Joint Communication and Computation Task Scheduling in Grids

Konstantinos Christodoulopoulos[1,2], Nikolaos Doulamis[2], Emmanouel (Manos) Varvarigos[1,2]
[1] *Computer Engineering and Informatics Department, University of Patras, Patras, Greece*
[2] *Research Academic Computer and Technology Institute, Patras, Greece*
kchristodou@ceid.upatras.gr

Abstract

In this paper we present a multicost algorithm for the joint time scheduling of the communication and computation resources that will be used by a task. The proposed algorithm selects the computation resource to execute the task, determines the path to route the input data, and finds the starting times for the data transmission and the task execution, performing advance reservations. We initially present an optimal scheme of non-polynomial complexity and by appropriately pruning the set of candidate paths we also give a heuristic algorithm of polynomial complexity. We evaluate the performance of our algorithm and compare it to that of algorithms that handle only the computation or communication part of the problem separately. We show that in a Grid network where the tasks are CPU- and data-intensive important performance benefits can be obtained by jointly optimizing the use of the communication and computation resources.

1. Introduction

Grids introduce new ways to share computing, storage resources and specific instruments between geographically distributed sites, the management of which requires scheduling at various levels [1]. The complexity of the grid applications, the user requirements and the system heterogeneity would result in inefficient scheduling in the case of a manual procedure. Scheduling tasks on a set of heterogeneous, dynamically changing resources is a complex problem that requires sophisticated algorithms that take into account multiple optimization criteria.

In order for Grid systems to be used in real world commercial applications and demanding scientific experiments, end-to-end Quality of Service (QoS) is desirable. The main approach to providing end-to-end QoS is reservations and especially reservations that are performed "in-advance". Reservations in Grids expand in various domains, such as computation, communication, storage resources and even instruments. The efficiency of a Grid system depends on the development of sophisticated resource management systems capable of allocating resources based on user requirements. However, in order to employ advance reservations, temporal information should be taken into account. This would considerably increase the complexity of the Grid management systems and the related algorithms.

In this paper we propose an algorithm that jointly addresses a communication and computation scheduling problem. We assume that task processing consists of two successive steps: (i) the transfer of data from the scheduler or a data repository site, which we will call source, to the cluster or computation resource (these terms will be used interchangeably in this paper) in the form of a connection or a data burst and (ii) the execution of the task at the cluster. The link utilization profiles, the link propagation delays, the cluster utilization profiles and the task parameters (input data size, computation workload and maximum acceptable delay) form the inputs to the algorithm. The proposed multicost algorithm selects the cluster to execute the task, determines the path to route the input data, and finds the starting times for the data transmission and the task execution at the cluster, by defining advance reservations. The algorithm takes its decisions based on the resources (link and cluster) utilization information available at the scheduler when the algorithm is executed. Note that the proposed algorithm is designed for a distributed architecture and thus the information maintained at the scheduler can be outdated. However, it can easily be extended to function in a centralized manner.

The proposed algorithm consists of three phases: it first uses a multicost algorithm to compute a set of candidate non-dominated paths from the source (scheduler or data repository site) to all network nodes. Secondly, the algorithm obtains the set of candidate non-dominated (path, cluster) pairs from the source to all clusters that can process the task. Finally, the algorithm chooses from the previously computed set a pair that minimizes the task completion time, or some other performance criterion. An important drawback of the algorithm outlined above is that in its first phase the number of non-dominated paths may be exponential. To obtain a polynomial-time heuristic algorithm we use a pseudo-domination relationship between paths to prune the solution space.

We evaluate the performance of the optimal task routing and scheduling algorithm and of its proposed polynomial-time heuristic variation using network simulation experiments, and compare it to that of algorithms that handle the computation or communication part of the problem separately. Our results indicate that when the tasks are CPU- and data-intensive it is beneficial for the scheduling algorithm to jointly consider the communicational and computational problems, as our proposed algorithms do. Comparing the optimal multicost algorithm to the proposed heuristic we observe that they exhibit similar performance in all our experiments. Thus the proposed heuristic combines the strength of the optimal algorithm with a low computation complexity.

The remainder of this paper is organized as follows. In Section 2 we report on previous work. In Section 3 we present the utilization profiles of the communication and computation resources. In Section 4 we formally define the

978-0-7695-3156-4/08 $25.00 © 2008 IEEE
DOI 10.1109/CCGRID.2008.46

problem and show how to compute the utilization profile of a path and the utilization profile of a cluster over a path based on the profiles defined in Section 3. The joint communication and computation scheduling algorithm is presented in Section 5 and its heuristic variation in Section 6. Section 7 presents performance results. Our conclusions follow in Section 8.

2. Related Work

In [2], the Grid Scheduling Architecture Research Group (GSA-RG) of the Open Grid Forum (OGF) presents different Grid scheduling use case scenarios and describes common usage patterns. Among them, the most complicated scenario deals with the scheduling of tasks requesting more than one and possibly different service guarantees. In this context, a "workflow" is defined as a task that consists of a number of "subtasks" with various interdependencies. A workflow requests the co-allocation of resources in different time frames and advance reservations are employed for the orchestration of the corresponding resources.

Grid applications can be categorized as CPU-intensive, data-intensive applications, or both. However, almost all tasks have a computation and a communication part, even if one part is negligible. For example, it is usual for a task to require the movement of a large chunk of data from the location of the user or a data repository site to the computation resource where the task will be executed. In this paper we address such a problem. Note that this problem can be also viewed as a simple form of workflow with two successive steps: a communication part followed by a computation part.

The Globus Architecture for Reservation and Allocation (GARA) [3] is a framework for advance reservations that treats in a uniform way various types of Grid resources. Although GARA has gained popularity in the Grid community, its limitations in coping with current application requirements and technologies led to the proposal of the Grid Quality of Service Management (G-QoSm) framework [4].

Up until now, a number of algorithms that use advance reservations have been proposed for task scheduling on computation resources. In [5] various scheduling algorithms for advance reservations in super-computers are proposed. The Nimrod/G scheduler is presented in [6]. Communication resources are separately taken into account in a bandwidth reservation system within the GARA framework in [7].

Some types of joint communication and computation problems have also been examined. In [8] the authors decoupled the data replication and the computation problems and evaluated the performance of data and task schedulers working in a cooperatively manner. In [9] the proposed scheduler selects the computation resource to execute a task based on the computation resource capability, the bandwidth available from the data host to the computation resource and the cost of the data transfer. Similar algorithms have been examined in multimedia networks where the co-allocation of computation, bandwidth and other resources are examined [10]. The authors in [11] introduced the concept of time scheduling and routing of advance reservation requests in the communication plane, proposed several algorithms for advance reservations, and also discussed complexity issues.

The multicost algorithm we propose considers jointly the time scheduling of communication and computation resources. In comparison to the solutions proposed in [8]-[10] our algorithm also uses advance reservations for the time scheduling of the communication resources, in a similar way to [11]. Multicost algorithms have mainly been used for QoS routing problems. In [12] the authors proved that QoS routing with parameters being the bandwidth and the delay is not NP-complete. The general Multiconstrained Path Problem (MCP) is discussed in [13]. To the best of our knowledge, the present work is the first time a multicost algorithm is used for the joint communication and computation problem in a Grid environment. Moreover, a key difference to other multicost approaches is that the proposed algorithm is designed to handle temporal information, using timeslots as cost parameters in the multicost formulation, in order to cope with the time scheduling of the resources.

3. Communication and Computation Utilization Profiles

3.1. Link Utilization Profiles

In a network that employs advance reservations, each node needs to keep a record of the capacity reserved on its outgoing links, as a function of time, in order to perform channel scheduling and reservations [14].

Assuming each connection reserves bandwidth equal to *r* for a given time duration, the utilization profile U_l(t) of a link *l* is a stepwise function with discontinuities at the points where reservations begin or end, and is updated dynamically with the admission of each new connection. We define the capacity availability profile of link *l* of capacity C_l as $C_l(t)=C_l-U_l(t)$. To obtain a data structure that is easier to handle in an algorithm, we discretize the time axis in steps (timeslots) of duration τ_l and define the *binary r-capacity availability vector* $\hat{C}_l(r)$, abbreviated CAV, as the vector whose *k*-th entry is:

$$\left\{\hat{C}_l(r)\right\}_k = \left\{\begin{matrix} 1, \text{ if } C_l - U_l(t) > r \\ 0, \quad \text{othewise} \end{matrix}\right\}, \text{ for all } (k-1)\cdot\tau_l \le t \le k\cdot\tau_l, \quad k = 1,2,\ldots,z_l$$

where z_l the dimension of the CAV (see Figure 1).

The data structures defined above can be useful in a number of network settings. For example, in an optical WDM network with full wavelength conversion and *w* wavelengths per link, each of capacity C_w, the total capacity of a link *l* is $C_l=w\cdot C_w$. A connection that wants to reserve *k* wavelengths, $1\le k \le w$, requests rate $r=k\cdot C_w$. If we can find a path of available capacity greater or equal to *r*, then the connection can be established. If no wavelength converters are available, each link needs to keep track of the utilization profile of each of its *w* wavelengths separately. Thus, a node has to maintain *w* binary (a wavelength can be reserved or not for a given time) utilization profiles for each outgoing link. In this case the network can be viewed as *w* "parallel" networks, each having a single wavelength.

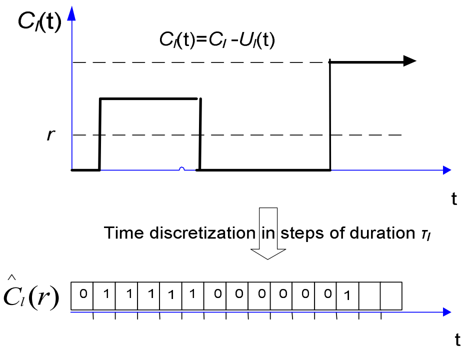

Fig. 1. The capacity availability profile $C_l(t)$, and the binary r-capacity availability vector $\hat{C}_l(r)$ of a link l of capacity C_l. In a similar way we can define the cluster availability profile $W_m(t)$ and the binary r-cluster availability vector $\hat{W}_m(r)$.

In order to simplify the notation, in the remainder of the paper and when no confusion arises, we will denote the profile $\hat{C}_l(r)$ of a link l by $\hat{C}$, suppressing the dependence on l and r.

For the rest of the paper we assume that the network connecting the clusters, the schedulers and the data repositories follows the Optical Burst Switching (OBS) paradigm [15]. An OBS network falls in one of the two aforementioned categories, depending on whether or not wavelength conversion is employed. Note that this assumption does not limit the applicability of our algorithms, which can be used in any network supporting advance reservations.

In OBS networks, the data exchanged are transmitted as data bursts that are switched though the network using a single label. This reduces the switching and processing requirements in the core network. The Grid Optical Bursts Switched (GOBS) solution has been proposed to the Open Grid Forum (OGF) as a candidate network infrastructure to support dynamic and interactive services [16].

3.2. Clusters Utilization Profiles

To have a consistent formulation, we define the cluster utilization profile in a similar way to the link utilization profile. We assume that a cluster-site m consists of W_m CPUs of equal processor speed C_m (measured, e.g., in MIPS). We also assume that when a task starts executing it cannot be preempted. A task requests to be executed in r CPUs ($r \le W_m$), and can be scheduled for execution in the future.

The utilization profile $U_m(t)$ of cluster m is defined as an integer function of time, which records the number of CPUs that have been committed to tasks at time t relative to the present time. The maximum value of $U_m(t)$ is the number of CPUs, W_m, and it has a stepwise character with discontinuities of height r (always integer number) at the starting and ending times of tasks. In case all tasks request a single CPU the steps are unitary. We defined the cluster availability profile, which gives the number of CPUs that are free as a function of time, as $W_m(t)= W_m - U_m(t)$. We discretize the time axis in steps of duration τ_m and defined the binary r-cluster availability vector $\hat{W}_m(r)$, as follows:

$$\left\{\hat{W}_m(r)\right\}_k = \begin{cases} 1, \text{ if } W_m - U_m(t) > r \\ 0, \quad \text{othewise} \end{cases}, \text{ for all } (k-1)\cdot\tau_m \le t \le k\cdot\tau_m, \quad k = 1,2,\ldots,z_m$$

where z_m is the maximum size of the $\hat{W}_m(r)$ vector. To simplify the presentation, we assume that each task requests r=1 CPUs, which is the most usual case. Then, we can denote $\hat{W}_m(r)$ by $\hat{W}_m$ suppressing the dependence on r.

The discretization of the time axis results in some loss of information, and provides a tradeoff between the accuracy and the size of the maintained information. The discretization steps τ_l and τ_m used in the link and cluster utilization profiles, respectively, can be different to account for the different time scales in the reservations performed in these different types of resources and to separately control the efficiency/accuracy we want to obtain in each case. Note that the timeslot-based management of Grid resources is a well established and efficient way to manage utilization information [11],[17]. This approach is already used in various environments, e.g. denoted as *timeslot table* in GARA [3], and used in the VIOLA testbed [18] .

3.3. Utilization Profiles in a Distributed Architecture

In a distributed architecture, each distributed scheduler maintains a “picture” of the utilization of all communication and computation resources. In our approach, this is done by maintaining a utilization database with link and cluster availability vectors for all resources. This picture can be different among the distributed schedulers, due to non-zero propagation delays. Update information (in the form of messages) is communicated to synchronize the locally maintained profiles with the actual utilization. Note that in the case of a centralized architecture only the single central scheduler would have to maintain such information, simplifying in this way the synchronization process. Although the design of a distributed algorithm is more complex, compared to a centralized scheme, a distributed algorithm is more general and applicable to more cases, since it scales better and avoids many drawbacks of a centralized architecture.

4. The Task Routing and Scheduling Problem under Consideration

We are given a Grid infrastructure consisting of a network with links l of known propagation delays d_l and capacity C_l (bps), and a set M of clusters. Cluster $m \in M$ has W_m CPUs of a given processor speed C_m (MIPS). A task is created by a user with specific needs: input data size I (bits) and computational workload W (MI). The user communicates this information to distributed scheduler S (using e.g. JSDL [19]). We assume that the input data are forwarded by the user to the scheduler S or are located at a data repository site R. Thus, S or R comprise the source of the input data. Also, S has (possibly outdated) information about the capacity availability vectors $\hat{C}_l$ of all links l, and the cluster-availability vectors $\hat{W}_m$ of all clusters M. We assume that there is an upper bound D on the maximum delay tasks can tolerate. Even when no limit D is given, we still assume that the dimension z_l and z_m of the link and cluster utilization vectors are finite, corresponding to the latest time (relative to the present time) for which reservations have been made. Given the previous information, we want to find a suitable cluster to execute the task, a feasible path over which to route the data, and the time at which the task should start transmission (from the source) and execution (at the cluster), so as to optimize some performance criterion, such as the completion time of the task. In other words we want to find a (path, cluster) pair and the corresponding Time Offsets, to transmit the data of the task (TO_{path}), and execute the task at the cluster ($TO_{cluster}$). Figure 2 presents an instance of the problem.

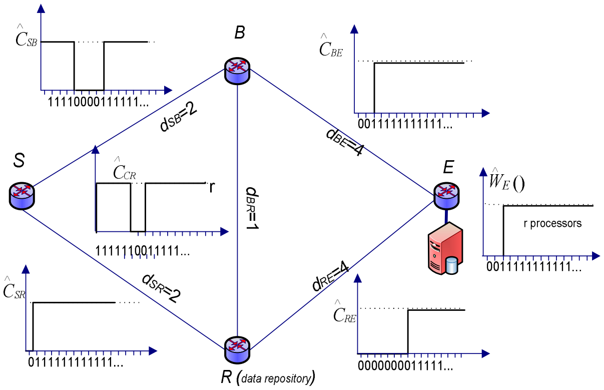

Fig. 2. A task request is forwarded to the distributed scheduler S. The task requires the transmission of a burst with duration $b=I/C_l$ from source (which can be S or a data repository site R) and r CPUs to execute. Each link is characterized by its propagation delay (in τ_l time units) and its binary capacity availability vector $\hat{C}_l$. Node E has a cluster with binary r-cluster availability vector $\hat{W}_E(r)$.

4.1. Binary Capacity Availability Vector of a Path

Assuming the routing and scheduling decision is made at the distributed scheduler S, the capacity availability vectors of all links should be gathered continuously at S. For this and Section 4.2 we assume that the input data are located at S. If the input data were located at a data repository site R, the scheduler S would have to compute the paths starting from R.

To calculate the CAV of a path we have to combine the CAVs of the links that comprise it, by defining an associative operator '**&**', as described in [20]. For example, for the topology of Figure 2, the CAV of path SBE (p_{SBE}), consisting of links SB and BE, is

$$\hat{C}_p = \hat{C}_{SBE} = \hat{C}_{SB} \,\&\, \hat{C}_{BE} = \hat{C}_{SB} \oplus \mathrm{LSH}_{2\cdot d_{SB}}(\hat{C}_{BE})\,, \qquad (1)$$

where $\hat{C}_{SB}$ and $\hat{C}_{BE}$ are the CAVs of links SB and BE, respectively, and LSH() defines the left shift of $\hat{C}_{BE}$ by $2{\cdot}d_{SB}$ (twice the propagation delay of link SB measured in τ_l-time units). Left shifting $\hat{C}_{BE}$ by d_{SB} positions purges utilization information corresponding to time periods that have already expired (time to transfer $\hat{C}_{BE}$ information from B to S), while left shifting it by another d_{SB} accounts for the propagation delay any burst sent from S suffers to reach node B, assuming the link propagation delay is the same in both directions. We finally execute a bit-wise AND operation, denoted by $\oplus$, between the CAVs of SB and BE to compute the binary availability vector of the whole path SBE. This process is depicted in Figure 3.

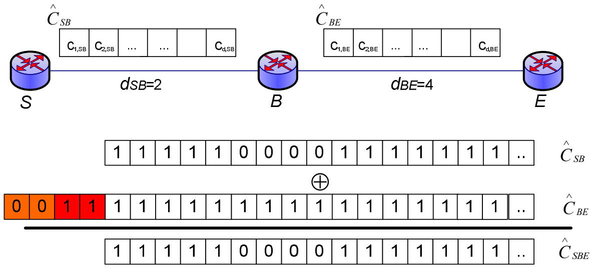

Fig. 3. Calculation of the path capacity availability vector $\hat{C}_{SBE}$. $\hat{C}_{BE}$ is left shifted by $2{\cdot}d_{SB}$ τ_l-time units (d_{SB}=2 in this example), before the AND operation is applied.

4.2. Binary Cluster Availability Vector over a Path

Let p be the path that starts at the distributed scheduler S and ends at cluster m, and let $\hat{C}_p$ be its capacity availability vector and d_p be its delay. We want to transmit a task with data duration b ($b=I/C_l$ where I is the data size) over the path p in order to execute it at cluster m. We define $R_p(b)$ as the first position after which $\hat{C}_p$ has b consecutive ones. In other words, $R_p(b)$ is the earliest time after which a burst of duration b can start its transmission on path p. The earliest time that the task can reach cluster m is then given by $\mathrm{EST}(p,b) = R_p(b) + b + d_p$. The distributed scheduler S has a partial (outdated) knowledge of the cluster availability vector $\hat{W}_m$ of m. We define $\mathrm{MUV}_k(\hat{W}_m)$ as the operation of setting zeros (making unavailable) the first k elements of vector $\hat{W}_m$. Then, vector $\hat{W}_m(p,b) = \mathrm{MUV}_{\mathrm{EST}(p,b)}(\hat{W}_m)$ gives the time periods that S can schedule the task at cluster m over path p.

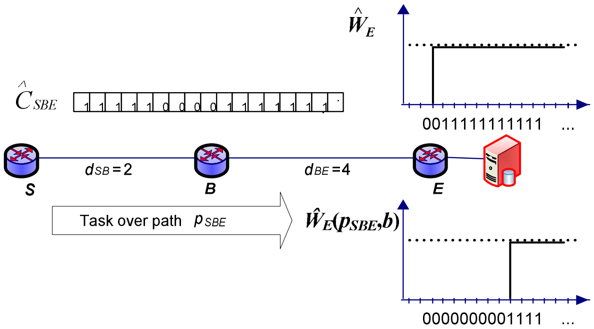

Fig. 4. Scheduler S wants to transfer a task of data duration b=3 over path p_{SBE}. We denote by EST(p_{SBE},b) the earliest time the task can reach E over p_{SBE}, and by $\hat{W}_E(p_{SBE},b)$ the cluster availability vector that gives the periods at which S can schedule the task on E. To calculate $\hat{W}_E(p_{SBE},b)$, we put 0's in the first EST(p_{SBE},b)=9 elements of $\hat{W}_E$.

With respect to Figures 2 and 3, we assume that we want to transmit a task of duration b=3 from S to the cluster at node E, over path p_{SBE} with propagation delay d_{SBE}=6. The capacity availability vector $\hat{C}_{SBE}$ was calculated in Section 4.1, and we have also calculated $R_{p_{SBE}}(b)=0$. The task reaches E after $\mathrm{EST}(p_{SBE},b)=R_{p_{SBE}}(b)+b+d_{SBE}=9$. Also, S has a (possibly outdated) knowledge of the cluster availability profile $\hat{W}_E$. The cluster availability vector that gives the periods that S can schedule the task on E is $\hat{W}_E(p_{SBE},b)=\mathrm{MUV}_{\mathrm{EST}(p_{SBE},b)}(\hat{W}_E)=$ $\mathrm{MUV}_9(\hat{W}_E)$, which is the operation of setting the 9 first entries of vector $\hat{W}_E$ to zero. This calculation is depicted in Figure 4.

5. Joint Communication and Computation Task Scheduling Algorithm in Grids

In what follows we present a multicost algorithm for the joint communication and computation scheduling of tasks. The algorithm consists of three phases: given that the input data are located at source (which can be the scheduler S or a data repository site R) we first calculate the set $P_{n\text{-}d}$ of non-dominated paths between the source and all the network nodes (Section 5.1). We then obtain the set $PM_{n\text{-}d}$ of candidate non-dominated (path, cluster) pairs from source to all the clusters that can process the task (Section 5.2). We finally choose from $PM_{n\text{-}d}$ the pair that minimizes the completion of the task execution, or some other performance criterion (Section 5.3).

5.1. Algorithm for Computing the Set of Non-Dominated Paths

In multicost routing, each link l is assigned a vector V_l of cost parameters, as opposed to the scalar cost parameter assigned in single-cost routing. In our initial formulation, the cost parameters of a link l include the propagation delay d_l of the link and its binary capacity availability vector $\hat{C}_l$:

$$V_l=(d_l,\ \hat{C}_l)=(d_l,\ c_{1,l},\ c_{2,l},\ \ldots,\ c_{z,l}),$$

but they may also include other parameters of interest (such as number of hops, the number of executed tasks in a cluster, etc). A cost vector [21] can then be defined for a path p consisting of links $l\in p$, based on the cost vectors of its links:

$$V(p)=\bigodot_{l\in p} V_l \overset{def}{=} \left(\sum_{l\in p} d_l,\ \mathop{\&}_{l\in p}\hat{C}_l\right), \qquad (2)$$

where **&** is the associative operator defined in Eq. (1).

We say that path p_1 dominates path p_2 for a given burst and source-destination pair, if the propagation delay of p_1 is smaller than that of p_2, and path p_1 is available for scheduling the burst (at least) at all time intervals at which path p_2 is available. Formally:

p_1 dominates p_2 (notation: $p_1 > p_2$) iff

$$\sum_{l\in p_1} d_l < \sum_{l\in p_2} d_l \ \text{ and } \ \mathop{\&}_{l\in p1}\hat{C}_l \geq \mathop{\&}_{l\in p_2}\hat{C}_l, \qquad (3)$$

where the vector inequality "≥" should be interpreted component-wise. The set of non-dominated paths $P_{n\text{-}d}$ for a given burst and source-destination pair is then defined as the set of paths with the property that no path in $P_{n\text{-}d}$ dominates another path in $P_{n\text{-}d}$.

An algorithm for obtaining the set $P_{n\text{-}d}$ of non-dominated paths from a given source to all destination nodes is given in [20] and [21], and is a generalization of Dijkstra's algorithm that only considers scalar link costs.

5.2. Set of Non-Dominated (path, cluster) Pairs

In the first phase of our proposed routing and scheduling algorithm we obtained the set of non-dominated paths between the source (S or R) and all the nodes of the network. We now expand the definition of the path cost vector to include the utilization profiles of the clusters. More specifically, we define the cost vector of a (path, cluster) pair pm (path p ending to cluster m) as:

$$V(pm)=\left(V(p),\ \hat{W}_m(p,b)\right)=\left(\sum_{l\in p} d_l,\ \mathop{\&}_{l\in p}\hat{C}_l,\ \hat{W}_m(p,b)\right), \qquad (4)$$

where $\hat{W}_m(p,b)=\mathrm{MUV}_{\mathrm{EST}(p,b)}(\hat{W}_m)$ is the binary cluster availability vector of m with 0's at the first EST(p,b) elements (Section 4.2).

We define a domination relationship between (path, cluster) pairs: A (path, cluster) pair p_1m_1 dominates another pair p_2m_2 for a given task, if p_1 dominates p_2 (Eq. 3), and also the cluster m_1 can execute the task (after the minimum transmission delay over p_1) at least at all time intervals at which cluster m_2 is available (after the minimum transmission delay over p_2). Formally:

p_1m_1 dominates p_2m_2 (notation: $p_1m_1 > p_2m_2$) iff

$$p_1 > p_2 \ \text{ and } \ \hat{W}_{m_1}(p_1,b) \geq \hat{W}_{m_2}(p_2,b)\ , \qquad (5)$$

where the vector inequality "≥" is interpreted component-wise. The set of non-dominated (path, cluster) pairs $PM_{n\text{-}d}$ is then defined as the set of (path, cluster) pairs with the property that no pair in $PM_{n\text{-}d}$ dominates another.

Clearly, we have $PM_{n\text{-}d} \subseteq P_{n\text{-}d}$. Therefore, to obtain the set $PM_{n\text{-}d}$ we apply Eq. (5) to the elements of $P_{n\text{-}d}$.

5.3. Finding the Optimal (path, cluster) Pair and the Transmission and Execution Time Offsets

In the third phase of the proposed algorithm we apply an optimization function $f(V(pm))$ to the cost vector, V, of each pair $pm \in PM_{n\text{-}d}$. The function f yields a scalar cost per pm in order to select the optimal path and cluster pair. The function f can be different for different tasks, depending on their QoS requirements. For example, if we want to optimize data transmission, which corresponds to the routing optimization problem, the function f will select the path minimizing the reception time of the data at the cluster. If we consider the optimization of the computation problem, the function f will select the cluster that has the fewer scheduled tasks, or the one that minimizes its completion time. A weighted combination of the above considerations can be also employed. Note that the optimization function f applied to the cost vector of a pm has to be monotonic in each of the cost components. For example, it is natural to assume that it is increasing with respect to delay, decreasing with increased capacity availability, decreasing with increased cluster availability, etc. The final step is to choose from $PM_{n\text{-}d}$ the *pm* pair that minimizes $f(V(pm))$.

In the context of this study we assume that we want to minimize the completion time of the task and that we are using a one-way connection establishment and reservation scheme. This is accomplished in the following way:

i) Compute the first available position to schedule the task
We start from the cost vector $V(p_i m_i)$ of pair $p_i m_i$ and calculate the first position $R_i(w_i)$ after which $\hat{W}_{m_i}(p_i,b)$ has $w_i = W/C_{mi}$ consecutive ones. In other words, $R_i(w_i)$ is the earliest time at which a task of computation workload W can start execution on m_i. Note that the way w_i is calculated accounts for the computation capacity of resource m_i, and that $\hat{W}_{m_i}(p_i,b)$, by definition, accounts for the earliest transmission time, the propagation delay of path p_i and the transmission delay (Section 4.2).

ii)Select the cluster with the minimum task completion time
Select the pair $p_i m_i$ that results in the minimum completion time $R_i(w_i)+w_i$ for the task. In case of a tie, select the path with the smallest propagation delay. The time offset of task execution ($TO_{cluster}$) is given by $R_i(w_i)$.

iii) Select the time to schedule the burst
Having chosen the pair $p_i m_i$ we transmit the task at the earliest time possible. The time offset TO_{path} for the data transmission is $R_{pi}(b)$, defined as the first position after which $\hat{C}_{p_i}$ has b consecutive ones (like in Section 4.2).

iv) Update the CAV of chosen (path, cluster)
Having chosen the (path,cluster) pair and the time offsets to transmit the input data and execute the task, the next step is to update the utilization profiles of the corresponding links and the cluster. Update messages must also be sent to update the utilization profiles maintained at the other distributed schedulers. Such update mechanisms are extensively presented in [20],[22], and are not described in this paper.

The procedure described above assumes a tell-and-go protocol. If we wish to use a tell-and-wait protocol we simply have to redefine $\hat{W}_{m_i}(p_i,b)$, $R_i(w_i)$, and $R_{pi}(b)$ to take into account the round trip time before the data transmission.

6. Polyonomial Algorithm for Computing the Set of Non-Pseudo-Dominated Paths

A serious drawback of the algorithm described in the previous section is that the number of non-dominated paths may be exponential, and the algorithm is not guaranteed to finish in polynomial time. The basic idea to obtain a polynomial time variation is to define a pseudo-domination relationship $>_{ps}$ between paths, which has weaker requirement than the domination relationship > defined in Eq. (3).

In [20], two such pseudo-domination relations were proposed and evaluated. For the scope of this study we present the better performing relation. We define a new link metric, called the slot availability weight of the link, as $weight(\hat{C}_l)$, which represents the total number of 1's in the vector $\hat{C}_l$.

The polynomial-time heuristic variation of the optimal multicost algorithm computes the set of non-pseudo-dominated paths following the same steps presented in Section 5.1. The algorithm still maintains the binary vectors of the paths but the domination relationship that is used to prune the paths is not Eq. (3) but the following:

$$p_1 \text{ pseudo-dominates } p_2 \ (p_1 >_{ps} p_2) \text{ iff}$$

$$\sum_{l\in p_1} d_l < \sum_{l\in p_2} d_l \ \text{ and } \ weight(\underset{l\in p_1}{\oplus} \hat{C}_l) > weight(\underset{l\in p_2}{\oplus} \hat{C}_l) \quad (6)$$

When the domination relationship of Eq. (6) is used, an upper limit on the number of non-pseudo-dominated paths is the dimension z_l of the capacity availability vectors. The heuristic algorithm obtained in this way avoids the tedious comparisons of the CAVs of the optimal multicost algorithm, by essentially converting a z_l+1 dimensioned cost vector into a cost vector of 2 dimensions that conveys most of the important information contained in the original vector. The reduced problem was proven to be polynomial in [12].

7. Performance Results

In order to evaluate the performance of the proposed multicost algorithm for the joint communication and computation task scheduling we conducted simulation experiments, assuming an OBS underlined network. We have extended the ns-2 platform [23] and tested the following algorithms:

- Optimal multicost algorithm for the joint communication and computation task scheduling (MC-T). MC-T algorithm minimizes the completion time of the task, as presented in Section 5.
- AW heuristic multicost algorithm for the joint communication and computation task scheduling (AWMC-T), as presented in Section 6.
- Optimal multicost burst routing and scheduling algorithm (MC-B), as presented in [20]. The MC-B algorithm takes into account only the communication part of the problem, and routes the input data to the cluster at which the data will arrive earlier, without using the utilization profiles of the related clusters.
- Earliest Completion time (ECT). The ECT algorithm considers only the computation part of the problem, and sends the task to the cluster where it will complete its execution earlier, using the shortest path and examining communication contention only at the first link.

In order to establish the connection and reserve the appropriate communication and computation resources we have implemented a one-way reservation protocol capable of supporting advance reservations. The protocol is similar to JET [14] with extensions to cope with the one-way reservation of computation resources.

The simulations were performed assuming a 5x5 mesh network with wraparounds, where the nodes were arranged along a two-dimensional topology, with neighboring nodes placed at a distance of 400 km. In this topology we placed 4 clusters, each having 25 CPUs and each CPU having a computational capacity C_m = 25000 MIPS (typical value for Intel Xeon CPUs). The 4 Clusters were placed randomly in the mesh network. Each link had a single wavelength of capacity C_l equal to 1 Gb/s. Users were placed at all the 25 nodes of the network and the tasks were generated according to a Poisson process with rate $\lambda/25$ tasks per second at each. The computation workload of each task was exponentially distributed with average value W MI (so the average task execution time was $w=W/C_m$). Finally, the size of the input data burst was also exponentially distributed with average I Bytes (so the average burst duration was $b=I/C_l$).

To assess the performance of the algorithms we used the following metrics:

- Average total delay: defined as the time between the task creation and its completion time.
- Burst blocking probability: the probability of an input data burst to content with another burst.
- Conflict probability: the probability of a task to find a cluster unavailable at the time predicted by the algorithm (equal to $TO_{cluster}$ - Section 5.3), due to another task that has already reserved that cluster.

If we used a centralized architecture the burst blocking and conflict probabilities would be negligible, since the "central" scheduler would have a complete knowledge of the utilization of the resources and would schedule the tasks accordingly. Thus, these two metrics depend on the update strategy used, as well as the examined algorithm.

We used the following parameters: b = 1 sec and w=10 sec. We classify the tasks as CPU- and data-intensive since w is considerable with respect to the total computation power, while b is considerable with respect to the total communication capacity of the Grid network. Note that the average total delay can take values less than 11sec, since a task of a user that is attached to a node with a cluster can be executed locally (without data transferring).

In Figure 5a we observe that the multicost algorithms that jointly consider the communication and computation resources (MC-T, AWMC-T) perform better than the other two algorithms (MC-B, ECT) with respect to the average total delay metric. The average total delay of the MC-T and AWMC-T algorithms increases slightly with the tasks' generation rate λ. The tasks have high demands for both communication and computation resources and these algorithms solve this joint problem efficiently, as can be seen by the corresponding low burst blocking probability (Figure 5b) and the low conflict probability (Figure 5c). On the other hand, the performance of ECT deteriorates as λ increases. ECT does not take into account the communication part of the problem, and thus exhibits a high burst blocking probability as λ increases (Figure 5b), which results in increased average total delay. Similarly, the performance of the MC-B algorithm deteriorates as λ increases. MC-B does not take into account cluster availability, and the clusters chosen are usually not the optimum ones, as can be seen by the high conflict probability (Figure 5c). This introduces additional delay to the total time of the task.

From these results it is clear that in a Grid network where tasks are both CPU- and data-intensive (or where some tasks are CPU-intensive and some data-intensive) performance improves significantly by jointly optimizing the use of the communication and computation resources.

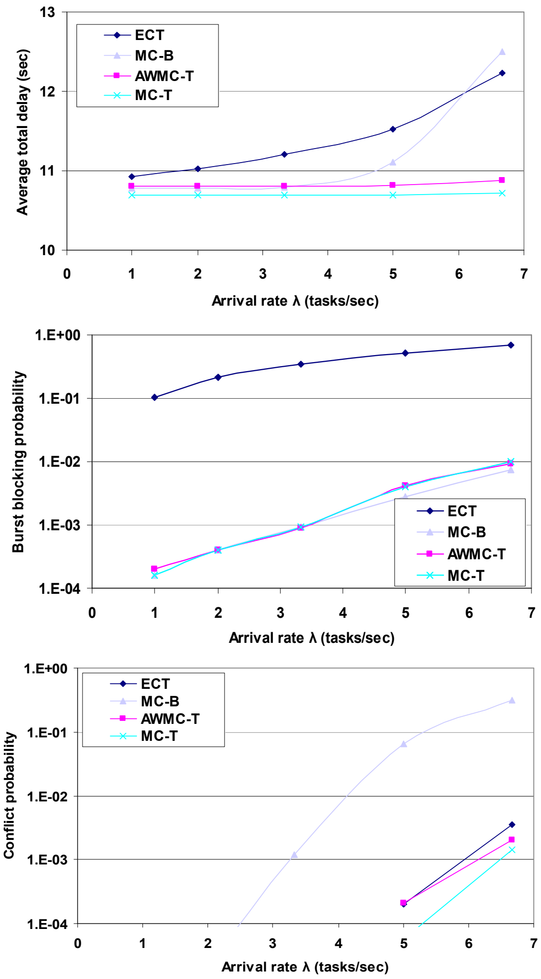

Fig. 5. Algorithms performance for tasks that are CPU- and data-intensive: (a) average total delay, (b) burst blocking probability and (c) conflict probability.

It is worth noting that the difference in the average delay performance between the optimal multicost (MC-T) and the heuristic multicost (AWMC-T) algorithms is small. In Figure 6a we show the average number of searched paths per task request (that is, the average size of the set P_{n-d}, presented in Section 5.1). We observe that the optimal multicost algorithm searches significantly more paths than the heuristic algorithm and the difference increases as the load (expressed by λ) increases. Figure 6b shows the average number of operations required to route and schedule a task. Note that the MC-T algorithm uses as cost parameters the delay and the link utilization profiles and thus can be viewed as an algorithm with $1+z_l$ costs. The average number of operations is defined as the number of operations (additions, Boolean operations (e.g. AND) or comparisons ($<$, $>$, $\neq$, etc)) required to manipulate this cost vector. As expected the heuristic algorithm requires fewer operations than the optimal multicost algorithm.

Thus, the proposed polynomial time AWMC-T algorithm yields delay performance that is very close to that of the optimal multicost algorithm, while maintaining the number of searched paths and required operations at low levels.

In future we plan to examine the performance of the proposed algorithms for uneven and heterogeneous resources and more complicated topologies.

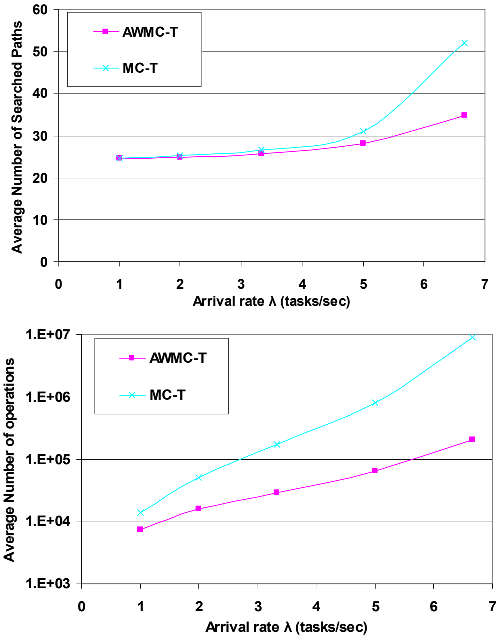

Fig. 6. Algorithms complexity: (a) average number of searched paths, and (b) average number of operations.

8. Conclusions

We presented a multicost algorithm for the joint selection of the communication and computation resources to be used by a task. We initially presented an optimal scheme of non-polynomial complexity and by appropriately pruning the set of candidate paths we also obtained a heuristic algorithm of polynomial complexity. We showed that in a Grid network where the tasks are CPU- and data-intensive important performance benefits, in terms of average total execution delay, can be obtained by jointly optimizing the use of the communication and computation resources as our proposed algorithms do. The proposed heuristic algorithm was shown to combine the strength of the optimal multicost algorithm with a low computation complexity.

Acknowledgments

This work has been supported by the European Commission through the Phosphorus project (http://www.ist-phosphorus.eu) and by the Greek General Secretariat for Research and Technology through the GRID-APP project.

References

[1] I. Foster, C. Kesselman, “The Grid 2: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann, 2003.

[2] R. Yahyapour, Ph. Wieder, “Grid Scheduling Use Cases”, Grid Scheduling Architecture Research Group (GSA-RG), Open Grid Forum (OGF), 2006.

[3] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, “A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation”, Intl. Workshop on Quality of Service (IWQoS), 1999.

[4] R. Ali, K. Amin, G. Laszewski, O. Rana, D. Walker, M. Hategan, N. Zaluzec, “Analysis and provision of QoS for distributed grid applications”, J. of Grid Computing, vol 2(2), 2004.

[5] W. Smith, I. Foster, V. Taylor, “Scheduling with advanced reservations”, Intl. Parallel and Distributed Processing Symposium (IPDPS), pp 127-132, 2000.

[6] R. Buyya, D. Abramson, J. Giddy, “Nimrod/G: An architecture for a resource management and scheduling system in a Global computational Grid”, HPC Asia, 2000

[7] G. Hoo, W. Johnston, I. Foster, A. Roy, “QoS as Middleware: Bandwidth Reservation System Design”, Intl. Symposium on High-Performance Distributed Computing (HPDC), pp. 345-346, 1999.

[8] K. Ranganathan, I. Foster. “Decoupling computation and data scheduling in distributed data-intensive applications”, Intl. Symp. on High-Performance Distributed Computing (HPDC), 2002.

[9] S. Venugopal, R. Buyya, L. Winton, “A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids”, Intl. Workshop on Middleware for Grid Computing, 2004.

[10] K. Nahrstedt, H. Chu, S. Narayan, “QoS-aware resource management for distributed multimedia applications”, J. High Speed Networks, vol. 7, no 3-4, pp. 229-257,1998

[11] R. Guérin, A. Orda, “Networks with Advance Reservations: The Routing Perspective”, Infocom, 2000.

[12] Z. Wang, J. Crowcroft, “Quality-of-service routing for supporting multi-media applications”, J. Selected Areas in Communications, vol. 14, no. 7, Sept. 1996.

[13] P. Van Mieghem, F. Kuipers, “Concepts of Exact QoS Routing Algorithms”, Transactions on Networking, 2004.

[14] E. Varvarigos, V. Sharma, “An efficient reservation connection control protocol for gigabit networks”, Computer Networks and ISDN Systems, vol. 30, no 12, pp. 1135–1156, 1998.

[15] C. Qiao, M. Yoo, “Optical burst switching (OBS)–a new paradigm for an optical Internet,” J. High Speed Networks, 1999.

[16] Grid Optical Burst Switched Networks: www.ogf.org/Public_Comment_Docs/Documents/Jan2007/OGF_GHPN_GOBS_final.pdf

[17] L. Burchard, “Analysis of Data Structures for Admission Control of Advance Reservation Requests”, Transactions on Knowledge and Data Engineering, vol 17, no 3, pp. 413–424, 2005.

[18] C. Barz, T. Eickermann, M. Pilz, O. Wäldrich, L. Westphal, W. Ziegler, “Co-Allocating Compute and Network Resources-Bandwidth on Demand in the VIOLA Testbed”, Springer, CoreGRID Series, Towards Next Generation Grids, 2007.

[19] Job Submission Description Language specification: www.ogf.org/documents/GFD.56.pdf.

[20] E. Varvarigos, V. Sourlas, K. Christodoulopoulos, “Routing and Scheduling Connections in Networks that Support Advance Reservations”, pending 2nd review at Computer Networks.

[21] F. Gutierrez, E. Varvarigos, S. Vassiliadis, "Multicost Routing in Max-Min Fair Networks", Allerton Conference, 2000.

[22] K. Manousakis, V. Sourlas, K. Christodoulopoulos, E. Varvarigos, K. Vlachos, “A Bandwidth monitoring mechanism: Enhancing SNMP to record Timed Resource Reservations”, J. Network and Systems Management, pp. 583-597, 2006.

[23] The Network Simulator (ns2): www.isi.udu/nsnam/ns