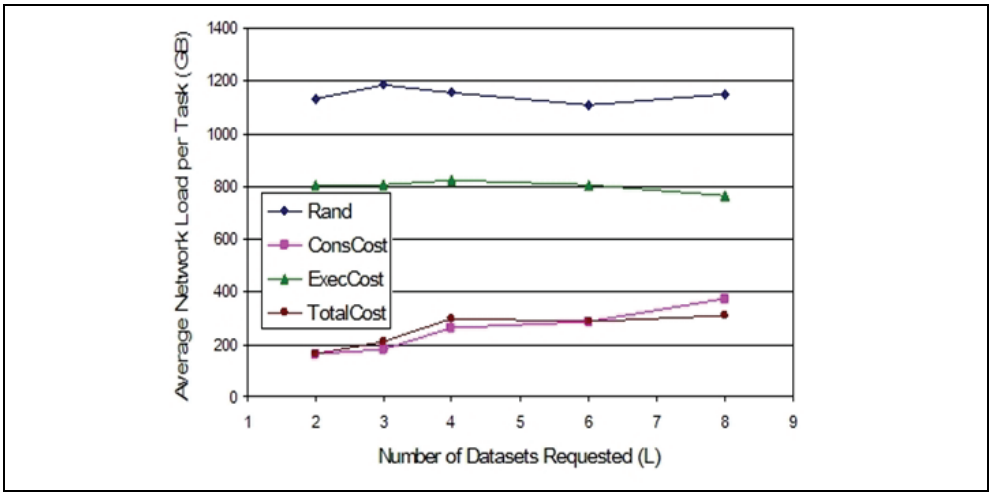

Fig. 6. The average network load per task (in GB) for the Rand, ExecCost, ConsCost and TotalCost DC algorithms, when tasks request a different number of datasets *L* for their execution. The average total data size per task is *S*=600 GB.

5.2 Information aggregation

5.2.1 Simulation settings

We consider a number of sites that are randomly grouped into domains, each having an approximately equal number of sites. Site *i* is characterized by its computational capacity C_i, measured in MIPS and chosen from a uniform distribution *UC*. In our simulations, tasks are created with exponentially distributed interarrival times with mean *I*, while their workload follows a uniform distribution *UW*. These tasks are submitted to the central scheduler that makes its decisions using either complete or aggregated resource information. In the simulation results we do not consider the effects of the information propagation delay, assuming that resource information (aggregated or not) is not outdated by the time it is used by the scheduler due, to the delay between measuring some parameter and the moment the measured value is available to the scheduler. However, we believe that in practice the aggregation operation can reduce the negative effects of this delay, since summarized information is usually less affected by the change in the values of the parameters measured. Network related issues are not considered in these simulation experiments.

The information vector V_i of site i contains its computational capacity C_i and the number of tasks N_i queued at it:

$$V_i = \{C_i, N_i\} .$$

In case no aggregation is used a new task T_m is assigned to the site i that minimizes

$$\min_i \left\{ \frac{C_i}{N_i} \right\} .$$

In case where aggregation is used then the central scheduler *CS* uses only the aggregated domain information vectors, and assigns task T_m to the domain D_j that minimizes

$$\min_j \left\{ \frac{\hat{C}_j}{\hat{N}_j} \right\} .$$

Next, the selected domain's scheduler, DS_j, receives the task and assigns it to a domain site, having complete knowledge of the information vectors of all the sites in the domain. The assignment again is performed based on the same optimization function:

$$\min_{i \in D_j} \left\{ \frac{C_i}{N_i} \right\} .$$

We implemented and evaluated the performance of the following schemes:

- *FlatCpuTasks*: In this scheme no information aggregation is performed of the sites' computational capacity and number of tasks parameters.
- *MinCpuSumTasks*: In this scheme the information vectors of the sites belonging to the same domain are aggregated (single point aggregation) using the minimum representative and the additive operators, respectively:

$$\hat{C} = \min_i C_i \text{ and } \hat{N} = \sum_i N_i .$$

- *DomMinCpuSumTasks*: This scheme is similar to the *MinCpuSumTasks*, except that domination relations are applied to the vectors of the sites of a domain, before they are aggregated using the single point aggregation scheme.
- *ICMinCpuSumTasks*: This scheme is similar to the *MinCpuSumTasks*, except that the intra-domain clustering aggregation scheme is applied, instead of the single point one, where sites are randomly clustered into intra-domain clusters.

We are interested in evaluating the degree to which the information produced by the proposed aggregation schemes leads to efficient scheduling decisions, while the size and the frequency of the information updates is kept low.

For the evaluation of the MinCpuSumTasks, DominanceMinCpuSumTasks, and ICMinCpuSumTasks aggregation schemes we use the *Stretch Factor* (*SF*), as the metric that measures the scheduling efficiency and in practice the quality of the aggregated information. The *Stretch Factor* (*SF*) is defined as the ratio of the task delay *D* when the task is scheduled using complete resource information (FlatCpuTasks) over the task delay when

an aggregation scheme is used (MinCpuSumTasks, DominanceMinCpuSumTasks, or ICMinCpuSumTasks). The task delay is defined as the time that elapses from the task's submission to the grid until the completion of its execution at a site. A stretch factor metric is also encountered in the hierarchical networks related literature (Lee, 1995), where it is defined as the ratio of the average number of hops (or average delay) between a source and a destination when flat routing is used over the corresponding value when hierarchical routing is used. Table 1 presents the *Stretch Factor* (*SF*) metrics we use in our work. In all cases $SF \leq 1$, since when a scheduler has complete resource information, it can make better decisions than when this information is aggregated. An aggregation technique is efficient when its corresponding *SF* is close to 1.

$SFMinCpuSumTasks = \frac{D_{FlatCpuTaks}}{D_{MinCpuSumTasks}}$
$SFDomMinCpuSumTasks = \frac{D_{FlatCpuTaks}}{D_{DomMinCpuSumTasks}}$
$SFICCpuTasks = \frac{D_{FlatCpuTaks}}{D_{ICMinCpuSumTasks}}$

Table 1. The *Stretch Factor* (*SF*) metrics we use in our simulation experiments.

5.2.2 Results

Figure 7a presents the *Stretch Factor (SF)* for the MinCpuSumTasks, the DomMinCpuSumTasks and the ICMinCpuSumTasks aggregation schemes when 1000 grid sites are clustered in a variable number of domains and 25000 tasks are created and scheduled. The sites' computational capacity and the tasks' workload follow uniform distributions (*UC* min/max = 10/10000 MIPS and *UW* min/max 1000/10000000 MI respectively). In general, all the stretch factor metrics measured behave similarly, that is, their value first decreases up to some point, after which it starts increasing towards 1. This is because when the number of domains is small, then the number of sites per domain is quite high, increasing the probability that more than one "best" sites or sites similar to the "best" site exist in different domains. This increases the probability that a domain with such a site will be chosen, even if aggregation is used. We represent this probability as $P_{multiple\text{-}best}$, and as our results will indicate it strongly affects the stretch factor; also, by "best" we mean the site that optimizes the metric of interest (task delay, or some other optimization function). Next, as the number of domains increase $P_{multiple\text{-}best}$ decreases and the stretch factors also decrease. After some point, as the number of domains increases and the number of sites per domain decreases, the quality of information produced by the aggregation schemes improves. This is because when there are few sites per domain, the aggregated information better represents the characteristics of its sites.

Comparing the different aggregation policies we observe that the ICMinCpuSumTasks produces the best results, followed by the DomMinCpuSumTasks and the

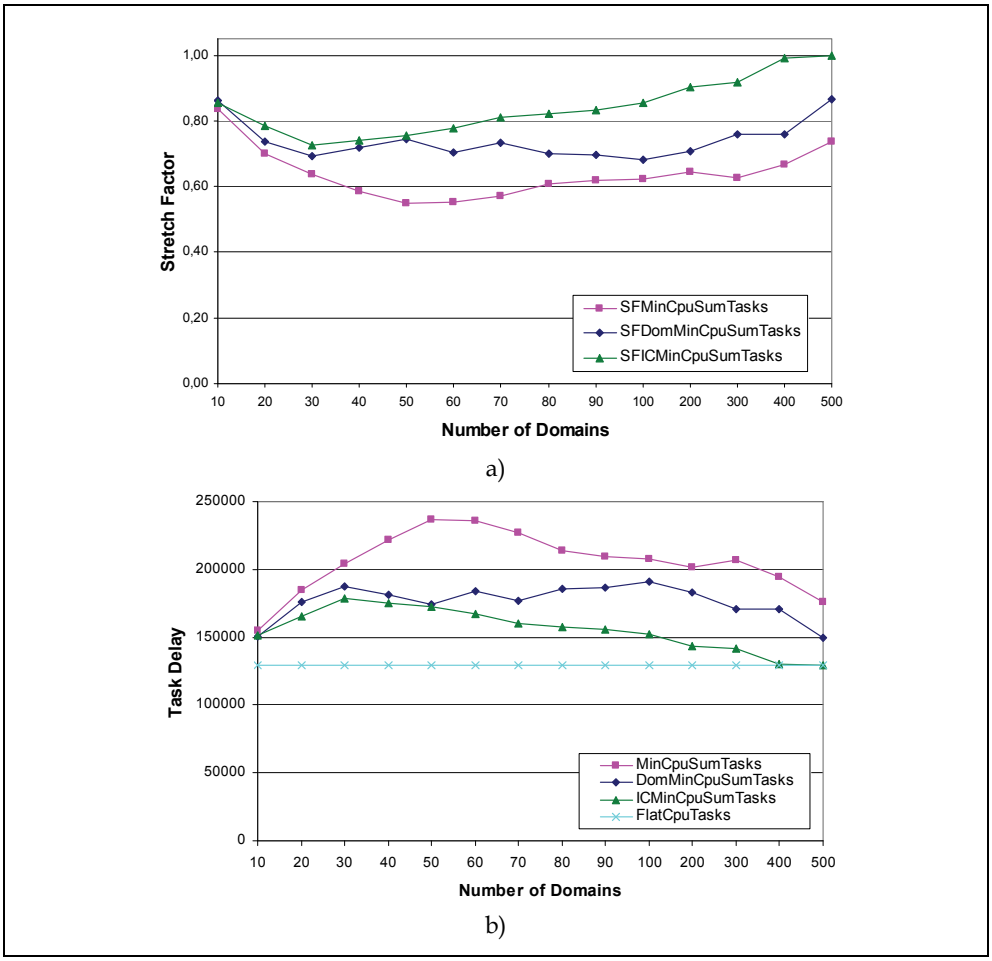

Fig. 7. a) The *Stretch Factor (SF)*, b) the average task delay (in secs) for the MinCpuSumTasks, the DomMinCpuSumTasks and the ICMinCpuSumTasks aggregation schemes, when 1000 grid sites are clustered in a variable number of domains and 25000 tasks are created and scheduled.

MinCpuSumTasks aggregation policies. The ICMinCpuSumTasks aggregation scheme use h=5 intra-clusters in each domain and its use leads to better scheduling decisions (as measured by the corresponding stretch factor), however this comes at the cost of increased number of information vectors advertised (Table 2). Reducing the number of intra-domain clusters, reduces the number of information vectors produced, but also reduces the quality of the information provided (as measured by the corresponding stretch factor). In addition, it seems that the domination operation, which discards dominated information vectors, improves the quality of the information provided to the scheduler. In Figure 7b we observe that the average task delay results, using the MinCpuSumTasks, the DomMinCpuSumTasks and the ICMinCpuSumTasks aggregation algorithms, are in accordance with the results

presented in Figure 7a. A large task delay indicates that the information produced by the corresponding aggregation scheme, leads the scheduler in wrong task scheduling decisions. We should also note that the average task delay when no information aggregation is applied (FlatCpuTasks) is not affected by the number of domains and this is the reason that it remains static.

Figure 8 illustrates the frequency with which the aggregated information vectors change. Since, our evaluated aggregation schemes consider only one dynamic parameter (that is the number of tasks scheduled in each site), this means that the information vector of a site changes only if a new task is scheduled into it. We observe that the MinCpuSumTasks and ICMinCpuSumTasks aggregation schemes result in a very large number of updates and almost equal, in most cases, to the maximum one. On the other hand using the DomMinCpuSumTasks aggregation scheme, we observe that when the number of domains is small, and as a result many sites exist in each domain, the domination operation achieves in absorbing a large percent of the changes in the sites' information vectors. As the number of domains increases and fewer sites exist in each domain, this capability declines almost linearly.

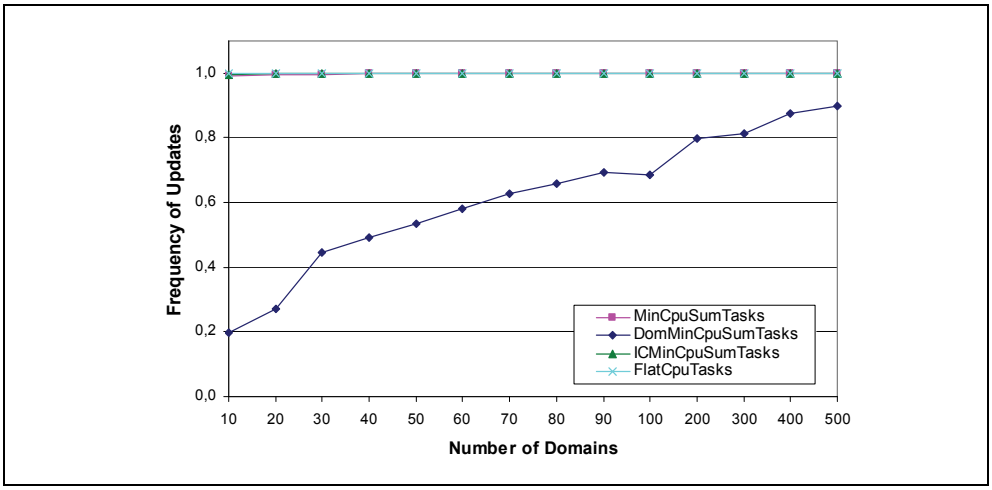

Fig. 8. The frequency of information vector updates for the FlatCpuTasks, the MinCpuSumTasks, the DomMinCpuSumTasks and the ICMinCpuSumTasks aggregation algorithms, when 1000 grid sites are clustered in a variable number of domains and 25000 tasks are created and scheduled.

We also performed several other experiments altering the number of sites, the number of tasks created or their creation pattern. In any case our results and observations were similar to the above.

6. Conclusions

In this chapter we examined the Data Consolidation (DC) problem and the application of information aggregation 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 sub-problems: (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 sub-problems can be solved jointly or independently. To the best of our knowledge this is the first time that these issues are jointly considered. We proposed a number of DC schemes, of polynomial number of required operations and 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 investigated for the first time resource information aggregation in grid networks, describing several information aggregation operators and methods. We performed a number of simulation experiments using the stretch Factor (*SF*), defined as the ratio of the task delay incurred when scheduling based on complete resource information over that incurred when an aggregation scheme is used, as the main metric for judging the extent to which the proposed aggregation schemes preserve the value of the information aggregated and assist the scheduler in making efficient decisions. We observed that in many scenarios the proposed schemes enabled efficient task scheduling decisions as indicated by the *SF* achieved, while achieving large information reduction because of the aggregation. We looked into the frequency of information vector updates resulted by the aggregation schemes, and observed that the changes in the dynamic characteristics of the resources will not always propagate to the central monitoring system, since aggregated information vectors sometimes absorb these changes.

7. References

K. Krauter, R. Buyya, M. Maheswaran, (2002), A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience, Vol. 32, No. 2, pp. 135-164, 2002.

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

R. Rahman, K. Barker, R. Alhajj, (2007), Study of Different Replica Placement and Maintenance Strategies in Data Grid, IEEE/ACM International Symposium on Cluster Computing and Grid, pp. 171-178, 2007.

R. Rahman, K. Barker, R. Alhajj, (2008), Replica Placement Strategies in Data Grid, Journal of Grid Computing, Springer, Vol. 6, No. 1, pp. 103-123, 2008.

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

J. Byers M. Luby, M. Mitzenmacher, (1999), Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads, IEEE INFOCOM, pp. 275-283, 1999.

P. Rodriguez, E. Biersack, (2002), Dynamic parallel access to replicated content in the Internet, IEEE/ACM Transactions on Networking, Vol.10, No.4, pp. 455-465, 2002.

R. Chang, M. Guo and H. Lin, (2008), A multiple parallel download scheme with server throughput and client bandwidth considerations for data grids, Future Generation Computer Systems, Vol. 24, No. 8, pp. 798-805, 2008.

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

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

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

W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger, F. Zini, (2003), OptorSim: A Grid Simulator for Studying Dynamic Data Replication Strategies, International Journal of High Performance Computing Applications, Vol. 17, No. 4, pp. 403-416, 2003.

D. Cameron, A. Millar, C. Nicholson, R. Carvajal-Schiaffino, F. Zini, K. Stockinger, (2004), Optorsim: a simulation tool for scheduling and replica optimisation in data grids, Computing in High Energy and Nuclear Physics, 2004.

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

T. Stevens, M. De Leenheer, C. Develder, B. Dhoedt, K. Christodoulopoulos, P. Kokkinos, E. Varvarigos, (2008), Multi-cost job routing and scheduling in Grid networks, Future Generation Computer Systems, Vol. 25, No. 8, pp. 912-925, 2008.

A. Dogan, (2009), A study on performance of dynamic file replication algorithms for real-time file access in Data Grids, Future Generation Computer Systems, Vol. 25, No. 8, pp. 829-839, 2009.

L. Kleinrock, F. Kamoun, (1977), Hierarchical routing for large networks. Performance evaluation and optimization, Computer Networks, Vol. 1, No. 3, pp. 155-174, 1977.

W. C. Lee, (1995), Topology aggregation for hierarchical routing in ATM networks, Computer Communication Review, Vol. 25, No. 2, pp. 82-92, 1995.

P. Van Mieghem, (1999), Topology information condensation in hierarchical networks, The International Journal of Computer and Telecommunications, Vol. 31, No. 20, pp. 2115 – 2137, 1999.

D. Bauer, J. Daigle, I. Iliadis, and P. Scotton, (2006), Topology aggregation for combined additive and restrictive metrics, Computer Networks, Vol. 50, No. 17, pp. 3284-3299, 2006.

C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, F. Silva, (2003), Directed diffusion for wireless sensor networking, IEEE Transactions on Networking, Vol. 11, pp. 2-16, 2003.

R. Renesse, K. Birman, W. Vogels, (2003), Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining, ACM Transactions on Computer Systems, Vol. 21, No. 2, pp. 164-206, 2003.

S. Schulz, W. Blochinger, H. Hannak, (2009), Capability-Aware Information Aggregation in Peer-to-Peer Grids, Journal of Grid Computing, Vol. 7, No. 2, pp. 135-167, 2009.

R. Zhou, K. Hwang, (2006), Trust overlay networks for global reputation aggregation in P2P grid computing, IPDPS, 2006.

M. Cai, K. Hwang, (2007), Distributed Aggregation Algorithms with Load-Balancing for Scalable Grid Resource Monitoring, IEEE International Parallel and Distributed Processing Symposium, 2007.

K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, (2001), Grid Information Services for Distributed Resource Sharing, IEEE International Symposium on High-Performance Distributed Computing (HPDC), 2001.

K. Muraki, Y. Kawasaki, Y. Mizutani, F. Ino, K. Hagihara, (2006), Grid Resource Monitoring and Selection for Rapid Turnaround Applications, IEICE - Transactions on Information and Systems, Vol. 89, No. 9, pp. 2491 – 2501, 2006.

I. Rodero, F. Guim, J. Corbalan, L. Fong, S. Sadjadi, (2010), Grid broker selection strategies using aggregated resource information, Future Generation Computer Systems, Vol 26, No. 1, pp. 72-86, 2010.

E. Laure, et al., (2006), Programming the Grid with gLite. Computational Methods in Science and Technology, Vol. 12, pp. 33-45, 2006.