Computer Communications 32 (2009) 1172–1184

Contents lists available at ScienceDirect

Computer Communications

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

A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in Grid networks

K. Christodoulopoulos [a,*], V. Sourlas [b], I. Mpakolas [a], E. Varvarigos [a]

[a] Computer Engineering and Informatics Department, University of Patras, Greece, and Research Academic Computer Technology Institute, Patras, Greece
[b] Department of Computer and Communications Engineering, University of Thessaly, Volos, Greece

ARTICLE INFO

Article history:
Received 12 November 2008
Received in revised form 4 March 2009
Accepted 11 March 2009
Available online 21 March 2009

Keywords:
Grid networks
Centralized vs. distributed architecture
Task scheduling
Routing and data scheduling
Online algorithms

ABSTRACT

The management of Grid resources requires scheduling of both computation and communication tasks at various levels. In this study, we consider the two constituent sub-problems of Grid scheduling, namely: (i) the scheduling of computation tasks to processing resources and (ii) the routing and scheduling of the data movement in a Grid network. Regarding computation tasks, we examine two typical online task scheduling algorithms that employ advance reservations and perform full network simulation experiments to measure their performance when implemented in a centralized or distributed manner. Similarly, for communication tasks, we compare two routing and data scheduling algorithms that are implemented in a centralized or a distributed manner. We examine the effect network propagation delay has on the performance of these algorithms. Our simulation results indicate that a distributed architecture with an exhaustive resource utilization update strategy yields better average end-to-end delay performance than a centralized architecture.

1. Introduction

The continuing deployment of high speed networks is making the vision of Grid networks a reality. Grids consist of geographically distributed and heterogeneous computational and storage resources that may belong to different administrative domains, but are shared among users by establishing a global resource management architecture [1]. The Grid scheduling problem deals with the coordination and allocation of resources so as to efficiently execute the users' tasks. Tasks are created by applications, belonging to individual users or Virtual Organizations (VOs), and request the Grid's services for their execution. Tasks may or may not depend on each other, and they generally require the use of different kinds of resources, e.g., computation, communication (network), storage resources, or specific instruments. Scheduling aims at meeting user demands (e.g., in terms of cost, response–time) and the objectives represented by the resource providers (e.g., in terms of profit, resource utilization efficiency), while maintaining a good overall performance/throughput for the Grid network. The complexity of Grid applications, the diverse user requirements and the system heterogeneity would result in inefficient scheduling in the case of a manual procedure.

Grid scheduling is usually viewed as a hierarchical problem with two levels. At the first level, usually called *meta-scheduling*, a meta-scheduler selects the resources to be used by a task. These resources can be computational, communication, storage, or other. At the second level, usually called *local-scheduling*, each resource (more precisely, the Local Resource Management System, LRMS, of that resource) schedules the tasks assigned to it. The meta-scheduler and the local-scheduler differ in that the latter only manages a single resource, e.g., a single computation machine, a single network link, etc., while the former considers more than one resources in making its decisions. In particular, the meta-scheduler receives tasks that require the use of computation, communication or other resources from Grid users and generates task-to-resource assignments, by optimizing some objective function. In this study we are more interested in the first level of Grid scheduling, the meta-scheduling level.

Grid applications can be categorized as cpu-intensive or data-intensive. However, almost all tasks have a computation and a communication part, even if one part is often more important than the other. Various meta-scheduling algorithms have been proposed for both the computation and communication scheduling problems. Cpu-intensive applications require the meta-scheduler to consider the characteristics and the availability of the computation resources in order to make task-to-resource assignments, while data-intensive applications require the meta-scheduler (bandwidth broker) to examine the utilization of the communication resources in order to make routing and data transmission

* Corresponding author. Tel.: +30 2610 996988.
E-mail addresses: kchristodou@ceid.upatras.gr (K. Christodoulopoulos), vsourlas@inf.uth.gr (V. Sourlas), mpakolas@ceid.upatras.gr (I. Mpakolas), manos@ceid.upatras.gr (E. Varvarigos).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.03.004

scheduling decisions. For the rest of this study we will refer to the problem related to cpu-intensive applications and computation resources as *task scheduling*, and the problem related to data-intensive applications and communication resources as *routing and data scheduling*.

An important issue in designing a meta-scheduler for a distributed environment, such as the Grid, is whether it will be implemented in a centralized or a distributed manner. In both the task, scheduling and the routing and data scheduling problems, we can either use a (single) central meta-scheduler to make such decisions or use multiple meta-schedulers to perform this in a distributed way. In a centralized scheme, a central meta-scheduler collects all requests and uses resource utilization information to schedule computation and communication tasks efficiently. We will assume that the central meta-scheduler has complete knowledge of the availability of all resources, since it is the one that makes all resource assignment decisions. On the other hand, in the distributed case, there exist a number of distributed meta-schedulers, each of which keeps track of the resources availability, and uses this information to make the appropriate task scheduling or routing and data scheduling decisions. The information used by distributed schedulers to perform their function may be outdated due to the non-zero network propagation delays.

Meta-scheduling algorithms, whether centralized or distributed, can be categorized as: (i) *online* algorithms, which assign tasks to resources immediately upon their arrival and (ii) *offline* algorithms, which wait for a period of time so that several tasks accumulate at the meta-scheduler, before making the task-to-resource assignment decisions. Offline algorithms usually consist of two phases: the *task-ordering* phase and the *resource assignment* phase. Online algorithms can be viewed as a special case of offline algorithms where the task-ordering phase uses the FCFS (First Comes First Serves) queuing discipline. Distributed scheduling schemes usually follow a single-phase procedure, while centralized scheduling schemes are employed with one or two phases.

In this work, we compare the performance of various (i) task scheduling and (ii) routing and data scheduling online algorithms, implemented either in a centralized or in a distributed manner in a Grid network. For scheduling computation tasks, we examine two typical online algorithms, namely, the Earliest Start Time and the Earliest Completion Time algorithm, which have been extended to support advance reservations. Both algorithms are examined assuming either a centralized or a distributed implementation. The performance metrics used to compare the two implementations include the average total task execution delay, the conflict probability, and the average number of update messages exchanged. Similarly, for the communication problem, we compare centralized and distributed versions of two online routing and data scheduling algorithms, namely, a multicost heuristic algorithm (which we call the Availability Weighting heuristic multicost algorithm) and a typical Dijkstra with conflict avoidance algorithm. The distributed versions of both algorithms were presented in [20]. The metrics used to evaluate the performance of their centralized and distributed implementations are the average end-to-end delay, the number of retrials for a successful transmission, and the average number of update messages exchanged.

Our results indicate that a distributed architecture with an exhaustive utilization update strategy results in better average end-to-end delay performance for cpu- or data-intensive tasks than a centralized architecture employing the centralized version of the same algorithm. The exhaustive update strategy requires a large number of utilization update messages. However, this does not make the distributed architecture impractical, since modern Grid networks exchange a large number of messages for monitoring and other purposes, which can also be used to communicate the resource availability information to the distributed schedulers.

The remainder of this paper is organized as follows. In Section 2, we report on previous work. In Section 3, we describe the Grid network model assumed and define the task scheduling and the routing and data scheduling problems. In Section 4, we introduce the cluster availability profile, which is a data structure that can be used to monitor the utilization profile of a computation resource. We also present a mechanism to exchange computing resources utilization information and describe the two task scheduling algorithms examined. Similarly, in Section 5, we present the link availability profile, a mechanism to exchange link utilization information and the two examined routing and data scheduling algorithms. Performance evaluation results are presented in Section 6, with separate sections addressing the task scheduling (Section 6.1) and the routing and data scheduling (Section 6.2) subproblems. Finally, Section 7 concludes the study.

2. Related work

The meta-scheduling algorithms to be investigated address the task scheduling and the routing and data scheduling problems. In Section 2.1, we report on previous work with respect to task scheduling algorithms, while in Section 2.2 we report on routing and data scheduling algorithms. The goal of this study is to compare these types of algorithms when they are employed in a centralized or α distributed manner. Previous work on centralized vs. distributed architectures is discussed in Section 2.3.

2.1. Prior work on the task scheduling problem

In [2], Li examines 128 combinations of algorithms for different levels of scheduling in Grids, assuming a centralized meta-scheduler. More specifically, the author assumes a two-phase meta-scheduling algorithm and examines four initial task ordering strategies, four resource assignment algorithms, and two local queue scheduling algorithms, when these are applied to four meta-computers. The authors in [3] consider a centralized meta-scheduling architecture and examine algorithms for the resource assignment phase. For a given set of tasks, they evaluate 11 heuristics algorithms to solve the optimal assignment problem, which is NP-hard, assuming that an accurate estimate of the execution time for each task on each machine is known prior to execution. In [4], the authors assume a central meta-scheduler that uses online task assignment, and examine the performance of various heuristic algorithms at the meta-scheduling and the local/cluster scheduling levels.

Buyya et al. [5] proposed a distributed economic-based approach, where scheduling decisions are made online and they are driven by the end-users requirements. In [6], the authors apply economic considerations in Grid resource scheduling and propose GridIS, a Peer-to-Peer (P2P) decentralized scheduling framework. In GridIS when a user has a task, he/she sends a related announcement via a portal. The task announcement is forwarded throughout the P2P network and resource providers that receive it bid for the task (auction). In [7], the proposed distributed meta-scheduler uses an "aggressive reservation" technique and forwards each task to the k least loaded resource sites, each of which schedules the task using its local queue scheduler. When a task is able to start execution at a resource, the task-originating site is informed, which in turn contacts the $k-1$ other resources to cancel the tasks from their respective queues. A distributed algorithm that tackles the assignment of divisible load applications is presented in [8]. The proposed distributed algorithm uses resource utilization information (feedback), and the authors quantify the level of global information that is required for efficient scheduling. Scheduling tasks in an environment that supports advance reservations has been con-

sidered in [10,11]. Finally, a taxonomy of Grid scheduling algorithms can be found in [9].

2.2. Prior work on the routing and data scheduling problem

As Grid applications evolve, the need for user controlled network infrastructures becomes apparent in order to support emerging dynamic and interactive services. By the term "user" we refer to an individual user, a Virtual Organization (VO) or even a general-type application. To provide quality of service (QoS) over the communication plane to the users, reservations and, in particular, reservations performed *in advance* are needed. Given a data transfer request, choosing the path to be used and the time instances the corresponding communication resources (link bandwidth, buffers, etc.) will be reserved is what we call the routing and data scheduling problem. Note that these resources might be requested and reserved in the future (in-advance reservations). We will focus on a special case of the routing and data scheduling problem that assumes an Optical Burst Switched (OBS) underlying network. Thus, a communication task requires the transfer of data from the scheduler or a data repository site, which we will call source, to another site in the form of a continuous connection with a constant rate, or a data burst. The role of the Bandwidth Broker is to choose the route and the time to schedule this transfer.

A draft submitted to the Open Grid Forum [13] proposes Grid Optical Bursts Switched (GOBS) as a candidate network infrastructure for supporting dynamic and interactive Grid services. In burst switched networks [12], the data that need to be exchanged between two sites (task originating sites, repositories, computational resources, etc.) is transmitted as a data burst that is switched through the network using a single label. This reduces the switching and processing requirements in the core network. The signaling protocols proposed for burst switched networks [14] can be categorized into two main classes: two-way (tell-and-wait) and one-way (tell-and-go) protocols. An example of a tell-and-wait protocol is the Efficient Reservation Virtual Circuit (ERVC) protocol proposed in [15], while recent research efforts include the WR-OBS [16]. One-way reservation schemes have received increasing attention in the recent years. Typical examples in this category are the Ready-to-Go Virtual Circuit protocol [17], Horizon [18], Just-Enough-Time (JET) [14], and Just-In-Time (JIT) [19].

The challenge in burst switched networks that employ no or minimal buffering in the core, is how to route and schedule the bursts so as to achieve efficient usage of resources and high throughput, while keeping dropping/blocking rates at low levels. The majority of burst routing and scheduling algorithms that have appeared in the literature are online algorithms employed in a distributed environment. The starting time of the burst transmission, referred to as the *time offset*, is usually calculated by taking into account the used protocol (one- or two-way) and the number of hops. The time offset can also be chosen so as to provide QoS differentiation, as in [23], where a high loss priority class is given a larger offset time in order to make earlier wavelength reservation than lower priority classes. The choice of the offsets is further examined in [24], where an adaptive offset calculation that depends on both the link utilization and the burst losses in the core is proposed. In [25], a dynamic Wavelength Routed OBS architecture is introduced, where centralized control is employed to provide resource reservation efficiency, low delay, and QoS differentiation.

A recent approach in scheduling communication tasks is the multicost routing and scheduling algorithm presented in [20]. This algorithm selects the paths to be followed by the bursts and the times the bursts should be transmitted so as to arrive at their destination with minimum delay, addressing in this way the burst routing and scheduling problem jointly. The algorithm uses advance reservations in order to schedule the time that the data will be transmitted and reserve capacity on the subsequent links down the path [21]. The proposed algorithm is designed to function in a distributed way but can be modified in a straightforward way to work in a centralized environment.

2.3. Prior work on distributed versus centralized implementations

Generally, a centralized architecture is vulnerable to its single point of failure and lacks scalability. The central scheduler is simpler to implement, easier to manage and quicker to repair in case of a failure, but becomes a performance bottleneck in large-scale networks. For these reasons, a centralized architecture is generally considered more suitable for small-sized networks, while a distributed architecture is preferable for large-scale networks. A distributed architecture leaves the scheduling decision to each node and distributes the associated computation overhead. So, it eliminates the bottleneck posed by the central scheduler and improves the reliability and scalability of the network. The main challenge in a distributed architecture is the coordination of the distributed schedulers and the control plane overhead that is required. This overhead has to be measured against the performance that can be achieved.

In [26], the authors compare centralized and distributed routing connection management schemes under different traffic patterns in WDM networks with wavelength conversion. They show that the network blocking probability improvement obtained by the centralized schemes over that of the distributed schemes is proportional to the reservation durations and inversely proportional to the average connection inter-arrival times. Beyond a certain value of the traffic load, the distributed and centralized mechanisms yield similar blocking probabilities. In [27], the authors examine a novel distributed lightpath reservation mechanism based on multiple tokens in a WDM ring topology, and compare it to centralized reservation mechanisms.

Regarding the task scheduling problem, the studies comparing centralized and distributed implementations are quite limited. In [28], the authors discuss centralized, distributed, and hierarchical task-scheduling in Grids and perform simulations to compare various alternative schemes. Their results show that, due to the complicated nature of the problem, the performance of the algorithms depends highly on the parameters, the machine configurations and the workload.

The centralized versus distributed problem has a number of other extensions apart from the ones we consider here. For example, the US Department of transportation has examined the strengths and weaknesses of a distributed and a centralized telecommunications infrastructure for implementing an Intelligent Transportation System (ITS) [29].

3. Problem statement

We consider a Grid network consisting of users, computing resources (clusters) and burst routers, connected through a network. Depending on the problem that we want to address (task scheduling, or routing and data scheduling) we assume that a user generates cpu- or data-intensive tasks in the form of specific requests and forwards them to the appropriate meta-scheduler. In order to separate the two Grid scheduling sub-problems, we will refer to the meta-schedulers that handle the task scheduling requests as Computation Schedulers (CS) and the corresponding data communication schedulers as Bandwidth Brokers (BB). The terms "meta-scheduler" and "scheduler" are used interchangeably for both problems. Each node in the network has a router that performs the data forwarding according to the Optical Burst Switching

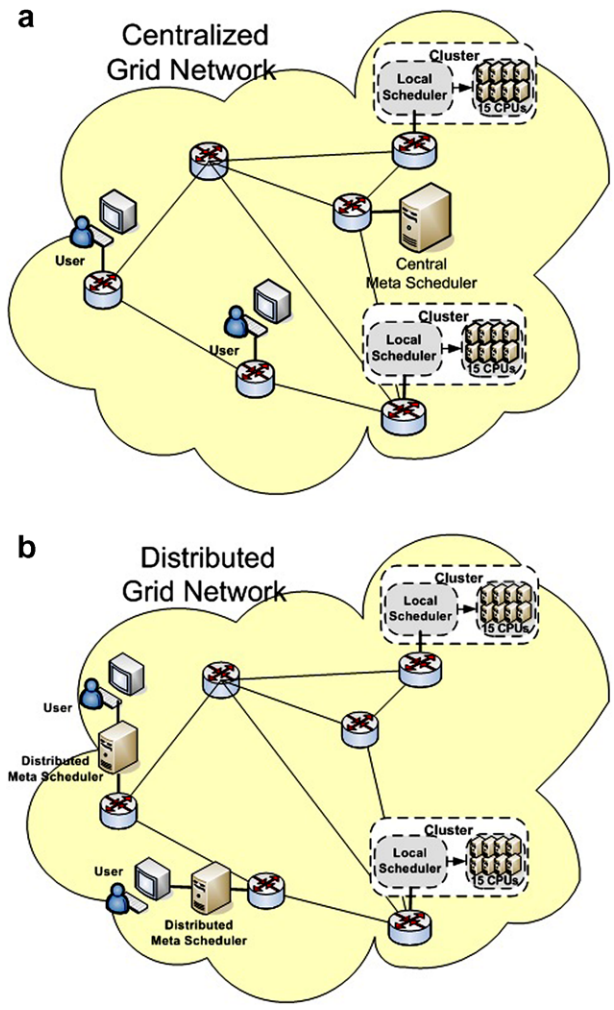

Fig. 1. (a) A centralized versus (b) a distributed Grid environment. Each node has a router that performs data forwarding according to the Optical Burst Switched paradigm and can have a user, a computation resource (cluster) consisting of a number of processors, a meta-scheduler, a combination, or none of these entities. Depending on whether we have a centralized or a distributed architecture we may accordingly have one or many meta-schedulers in the grid network.

paradigm. Users, computation resources, and scheduler(s) are placed at nodes, so that a node may have all, some or none of these entities. In the distributed approach, there is one distributed scheduler for every node that has a user, while in the centralized approach there is only one scheduler in the whole Grid network. Fig. 1a and b presents an example of a centralized and distributed architecture.

3.1. Task scheduling problem

A task generated by a user is characterized by several parameters. Specifically, in our model, a task i (e.g., an MPI parallel program) requests to be executed on r_i processors, has a computation workload W_i per processor (measured in millions instructions, MI), and a non-critical deadline D_i (measured in seconds). By "non-critical" we mean that if the task deadline expires, the task remains in the system until it is executed, but is recorded as a deadline miss. Note that we have assumed that we know the exact computation workload of a task before its execution, which may be provided by user estimates or through a prediction mechanism, such as script discovery algorithms, databases containing statistical data on previous runs of similar tasks, or some other method. Although this assumption is rather strong, it affects in similar ways the centralized and the distributed architectures we investigate.

Each computation resource (cluster) P contains a number of processors N_P, each with the same computation capacity C_P (measured in millions instruction per second, MIPS). The computation resources employ a space-shared non-preemptive discipline, so that once a task starts its execution on a set of cpus it cannot be stopped and has to be completed on those cpus. Each resource has a local queue for storing the arriving tasks, and a corresponding local scheduler that assigns the tasks in the local queue to the available processors. We assume that the local scheduler serves tasks in the FCFS order and can use advance reservations to schedule tasks to be executed in the future. In this way, a task that arrives at the queue and requests to be executed at some specific time, called Starting Time (ST), reserves r processors for the duration that it requests. If another task arrives later and its requested execution period overlaps with a previously reserved period, this new task is not served (assuming no other processors are available), even if it requests to start earlier. This problem is called a *racing conflict*, which we solve by running a re-scheduling algorithm locally at the resource.

In the case of a centralized architecture, the central meta-scheduler maintains utilization information for all resources that is always up-to-date, by keeping track of a *cluster availability vector* per resource (see Section 4). In the distributed architecture, each distributed meta-scheduler maintains a "picture" of the utilization of all the cluster resources. The locally maintained picture can be different among the distributed meta-schedulers, due to the different propagation delays, and depends on their position in the Grid network. Update messages are exchanged among resources and distributed meta-schedulers, as described in Section 4, to synchronize the locally maintained clusters' profiles with the actual utilization information.

The parameters of a newly created task are immediately forwarded to the corresponding meta-scheduler (central or distributed meta-scheduler), in the form of a task request. Depending on whether we have a centralized or distributed architecture, the meta-scheduler has a complete or partial knowledge of the resource utilizations. Based on this information, it executes the scheduling algorithm and returns to the user the assignment decision and an estimated *starting time* (ST_i) after which the task i is going to be executed at the selected resource. The user immediately sends the task to the selected resource. We assume that shortest path routing is used, and the communication delays consist of the propagation and transmission delays. Queuing delays at intermediate nodes are assumed to be negligible.

In the centralized architecture the starting time ST_i estimation is always correct, since the central meta-scheduler has assigned all the previous tasks to the resources and thus has a complete knowledge of the resources' utilization, assuming that tasks' workloads are known in advance. In the case of a distributed scheduling architecture, however, the computed ST_i can be incorrect, since the utilization of a resource may have changed by the time the task arrives at that resource. This is a problem that cannot be avoided by any distributed algorithm in a network that has non-zero propagation delays.

Upon its arrival at a computation resource a task is locally queued and requests to be executed at the estimated ST_i. If we are using a distributed scheduling architecture and one or more other tasks have arrived at this specific resource after the task assignment decision at the meta-scheduler was performed, we have a racing conflict and task i may not be executed at the predicted ST_i. When a task finishes its execution, the resource sends the results to the originating user.

3.2. Burst routing problem

The routing and data scheduling problem in a burst routed Grid network is defined as follows. We are given a network with links l of known propagation delays d_l and capacities C_l and a source node that requests to send a burst i of size equal to I_i bits to some destination node. We are also given the utilization profiles of all links l. We want to find a feasible path over which the burst should be routed and the time at which the burst should start transmission, so as to optimize some performance criterion, such as the reception time of the burst at its destination.

In the centralized scenario a node of the network plays the role of the centralized Bandwidth Broker (BB). In that case, similar to [22], there is a single BB in the network domain that records information, including topology, routing information, bandwidth reservations, existing data flows, etc., for the whole domain. When a burst wishes to be transmitted, a request is sent to the centralized BB, which executes a routing and data scheduling algorithm and communicates the outcome (path and time offset after which the transmission should start) back to the source. On the other hand, in the distributed scenario each node separately maintains information for the whole network. A node wishing to send a burst executes locally the algorithm to obtain the path to be followed and the time to start transmission, and informs the other nodes of its decision.

After choosing the best available path, a tell-and-go or a tell-and-wait reservation scheme is used to transmit the burst. If we have a distributed architecture, contentions may occur. Specifically, if a tell-and-go scheme is used, the burst is not guaranteed to arrive at the destination, since the utilization profile at some intermediate link may have changed by the time the setup packet arrives at that link, in which case the burst will be dropped. Similarly, if a tell-and-wait scheme is used, the reservation of the path may fail, in which case the burst will be blocked but not dropped. This is a problem that cannot be avoided by any routing and scheduling algorithm in a network that has non-zero propagation delays. In a centralized architecture, however, the central BB has complete knowledge of the utilization of all the links, since it has routed and scheduled all previous bursts, and thus no contention occurs. In the centralized version there is no reason for employing a two-way reservation scheme and thus in order to obtain smaller delays we assume that a one-way JET protocol is always employed.

4. Clusters utilization profiles, utilization update messages and meta-scheduling algorithms

In this section, we present the *clusters availability profile*, a data structure that can be used by a meta-scheduler (centralized or distributed) to monitor the utilization profile of a computation resource (cluster). We assume that a computation resource P consists of N_P number of processors that all have equal computation capacity C_P (MIPS). The utilization profile U_P (t) of cluster P is defined as an integer stepwise function of time, which records the number of processing elements already committed to tasks at time t relative to the present time. The maximum value of this function is the number of processors N_P, while the stepwise character of $U_P(t)$ is due to the discontinuities of height r_i (always integer) at the starting and ending times of task i. In case all tasks request a single processor, the discontinuities are always unitary. We also define the *clusters availability profile*, which gives the number of processors $C_P(t) = N_P - U_P(t)$ that are free as a function of time. Fig. 2 shows an example of a cluster availability profile.

In the case of the centralized architecture, the central meta-scheduler maintains a *cluster availability vector* $C_P(t)$ for every computation resource P. These profiles always contain up-to-date information on the resources' utilization, assuming the task execution times are known in advance. In the distributed architecture, each distributed meta-scheduler maintains a "picture" of the resource utilizations by storing one vector $C_P(t)$ for every computation resource P. The locally maintained availability vectors are updated by exchanging update messages (as described in Section 4.2).

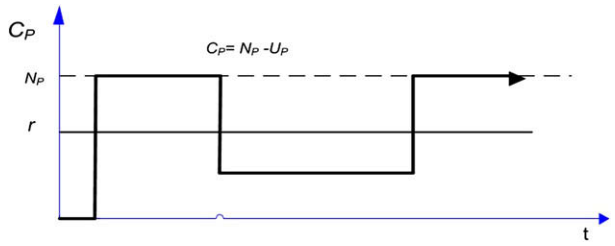

Fig. 2. The cluster availability profile $C_P(t)$ of a given cluster (computation resource) P with N_P processors, each of equal processor computation capacity C_P. A task requests to be executed in r processors.

The cluster availability vectors maintained at a meta-scheduler are used in the meta-scheduling algorithm. More specifically, when a task i requesting r_i processors and having computation workload W_i per processor arrives at the meta-scheduler, the algorithm searches the profiles $C_P(t)$ of all computation resources P for suitable placements of the task. A cluster P can execute a task if we can find a continuous period of duration W_i/C_P during which more than r_i processors are available.

To simplify the presentation and the experiments, we will assume in the remaining of this study that each task i requests $r_i = 1$ processors, which is the most usual case.

4.1. Utilization update messages

A centralized architecture does not require the exchange of resource utilization information, since the central meta-scheduler assigns all the tasks to the computation resources and can compute exactly the time periods the tasks are going to execute on them. However, in a distributed architecture, each distributed meta-scheduler needs to keep track of the utilization of all the resources to efficiently assign the tasks. In order to synchronize the locally maintained "picture" with the actual resources' utilization, update messages should be sent to each distributed meta-scheduler.

In an *exhaustive update* strategy, update messages are sent every time the utilization of a resource changes. In particular, a utilization update message can be sent at the time a task arrives to be executed at a resource. At that time, the local queue scheduler can calculate the time δ_i at which that specific task will start execution, taking into account the tasks already scheduled ahead of it or under execution. Update messages are routed in the Grid network over shortest paths. However, in order to maintain the control plane overhead low, update messages are not sent to all distributed meta-schedulers, but only to those that can use this information. A meta-scheduler is not informed about a task assignment to a resource if the task will complete its execution before the earliest time at which the meta-scheduler could send a task to that specific resource.

More precisely, consider a resource (cluster) P on which task i, is going to execute after time δ_i. Resource P informs the distributed meta-scheduler M that a task is going to start execution after time δ_i and for duration W_i/C_P if

$$2 \cdot d_{P,M} \leqslant \delta_i + \frac{W_i}{C_P}, \tag{1}$$

where $d_{P,M}$ is the shortest path delay from P to M. Update messages are sent to all meta-schedulers that satisfy Eq. (1). All other meta-schedulers are not informed about this particular reservation since they cannot use this information. This is because the update packet will reach M after time $d_{P,M}$, and the earliest time after which M can assign a task to P is after an additional $d_{P,M}$, for a total time of $2 \cdot d_{P,M}$, assuming propagation delays are the same in both link directions. A distributed meta-scheduler that receives a utilization update message, uses this information to update the cluster availability profile $C_P(t)$ maintained locally for computation resource P.

4.2. Employed meta-scheduling algorithms

For the purposes of this study we have employed two typical meta-scheduling algorithms and extended them to support advance reservations. A centralized and a distributed version of each algorithm have been implemented.

4.2.1. Earliest Start Time algorithm (EST)

The EST algorithm selects the computation resource (cluster) where the task can start execution earlier. In computing the time at which the task can start execution at a cluster, the algorithm takes into account the propagation delay and the cluster's utilization information (accurate or possibly outdated, depending on the selected architecture). More formally, assuming that user U wants to execute a task i that has computation workload W_i and deadline D_i, the EST algorithm calculates for each cluster P of computation capacity C_P the earliest start time $EST_{P,i}$, defined as the earliest time, greater than the propagation delay $d_{U,P}$ from U to P, after which one processor is available for duration W_i/C_P, given P's availability profile $C_p(t)$. The EST algorithm selects the resource

$$R = \arg\min_{\text{all } P}(EST_{P,i})$$

with the minimum $EST_{P,i}$, and the starting time of task i is defined as $ST_i = EST_R$.

4.2.2. Earliest Completion Time algorithm (ECT)

The ECT algorithm computes for a cluster P the earliest completion time $ECT_{P,i}$, of task i on resource P, defined as the earliest time, greater than the propagation delay $d_{U,P}$ from U to P, at which one processor is available for duration W_i/C_R, given $C_p(t)$. The algorithm selects the resource

$$R = \arg\min_{\text{all } P}(ECT_{P,i})$$

with the minimum $ECT_{P,i}$, and the starting time of task i is defined as $ST_i = ECT_{R,i} - W_i/C_R$.

When a task arrives at the resource it requests to be executed at time ST_i. In the distributed architecture, it is possible that this is no longer feasible (because other distributed meta-schedulers have in the meantime sent tasks to that resource so that it is occupied at time ST_i), in which case we say that we have a racing conflict. A conflict is solved locally at the cluster by applying a simple algorithm that searches for the earliest placement of the task, taking into account the already scheduled tasks. An update message is then sent to inform the distributed meta-schedulers for this particular reservation.

5. Link utilization profiles, utilization update messages and burst routing and scheduling algorithms

In the optical burst routing paradigm, each node needs to keep a record of the capacity reserved on its outgoing links as a function of time [15] in order to perform channel scheduling and reservation. In this section, we present the *link availability profile*, which is a data structure that can be used by a meta-scheduler (Bandwidth Broker) to track and monitor the utilization profile of a communication resource (link).

Similarly to Section 4, the *utilization profile* $U_l(t)$ of a link l is defined as a stepwise binary function with discontinuities at the points at which bandwidth reservations begin or end, and is updated dynamically with the admission of each new session or burst that reserves bandwidth for a given time duration. We define the *capacity availability profile* of link l of capacity C_l as $C_l(t) = C_l - U_l(t)$.

In the case of the centralized architecture, the central BB keeps track of the utilization of all links by maintaining a utilization database for all the links that is always up-to-date. In the distributed architecture, each node maintains an estimate of the utilization of all the links. This local picture can be different among the nodes and depends on their position in the network, for networks with non-zero propagation delays. Update information is communicated among nodes to synchronize the locally maintained profiles with the actual utilization information.

5.1. Link utilization update messages

Similarly to Section 4.1, link utilization update messages are required only in a distributed architecture. In an exhaustive update strategy, link update messages are sent at the time a link is reserved. Assuming a reservation is made for link l starting at time σ_i relative to the current time, the router R that handles link l informs the Bandwidth Broker of node M that a burst is going to pass after time σ_i and duration I_i/C_l if

$$2 \cdot d_{R,M} \leqslant \sigma_i + \frac{I_i}{C_l}, \tag{2}$$

where $d_{R,M}$ is the shortest path delay from R to M.

5.2. Burst routing and scheduling algorithms

We have implemented two burst routing and scheduling algorithms, each in a centralized and a distributed version. The algorithms presented here consider the burst routing and scheduling problem jointly and thus return not only the path to be followed (routing decision) but also a time offset (TO) after which the source should transmit the burst (scheduling decision).

The first algorithm we used is the Availability Weighting heuristic multicost algorithm (AW) introduced in [20]. The AW algorithm consists of two phases. In the first phase it computes a set of candidate so-called non-dominated paths for the given source–destination pair. More precisely, in the AW algorithm each link is assigned a vector of cost parameters that includes a discretized utilization profile, and by defining appropriate "combining" operations for the link cost parameters the cost vectors of the paths are calculated. In order to compute the set of non-dominated paths for a given burst and source–destination pair, the multicost algorithm also requires the definition of a domination operation between two paths. More specifically, we say that a path p_1 dominates another path p_2, for a given burst, if all the parameters in the cost vector of p_1 are better than the corresponding parameters of p_2. In the second phase of the algorithm, we apply an optimization function to the cost vectors of the non-dominated paths, which takes into account the burst parameters and QoS requirements, in order to select the optimal path. Specifically, the optimal path is defined as the one that results in the minimum reception time of the burst at the destination.

The second algorithm examined is the Dijkstra shortest path algorithm with collision avoidance (D/CA) that was also presented in [20]. In this algorithm, the shortest paths of all source–destination pairs are computed at the beginning of the simulation. Upon

receiving a burst transmission request, the source/ingress node combines the utilization profiles of the links over the shortest path and schedules the transmission so as to avoid contention at subsequent nodes.

To serve a burst, in the case of the distributed architecture, the source executes the algorithm, taking into account whether a one- or a two-way reservation scheme is used, and schedules the burst accordingly. A SETUP packet is forwarded on the chosen path. If the SETUP phase is dropped or blocked at an intermediate node (which may happen when another reservation has been performed in the meantime) a REJ packet is sent to notify the source about the rejection. The source waits for a small period of time (back-off) and handles the blocked request as a new transmission request, by recomputing a path and scheduling the burst transmission accordingly. In order to reserve the appropriate resources and set up the lightpath, we have experimented with two connection establishment protocols, one from each family of reservation protocols. More specifically, we have used a variation of the JET protocol (that employs REJs for burst retransmissions) and a two-way protocol with timed reservations that we call Wait-For-Reservation (WFR) protocol.

In the case of the centralized architecture, the source forwards the transmission request to the central BB, which, in contrast to the distributed case, has complete knowledge of the utilization of the links. Since contentions do not occur, there is no reason for employing a two-way scheme and thus to obtain a low average end-to-end delay we have used a one-way JET protocol.

To sum up, we have experimented with two routing and data scheduling algorithms (AW and D/CA) when these are implemented either in a centralized or distributed scheduling environment. In the distributed approach, we have examined the performance of AW and D/CA algorithms when a JET and a WFR protocol is used, while in the centralized approach we only examined the use of the JET protocol.

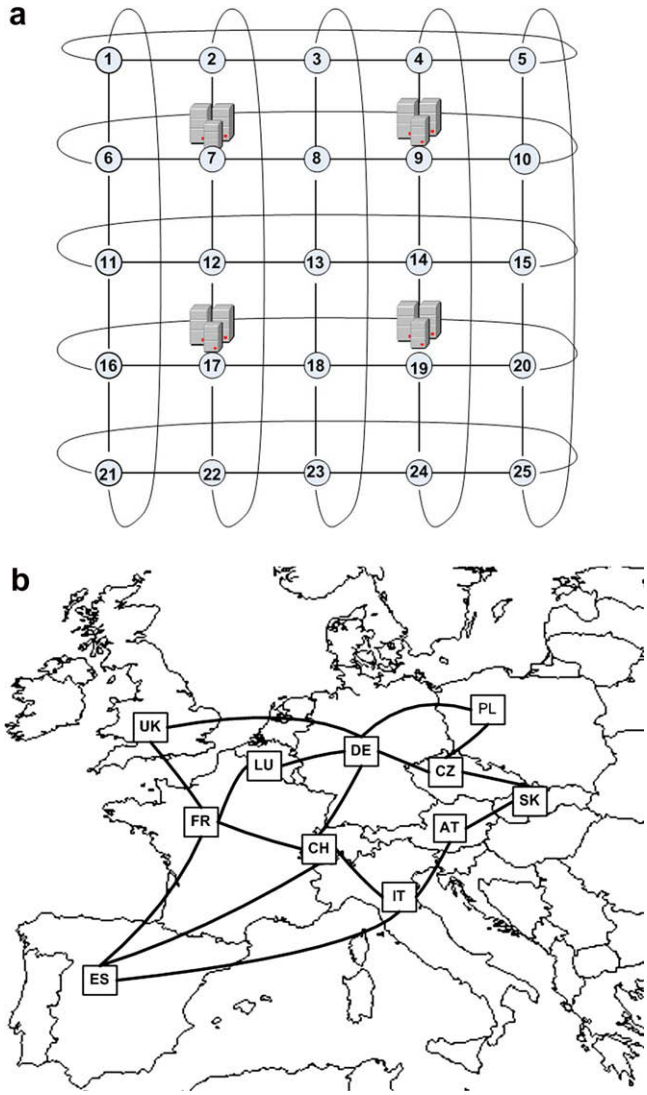

Fig. 3. (a) 5×5 mesh with wraparounds topology and (b) a realistic Pan-European network topology. Both topologies were used in the simulation experiments.

6. Performance results

In order to assess the performance of the distributed and the centralized versions of the aforementioned algorithms, we have conducted full network simulation experiments, by extending the ns-2 platform [30] and incorporating the appropriate entities. ns-2 is a discrete event simulator and is a manageable environment for simulating the network resources of the Grid, which is particularly important for the evaluation of the examined distributed and centralized scheduling architectures. More specifically, for the task scheduling problem we have designed appropriate cluster utilization data structures, developed appropriate ns-objects for users, computation resources and meta-schedulers, and defined a format for the update messages. Similarly, for the routing and data scheduling problem, we have designed appropriate link utilization profiles, developed ns-objects for users, burst routers and bandwidth brokers, and defined a format for the link update messages.

6.1. Task scheduling performance results

6.1.1. Parameters and metrics

For the simulations, we have assumed two network topologies: (i) a regular 5×5 mesh with wraparounds topology (Fig. 3a) and (ii) a more realistic topology that is based on a Pan-European Grid network (Fig. 3b). In the mesh topology, the distance between neighboring nodes was initially set to 400 km (20 ms link propagation delay), but we also experimented with distances of 100 and 1600 km (5 and 80 ms propagation delay, respectively). Four computation resources (clusters) were placed randomly in the network and the results were averaged over 5 runs with 30,000 tasks per run. In the Pan-European topology, the distances between the nodes are given by the actual strait-line distances of the capitals of the countries that participate in the corresponding topology, multiplied by a factor of 1.2. When we used a centralized architecture, the central meta-scheduler was also randomly placed. We have examined the performance for both homogenous and inhomogenous resources, assuming that we have either (i) an equal number of processors per resource all with equal computation capacity (Section 6.1.2), or (ii) an uneven number of processors per resource and different processor computation capacities between resources (Section 6.1.3).

Users were placed at all the 25 nodes (11 in the Pan-European Grid network) of the network and tasks were generated following a Poisson distribution with rate λ tasks per second. When a task is created its owner is chosen with equal probability among the 25 users (11, respectively). The assumption of a Poisson task generation process is driven by a modeling study that was performed for an operational Grid network, and in particular for the EGEE/LCG Grid infrastructure, and was reported in [31]. It was shown in that work that the task arrival process at the EGEE/LCG Grid infrastructure at the Grid level can quite accurately be modeled as a Poisson process. The computation workload of a task follows

an exponential distribution with average W millions instructions. Unless explicitly stated otherwise, the average computation workload of the tasks was taken to be W = 1,500,000 MI.

In order to assess the performance of the task scheduling algorithms, we used the following performance metrics:

- Average total execution delay. The total execution delay of a task is defined as the time between the task's creation and the time at which the task finishes its execution on a computation resource.
- Conflict probability. This metric is defined only for the distributed architectures and records the frequency of racing conflicts observed. A racing conflict occurs when the starting time ST of a task predicted by the scheduling algorithm is not met due to reservations made by other distributed schedulers in the meantime.
- Average number of exchanged messages, which measures the communication overhead of the employed distributed algorithms. In the centralized version the average number of exchanged messages is always two messages per task request (question to/reply from the central meta-scheduler). In a distributed scheduling architecture, however, there is a considerable control plane overhead (update messages, as presented in Section 4.2) needed to "synchronize" the meta-schedulers. We have assumed an exhaustive update strategy, where a resource, upon receiving a task, informs all meta-schedulers that satisfy Eq. (1) about the period during which the task is going to be executed.

6.1.2. Resources with equal number of processors and equal computation capacity

In the experiments of this subsection, we assume that the four computation resources have an equal number of processors of the same computing capacity. More specifically, each resource consists of a cluster of $N = 15$ processors, each of computational capacity $C_P = 25,000$ MIPS (a typical value for Xeon processors). Note that when all the resources have processors of equal computation capacity, the performance of the ECT algorithm is identical to that of the EST algorithm. Thus, in the figures of this section we graph only the performance of EST.

Note that, since in our experiments the tasks do not have deadlines, and the queue sizes at the clusters (computation elements) are assumed to be infinite, all the tasks that are generated are served, as long as we operate in the stability region. If we view the Grid network as a single system, then the workload (computation or communication) that is generated per unit of time should be on the average less than some upper bound determined by the topology and the (computation or communication) capacities of the Grid network. For example, in this set of experiments, we have four clusters each with 15 CPUs each of computation capacity equal to 25,000 MIPS. Thus, the total computation capacity of the Grid network is 1,500,000 MIPS. For task generation, we use a Poisson process with average rate λ tasks per second and each task has on average workload 1,500,000 MI. Based on these values, we can see that the arrival rate λ cannot exceed 1 tasks/s. Otherwise, the size of the queues will continuously grow and the Grid will no longer operate in the stable region. This, of course, has to do with the number of cluster sites, the number of CPUs and the processing capacity of these CPUs. By changing these parameters, arrival rates higher than 1 tasks/second can be served.

Fig. 4 shows the performance of the EST algorithm when the average task arrival rate takes values between 0.1 and 1 tasks/s. From Fig. 4, we can observe that the distributed version of the EST algorithm exhibits smaller average total delay. The average total delay is dominated by the task execution time $W/C_P \sim 60$ s, while the task load affects the performance when the load takes large values. The conflict probability of the distributed case is presented in Fig. 5.

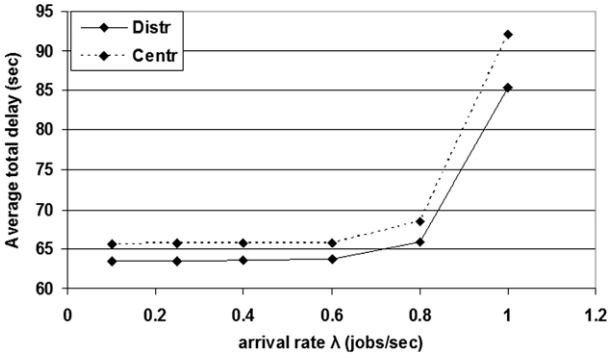

Fig. 4. Effect of task arrival rate λ on the average total task delay. This scenario corresponds to homogeneous computation resources.

Fig. 5 shows the performance of the EST algorithm when the distance between neighboring nodes is set equal to 100, 400, or 1600 km. As expected, the average task delay increases with increasing distance. From Fig. 5a, we can observe that as the distances between the nodes increase, the performance difference between the distributed and centralized architectures increases as well. This is because the increase in the network distances affects vastly the centralized architecture by increasing the time required to question and obtain the reply from the central meta-scheduler. The average total delay is dominated by the task execution time, while the propagation delay plays a secondary role. Fig. 5b presents the conflict probability only for the distributed version of the EST algorithm. As expected, the conflict probability increases as the task arrival rate and the network distances increase. This is because the increase in the arrival rate or the network distance

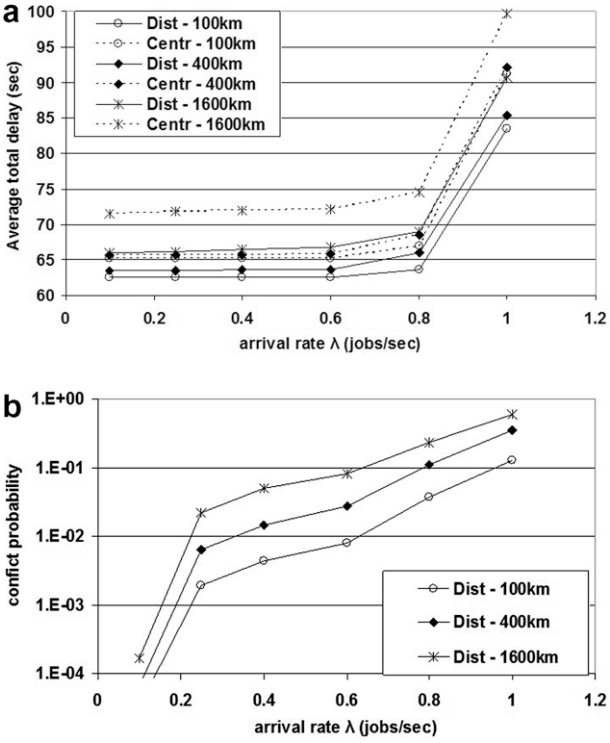

Fig. 5. Effect of the network distances (propagation delay) on the: (a) average total task delay and (b) conflict probability (only in distributed algorithm). The distances between neighboring nodes were 100, 400, and 1600 km.

increases the probability of having two (or more) distributed meta-schedulers deciding to schedule a task on the same resource at overlapping periods, before each of them learns of the decision of the other.

The average number of update messages exchanged, obtained only for the distributed architecture, was almost in all cases constant and close to the upper bound 25 (equal to the number of users and thus equal to the number of distributed meta-schedulers), and for this reason we do not provide graphs for this metric. This is because the average execution time of a task is $W/C_P \sim 60$ s, which according to Eq. (1) is quite large and gives enough time for the update information to reach all the distributed meta-schedulers, even in the case that the distance between adjacent nodes is 1600 km.

6.1.3. Resources with uneven number of processors and uneven computation capacity

In this subsection, we present results for the case where the four computation resources have an uneven number of processors and different processor computational capacities. More specifically, two of the computation resources are clusters consisting of 20 processors of computational capacity 20,000 MIPS per processor, and the remaining two resources are clusters consisting of 10 processors of computational capacity 35,000 MIPS per processor. Note that the overall computation capacity of the Grid network is equal to that assumed in the previous subsection. With this in-homogeneous setting of computation elements, we have performed experiments for both the regular 5×5 mesh network and the more realistic Pan-European network.

Fig. 6 shows the performance of the distributed and the centralized versions of the EST and ECT algorithms when the average task arrival rate takes values between 0.1 and 1 tasks/s. We observe that the EST exhibits close to constant performance, while the ECT performs much better for low load, and deteriorates rapidly as the load takes large values. This can be explained by the fact that ECT is a greedy algorithm (when compared to EST) that tries to schedule each task to the more powerful cluster. For a series of tasks of various lengths, the EST algorithm will distribute them more evenly, while the ECT will try to achieve the best for each task separately, occasionally having a negative effect on the overall performance. The performance of EST algorithm is similar to that presented in the previous section where all the resources had the same number of processors and equal computation capacities.

Fig. 7 shows the performance of the ECT algorithm when the distance between neighboring nodes was set to 100, 400, and 1600 km. We observe that as the network distances increase, the improvements obtained by employing a distributed architecture become more pronounced.

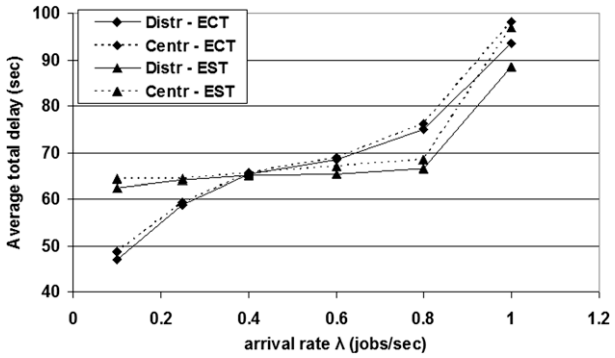

Fig. 6. Effect of task arrival rate λ on the average total task delay for the 5×5 mesh topology. This scenario corresponds to in-homogeneous computation resources.

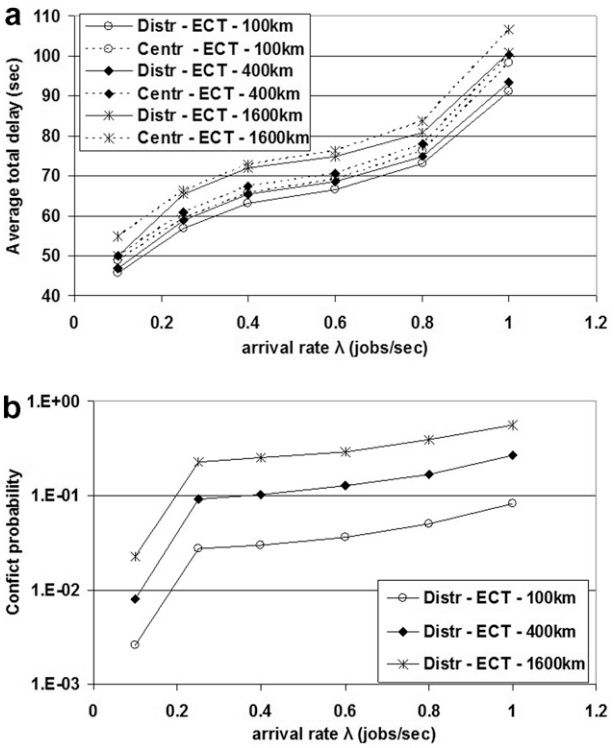

Fig. 7. Effect of the network distances (propagation delays) on the: (a) average total task delay and (b) conflict probability (only in distributed algorithm). The distances between neighboring nodes of the 5×5 mesh network were set to 100, 400, and 1600 km.

Fig. 8a shows the average total delay for the distributed and centralized versions of the EST and ECT algorithms assuming the realistic Pan-European Grid network topology. The average task arrival rate λ takes values between 0.1 and 1 tasks/s. We again observe that the ECT algorithm exhibits better performance for low arrival rates λ, while when λ increases the performance of the EST becomes better, since it has a larger maximum achievable throughput (stability region). In this case, the performance of the ECT algorithm deteriorates faster than in the case of the 5×5 mesh topology. Again, from these results we observe that the distributed algorithms perform slightly better than the corresponding centralized algorithms. Comparing the delay reported for the Pan-European Grid network case to the delay reported for the 5×5 mesh topology we also observe that the tasks in the Pan-European network tend to be executed with less delay (notice the different scales used in the corresponding graphs). This is because the Pan-European network is more compact and its physical diameter is almost half that of the mesh 5×5 network. Although the connectivity degree of the Pan-European network is smaller, this does not affect the execution of the computation tasks, but mainly the network performance.

Fig. 8b shows the conflict probability for the distributed versions of the EST and ECT algorithms. As expected, the ECT algorithm exhibits higher conflict probability than EST due to its greedy nature. As the arrival rate λ increases, the conflict probabilities of both algorithms converge. This might seem at first glance to contradict the results reported for the average delay but can be explained as follows. When the arrival rate λ is low, although the ECT algorithm exhibits higher conflict probability, the percentage of tasks that experience a conflict is low and finally ECT manages to utilize more efficiently the available in-homogeneous clusters. As the task arrival rate increases the conflict probability also increases. The EST algorithm distributes the computational load

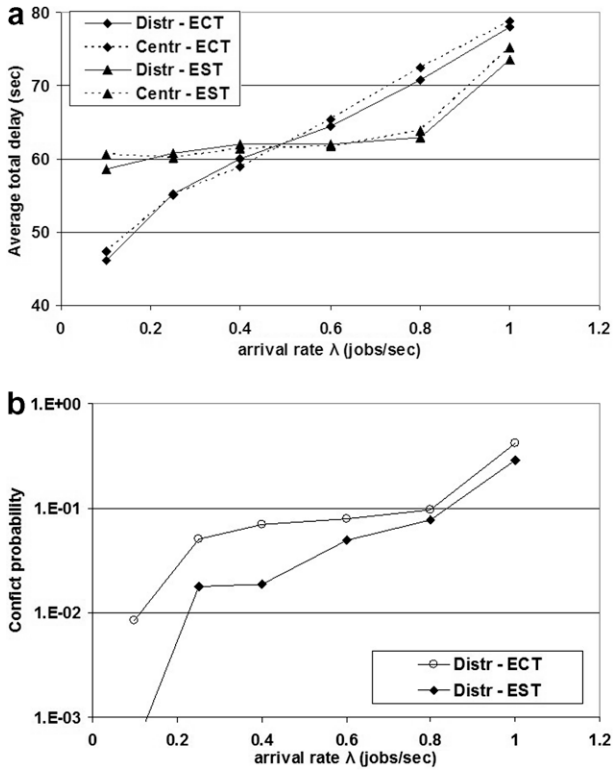

Fig. 8. Effect of task arrival rate λ on the average total task delay for the realistic Pan-European network topology. (a) Average total delay per task and (b) conflict probability for the distributed versions of the examined algorithms.

more evenly among the processors and also balances the load in the network, while the ECT algorithm tries to greedily assign the tasks to the best cluster(s). However, for high task arrival rates the utilization update process becomes increasingly inefficient. It stands to reason that the ECT algorithm is more sensitive to this outdated utilization information effect than the EST algorithm.

6.2. Routing and burst scheduling performance results

6.2.1. Parameters and metrics

In this section, we turn our attention to the burst routing and scheduling (communication) problem, and present simulation results obtained for the same 5×5 mesh with wraparounds topology of Fig. 3a. The neighboring nodes were again placed at a distance of 100, 400, and 1600 km from each other (with link propagation delays 5, 20, and 80 ms, respectively). The processing delay of the set-up packets was set to 0.02 μs and the link bandwidths C_l were assumed equal to 1 Gb/s. Bursts transmission requests arrive at each edge node according to a Poisson process with rate λ requests per second and their destinations are uniformly distributed over all remaining network nodes. Burst sizes were assumed to follow the exponential distribution with mean value equal to $\bar{l}$ bits.

For the experiments in this section, unless explicitly stated, the distance between adjacent node was 400 km, the mean burst size $\bar{l}$ was set equal to 100 MB, and the arrival rate λ of the bursts varied between 0.1 and 1 bursts/s per source node.

We have used the following metrics to evaluate the performance of the routing and burst scheduling algorithms:

- Average end-to-end delay, defined as the average time that elapses between the generation of the burst and the time that it has been completely transferred to its destination.
- Average number of retrials for a successful reservation. This metric is calculated only for the distributed implementations. A retrial occurs when the burst transmission request is blocked (when the JET protocol is used) or dropped (when the WFR protocol is used) in the core.
- Average number of update messages exchanged, which measures the communication overhead of the distributed algorithms. In the centralized versions (as mentioned in Section 6.1.1) the average number of updates is always 2 messages per request (question to/reply from the central BB), while in the distributed scheduling architectures considerable control plane overhead is required to "synchronize" the distributed BBs (as presented in Section 5.2).

6.2.2. Effect of arrival rate

Fig. 9a illustrates the average-end-to-end delay experienced by the bursts. We observe that the multicost heuristic algorithm (AW) outperforms the D/CA algorithms in all examined cases. Regarding the architecture and the signaling protocol used, the distributed JET case exhibits the best performance, followed by the centralized JET case, while the worst performance was observed for the distributed WFR architecture. By comparing the results presented in Fig. 9a to those presented in Figs. 4 and 6, we observe the difference between the time scales of the computation and the communication scheduling problems. Specifically, in the task scheduling problem the execution times dominate the total task delays, which take values in the order of minutes (60 s), while in the burst routing problem the transmission and propagation delays are comparable and thus the network distances play a significant role. Note that in the burst routing problem and under the parameters assumed, the average end-to-end delay takes values in the order of seconds.

Fig. 9b shows the average number of retrials required for a successful burst transmission only for the distributed architecture (recall that there are no contentions in the centralized case). We observe that the WFR protocol results in fewer retrials, while the choice of the routing algorithm (AW or D/CA) also affects this metric, especially at heavy loads. Finally, Fig. 9c shows the average number of messages per transmission request required for the reservation updates in the network (again only for the distributed scenario). We observe that the use of the WFR protocol results in the exchange of more messages, since that protocol requests the reservation of a link one round trip time (RTT) after the setup packet has reached a node, enabling the communication of this reservation information to more distant nodes (according to Eq. (2)).

In Fig. 10, we report the corresponding results obtained assuming the realistic Pan-European Grid network topology. From these graphs, we can again see that the distributed algorithms outperform the corresponding centralized algorithms and that the multicost AW-JET algorithm outperforms the Dijkstra-based D/CA algorithm. However, in these results, we can see that the performance of all the algorithms starts to deteriorate at lower arrival rates than in the case of the 5×5 mesh network. This has to do with the connectivity degree of the two topologies. In particular, the used Pan-European network has smaller connectivity degree than the mesh 5×5 network. As the arrival rate increases, the network becomes more congested and certain links become bottlenecks. This was avoided in the regular mesh network where the topology is homogeneous and highly connected, and there are always a lot of alternative routes to serve the connections. In contrast, in the used Pan-European network, the available routes are rather limited, and contention becomes high even for low arrival rates (Fig. 10b), increasing the average end-to-end delay (Fig. 10a). With respect to the average number of update messages (Fig. 10c) exchanged, we can see that utilization information is able

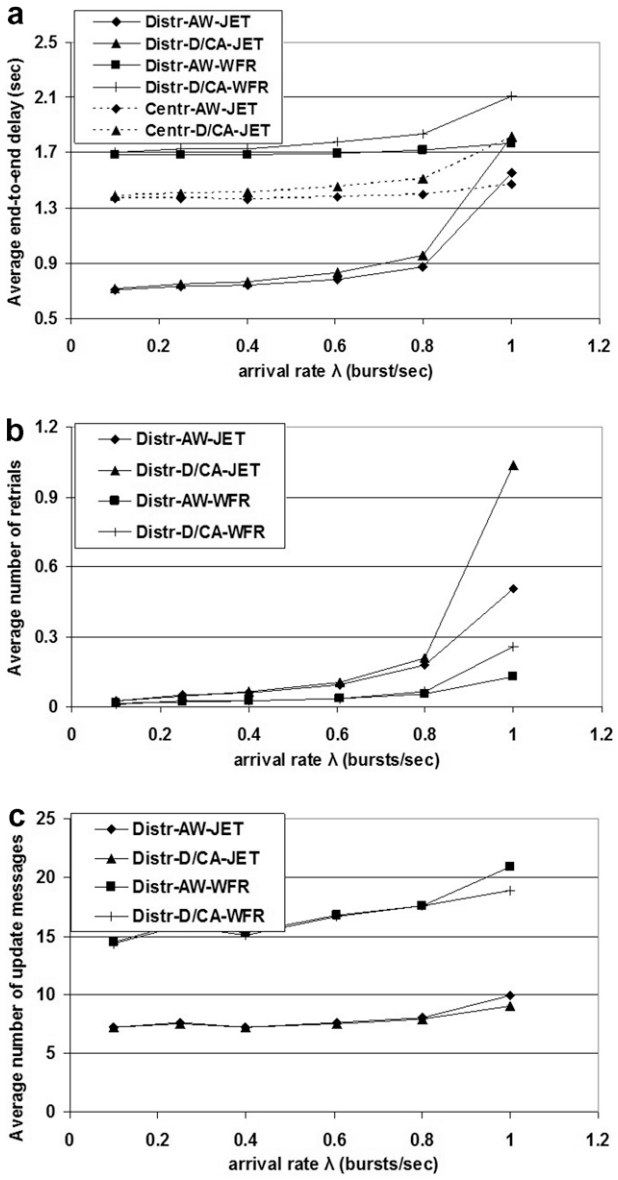

Fig. 9. Effect of the arrival rate λ. (a) Average end-to-end delay per burst, (b) average number of retrials for a successful burst transmission, and (c) average number of exchanged messages. These results correspond to the 5×5 mesh network topology with 400 km distance between adjacent nodes.

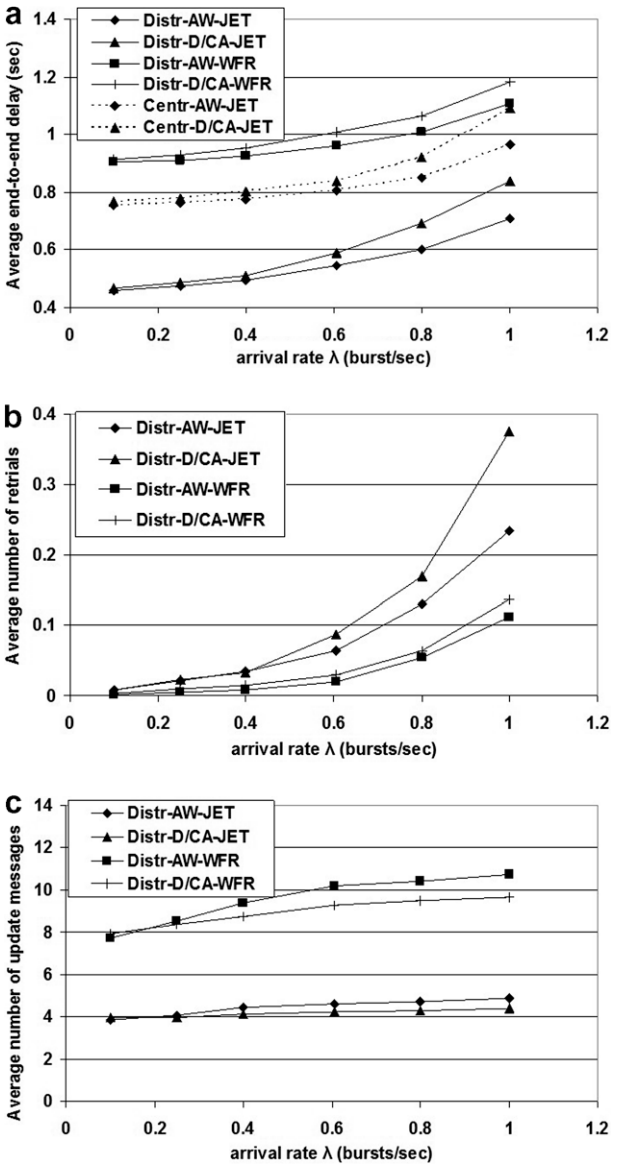

Fig. 10. Effect of the arrival rate λ. (a) Average end-to-end delay per burst, (b) average number of retrials for a successful burst transmission, and (c) average number of exchanged messages. These results correspond to the realistic Pan-European network topology.

to reach almost all nodes (12 in total) of the network when two-way reservation is performed (WFR protocols).

6.2.3. Effect of the network distances

The network propagation delays play a significant role on the link state update mechanism used in the distributed architecture. Cleary, information regarding a link reservation cannot be used by a distributed BB if it reaches that BB after the time that it is useful Eq. (2). On the other hand, the increase in the propagation delays also has a negative effect in the case of a centralized architecture, since it increases the delay required to question and obtain an answer from the central BB. Thus, we expect the increase in network distances to negatively impact the performance of both the centralized and the distributed architectures.

Fig. 11 shows the performance of the better performing AW algorithm, in its centralized and distributed JET versions, when the adjacent node distances were set to 100, 400, or 1600 km. As expected, the average end-to-end delay deteriorates as the network propagation delays increase, in both the centralized and distributed versions (Fig. 11a). The distributed version of the AW algorithm always performs better than its corresponding centralized version. Fig. 11b shows the average number of retrials required for a successful burst transmission only for the distributed architecture. As the network distances increase, more time elapses between a reservation and the time this information reaches the distributed schedulers. Thus, the bandwidth brokers do not use up-to-date information, more burst are dropped in the core of the network and we end up with more retrials. This phenomenon is similar to the racing conflict observed in the task scheduling problem. Fig. 11c shows the average number of update messages exchanged per request. We observe that as the network distances increase, fewer update messages are exchanged (something that can be explained by Eq. (2)). Note that an update message is generated when a link is reserved. Thus, although in total we have

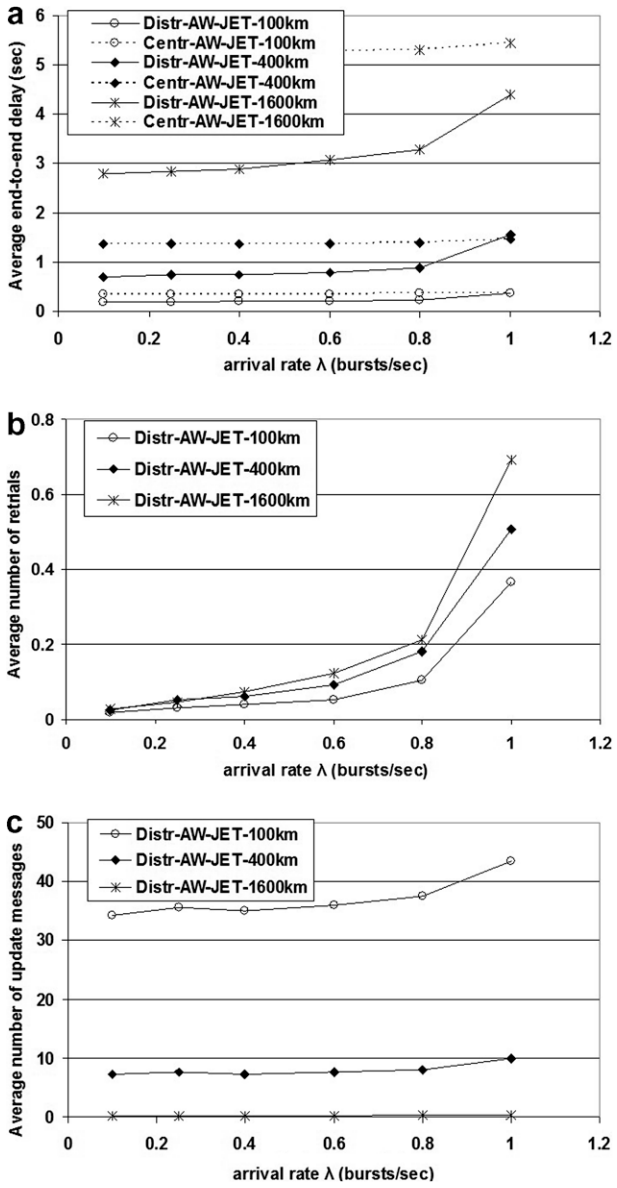

Fig. 11. Effect of the network distances (propagation delays) on the: (a) average end-to-end delay per burst, (b) average number of retrials for a successful burst transmission, and (c) average number of exchanged messages. The distances of adjacent nodes were 100, 400, and 1600 km.

25 distributed bandwidth brokers, more than 25 messages can be generated per burst, depending on the chosen path.

7. Conclusions

We compared the performance of centralized and distributed architectures for scheduling computation and communication tasks in Grid networks. Regarding computation tasks, we examined two typical online task scheduling algorithms that incorporate advance reservations and performed full network simulation experiments to measure their performance when they are implemented in a centralized or in a distributed manner. Similarly, for communication tasks, we compared two routing and data scheduling algorithms that were implemented in a centralized or a distributed manner. We also examined the effect network propagation delays have on the performance of these algorithms. Our simulation results indicate that a distributed architecture with an exhaustive utilization update strategy results in better average end-to-end delay performance for computation or communication intensive tasks than a centralized architecture employing the same algorithm. The exhaustive update strategy requires a larger number of utilization update messages, but this does not make this strategy unrealistic, since in operational Grid networks a high number of messages are exchanged for monitoring and other purposes.

Acknowledgements

This work has been supported by the European Commission through the Phosphorus project. http://www.ist-phosphorus.eu/.

References

[1] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, second ed., Morgan Kaufmann, Los Altos, CA.
[2] K. Li, Experimental performance evaluation of job scheduling and processor allocation algorithms for grid computing on metacomputers, in: International Parallel and Distributed Processing Symposium (IPDPS), Santa Fe, NM, 2004, pp. 170–177.
[3] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, R. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837(28).
[4] S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, D. Grushin, N. Kuzjurin, A. Pospelov, A. Shokurov, Comparison of scheduling heuristics for grid resource broker, in: Mexican International Conference in Computer Science (Enc'04) – Volume 00, ENC, IEEE Computer Society, Washington, DC, September 20–24, 2004, pp. 388–392.
[5] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic models for resource management and scheduling in grid computing, special issue on grid computing environments, Journal of Concurrency and Computation: Practice and Experience 14 (2002) 1507–1542.
[6] J. Xiao, Y. Zhu, L. Ni, Z. Xu, GridIS: An incentive-based grid scheduling, in: International Parallel and Distributed Processing Symposium (IPDPS), 2005.
[7] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: High Performance Distributed Computing Symposium, (HPDC), 2002.
[8] Y. Cardinale, H. Casanova, An evaluation of job scheduling strategies for divisible loads on grid platforms, in: High Performance Computing & Simulation Conference (HPC&S'06), Bonn, Germany, 2006.
[9] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience 32 (2) (2002) 135–164.
[10] W. Smith, I. Foster, V. Taylor, Scheduling with advance reservations, in: Proc. of the 14th International Parallel and Distributed Processing Symposium, IEEE, Washington, Brussels, Tokyo, 2000, pp. 127–132.
[11] E. Elmroth, J. Tordsson, Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions, Future Generation Computer Systems 24 (6) (2008).
[12] C. Qiao, M. Yoo, Optical burst switching (OBS) – a new paradigm for an optical Internet, Journal of High Speed Networks 8 (1) (1999) 69–84.
[13] Grid optical burst switched networks, Available from: <http://www.ogf.org/Public_Comment_Docs/Documents/Jan-2007/OGF_GHPN_GOBS_final.pdf/>.
[14] Y. Chen, C. Qiao, X. Yu, Optical Burst Switching (OBS): a new area in optical networking research, IEEE Network Magazine 18 (3) (2004) 16–23.
[15] E. Varvarigos, V. Sharma, An efficient reservation connection control protocol for gigabit networks, Journal of Computer Networks and ISDN Systems 30 (1998) 1135–1156.
[16] M. Dueser, P. Bayvel, Analysis of a dynamically wavelength-routed optical burst switched network architecture, Journal of Lightwave Technology 20 (2002) 574–585.
[17] E.A. Varvarigos, V. Sharma, The ready-to-go virtual circuit protocol: a loss-free protocol for multigigabit networks using FIFO buffers, Transactions on Networking 5 (5) (1997) 705–718.
[18] J. Turner, Terabit burst switching, Journal of High Speed Networks 8 (1999) 3–16.
[19] J. Teng, G. Rouskas, A comparison of the JIT, JET, and horizon wavelength reservation schemes on a single OBS node, in: International Workshop on Optical Burst Switching (WOBS), 2003.
[20] E.A. Varvarigos, V. Sourlas, K. Christodoulopoulos, Routing and scheduling connections in networks that support advance reservations, Computer Networks 52 (15) (2008).
[21] R. Guérin, A. Orda, Networks with advance reservations: the routing perspective, Infocom, 2000.
[22] Z. Zhang, Z. Duan, Y.T. Hou, On scalable network resource management using bandwidth brokers, NOMS 2002, pp. 169–183, 15–19.
[23] M. Yoo, C. Qiao, S. Dixit, QoS performance of optical burst switching in IP-over-WDM networks, Journal on Selected Areas in Communication 18 (2000) 2062–2071.

[24] T. Coutelen, H. Elbiaze, B. Jaumard, An efficient adaptive offset mechanism to reduce burst losses in OBS networks, GLOBECOM 2005.
[25] A. Zapata, P. Bayvel, Dynamic wavelength-routed optical burst-switched networks: scalability analysis and comparison with static wavelength-routed optical networks, OFC (2003) 212–213.
[26] Lu Shen, B. Ramamurthy, Centralized vs. distributed connection management schemes under different traffic patterns in wavelength-convertible optical networks, ICC 5 (2002) 2712–2716.
[27] J. Cai, A. Fimagalli, C. Guan, Centralized vs. distributed on-demand bandwidth reservation mechanisms in WDM Ring, OFC, 2001.
[28] V. Hamscher, U. Schwiegelshohn, A. Streit, R. Yahyapour, Evaluation of job-scheduling strategies for grid computing, in: International Workshop on Grid Computing, 2000.
[29] Available from: <http://www.itslessons.its.dot.gov/its/benecost.nsf/Lesson?OpenForm&3176B4EBDFF65BD78525728A00766A08%5ELLCats/>.
[30] Network simulator (ns-2), Available from: <http://www.isi.edu/nsnam/ns/>.
[31] K. Christodoulopoulos, V. Gkamas, E. Varvarigos, Statistical analysis and modeling of jobs in a grid environment, Springer Journal of Grid Computing 6 (2007) 77–101.