Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2012, Article ID 415182, 8 pages
doi:10.1155/2012/415182

Research Article

Multidomain Hierarchical Resource Allocation for Grid Applications

Mohamed Abouelela and Mohamed El-Darieby

Software Systems Engineering Department, University of Regina, Regina, SK, Canada S4S 0A2

Correspondence should be addressed to Mohamed Abouelela, mmostafa79@gmail.com

Received 4 May 2012; Revised 4 August 2012; Accepted 5 August 2012

Academic Editor: Fangwen Fu

Geographically distributed applications in grid computing environments are becoming more and more resource intensive. Many applications require the collaboration between different domains, may be independently administrated domains, to exchange data and share computing and storage resources. This collaboration should be done in a way that maintains the privacy of each participant domain. This calls for new architectures and approaches to deal with such multidomain environments. We propose a hierarchical-based architecture as well as multidomain hierarchical resource allocation approach. The resource allocation is performed in a distributed way among different domains such that each participant domain keeps its internal topology and private data hidden while sharing abstracted information with other domains. Both computing and networking resources are jointly scheduled while optimizing the application completion time taking into account data transfer delays. Simulation results show the scalability and feasibility of the proposed approach.

1. Introduction

An increasing number of scientific and enterprise applications are becoming dependent on high performance computing (HPC) environments. In general, these applications are computation- and communication-intensive as they process very large amounts of datasets. The datasets of the applications and the resources required are geographically distributed across the grid.

The grid is an interconnected multidomain environment where each domain consists of computational, storage, and communication resources grouped together for business or administrative reasons. Each domain is independently administrated and is free to deploy different technologies. Meeting resource requirements of HPC applications generally requires allocating resources across a number of grid domains without sacrificing domain security or privacy requirements. This calls for novel multidomain scalable and reliable grid architectures, mechanisms, and algorithms that keep the balance between integration and privacy.

In general, grid systems should maintain scalability, reliability, domain privacy, and integration requirements. a scalable grid system implies maintaining acceptable performance as the number of domains increase and as the workload on the system intensifies. Reliability implies the ability of the architecture to recover from resource failures in acceptable time. Grid resource integration is a basic concept in grid computing systems that results in better overall system performance and resource utilization. The privacy of a grid domain must be maintained in for confidentiality and commercial competition.

In this paper, we propose hierarchical-based architecture. Hierarchical architecture is typically used to handle scalability and privacy problems [1]. The proposed hierarchical architecture helps in keeping domain privacy while integrating with other domains. For each domain, different computing and networking resource parameters including internal topology and resource status information are kept internally, while abstracted values for these parameters are shared with other domains. The abstracted values are to be sent to a higher level resource manager to help in taking the resource allocation decisions at the interdomain level. A multidomain hierarchical resource allocation approach is used for resource allocation.

The multidomain hierarchical allocation approach is carried out in a distributed manner. Each domain executes intradomain coallocation algorithms to allocate its own resources. Moreover, different domains coordinate with each other for resource allocation at the interdomain level and over all hierarchical levels. The approach relies on coallocation algorithm that jointly allocate computing and networking resources considering both data execution and data transfer times. We focus on a computation- and communication-summarized in section intensive application where data is stored at different sites across multidomain network and can be divided into independent subsets to be processed in parallel at different locations. This type of application is called *Divisible Load application*.

The rest of the paper is organized as follows: related work is summarized in Section 2. The proposed architecture is described in Section 3, while the multidomain hierarchical resource allocation approach is explained in Section 4. Experiments setup is explained in Section 5, while results and discussions are provided in Section 6. Finally, conclusions are offered at the end of the paper.

2. Related Work

Resource allocation in high performance grid computing is an area of ongoing research and development. Many researches were conducted illustrating the joint allocation approach and showing the advantage of it over the separated one [2–6]. Most of these efforts assume a centralized resource manager that has a complete vision of network topology as well as networking and computing resources status. This assumption is not valid for large-scale worldwide grid networks. Practically, grid network comprises geographically distributed heterogeneous resources interconnected by multidomains networks. Each domain is managed by a local domain grid manager that is usually not willing to share its internal domain information to others due to security and business confidentiality reasons. Moreover, maintaining and managing, in one centralized location, dynamic data coming from heterogeneous resources located in multidomain environments added a serious difficulty to the resource allocation process. To deal with such multidomain environments, two solutions were presented in the literature: network virtualization [5, 7, 8] and harmony [9].

Network virtualization separates logical network, called virtual network, from the substrate infrastructure network resources by dividing the role of the traditional service provider into two independent entities: infrastructure provider, who manages the substrate infrastructure network resources and service provider, who creates the virtual network by aggregating network resources from multiple infrastructure providers to build the network topology. A number of projects were already developed providing *network virtualization* over multiple domains [5, 8].

Using *network virtualization* as a solution of multidomain joint scheduling problem in grid computing environment is proposed in [5]. The authors proposed a virtualized optical network (VON) service composition framework for grid applications. Upon application task arrival, the virtual network topology is generated, and the joint scheduling starts over the virtualized network. Then, the virtualized network is release after finishing the task. Using *network virtualization* as described in [5] has many drawbacks. *Network virtualization* is still an evolving technique facing many challenges and enclosing lots of complexities [7]. Integrating multidomain joint scheduling problem within the *network virtualization* framework increases the complexity of the system without any clear advantage. One of the major drawbacks is the proposed virtualized network topology design. The proposed topology design uses bounds for the maximum amount of the expected traffic to calculate the minimum bandwidth to be reserved. This topology design does not take into account the availability of computational resources. It is not acceptable to consider just the expected traffic bounds while designing a topology that will be used in joint computing and networking resource scheduling. Ignoring computational resources capacity and availability may affect the overall performance significantly specially in computational intensive applications.

Harmony [9] is the network resource brokering system in Phosphorus Research Project [10]. The objective of Phosphorus project is to provide on-demand and end-to-end provisioning of computing and networking resources in multidomain and multitechnology environments. The workflow when a grid task received is as follows. After the authentication, the availability of the requested resources is verified. Then, the end-to-end path is allocated in two phases. In the first phase, the interdomain path is selected by the interdomain broker (IDB) module. In the second phase, a Network Resource Provisioning System (NRPS) module in each independent domain calculates the intradomain path. The intradomain topology for each domain is totally hidden from other domains and from IDB. Only border endpoints and interdomain links are exported. The organization of IDBs can be done in centralized, hierarchical, and distributed manner.

The proposed Harmony system for multidomain reservation is promising. It shows how different domains can interact to provide end-to-end connectivity and allocate the required networking resources, while maintaining the confidentiality for each domain. The main concern of this work is the allocation of the networking resource in multidomain environment, while the joint allocation of the computing and networking resources is not presented. It is just stated that they assumed that the computing resources are scheduled prior to path setup request.

In this paper, we extended the hierarchical architecture of the Harmony system to jointly schedule both computing and networking resources in multidomain environment. Each domain will maintain its structure and topology internally, while share an abstracted data about its computing and networking resources status with its Resource Manager (RM). The RM is similar to IDB in Harmony system with extended functionality to manage both computing and networking resources. The RMs are to be arranged in a multilevel hierarchical architecture. New approaches to

schedule divisible load applications in such multidomain hierarchical architecture are to be introduced.

Scheduling divisible load applications in distributed environments is frequently discussed in the literature. divisible load theory (DLT) has been successfully applied to parallel and distributed systems, as well as to grid computing environment [11–13]. Genetic-algorithms (GA)-based approaches were also proposed to schedule Divisible Loads [2, 14]. Integer linear programming has also been introduced to model such problems [13].

3. Proposed Architecture

3.1. Problem Statement. HPC grid computing applications require heterogeneous and geographically distributed computing resources interconnected by multidomain networks. Cooperation among domains, without sacrificing domain privacy, to allocate resources is required to execute such applications. For example, internal topology information of a domain should not be revealed to other domains [9]. This calls for a novel and scalable architecture allowing the integration between domains while keeping the privacy of each domain is required. We focus on divisible load applications in multidomain environment where application data is originally stored in geographically distributed sites and is divided into independent subsets to be executed in parallel at distributed data-processing sites. Those sites belong to different independently administrated domains. The performance of these applications can be optimized by concurrent execution of data processing tasks at different processing sites with different input datasets.

Such applications are modelled as data processing jobs requiring large logical input dataset, D, of total size L. D is divided into n physical datasets stored at different data sources DS_k, where $k = 1, \ldots, n$. Each physical dataset k has a size L_k, where $\sum_{k=1}^{n} L_k = L$. Those datasets are to be divided into n datasets to be executed at n different sites, and assign the required computing and networking resources. We assume that divisible data can be executed at any site using the same data processing algorithm.

The optimization (scheduling) problem is to minimize the maximum completion time by deciding on portions of datasets to be executed at each site (either executed at sites belonging to the same domain or different domains) and assigning necessary inter- and intradomain computing and networking resources.

3.2. Hierarchical Architecture. Hierarchical architecture is typically used to handle scalability and privacy problems [1]. In hierarchical architecture, sites with storage and computing resources are organized into different interconnected subnetworks (domains). A domain consists of a number of interconnected sites. A RM manages and maintains topological and state information about different computing and networking resources in a domain. The process of grouping sites (at one hierarchy level) into logical domains and abstracting such domains via a RM (at the next higher level) is done at all levels of the hierarchy (see Figure 1).

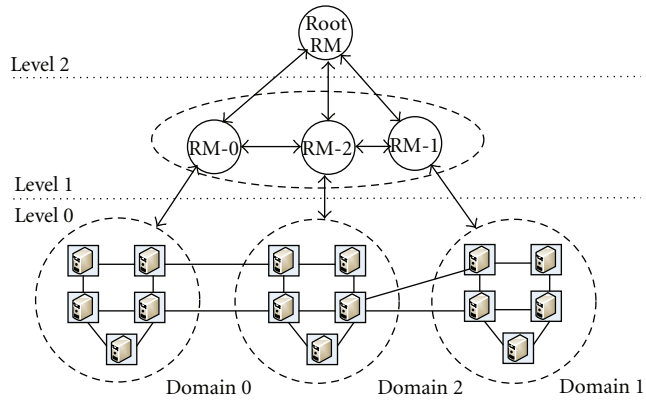


Figure 1: Two levels of hierarchical grid architecture.

Figure 1 shows a screenshot for a two levels hierarchical grid architecture. Level-0 nodes (square shape) represent data-processing sites, for example, computing clusters, or super computers containing storage and processing capabilities. Different sites and links have different computational and networking capabilities. Different sites are grouped into domains. This does not violate the special case by which single site can be considered as a domain. Each domain is managed by a level-1 RM (circular shape). Vertical line represents dedicated control channels between level-1 RM and the corresponding domain sites. Level-1 RMs are grouped into domains. Level-1 domains are managed by Level-2 RM which aggregates the collected information by level-1 RMs. In this two levels example, Level-2 RM is known as the root RM.

Level-0 horizontal links presents inter- and intradomain links available for data transfers. Level-1 horizontal links connecting level-1 RMs are virtual links. Virtual links represent the aggregated topology of the corresponding level-0 interdomain links, by which Level-0 interdomain links are aggregated and represented by level-1 links. The capacity of level-1 virtual link is the summation of the capacities of the corresponding level-0 interdomain links. At each level in the hierarchy, networking and computing resource status information is aggregated by the corresponding RM, abstracted and sent to parent RM. Within this architecture, we assume the following.

(i) RMs are connected to each other and to physical sites with fault tolerant connections (control channels).

(ii) Due to privacy considerations, complete data for each site, including its internal topology and static and dynamic resource status data, is available only for its domain RM.

(iii) An RM shares border end points and interdomain links data for its managed domain with its parent RM.

(iv) An RM maintains complete vision for the sites connected directly to it, and summarized vision (abstracted parameters) of the sites managed by its children RMs.

4. Multidomain Hierarchical Resource Allocation Approach

The resource allocation is carried out in a distributed manner at RMs from different domain and different hierarchical levels. The first step starts by executing resource coallocation algorithm at the root RM with the objective of achieving load balance and minimizing the application completion time. The algorithm defines interdomain data transfer requests. In the following step, the data transfer requests are sent down the hierarchy to children RMs. Children RMs apply intradomain resource allocation algorithm to allocate their own resources independently. This step is repeated down the hierarchy until the data transfer requests reaches Level-0 sites. If a RM couldont find enough resources to fulfil the requests, a relocate message is to be sent up the hierarchy to its parent RM to relocate the request load to another RM.

The process starts, as the system receives a job request for divisible load application, at the root RM, assumed at level k. The RootRM calculates the level $k-1$ RMs that have enough resources to meet application request. The rootRM defines a list of data transfer requests for each of the level $k-1$ RM that is expected to participate in serving the application request. Each *data transfer request* is defined by five components: source, destination, value-to-transfer, path and start-time. The *source* is a node with a number of datasets (equals to the value-to-transfer) to be executed remotely at the destination. Those datasets should be sent at a certain time (start-time) and should follow a certain *path*. The *path* is defined as a number of links connecting *source* and *destination* nodes. At level $k-1$, the RMs schedule their resources according to the requests by their parent RM. This process is repeated at each level in the hierarchy until Level 0 sites receives the resource allocation requests. This completes the scheduling process.

For example, consider the grid architecture introduced in Figure 1. The resource allocation is done first at root RM, which defines a list of data transfer requests. Assume that one of the defined requests is source = RM-0, destination = RM-1, start time = 40 s, value-to-transfer = 20 datasets, and path = RM-0 ⇒ RM-2 ⇒ RM-1 (the interdomain path). This request is to be sent to all the RMs involved in this task (RM-0, RM-1, and RM-2). Then, the scheduling starts at those RMs to allocate the required internal resources to complete those requests and provide end to end connectivity. The scheduling at each of the three RMs results in new lists of requests. Those lists are to be sent to the sites at level 0.

The detailed algorithm at each RM is comprised of the following three steps, described in the following subsections.

4.1. Handling Parent Data Transfer Requests. The process at a level j RM starts by receiving a list of data transfer requests from its parent RM at level $j+1$. Those requests should be handled first by allocating the necessary computing and networking resources. A resource allocation greedy algorithm is called to allocate the needed resources. This greedy algorithm will be explained in Section 4.3. The RM defines the set of need computing and networking resources according to its role in parent request. Generally, the RM can play one of the following roles: *source role*, *destination role,* and *transit role*.

(i) *Source Role*. If the data transfer request defines the RM as a source node, then a number of datasets, equals to the *value-to-transfer*, should be sent out of the domain managed by this RM to a Predefined interdomain link at, or before, task start-time. The RM should allocate the required internal networking resources to transfer the task data to the border node connected to the predefined interdomain link.

(ii) *Destination Role*. If the data transfer request defines the RM as a destination node, then the domain managed by this RM expects a certain number of datasets to arrive to a certain border node through a certain interdomain link. The required computing and networking resources should be assigned to execute or analyze the coming data internally.

(iii) *Transit Role*. If the data transfer request defines the RM as a transit node (one of the intermediate nodes defined in the path field of the request), then the managed domain expects a certain amount of data to arrive to a certain border node and the same amount of data to send out from another border node. The RM should provide the internal networking resources to connect those two border nodes to complete the interdomain path end to end connectivity.

4.2. Optimal Load Distribution Calculation. Allocating the necessary resources to handle parent request may result in unbalanced-load distribution among different computing units. Therefore, load balancing is needed to ensure that the computing units in the participant sites (or domains) will finish the load processing at the same time. Assuming n sites, L_i, for all $i \in 1,\ldots,n$ defines the current load distribution (before load balancing). The objective of the load balancing is to define the optimal load distribution α_i, for all $i \in 1,\ldots,n$. α_i defines the number of datasets that should be allocated for each site i for optimal load distribution. Different algorithms could be used to calculate the optimal load distribution. in this paper, we will use Network Aware Divisible Load Algorithm (NADLA) [15]. NADLA is a simple, light-weight and fast load balancing algorithm based on divisible load theory. It considers network availability and connectivity while deciding on load distribution.

4.3. Resource Allocation Greedy Algorithm at Each RM. After defining the optimal load distribution, a set of *data transfer requests* should be defined to execute the new load distribution, and different computing and networking resources should be allocated. The resource allocation greedy algorithm (Algorithm 1) is used to define the requestlist (the set of data transfer requests). The algorithm starts with an empty requestlist (step 1). Then, the difference between the optimal load distribution α_i and the current load distribution L_i is calculated for each site i. This difference represents the portions of data to be transferred to/from each site i. This value can be positive, negative, or zero. Positive

1: Set $RequestList = \{\}$
2: Calculate $\alpha_i - L_i, \forall i \in 1, \ldots, n$
3: **while** $\alpha_i - L_i \neq 0, \forall i \in 1, \ldots, n$ **do**
4 : Set $dest = i$, such that $(\alpha_i - L_i)$ is max
5 : Set $SourceList = \{i\}, \forall i \in 1, \ldots, n \& \alpha_i - L_i \leq 0$
6 : **for each** $source \in SourceList$ **do**
7 : Calculate $Path_{source,dest}$ and $PathWaitingTime_{source,dest}$
8 : **end for**
9 : Select $source$ from $SourceList$ such that $PathWaitingTime_{source,dest}$ is minimum
10 : Set $ValueToTransfer = min[abs(\alpha_{source} - L_{source}), abs(\alpha_{dest} - L_{dest})]$
11 : $RequestList+ = newTask(source, dest, ValueToTrasfer, Path_{source,dest}, TransferTime)$
12 : Set $L_{source} - = ValueToTransfer$
13 : Set $L_{dest} + = ValueToTransfer$
14 : Update links with the new reservations
15: **end while**
16: Populate $RequestList$

Algorithm 1: Resource allocation greedy algorithm.

values mean data sink site (destination receiving data to be executed internally), while negative values mean data source sites (sites containing extra-data to be executed in remote site).

The algorithm iterates until the current load distribution equals to the optimal load distribution at all sites. In each iteration, the *dest* site is selected first as the less loaded site; $\alpha_i - L_i$ is maximum. Then, a *SourceList* list is defined containing all sites having extra-load; $\alpha_i - L_i$ is negative. The *source* site is selected from this list with the objective of minimizing the path waiting time. The shortest paths between the *dest* site and each site in the *SourceList* are calculated, and the site with minimum path waiting time is selected. A new *data transfer request* is added to the *RequestList*. Finally, the source and destination Loads (L_{source} and L_{dest}) and Links schedules should be updated accordingly.

5. Experiment Setup

Simulation experiments were conducted to evaluate the performance of proposed architecture as well as the multidomain resource allocation approach. A wide range of different parameters was considered to cover different network topologies, application types, and algorithms. Up to 10 runs are carried out for each experiment and their results are averaged for 95% confidence intervals.

Simulations were conducted using OMNET++ network simulator (http://www.omnetpp.org/). OMNET++ is a C++ open source discrete event simulator. OMNET++ is highly modular and well-structured simulator. It provides realistic and accurate network models for different protocols and architectures. We developed our own modules to support multilevel hierarchical architecture and grid computing functionality.

Different network topologies were generated with a wide range of parameter variations matching the network architecture proposed in Section 3.2. Different network sizes, the number of level 0 sites, were considered, varying from 16 sites up to 1000 sites. Sites were grouped into domains to construct multilevel hierarchies up to 5 levels. Networks with different average node degree d: the number of links connecting this node to other nodes, were considered. The average node degree values are varying from 2 to 8. Different bandwidth values for interdomain and intradomain links were considered.

Moreover, different application load sizes were examined starting from an average of 25 datasets per source site to 3000 datasets per site, while the unit dataset size was fixed to be 1 Gbit. As the datasets per source site increases, the application becomes more data-intensive. Different applications may have different processing capacities (time to process a unit dataset) even on the same site. As the processing capacity increases, the application goes to be more computationally intensive. In our simulations, we considered three application categories: data intensive applications, intermediate applications, and computationally intensive applications. To differentiate between those three categories, the average site processing capacities is set to 5, 25, and 100 second/unit dataset for the three categories, respectively.

The following metrics have been used to evaluate the performance of the hierarchical scheduling as well as the proposed architecture.

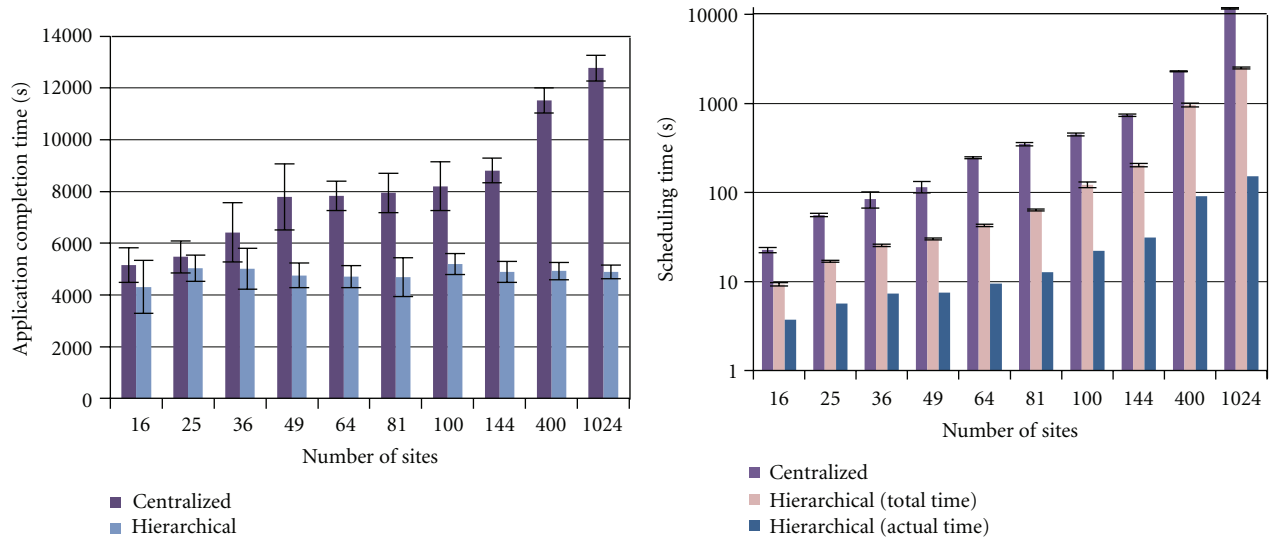
(i) *The application completion time*, which is the maximum task completion time over all the sites. It is measured from the task arrival time, until the last site finishes data processing.

(ii) *Scheduling time*, which is the time consumed inside the RMs to come to a decision on scheduling and resource allocation. For hierarchical scheduling, the scheduling time is calculated as the summation of scheduling times over all RMs.

(iii) *Standard Deviation in resource utilization* (SD), which is a metric of the system load balancing by measuring the variations in resource utilization. It is calculated as the standard deviation in resource utilization for both links and computing resource units. Resource utilization is calculated as the percentage of time by which the resource is busy, so the SD is calculated as a percentage. SD = 0% reflects optimal load balancing, that all the resources are utilized equally.

6. Results and Discussion

6.1. Hierarchical versus Centralized. The hierarchical and centralized architectures are compared for different network sizes varying from 16 sites network up to 1024 sites network. The number of hierarchical levels is fixed to two levels for all hierarchical networks; also the average node degree is fixed to 4.



(a) Application completion time

(b) Scheduling time

Figure 2: Hierarchical versus centralized for different network sizes.

Figure 2(a) shows the application completion time for both centralized and hierarchical architectures. It is clear that for small size networks (16,25 & 36 sites) the two architectures results in almost the same application completion time. As increasing the network size the centralized architecture results in significant increase in the application completion time, while the hierarchical architecture results in slight increase in the application completion time.

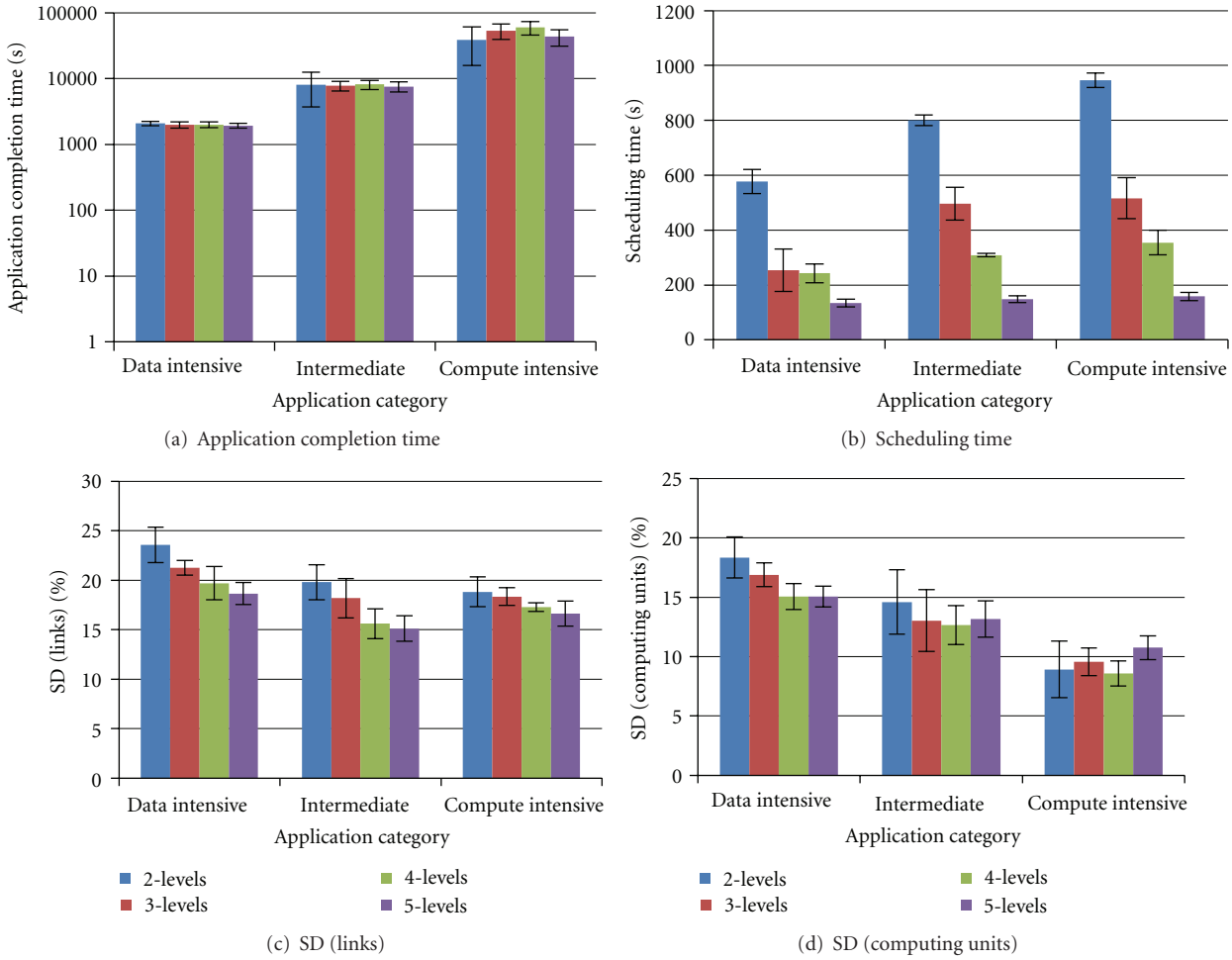
The *scheduling time* for both architectures, shown on the *y*-axis in Figure 2(b), (log scale is used for the *y*-axis for better visualization of results). For hierarchical architecture, two values for the scheduling time were measured: the total *scheduling time* and the *actual scheduling time*. The total time is measured as the summation of the scheduling task times in all the RMs, while the actual time is the actual task scheduling time by considering that the RMs at the same level are running in parallel. This is one of the advantages of the hierarchical architecture, that the scheduling is distributed among a number of RMs running in parallel, which results in significant deduction in scheduling time. As shown in Figure 2(b), the hierarchical architecture outperforms the centralized one for the *scheduling time*.

A significant reduction in the *scheduling time* is measured when using the hierarchical one especially for large size networks when compared to centralized architectures. The reduction is around 90% for small size networks (16 & 25 sites networks), while it reaches around 98% for large size networks (1024 sites network). This comes at a specific cost that will be discussed below. The *scheduling time* for centralized architecture increases significantly as increasing the network size. For example, as increasing the number of sites from 25 to 49, the *scheduling time* increases by a factor of 2, while increasing the number of sites from 400 to 1024 results in increase by a factor of 5. On the other hand, for hierarchical architecture, the *scheduling time* increases by a factor of 1.65 while increasing the number of sites from 400 to 1024.

6.2. Effect of Hierarchy Depth. To evaluate the effect of the depth of hierarchy, different networks with fixed number of sites (250 site) were used while varying the number of hierarchical levels from 2 to 5. The depth of the hierarchy is evaluated for the three application categories and the results are shown in Figures 3(a), 3(b), 3(c), and 3(d). It is shown in Figure 3(a) that *application completion time* is not affected by increasing the depth of the hierarchy for networks with different hierarchical levels. Figure 3(b) shows a decrease in the *scheduling time* as increasing the depth of the hierarchy for the three application categories. In addition, a small decrease in the standard deviation *in links utilization* is reported (Figure 3(c)). This means better load balancing among different links as increasing the depth of the hierarchy. Figure 3(d) shows no notable change in the *standard deviation in computing units utilization* as increasing the depth of the hierarchy.

Those advantages in *scheduling time* could be verified analytically. The time of the scheduling algorithm executed at each RM depends mainly on the number of sites in the managed domain. Assuming that we have a total of N sites grouped into domains in L hierarchical levels. Then, the number of nodes in each domain follows the exponential function $N^{1/L}$. Then, the scheduling time at each RM will follow the same function and decrease exponentially with respect to the number of levels. Assuming that all the RMs at the same level are running in parallel, then the actual scheduling time equals to the result of multiplying the scheduling time at one RM by the number of levels.

Those advantages in the *scheduling time* as using the hierarchical architecture as increasing the depth of the hierarchy come at the cost of increasing the control overhead. Increasing the depth of the hierarchy increases the required number of RMs to manage the system for networks with the same size. For example, increasing the depth of the hierarchy from 2 to 5 increases the number of RMs by



(a) Application completion time

(b) Scheduling time

(c) SD (links)

(d) SD (computing units)

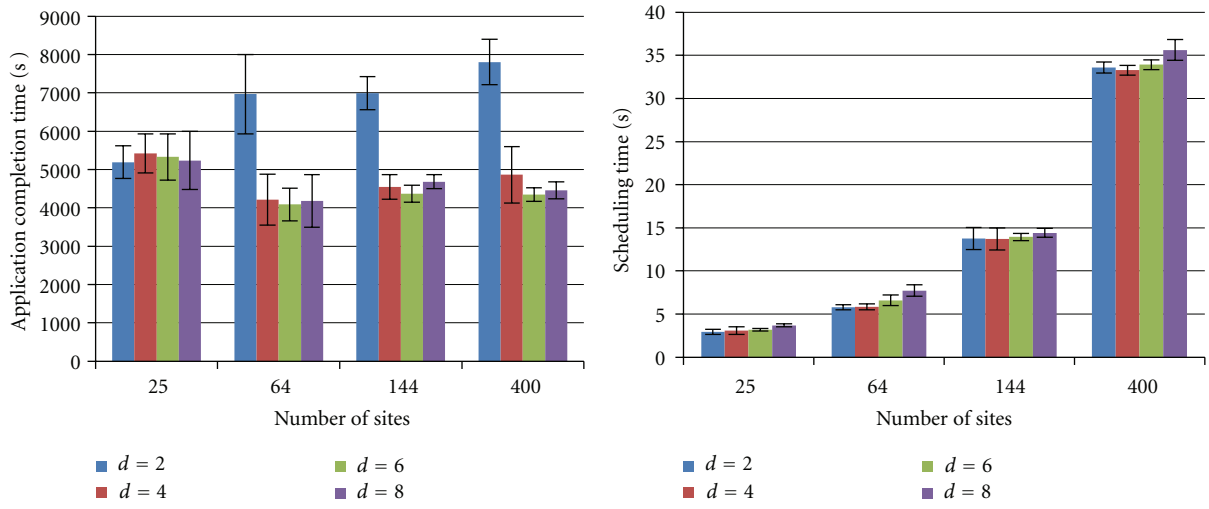
Figure 3: The Effect of depth of hierarchy for different application categories.

a factor of 7. Controlling and maintaining this hierarchical structure increases the cost and complexity as increasing the number of RMs. This increases the control overhead and communication complexity. In [1], a study of a hierarchical routing protocol reported a notable increase in path setup time and communication overhead as increasing the depth of the hierarchy.

6.3. Effect of Node Degree. To study the effect of average node degree (d), networks with different sizes and average node degrees were considered. As shown in Figure 4(a), for small size network (25 sites network), no significant change in *application completion time* is reported as increasing the average node degree, while for larger size networks (64, 144 & 400 sites networks), a notable deduction in *application completion time* is reported while increasing the average node degree from 2 to 4. Increasing the average node degree more than 4 results in no notable improvement in the application completion time for all the tested network sizes. Figure 4(b) shows a very small increase in *scheduling time* as increasing the average node degree. Those results can help in network dimensioning problem to select the optimal value for the average node degree that reduces the application completion time, while minimizing the number of links. The dimensioning problem is out of scope of this paper and may be considered in the future.

7. Concluding Remarks

The proposed hierarchical architecture as well as the multidomain hierarchical resource allocation approach provided a novel solution for the joint resource allocation problem in multidomain grid environments. The hierarchical architecture maintained the scalability and privacy of the grid system. The proposed architecture helped in keeping the domain privacy while integrating with other domains. Domain internal topology and resource status information were kept internally, while sharing abstracted parameters with other domains. The multidomain hierarchical resource allocation approach was carried out in a distributed manner where each domain executes intradomain joint scheduling algorithm to schedule its own resources. Moreover, the process involved coordinating the resource allocation at the interdomain level and over all hierarchical levels.



(a) Application completion time

(b) Scheduling time

Figure 4: Effect of average node degree for different network sizes.

Simulations were conducted to evaluate the performance in terms of application completion time, scheduling time, and resource utilization for different network topologies, application types, and algorithms. The proposed hierarchical architecture proved its scalability and feasibility. Increasing the hierarchical depth results in better scheduling time and load balancing. Those advantages came at the cost of increasing control overhead. In the future, other research work will be conducted based on the proposed hierarchical architecture and hierarchical resource allocation approach. Analysing and evaluating different aggregation procedures is one of our future goals. In addition, introducing fault management mechanism will be considered.

References

[1] M. El-Darieby, D. Petriu, and J. Rolia, "Load-balancing data traffic among inter-domain links," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 5, pp. 1022–1033, 2007.

[2] M. Abouelela and M. El-Darieby, "Co-scheduling computational and networking resources in E-science optical grids," in *Proceedings of the 53rd IEEE Global Communications Conference (GLOBECOM '10)*, pp. 1–5, Miami, FLa, USA, December 2010.

[3] N. Charbonneau, V. M. Vokkarane, C. Guok, and I. Monga, "Advance reservation frameworks in hybrid IP-WDM networks," *IEEE Communications Magazine*, vol. 49, no. 5, pp. 132–139, 2011.

[4] M. Koseoglu and E. Karasan, "Joint resource and network scheduling with adaptive offset determination for optical burst switched grids," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 576–589, 2010.

[5] Y. Wang, Y. Jin, W. Guo, W. Sun, and W. Hu, "Virtualized optical network services across multiple domains for grid applications," *IEEE Communications Magazine*, vol. 49, no. 5, pp. 92–101, 2011.

[6] G. Zervas, E. Escalona, R. Nejabati et al., "Phosphorus grid-enabled GMPLS control plane (G2MPLS): architectures, services, and interfaces," *IEEE Communications Magazine*, vol. 46, no. 6, pp. 128–137, 2008.

[7] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications Magazine*, vol. 47, no. 7, pp. 20–26, 2009.

[8] I. Houidi, W. Louati, W. Ben Ameur, and D. Zeghlache, "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011–1023, 2011.

[9] A. Willner, C. Barz, J. A. Garcia Espin, J. Ferrer Riera, S. Figuerola, and P. Martini, "Harmony—advance reservations in heterogeneous multidomain environments," in *Proceedings of the 8th International IFIP-TC 6 Networking Conference (NETWORKING '09)*, pp. 871–882, Springer, Berlin, Germany, 2009.

[10] S. Figuerola, N. Ciulli, M. De Leenheer, Y. Demchenko, W. Ziegler, and A. Binczewski, "Phosphorus: single-step on-demand services across multi-domain networks for e-science," in *Proceedings of the Network Architectures, Management, and Applications V (SPIE '07)*, vol. 6784, Wuhan, China, November 2007.

[11] S. Viswanathan, B. Veeravalli, and T. G. Robertazzi, "Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1450–1461, 2007.

[12] C. Yu and D. C. Marinescu, "Algorithms for divisible load scheduling of data-intensive applications," *Journal of Grid Computing*, vol. 8, no. 1, pp. 133–155, 2010.

[13] P. Thysebaert, B. Volckaert, M. De Leenheer, F. De Turck, B. Dhoedt, and P. Demeester, "Dimensioning and on-line scheduling in Lambda Grids using divisible load concepts," *The Journal of Supercomputing*, vol. 42, no. 1, pp. 59–82, 2007.

[14] S. Kim and J. B. Weissman, "A genetic algorithm based approach for scheduling decomposable Data Grid applications," in *Proceedings of the International Conference on Parallel Processing (ICPP '04)*, pp. 406–413, August 2004.

[15] M. Abouelela and M. El-Darieby, "Towards network-aware divisible load theory for optical grids," in *Proceedings of the IEEE 13th International Conference on High Performance Computing and Communications (HPCC '11)*, pp. 425–431, September 2011.