Future Generation Computer Systems 25 (2009) 912–925

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Multi-cost job routing and scheduling in Grid networks

T. Stevens [a,*], M. De Leenheer [a], C. Develder [a], B. Dhoedt [a], K. Christodoulopoulos [b,c], P. Kokkinos [b,c], E. Varvarigos [b,c]

[a] Department of Information Technology, Ghent University – IBBT, Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium
[b] Department of Computer Engineering and Informatics, University of Patras, Greece
[c] Research Academic Computer Technology Institute, Patras, Greece

ARTICLE INFO

Article history:
Received 9 November 2007
Received in revised form
12 June 2008
Accepted 18 August 2008
Available online 9 September 2008

Keywords:
Algorithms
Network problems
Scheduling
Optimization

ABSTRACT

A key problem in Grid networks is how to efficiently manage the available infrastructure, in order to satisfy user requirements and maximize resource utilization. This is in large part influenced by the algorithms responsible for the routing of data and the scheduling of tasks. In this paper, we present several multi-cost algorithms for the joint scheduling of the communication and computation resources that will be used by a Grid task. We propose a multi-cost scheme of polynomial complexity that performs immediate reservations and selects the computation resource to execute the task and determines the path to route the input data. Furthermore, we introduce multi-cost algorithms that perform advance reservations and thus also find the starting times for the data transmission and the task execution. 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. Our performance results indicate that in a Grid network in which tasks are either CPU- or data-intensive (or both), it is beneficial for the scheduling algorithm to jointly consider the computational and communication problems. A comparison between immediate and advance reservation schemes shows the trade-offs with respect to task blocking probability, end-to-end delay and the complexity of the algorithms.

1. Introduction

Grid computing aims to offer a unified interface to various resources such as computational clusters, data storage sites and scientific instruments. In general, these resources are heterogeneous in nature, have different access and control policies, and are distributed on a large scale (possibly global) network. The driving force for the realization of such Grid technology is the highly challenging applications emerging from large-scale collaborations and eScience experiments. Recently, several proposals have been made to extend the concept for supporting enterprise- and consumer-oriented Grid applications [1]. Management and control of an efficient Grid system will therefore require intelligent *scheduling* at various levels [2]. Indeed, the complexity of the Grid applications, the user requirements and the system heterogeneity would result in suboptimal system performance in case manual procedures are used. 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. Such algorithms should try to balance the users' individual demands (e.g., cost, response-time), the objectives represented by the resource providers (profit, utilization) while at the same time also maintain a good overall performance for the Grid network.

Grid applications usually pose challenging demands to the networking fabric, as data transfers demand high-bandwidth and low latency connections. As such, optical networks have been identified as the most suitable technology to interconnect the distributed resources. Irrespective of the transport technology, efficient *routing* algorithms for data transfer between Grid sites is of great importance for the performance of any Grid deployment, and especially for Data-Grids.

Another issue for efficient management of Grid resources is how to take into account temporal information in the scheduling and routing decisions. Traffic demands are known to fluctuate over time, and both network and computational resources are subject to failures and disconnections. Thus, the option of immediately reserving the resources for a task is not always the best choice. In a Grid scenario, a user typically submits a task to have it processed within a predetermined deadline. In this context, network transfers and/or task executions can be delayed, allowing communication and computational loads to be more spread out

* Corresponding author.
E-mail addresses: tim.stevens@intec.ugent.be (T. Stevens), marc.deleenheer@intec.ugent.be (M. De Leenheer), kchristodou@ceid.upatras.gr (K. Christodoulopoulos), kokkinop@ceid.upatras.gr (P. Kokkinos), manos@ceid.upatras.gr (E. Varvarigos).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.08.004

over time in order to efficiently serve/manage all the requests. Moreover, a number of Grid tasks require the utilization of certain resources for specific time periods. In both the above cases, the starting time of the resource reservation has to be (or is relaxed to be) in the future. This is generally referred to as *advance reservations*, as opposed to *immediate reservations* which are made in a just-in-time manner. However, inclusion of this temporal information in routing and scheduling algorithms will naturally increase the complexity of these algorithms. This paper will provide insight into this matter by comparing the delay, the acceptance performance, and the computational complexity of algorithms for both immediate and advance reservations.

In this work we address several of the previously introduced issues, by proposing two separate techniques in order to solve a communication and computation co-allocation problem of significant importance. The general assumption is that a task consists of two phases: (i) the transfer of data from the scheduler or a data repository resource (which we will call source) to a Computing Element (CE) or a cluster (which we will call destination) and (ii) the task's execution. Without loss of generality, we assume there is no output data associated with the tasks: this can be modelled as an additional task with only data transfer, to a specified destination. (Note that the algorithms can be extended to include output data transfer explicitly, but to limit the complexity we chose not to present those extensions in this paper.) The algorithms return the destination (CE) to execute the task and the path over which to route the input data. The first proposed technique, which is an extension of SAMCRA, is a multi-cost algorithm that uses the path delay and the computation load as selection metrics and is proven to be of polynomial complexity. This algorithm employs immediate reservations and can be applied in cases that the data size and the computation load of a task are either known or not known in advance. The second technique, called MC-T, is a multi-cost algorithm that is mainly used in the cases where the data sizes and the task execution times are known in advance. MC-T uses utilization vectors of the communication and computation resources in the multi-cost formulation in order to cope with advance reservations. This technique provides advance reservations by orchestrating the corresponding network and computational resources, and specifically returns not only the path and the destination (CE), but also the time that the data transmission should start and the time that the task execution should begin at the CE. Due to the large number of cost parameters that MC-T utilizes, the number of paths that it calculates can be exponential. By appropriately pruning the set of candidate paths we also present a heuristic algorithm of polynomial complexity. Table 1 provides an overview of the proposed algorithms.

We evaluate the performance of the various algorithms for multi-cost task routing and scheduling using full network simulation experiments. Our results indicate that in a Grid network in which tasks are either CPU- or data-intensive (or both), it is beneficial for the scheduling algorithm to jointly consider the computational and communication problems. A comparison between immediate and advance reservations shows the trade-offs with respect to task blocking probability and end-to-end delay. Finally, an analysis of the computational complexity of the proposed algorithms is also presented. Ultimately, we will show that the proposed algorithms combine the strength of multi-cost optimization (both for immediate and advance reservations) with a low computational complexity and running time.

The remainder of this paper is organized as follows. In Section 2 we report on previous work. Section 3 presents our model for Grid networks, with details on the assumptions for both computation and communication resources. In the same section we also propose an algorithm for immediate joint reservation of network and computation Grid resources. In Section 4, we initially present network and computation resource state models in the form of utilization profiles. Based on these profiles we present our advance reservation multi-cost algorithms (optimal and heuristic). In Section 5 we compare the proposed algorithms for a wide range of input parameters and scenarios, and analyze the performance results. Finally, our conclusions are summarized in Section 6.

2. Related work

The Grid Scheduling Architecture Research Group (GSA-RG) of the Open Grid Forum (OGF) in [3] provides different Grid scheduling use case scenarios and describes common usage patterns. Among them the most complicated scenario is scheduling tasks requesting more than one and possibly different service guarantees. In this content, a "workflow" is defined as a task that consists of a number of other "subtasks" with various interdependencies. Thus a workflow requests the co-allocation of resources in different time frames.

In general, Grid applications can be categorized as CPU- or data-intensive [4]. Different Grid environments have been created to cope with these two types of applications. Computational Grids normally deal with CPU-intensive problems on small data sets. In contrast, Data Grids mostly deal with problems that require the transfer of large amounts of data. However, in general 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 transfer of a large chunk of data, as a connection with a constant rate or a data burst, from the location of the user or a storage repository to the computation resource where it will be executed. This problem of communication and computation resources co-allocation will be addressed in this paper.

Resource co-allocation is one of the most challenging problems in Grids. The co-allocation problem for Computational Grids has been defined in [5]. In the Condor project, the gang matchmaking scheme [6] extends the matchmaking model in order to support the co-allocation of resources. In order to co-allocate resources as defined in workflows, the scheduler has to orchestrate resources belonging to different sites and different administrative domains. To do so, advance reservation of these resources has to be supported by the local resource management systems. The Globus Architecture for Reservation and Allocation (GARA) [7] is a framework for advance reservations that treats communication, computation, and storage resources in a uniform way. 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 [8]. The WS-Agreement protocol [9] has been proposed by the GRAAP working group in the Open Grid Forum (OGF) for the negotiation of advance reservations.

Some specific instances of communication and computation co-allocation problems have previously been examined in the literature. In [10,11] the authors study the relation between job scheduling decisions and data replication strategies within a data Grid environment, in order to minimize job turnaround time. Similar algorithms have been examined in multimedia networks where the co-allocation of computation, communication (bandwidth) and other resources are examined [12]. A framework for specifying and handling co-reservations in Grid environment is presented in [13]. This framework can be applied to the reservation of applications running concurrently on multiple resources and to the planning of job flows, where the components may be linked by some temporal or spatial relationship. More general Grid workflow management systems have been developed by several projects: Condor DAGman [14], GridFlow [15], Gridbus [16], and

Table 1
Main characteristics of the proposed algorithms

Algorithm	Topic	Options
SAMCRA (Section 3)	Input	Known or not known input data size and computation complexity of tasks. Utilization of links and clusters at current time
	Output	Assignment of computation resources, routing of data over the communication resources
	QoS parameters	*Communication*: path availability, delay *Computation*: processing availability
	Objectives	Delay minimization, load balancing
	Reservation type	Immediate
	Complexity	Polynomial for the specific problem (in general NP-hard)
MC-T (Section 4)	Input	Known input data size and computation complexity of tasks Utilization of links and clusters as a function of time
	Output	Assignment and time scheduling of computation resources, routing and time scheduling of communication resources
	QoS parameters	*Communication*: path availability (also in future), delay *Computation*: processing availability (also in future), delay *Total*: availability (also in future), end-to-end delay
	Objectives	Delay minimization, load balancing
	Reservation type	In-advance (can perform immediate as well)
	Complexity	NP-hard, (heuristic: polynomial)

UNICORE [17]. A taxonomy of workflow management systems is presented in [18].

In the related literature, multi-cost algorithms have mainly been used for QoS routing problems [22–27]. In [22], the authors show that QoS routing with the specific parameters bandwidth and delay is not NP-complete, while the general QoS routing problem is discussed in [23]. To the best of our knowledge, the present work is the first time to use a multi-cost algorithm for the joint communication and computation problem in a Grid environment. Moreover, the algorithm presented in Section 4 is significantly different from other multi-cost approaches, since it is designed to handle temporal information, using timeslots as cost parameters in the multi-cost formulation, in order to cope with the time scheduling of the resources.

3. Exact multiple constraints routing: Immediate reservations

Upon submitting a job to a computing grid, *scheduling* and *routing* actions determine to which computing resource(s) the job is assigned and how it can be transmitted over the network. If there is no possibility to delay the job transmission time or the job execution time, *immediate reservation* of network and computational resources is mandatory. In this case, scheduling and routing can be tackled simultaneously by transforming the problem to a Multi-Constrained Path (MCP) problem with multi-dimensional weight vectors. When *optimal* use of the available network- and computational resources is desired, the Multi-Constrained Optimal Path (MCOP) should be computed for each generated job. For this purpose, we embrace the SAMCRA algorithm [21].

In the following sections we present a model for Grid networks, and show how cluster sites can be virtualized into a single anycast group, which then allows multi-constraint routing to be used to balance network and cluster availability.

3.1. Grid network and resources model

As illustrated in Fig. 1, we consider a Grid network consisting of links l of known propagation delays d_l, and a set M of clusters or CEs (these terms will be used interchangeably in this paper). Cluster $m \in M$ has W_m CPUs of a given computation capacity C_m per CPU (e.g. in MIPS). A task or job (also used interchangeably) is created by a user with specific needs: input data size I (bits), computational workload W (in MIs) and requires r CPUs for its execution. The parameters I and W may or may not be known in advance. We assume that the task consist of two sub-tasks: (i) the transfer of data from the scheduler or a data repository resource (which we will call source) to a computation resource (CE) in the form of a connection with a constant rate or a data burst, and (ii) the task execution at that CE. Observe that we do not consider the possible transfer of output data, as this can be considered an additional job without computational demand. Another implicit assumption is that each task can be processed at a single resource site, but multiple CPU's can be allocated to that task. Although this model does not allow complex workflows between processing sites, we adopt it here for simplicity and tractability purposes. Furthermore, there is an upper bound D on the maximum delay the task can tolerate; if this deadline cannot be met, the task is rejected. Each user communicates this information to its corresponding meta-scheduler S. We assume a centralized scheduling architecture, where all users forward their task scheduling requests to a single central scheduler S, which maintains information about the utilization of the communication and computation resources throughout the network.

We will examine two cases that differ in the type and the amount of information the scheduler maintains. In case (i) we assume that the data size and the computation workload of a task are not known in advance. In this case the scheduler cannot use advance reservations and thus maintains resources' utilization information that is not time dependent. In case (ii) we assume the data size and the computation workload of a task are known in advance and thus the scheduler maintains profiles that record the utilization of the resources as a function of time. Such communication and computation utilization profiles are presented in the next subsection. We are interested in scheduling algorithms that decide the destination (CE) at which to execute the task and also the path over which to route the input data. These scheduling algorithms take into account the utilization information available at the scheduler in order to efficiently utilize the available Grid resources.

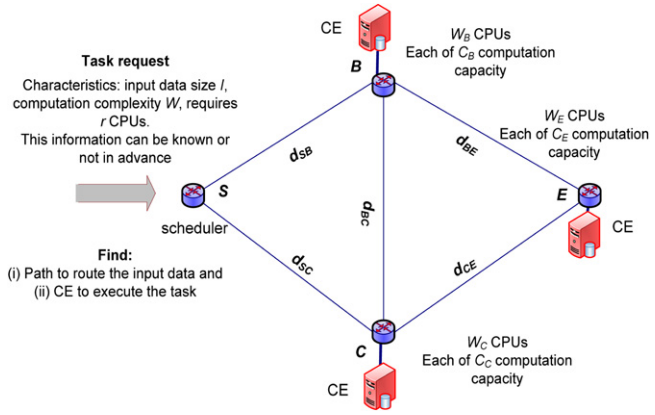

Fig. 1. A request for task execution is generated by a user and is forwarded to the scheduler S. The task has input data with size I, computation complexity W and requires r CPUs to execute.

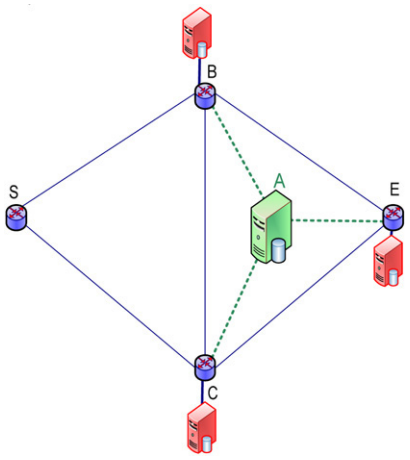

Fig. 2. Virtual topology.

3.2. Routing towards anycast destinations

From a routing perspective, cluster sites can be grouped in a single virtual anycast node A, as depicted in Fig. 2. Such an abstraction is only applicable if the virtual anycast group is the *final routing destination*, because this virtual anycast node cannot forward packets over virtual links. For the application in mind, anycast nodes are computational resources ($m \in M$) and not routers, making this a realistic assumption. Furthermore, this approach requires a distinct logical network topology – and hence a distinct routing component instance – for each anycast group present in the routing domain. In general, however, the number of anycast groups in an autonomous system will probably be small because of their focused applicability.

When traditional single constraint shortest path routing is applied to the logical topology, *shortest shortest path (SSP) routing* as discussed in [21] is achieved. Using SSP, the nearest anycast target node is contacted and data is sent over the shortest path. In this paper, additional metrics are also taken into account (e.g., resource availability), so Dijkstra shortest path routing or other single constraint routing algorithms are not sufficient. In a general multi-constrained anycast routing problem, each network link l is characterized by k link weights $w_i(l) \geq 0$ for all $1 \leq i \leq k$. Besides additive network-related constraints such as delay or hop count, also server-related constraints can be taken into account (e.g., server load). In this case, edges not directly attached to a member of the anycast server group have a weight equal to zero for all server-related constraints. Therefore, only the last edge of a path p from a source node to an anycast member can have a non-zero value for the server-related components of the weight vector. Based on [3], the corresponding anycast MCOP problem can be defined as follows: Given k constraints L_i ($1 \leq i \leq k$), find a path P from a source node (which in Fig. 2 is S) to the virtual anycast node A which satisfies the following condition:

$$w_i(p) \overset{\text{def}}{=} \sum_{l \in P} w_i(l) \leq L_i.$$

Additionally, for some length function $d(.)$, the condition $d(p) \leq d(p')$ should hold for all paths p' between S and A.

This anycast multiple constraints routing problem can now be solved by applying the SAMCRA algorithm, which is discussed in the next section.

3.3. Self-adaptive multiple constraints routing algorithm: SAMCRA

In this section, the SAMCRA algorithm with look-ahead extension [23] is briefly summarized; in the next section we present an adaptation of the original SAMCRA sub-path evaluation ordering. For an in-depth investigation of the algorithm, the reader is referred to [23].

SAMCRA is based on four key concepts: non-linear path length, k-shortest paths, non-dominated paths and look-ahead. Before presenting the complete algorithm, these key concepts are clarified.

The q-vector norm of path P from source S to destination E can be computed as follows:

$$d_q(p) = \left(\sum_{i=1}^{k} \left(\frac{\sum_{l \in S \to E} w_i(l)}{L_i} \right)^q \right)^{\frac{1}{q}}.$$

For $q \to \infty$, this length function can be rewritten as follows:

$$d_\infty(p) = \max_{1 \leq i \leq k} \left(\frac{\sum_{l \in S \to E} w_i(l)}{L_i} \right).$$

According to Van Mieghem et al. [23], finding the shortest path between S and E using the *non-linear length function* d_∞ in the equation above, solves the MCOP routing problem.

The *k-shortest paths* algorithm is similar to Dijkstra's algorithm, but stores the k shortest paths instead of the single previous *hop* in each node. This is necessary because in a multi-constrained environment, sub-paths of shortest paths are not necessarily shortest paths themselves. If a fixed value for the number of paths k is specified, the multi-constrained shortest path from a source to a destination may not be found. For SAMCRA, the number of paths stored in each node is unrestricted, meaning that *all* possible paths may need to be stored before the shortest one can be selected. This property leads to the worst-case NP-complete behavior of SAMCRA [24].

The *non-dominance* concept allows for a drastic reduction of the number of sub-paths that need to be stored in the router nodes. This optimization dismisses newly computed sub-paths from the source to the current routing node that a priori lead to a non-optimal path from the source to the final destination, based on previously computed sub-paths stored in the node. More concrete, new sub-paths between the source and the current routing node are dismissed if a previous sub-path to the same routing node has a weight vector for which each vector component is smaller than the corresponding component of the new sub-path weight vector.

The *look-ahead* extension further reduces the search space of possible paths by predicting the total length from source to destination for each sub-path stored in intermediate routers. This path length prediction is based on the sub-path history and Dijkstra

```
1.  SAMCRA(S,D)
2.  MAX_LENGTH ← 1, Q ← {∅}
3.  for all nodes v do
4.      K[v] ← 0
5.      Store Dijkstra look-ahead info for all constraints in b[v]
6.  for all constraints do
7.      if d_∞(Dijkstra path S → D for current constraint) < MAX_LENGTH then
8.      MAX_LENGTH ← d_∞(Dijkstra path S → D for current constraint)
9.  priority(S[1]) ← 0, path(S[1]) ← NULL
10. insert (Q,S[1])
11. while Q ≠ {∅} do
12.     u[i] ← extract_min(Q)
13.     u[i] marked GREY
14.     if u = D then
15.         stop and return path(u[i])
16.     else
17.         for all v in adjacency_list(u) do
18.             if v not in path(u[i]) then
19.                 PATH ← path(u[i]) + (u → v)
20.                 DOMINATED ← dominated(PATH)
21.                 mark all obsoleted paths v[j] BLACK
22.                 PRED_LENGTH ← d_∞( d[PATH] + d[b[v]] )
23.                 if PRED_LENGTH ≤ MAX_LENGTH and DOMINATED = false then
24.                     if all v[j] ≠ BLACK then
25.                         K[v] ← K[v] + 1
26.                         path(v[K[v]]) ← PATH
27.                         priority(v[K[v]]) ← PRED_LENGTH
28.                         insert(Q,v[K[v]])
29.                     else
30.                         path(v[BLACK_k]) ← PATH
31.                         decrease_key(Q, v[BLACK_k], PRED_LENGTH)
32.                     if v = D and PRED_LENGTH < MAX_LENGTH then
33.                         MAX_LENGTH ← PRED_LENGTH
```

Fig. 3. Meta-code for the SAMCRA algorithm.

shortest path information for all constraints from the intermediate router to the final destination. By applying the look-ahead concept, sub-paths with the lowest end-to-end predicted path length are evaluated *first*.

Meta-code for the complete algorithm is presented in Fig. 3. Lines 1–10 take care of the initialization. For each node v, Dijkstra look-ahead information $b[v]$ is computed and $K[v]$, the number of stored sub-paths between S and v, is initialized to 0. If the length of a Dijkstra path for one of the constraints is smaller than the maximum length MAX_LENGTH, the value of MAX_LENGTH is updated. Furthermore, the source node S is inserted in the priority queue Q with an empty path history. The iterative search for the shortest path starts at line 11. First, the sub-path with the lowest predicted end-to-end path length is dequeued and marked "GREY", which means this path is not considered anymore during the next iterations. If the destination has been reached, the algorithm stops. Starting at line 17, the current sub-path is extended by examining all nodes that are adjacent to the current intermediate router and are not listed in the sub-path history. For each of these path extensions, path dominance is evaluated and previously computed sub-paths that have become obsolete, are marked "BLACK". If the extended path is non-dominated and the predicted length is less than or equal to the maximum length, the priority queue is updated. When an arbitrary sub-path $v[BLACK_k]$ has become obsolete, it is replaced by the new sub-path in the priority queue (by decreasing the key value and updating the path). Otherwise, a new item is created and inserted into the priority queue. On lines 32–33, the maximum length MAX_LENGTH is updated if a shorter path reaching D is found.

In Fig. 4, a two-dimensional example illustrates the SAMCRA routing algorithm steps to compute a path from C (bottom) to E (right). Two-dimensional link weights and Dijkstra look-ahead vectors are depicted next to links and nodes, respectively. Step 1 depicts the algorithm state after the initialization phase (lines 1–10): look-ahead information is computed and MAX_LENGTH is reduced according to lines 6–8 in the SAMCRA meta-code. Subsequently, the priority queue is initialized by inserting a single element with empty path history. Steps 2–3 represent successive iterations of the main algorithm loop (lines 11–33). Each step starts by dequeueing the sub-path with the smallest predicted end-to-end cost (drawn in bold), whereupon possible path extensions (indicated by dotted arrows) are investigated. Path extensions that do not create a routing cycle and are not dominated by other sub-paths to the same intermediate router are inserted into the priority queue if their predicted end-to-end (i.e., $C \rightarrow E$) length (the priority queue key value) does not exceed MAX_LENGTH. During step 4, an end-to-end path is dequeued from the priority queue and the SAMCRA algorithm returns the solution $C \rightarrow B \rightarrow E$.

3.4. Avoiding sub-optimal SAMCRA results

Unfortunately, the path computed in the example above ($C \rightarrow B \rightarrow E$) is not the optimal solution: when taking into account the path length in the non-dominating dimension(s), the SAMCRA algorithm should have returned the path $C \rightarrow E$. Actually, it is not the SAMCRA algorithm itself but the $d_\infty(.)$ metric which leads to sub-optimal results. Because the path evaluation order (priority queue) is determined only by the dominating component of the predicted path length vector, *optimality in the non-dominating vector components cannot be guaranteed* in the evaluation order between equal length (sub-)paths (according to the $d_\infty(.)$ metric).

We adapted the SAMCRA algorithm to cope with this issue. Instead of using the $d_\infty(.)$ metric to determine the order in which sub-paths are evaluated, the information contained in the *entire path weight vector* p^w is used when ordering the sub-paths. Let $\rho(p^w)$ be a rescaled vector with components $\rho(p^w)_i = p_i^w/L_i$. Let $o(\rho(p^w))$ be the vector $\rho(p^w)$ with its indices reordered so that the components are in non-increasing order. The vector ordering $p^w < q^w$ holds if the first nonzero component of $o(\rho(p^w)) - o(\rho(q^w)) < 0$. In contrast, the original SAMCRA algorithm only considers the outcome of the first component of $o(\rho(p^w)) - o(\rho(q^w))$. It is clear that the proposed vector ordering is a refinement of the $d_\infty(.)$ metric that allows to differentiate between (sub-)paths with equal length according to the $d_\infty(.)$ metric. For unequal path lengths according to the $d_\infty(.)$ metric, the same ordering is preserved. With the new vector ordering, path length can only be equal if $o(\rho(p^w)) = o(\rho(q^w))$.

When revisiting the example depicted in Fig. 4 and applying the modification presented above, step 4 will dequeue the optimal path $C \rightarrow E$ because the non-dominating vector component is smaller. Indeed, we have

$$o(\rho(C \rightarrow E)) = (0.4, 0.1) \quad \text{and} \quad o(\rho(C \rightarrow B \rightarrow E)) = (0.4, 0.2)$$

and consequently

$$C \rightarrow E < C \rightarrow B \rightarrow E.$$

This adaptation is particularly important when the SAMCRA algorithm is applied in a hop-by-hop scenario [25]. If zero link weight components are allowed, hop-by-hop SAMCRA requires an exact solution in each intermediate node in order to prevent *routing loops*.

3.5. Complexity analysis

In the general case, the SAMCRA algorithm cannot guarantee to find the optimal path in polynomial time. According to Kuipers [26], the worst-case complexity of SAMCRA is:

$$C_{SAMCRA} = O(p_{\max} N \log(p_{\max} N) + p_{\max}^2 kE),$$

where N and E depict the number of nodes and edges in the network, respectively. $p_{\max}$ corresponds to the maximum number of sub-paths in an intermediate node that need to be evaluated before the optimal solution is found and is bounded by

$$p_{\max} = O(N!) = O(\exp(N \ln N)),$$

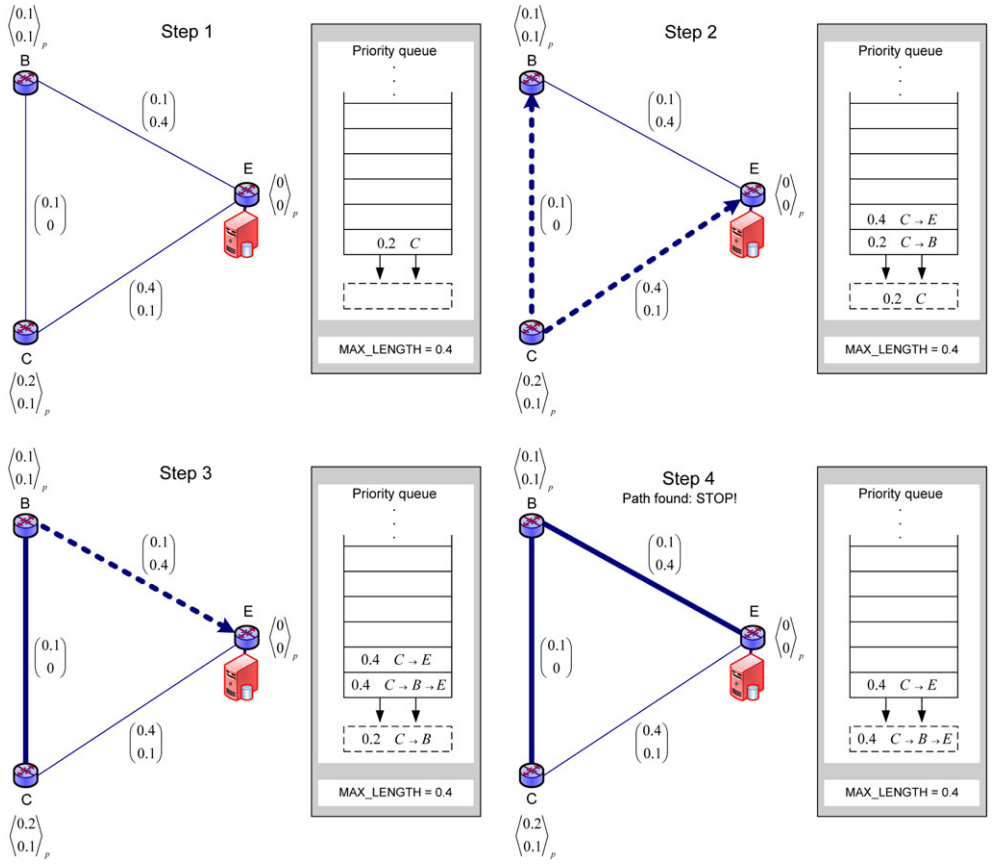

Fig. 4. Illustration of the SAMCRA algorithm.

thereby potentially leading to intractability. In practice, link weights will have a finite granularity and can be represented by integer values. In this case

$$p_{\max} = \frac{\prod_{i=1}^{k} L_i}{\max_{1 \le i \le k} L_i},$$

where L_i are the constraints (higher possible values), and SAMCRA has a pseudo-polynomial-time complexity.

For the two-constraint scenario considered, a link has two weights, being the delay, which is a float cost, and the availability of CPUs at the Computing Element of the ending node, which is a bounded integer cost (and thus, by definition, has finite granularity). Link delays will be zero for the virtual links leading to the computation resources, and resource's load will be zero for ordinary network links. By definition, each regular node will store at most one sub-path and the virtual target node will store at most M paths, where M stands for the number of Computing Elements. To this end the maximum number of sub-paths at an intermediate node is $p_{\max} = O(M)$. The resulting worst-case complexity is given by:

$$C_{SAMCRA-2costs} = O(MN \log(MN) + M^2 kE),$$

which is polynomial.

Note that this only applies to the case of these two specific cost weights and not the general case of SAMCRA algorithm. For an in-depth complexity analysis of the SAMCRA algorithm, we refer to [26].

4. Multi-cost task routing and scheduling: Employing advance reservations

The second algorithm that is presented in this section requires that the task's input data size and the computation load are known in advance (or an accurate estimation is available) and employs advance reservations of communication and computation resources.

4.1. Time dependent utilization profiles

Link utilization profile

The immediate reservation algorithm proposed in Section 3 poses no constraints on the underlying network, while the algorithm proposed in this section requires a network that supports advance reservations, as discussed in [20,28].

We require that each node records the capacity reserved on its outgoing links as a function of time, in order to perform channel scheduling and reservations. Assuming each connection or data burst reserves bandwidth equal to the link capacity for a given time duration, the *utilization profile* $U_l(t)$ of a link l is a stepwise binary 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 $C_l(t) = C_l - U_l(t)$. In order to obtain a data structure that is easier to handle in an algorithm, we discretize $C_l(t)$ in time steps of duration τ_l to obtain the *binary capacity availability vector* $\hat{C}_l$, abbreviated CAV, as the vector whose k-th entry is:

$$\left\{\hat{C}_l\right\}_k = \begin{Bmatrix} 1, & \text{if } C_l(t) = 1 \\ 0, & \text{otherwise} \end{Bmatrix},$$

$$\text{for all } (k-1) \cdot \tau_l \le t \le k \cdot \tau_l,\ k = 1, \ldots, u_l$$

where u_l is the dimension of the CAV (see Fig. 5).

Cluster utilization profile

To have a consistent formulation, we define the utilization profile $U_m(t)$ of cluster m as an integer function of time, which records the

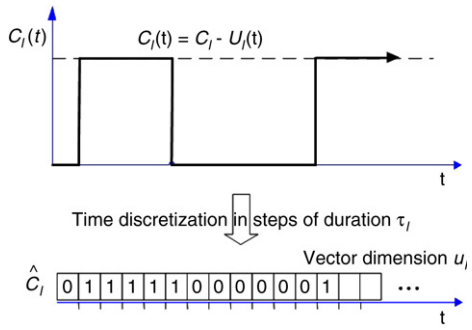

Fig. 5. The capacity availability profile $C_l(t)$, and the binary capacity availability vector $\hat{C}_l$ of a link l of capacity C_l, when the discretization step is τ_l.

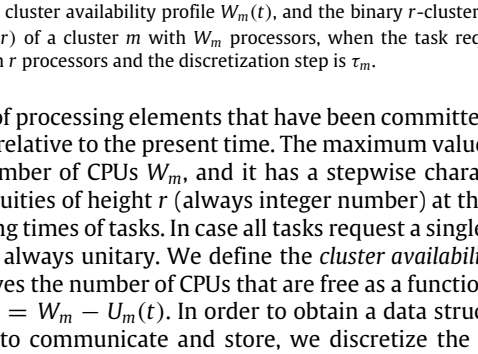

Fig. 6. The cluster availability profile $W_m(t)$, and the binary r-cluster availability vector $\hat{W}_m(r)$ of a cluster m with W_m processors, when the task requests to be executed on r processors and the discretization step is τ_m.

number of processing elements 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 always unitary. We define the *cluster availability profile*, which gives the number of CPUs that are free as a function of time, as $W_m(t) = W_m - U_m(t)$. In order to obtain a data structure that is easier to communicate and store, we discretize the time axis in steps of duration τ_m and define the *binary r-cluster availability vector* $\hat{W}_m(r)$, as follows:

$$\left\{\hat{W}_m(r)\right\}_k = \begin{Bmatrix} 1, & \text{if } W_m - U_m(t) > r \\ 0, & \text{otherwise} \end{Bmatrix},$$

$$\text{for all } (k-1)\cdot\tau_m \le t \le k\cdot\tau_m,\ k = 1, 2, \ldots, u_m$$

where u_m is the maximum size of the cluster availability vector (see Fig. 6).

To simplify the presentation and the experiments, we assume for this study that each task requests $r = 1$ processors, 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 and the dimensions u_l and u_m used in the link and cluster utilization profiles, respectively, can be different to account for the different time scales in the reservations performed on the communication and computation resources, and to separately control the efficiency–accuracy we want to obtain in each case.

The timeslot-based management of allocated resources [27–29] has been used in different environments for advance reservations, e.g. denoted as "slot table" in GARA [7]. A recent paper [30] has focused on enhancing the timeslot-based approach by introducing dynamic timeslots and the notion of granularity, and developed analytical models of the interrelation between user and system parameters. A different approach to maintain utilization information as a function of time is instead of bit-vectors to have linked lists that store the changes of states (from 0 to 1 and from 1 to 0) [31]. Manipulating such linked lists is also straightforward.

Fig. 7. The co-allocation problem when the task's input data size I and computation workload W are known in advance. Each link is characterized by its propagation delay and its binary capacity availability vector. Node E is a cluster with binary r-cluster availability vector $\hat{W}_E(r)$.

4.2. Grid network and resources model in the case of advance reservations

The assumptions of the Grid Network and the traffic are the same as Section 3.1, but we also assume that we know in advance or have an accurate estimation of the input data size I and the computational workload W of a task. Moreover, we assume that the scheduler S has 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 u_l and u_m of the link and cluster utilization vectors are finite. Given the previous information, we want to find a suitable cluster to execute the task, a feasible path over which to route the input data from the source (which can be the scheduler or a data repository site), 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 ($TO_{cluster}$). Fig. 7 presents an instance of the problem.

4.3. Binary capacity availability vector of a path and binary cluster availability vector over a path

Calculating the binary capacity availability vector of a path

Assuming the routing and scheduling decision is made at the scheduler S, the capacity availability vectors of all links should be gathered continuously. To calculate the Capacity Availability Vector (CAV) of a path we have to combine the CAVs of the links that comprise it, as described in [20].

For example, for the topology of Fig. 7, the CAV of path SBE (p_{SBE}), consisting of links l_{SB} and l_{BE}, is

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

where $\hat{C}_{SB}$ and $\hat{C}_{BE}$ are the CAVs of links l_{SB} and l_{BE}, respectively, and $\mathrm{LSH}_{2\cdot d_{SB}}$ defines the left shift of $\hat{C}_{BE}$ by $2\cdot d_{SB}$ (twice the propagation delay of link l_{SB} measured in τ_l-time units). Left shifting $\hat{C}_{BE}$ by