Proc. MASCOTS 2004, Volendam, The Netherlands, October 5-7, 2004.

A Framework for Resource Allocation in Grid Computing

Daniel A. Menascé
Dept. of Computer Science
George Mason University
menasce@cs.gmu.edu

Emiliano Casalicchio
Dip. Informatica Sistemi e Produzione
Univ. Roma "Tor Vergata"
casalicchio@ing.uniroma2.it

Abstract

Grid computing is the future computing paradigm for enterprise applications. An enterprise application running on a grid is composed of a set of SLA-constrained sub-tasks demanding different types of services and resources such as processors, data storages, service providers, and network links. This paper formalizes the resource allocation problem for SLA-constrained grid applications. The paper considers a very general case in which applications are decomposed into tasks that exhibit precedence relationships. The problem consists in finding the optimal resource allocation that minimizes total cost while preserving execution time service level agreements. The paper provides a framework for building heuristic solutions for this NP-hard problem, presents an example of such heuristic, and provides a numerical example.

1. Introduction

Grid computing [4, 7] is the future computing paradigm for enterprise applications [5, 6]. Large scale grids are complex systems, composed of thousands of components belonging to disjoined domains. Planning the capacity to guarantee quality of service (QoS) in these environments is a challenge because global Service Level Agreements (SLA) depend on local SLAs, i.e., SLAs established with components that make up the grid. These components are generally autonomous and join the grid as part of a loose federation. Only if all these partial SLAs are satisfied, the global SLA will be satisfied.

An enterprise application running on a grid is composed of a set of SLA-constrained sub-tasks demanding different types of services and resources. The grid middleware must be smart enough to schedule tasks among the available resources (processors, data storages, service providers, network links) in order to satisfy the SLAs at the lowest possible cost.

The main goal of this paper is to formalize the resource allocation problem for SLA-constrained grid applications. The problem consists in finding the optimal resource allocation that minimizes total cost while preserving execution time SLAs. It is well-known that this problem is NP-hard. Thus, we provide an extensible framework for building heuristic solutions, present an example of such heuristic, and provide a numerical example.

To the best of our knowledge, this is the first work that addresses the problem of optimal resource allocation in grid systems in a holistic maner. We consider different types of resources, the dependency among tasks, cost, and SLA constraints. Existing work on task scheduling and resource allocation in heterogeneous multiprocessor environments [13, 14] typically addresses the problems of minimizing the execution or completion time and maximizing the throughput. Existing work on task scheduling in grid systems [3, 17, 18] assumes independent tasks that are executed on compute nodes. In [3] the authors propose an application-level scheduling system for parameter sweep applications, which are applications composed of a set of independent (no task precedences) tasks. The heuristic scheduling algorithms implemented—min-min, max-min, and Sufferage [11, 10]—try to minimize the task completion time. The Grid Harvest Service [18, 17] is a performance prediction and task scheduling system. Its heuristic scheduling algorithm assigns a task to the least loaded machine in order to minimize the completion time. The scheduler used in the ICENI's Grid Middleware considers scheduling of resources to components connected as a directed acyclic graph [19]. Four scheduling algorithms were evaluated (random, simulated annealing, game theory, and best of n random). Only computation and network resources were considered. Other relevant work in the area of resource allocation, reservation, brokering, and scheduling can be found in [8, 15, 16]. Computational economy has been used in resource managers and schedulers for the grid [1, 2].

The rest of this paper is organized as follows. Section two presents a motivating example that highlights the need for QoS in enterprise grid applications. The following sec-

tion formulates our assumptions on the grid architecture and presents a task graph model for enterprise grid applications. Section four presents the task graphs for our motivating example application. Section five presents a formal model for the cost minimization SLA-constrained resource allocation problem. Then, a framework for building heuristics along with an example heuristic and a numerical example are presented. The paper concludes with final observations and discussion of future work.

2. Motivating Example

To motivate the discussion and illustrate the concepts presented in the remaining sections, we consider a large insurance company (IC) that offers many types of insurance products (e.g., automobile, boat, home, and business). The primary goal of the IC is to increase its profit by minimizing risks and attracting/retaining more customers. In the insurance business, the premiums paid by customers are a function of the risk posed by the insurance policy. Traditionally, insurance companies use a combination of actuarial data with some minimal amount of personal data to establish the risk associated with a policy and thus its premium. If the risk assessment yields an excessive risk, premiums increase and the insurance company may lose customers.

In an effort to increase its profits, the insurance company of our motivating scenario is moving towards a highly customized risk assessment model (RAM). Under this approach, a much larger number of information sources about a customer are queried to obtain a much richer set of inputs to the RAM. Contrary to the previous model that places a customer under a large category (e.g., all non-smoking male straight A college students under the age of 25), the new model provides a risk rating specific to a given customer (e.g., John Doe who besides being a non-smoking male straight A college student under the age of 25, is 23 years old and is a senior at the Computer Science Department at George Mason University, is a member of the IEEE, a member of a programming team that won a regional prize in an ACM-sponsored programming contest, has a very good credit record, undergoes a physical exam every year and is in perfect health, and has a clean record with federal, state, and local law enforcement agencies). Clearly, these customized models are much more sophisticated and significantly more compute- and data- intensive. These models are decomposed into many parallel tasks that run at the various resources provided by the compute grid.

The IC plans to establish a Web portal through which prospective customers can obtain insurance policy quotes. Users will be able to request three different types of quotes: immediate, non-immediate, and delayed. Immediate requests use simpler RAMs and can return results in a few seconds. Non-immediate requests return results in a few minutes while the user is still online and use RAMs that are more complex than the ones used by immediate requests. Finally, delayed requests may take hours to process and use fairly sophisticated RAMs. In the latter case, users are notified by an e-mail containing a link to the portal in order to view the details of their quote. Thus, customers can obtain potentially lower premiums if they are willing to wait longer for sophisticated and complex RAMs to execute.

The IC does not want to invest in additional computing resources to run the new models and decides to use a compute grid to harness unused cycles of all its computers (from desktops to mainframes) connected to its worldwide network. The grid that supports the IC application will schedule resources according to the type and compute requirements of the different types of RAMs needed to evaluate the three categories of requests.

The new customized RAMs draw their inputs from a large number of public and private data sources. These sources include health insurance companies, law enforcement agencies, financial organizations, departments of motor vehicles, professional and scientific organizations, federal, state, and local government agencies, and weather-related sources used to assess the risk of home and business insurance policies.

When a user logs in to the new application, the risk assessment application can have access to user's private data on behalf of the user at the various databases owned and managed by a set of organizations selected by the user. This way, the customer selectively trades-in access to some of his/her private data for lower premiums.

We use Fig. 1 to illustrate a scenario in which an insurance quote is requested by a customer of the IC through its Web portal. The notation used in the figure is patterned after the data mining example presented in [5].

The Web portal application contacts the factories at the RAM and DB service providers to request the creation of an instance of a RAM and of a database to be used to evaluate the risk of the policy (1). These instances have a specified lifetime, which can be extended by the application if needed. A RAM object is then instantiated along with the appropriate database (2). As the RAM is evaluated, it will generate queries (3) to various database services belonging to financial organizations, law enforcement agencies, health insurance organizations, and others not shown in the figure. The results (4) populate the database at the database service provider. The RAM instance queries this database (5) and uses the results (6) to provide an insurance quote (7) to the Web portal application.

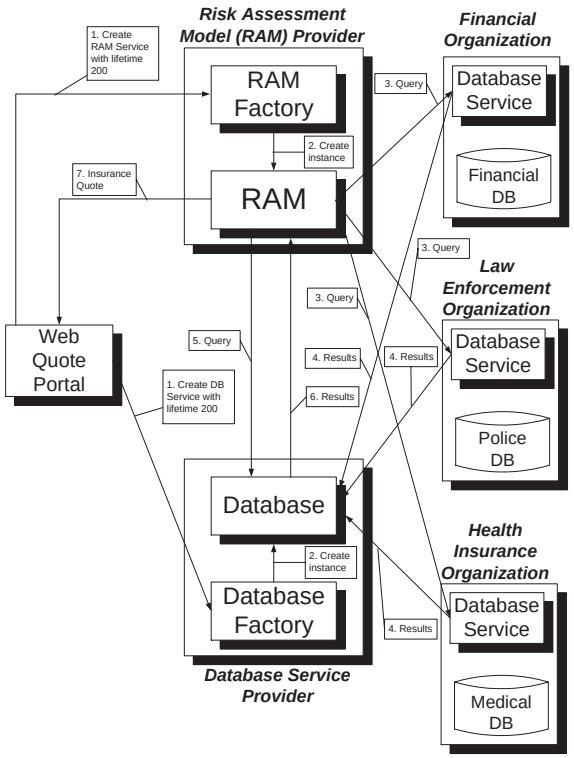

Figure 1. Quote request scenario.

3. Grid Applications and Resources

There are several important optimization problems related to the allocation of resources in a grid environment. First, we state the assumptions associated with resources and with the applications that use these resources.

Resources are assumed to be shared by more than one grid application. Four classes of resources are considered: compute resources, service provider (SP) resources, network resources, and data storage resources. Compute resources provide cycles to computations (e.g., RAM evaluation). Service providers are nodes that provide services upon request (e.g., access to financial, law enforcement, and health insurance databases). Network resources provide the bandwidth necessary for nodes of a grid to communicate and transfer data. Finally, data storage resources provide storage for data generated during the computation of a grid application. For instance, a delayed insurance premium quote request may generate a significant amount of temporary data to be used during the evaluation of a RAM.

There may be several instances of a resource of a given class (e.g., several SPs that provide access to law enforcement databases). Service provider nodes may offer the same functionality (e.g., access to the law enforcement database) with different Service Level Agreements (SLA) and at different costs. SLAs are contracts between SPs and its users. These contracts specify acceptable levels for various QoS metrics and usually associate a cost with (and sometimes penalties for non-compliance) a desired level of service.

Thus, application-level (i.e., global) SLAs have to be mapped into resource-level (i.e., local) SLAs in a way that minimizes total application cost. This problem is called the *SLA mapping problem* [12]. To illustrate, consider the example of an application that requires a global execution time SLA equal to 60 seconds. Suppose that three different types of service providers (SP) must be used by the application. Each offers two types of services, slow and fast, at different costs as illustrated in Table 1. Then, assuming that each type of service is used only once and that the services are used in sequence, the optimal solution (i.e., the one with the lowest cost that satisfies the SLA) is the one that uses the slow service of provider 1 and the fast service of providers 2 and 3. This yields an execution time of 45 seconds at a cost of 135 cents. Of the eight possible combinations of service providers and SLAs, four are not feasible because they violate the global SLA of 60 seconds.

The assumptions about a grid application are as follows. An application is formally specified by a task graph (see Figs. 2-4) composed of three types of nodes:

- *Computation task nodes.* Specify nodes where computation is performed (an example is the execution of part of a RAM evaluation); these nodes are represented by circles in the task graph.
- *Service providing nodes.* These nodes are represented by rectangles and indicate the invocation of a service by a computation task.
- *Logical data storage nodes.* These nodes are represented by disk icons and denote data stores used by tasks. These logical data storage elements have to be mapped into physical data storage devices, which may be accessed over a network. We will use the term data storage to indicate logical data storage and will explicitly use physical data storage to differentiate between the two.

For better readability of a task graph, service providing nodes and data storage nodes may appear more than once in the graph to avoid line crossings.

SP Type	Type of Service	Response Time SLA (sec)	Cost/Request (cents)
1	fast	5	50
1	slow	10	40
2	fast	20	65
2	slow	35	95
3	fast	15	30
3	slow	25	65

Table 1. Example of SLA mapping

An application task graph has the following types of arcs:

- *Precedence arc.* An arc from computation task t_i to task t_j indicates that t_j can only start after t_i finishes and after t_i has transmitted all the data it needs to send to t_j. A label on this type of arc indicates the number of bytes transmitted from t_i to t_j.
- *Service request arc.* This arc goes from a computation task node t_i to a service providing node s_j and indicates that task t_i invokes service s_j during its execution. A label on this arc indicates the number of times s_j is invoked per execution of t_j. A given task t_j may use any number of services in any combination of sequential and parallel invocations. To indicate that services are invoked sequentially, arcs are introduced between service provider nodes: if t_i uses s_j and s_k sequentially, an arc from s_j to s_k means that s_k will be used after s_j.
- *Data storage arc.* These arcs can go from task to data storage nodes and vice-versa. An arc connecting a computation node t_i to a data storage node d_k indicates that task t_i generates data storage d_k. An arc from d_k to t_i indicates that t_i uses data from storage element d_k and therefore cannot start until the data storage is generated. A label in a data storage arc indicates the number of bytes read/written from/to the data storage.

An application described by this type of task graph is a logical description and may be mapped in many different ways to physical resources such as computing resources, physical data storage resources, networking resources, and specific service providers.

4. Examples of Task Graph Representations

This section presents the logical structure of the three different types of RAM applications using the task graph notation.

Immediate services must provide a fast response time ($1-8$ seconds) to the customers to avoid that they abandon the IC web site before receiving an insurance premium quote. To meet this SLA, the instance of the RAM process is executed on a single high-capacity node to reduce communication and overhead costs for process coordination and data collection. Customer-related information is obtained from high-speed database service providers. The RAM used in this case is a simple model and the results are valid for large classes of customers that have common characteristics. Thus, customer-related information cached in the IC's network may be used. Figure 2 shows the task graph for the immediate service. Customer data are collected by task t_1 and are used by the model initialization task t_2, which obtains some local information from data storage d_1 and queries the service provider databases S_1, S_2 and S_3 (financial, law enforcement, and health insurance databases) through the interface tasks t_3, t_4, and t_5, using grid technologies. The query results are collected by task t_6, which updates internal database d_2, which is used by the final output task t_7 that provides the results to the customer. This scenario is the simplest of the three: there is no parallel execution of the RAM evaluation task, only parallel access to the external databases. On the other hand, this scenario has the highest cost since it uses high-performance service providers and high capacity computation resources.

Non-immediate services use more complex RAMs than immediate services and have a less stringent SLA; the customer agrees to wait between 10 and 60 seconds for a premium quote. The RAM evaluation application may be parallelized in subtasks based on functional criteria. As shown in Fig. 3, after data collection by task t_1, three specialized RAM evaluation tasks—t_2, t_3 and t_4—are started in parallel. These tasks obtain information cached at the IC's internal network data storage d_0. Additionally, each of these tasks read data from specialized external data storages d_1, d_2, and d_3 and access specialized service providers s_1 through s_3 through the interface tasks t_5 through t_7, respectively. Task t_8 collects the results of the previous subtasks and creates data storage d_4, which is then used by task t_9 to generate the output to the user. This more complicated scenario can be executed with less expensive resources than the previous one. Compute cycles in desktop workstations can be used but communica-

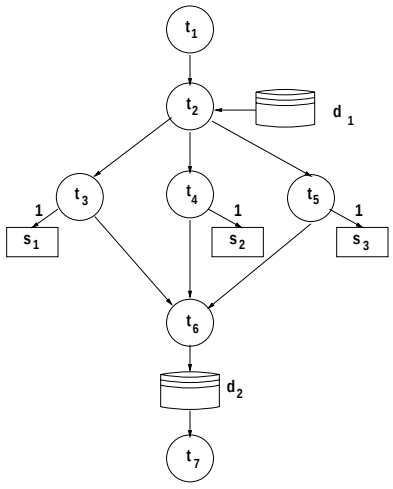

Figure 2. Task graph for immediate quote service.

tion overhead must be controlled and again high speed DB services must be used.

Delayed services are computationally-intensive since they use complex RAMs but no stringent execution time limits are imposed; thirty minutes to three hours is acceptable to return the best possible premium quote. Each RAM functionality (financial logic, law enforcement logic, and health insurance logic) is parallelized into sub tasks. For example, in the task graph of Fig. 4, tasks t_2 and t_3 deal in parallel with the same functional aspect (e.g., financial logic) of the RAM application. In fact, tasks t_2-t_4 in Fig. 3 are broken down into tasks t_2-t_7 in Fig. 4. Each of these six tasks communicate with its own service access interface task (i.e., tasks t_8-t_{13}). Tasks t_{14}-t_{16} collect the results of the parallel subtasks of each functional aspect of the RAM evaluation and generate databases d_4-d_6 containing these results. Task t_{17} reads these databases and generates a consolidated database d_7 that is read by task t_{18}, which produces results to the user. The task decomposition in this scenario allows the IC to use rather inexpensive computational, communication, and service providing resources.

5. A Formal Model of Resource Allocation

Before formalizing the resource allocation problem, some notation has to be defined. Consider the following notation for application-related parameters.

- $\mathcal{T}$: set of tasks of the application,
- T: execution time, in sec, of an application,

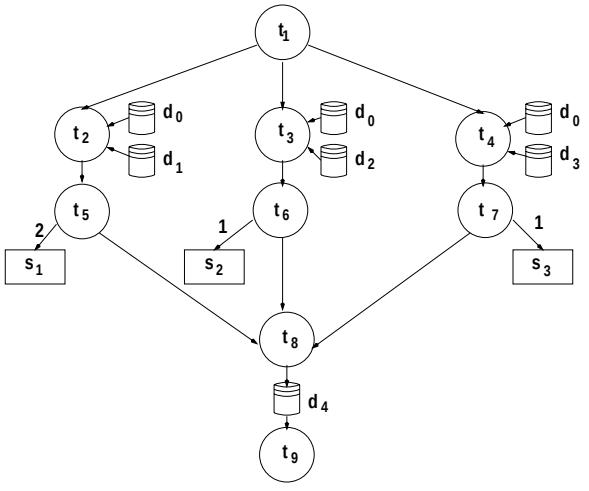

Figure 3. Task graph for non-immediate quote service.

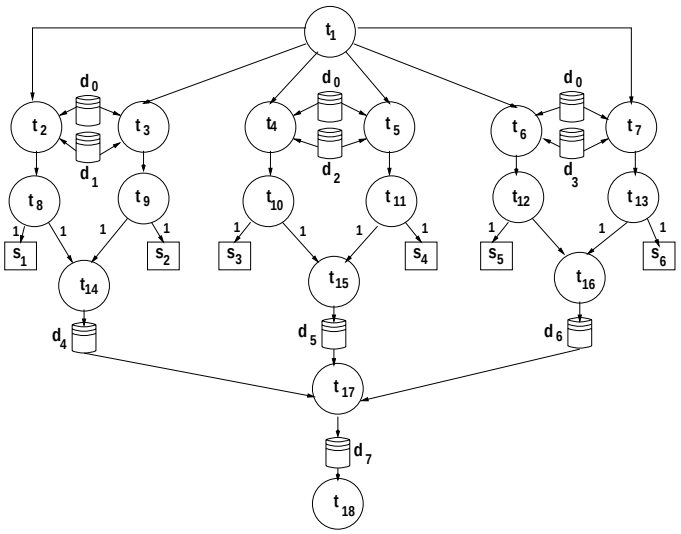

Figure 4. Task grah for a delayed service.

- NC_i: number of compute cycles, in millions of cycles, required by task i,
- C_i: execution time, in sec, of task i,
- $V_{i,s}$: average number of times that task i invokes logical service provider s,
- $x_{i,j}$: total amount of data, in Mbits, transferred from task i to task j,
- $T^k_{i,j}$: network time, in seconds, needed by task i to send data to task j over network resource k,
- $w_{i,m}$: amount of data, in Mbytes, transferred (i.e., written) from task i to logical data storage m,
- $r_{m,i}$: amount of data, in Mbytes, transferred (i.e., read) by task i from logical data storage m.
- $W_{i,m}$: time, in seconds, needed by task i to write into logical data storage m,
- $R_{m,i}$: time, in seconds, needed by task i to read from logical data storage m,
- N_{LDS}: number of logical data storage nodes used by the application, and
- N_{LSP}: number of logical service providers used by the application.

The following notation is related to resources and their performance.

- $\mathcal{C}$: set of computing resources. $\mid \mathcal{C} \mid = C$,
- $\mathcal{S}$ set of available physical service providers. $\mid \mathcal{S} \mid = S$,
- $\mathcal{N}$: set of available network resources. $\mid \mathcal{N} \mid = N$,
- $\mathcal{D}$: set of physical data storage resources. $\mid \mathcal{D} \mid = D$,

- s_k: speed, in millions of cycles per second, of computing resource k ($k = 1, \cdots, C$),
- R_k: response time, in seconds, of a service request at service provider k ($k = 1, \cdots, S$),
- B_k: bandwidth, in Mbps, of network resource k ($k = 1, \cdots, N$),
- L_k: latency, in seconds, of network resource k ($k = 1, \cdots, N$), and
- X_k: transfer rate, in Mbytes/sec, of physical data storage resource k ($k = 1, \cdots, D$).

The following notation is used to indicate cost parameters.

- c_k^c: cost, in \$/sec, per second of usage of compute resource k ($k = 1, \cdots, C$),
- c_k^b: cost, in \$/Mbps, per unit of bandwidth measured in Mbps provided by network resource k ($k = 1, \cdots, N$),
- c_k^d: cost, in \$/Mbyte, for each Mbyte of data transferred to/from data storage resource k ($k = 1, \cdots, D$), and
- c_k^s: cost, in dollars, for service provider k ($k = 1, \cdots, S$) to fulfill a service request. Physical service providers that provide the same functionality with different response time SLAs are considered different resources.

The variables defined in this section are assumed to represent either a constant known value, an estimate of the true value, or an average value obtained from previous observations.

5.1. Elements of the Execution Time of a Grid Application

The execution time, T, of a grid application can be computed as a function of the following components:

- *Task execution time.* The execution time of task t_i when running on computation resource k is given by

$$C_i = \frac{NC_i}{s_k} + \sum_{k=1}^{K_s} V_{i,k} \times R_k + \sum_{\mathcal{S}_p\ (t_i)\ \in \Pi(t_i)} \max_{s \in \mathcal{S}_p\ (t_i)} \{R_s\} + \sum_{m=1}^{N_{\mathrm{LDS}}} (W_{i,m} + R_{m,i}) \quad (1)$$

  where K_s is the number of services sequentially invoked by task t_i, $\mathcal{S}_p\ (t_i)$ is a set of services invoked in parallel by task t_i, $\Pi(t_i)$ is the set of sets of parallel service invocations of task t_i.

- *Network time.* The network time $T_{i,j}^k$ associated with the transmission of $x_{i,j}$ Mbits of data between tasks i and j over network resource k is a function of the latency and bandwidth of this network resource. As a first approximation, we assume that

$$T_{i,j}^k = L_k + x_{i,j}/B_k. \quad (2)$$

- *Data transfer time.* The write and read times for task i on data storage m, when it is mapped to physical data storage resource k are given by

$$W_{i,m} = \frac{w_{i,m}}{X_k} \quad (3)$$

$$R_{m,i} = \frac{r_{m,i}}{X_k} \quad (4)$$

5.2. Logical to Physical Mapping

The following mappings are required to compute the execution time T of a grid application: a) tasks to computing resources, b) communication patterns to network resources, c) service providers to physical service providers, and d) data storage elements to physical storage elements. More formally, a mapping $\mathcal{M} = (TC, CN, SPA, ST)$ where:

- TC: a task to compute resource mapping matrix. This is a $\mid \mathcal{T} \mid \times C$ matrix such that $TC[i,k] = 1$ if task i is allocated to compute resource k and zero otherwise.
- CN: matrix of network resource allocations. This is a $\mid \mathcal{T} \mid \times \mid \mathcal{T} \mid$ matrix such that $CN[i,j] = k$ if $x_{i,j} \neq 0$ and if network resource k is used for the communication between tasks i and j. $CN[i,j] = 0$ otherwise.
- SPA: set of service provider allocation matrices. Each element SPA_i of this set corresponds to a task t_i. The matrix SPA_i is an $N_{\mathrm{LSP}} \times S$ matrix. $SPA_i[s,k] = 1$ if service provider s is instantiated into physical service provider k for task t_i, and zero otherwise.
- ST: a data storage allocation matrix. This is a $N_{\mathrm{LDS}} \times D$ matrix such that $ST[m,k] = 1$ if storage element m is mapped into physical storage element k, and zero otherwise.

Once the mapping $\mathcal{M}$ is known, the execution time T can be easily computed from the structure of the task graph using the execution time components described previously. For example, from Fig. 3 we can see that the execution time, T, of the task graph of Fig. 3 is

$$T = C_1 + \max\{T_{1,2}^* + C_2 + T_{2,5}^* + C_5 + T_{5,8}^*, T_{1,3}^* + C_3 + T_{3,6}^* + C_6 + T_{6,8}^*, T_{1,4}^* + C_4 + T_{4,7}^* + C_7 + T_{7,8}^*\} + C_8 + C_9. \quad (5)$$

The terms with an "*" as a superscript indicate that the actual value of the term depends on the allocation of resources, the network in this case.

6. The Optimization Problem

The cost $C(A, \mathcal{M})$ of running application A using mapping $\mathcal{M}$ is given by

$$
\begin{aligned}
C(A,\mathcal{M}) \quad = \quad & \sum_{i=1}^{|\mathcal{T}|}\sum_{k=1}^{C} TC[i,k] \,.\, \frac{NC_i}{s_k} \,.\, c_k^c + \\
& \sum_{i=1}^{|\mathcal{T}|}\sum_{j=1}^{|\mathcal{T}|} c^b_{CN[i,j]} \,.\, B_{CN[i,j]} + \\
& \sum_{i=1}^{|\mathcal{T}|}\sum_{s=1}^{N_{\mathrm{LSP}}} V_{i,s} \sum_{k=1}^{S} SPA_i[s,k] \,.\, c_k^s + \\
& \sum_{m=1}^{N_{\mathrm{LDS}}} \left(\sum_{i=1}^{|\mathcal{T}|} (w_{i,m} + r_{m,i}) \right) \sum_{k=1}^{D} ST[m,k] \,.\, c_k^d
\end{aligned}
\tag{6}
$$

The first term in Eq. (6) represents the total computing cost. The second term indicates the cost of using networking resources. The third term represents the cost of using the various service providers and the last term is the cost of using data storage nodes.

So, the optimization problem to be solved is: given a grid application A, find the mapping $\mathcal{M}$ that minimizes the cost $C(A, \mathcal{M})$ while satisfying the execution time SLA of $T \leq T_{\max}$. This problem is an extension of the task-processor allocation problem, which is NP-hard. However, heuristics can be devised to tackle the problem, as described in the next section.

7. Heuristics

The optimization problem described in previous sections is a multi-resource assignment problem. Simpler versions of this problem appear in the context of task to processor assignment in multiprocessors [13, 14]. In the case of a grid, the resources that must be scheduled are not only processors (as in the case of multiprocessors) but also service providers, network resources, and storage resources. It is outside the scope of this paper to present a complete analysis of possible heuristics and their efficacy. However, we present a framework that can be used to design several heuristics and illustrate the framework through an example.

The heuristic scheduling framework is based on the work of Menascé, Porto, and Tripathi [13]. The authors of that work consider that a scheduling algorithm is composed of an envelope and a heuristic. Therefore, we envision a scheduling algorithm as consisting of a loop that is executed until all logical entities (tasks, communication pairs, logical SPs, logical data storage elements) are allocated to physical resources (computing resources, networks, physical SPs, and physical data storage resources). Inside the loop, a *domain selection* procedure determines the subset of the not yet assigned tasks and resources that are considered for assignment at that step. Let $\mathcal{T}^+$, $\mathcal{C}^+$, $\mathcal{S}^+$, $\mathcal{N}^+$, and $\mathcal{D}^+$, represent the task, compute resource, service provider, network, and data storage domains, respectively. The *heuristic* procedure allocates elements of each domain to physical resources. Finally, a *domain update* procedure updates the domains according to the results of the heuristic.

It was shown in [13] that a very effective envelope is the Deterministic Execution Simulation (DES). This envelope simulates the execution of the application according to the task graph. A task t enters the task domain $\mathcal{T}^+$ when it becomes "schedulable" according to the task graph (i.e., all the predecessors of t in the task graph have finished in DES and the communication from all its predecessors was received by t). A task leaves the task domain $\mathcal{T}^+$ when it is allocated by the heuristic. The domain of physical resources is updated as these resources become busy or available in the deterministic execution simulation.

We illustrate the approach by describing first a heuristic for the allocation of tasks to computing resources with the goal of minimizing cost while keeping the total execution time below an SLA value of $T_{\max}$. This description ignores the allocation of other resources. We discuss later the extension to these other resources.

Using DES, every time that a task finishes, new tasks may become enabled according to the task graph structure. We call these instants *allocation instants* because new resource allocations may have to be performed at these points. The following two rules explain the heuristic used at an allocation instant.

1. If $\mid \mathcal{T}^+ \mid = 1$, then assign the task in $\mathcal{T}^+$ to the least expensive computing resource in $\mathcal{C}^+$.
2. If $\mid \mathcal{T}^+ \mid > 1$ then the tasks in $\mathcal{T}^+$ are allocated to the $\mid \mathcal{T}^+ \mid$ least expensive computing resources in $\mathcal{C}^+$. The allocation of tasks to this subset of $\mathcal{C}^+$ is done in such a way that tasks with the largest computational demand (i.e., largest number of cycles) are allocated to the fastest computing resources.

After the execution of rules 1 or 2, the total cost is updated as a result of the allocation and the completion time of each task allocated is updated according to the speed of the computing resources they have been assigned to. If the allocation done in rules 1 or 2 above results in a violation of the total execution time $T_{\max}$, then the least expensive computing resource is removed from $\mathcal{C}^+$ and the algorithm backtracks to the previous allocation instant. The allocation is then attempted again with the reduced $\mathcal{C}^+$. If $\mathcal{C}^+$ becomes empty the heuristic backtracks to the previous allocation instant.

We illustrate the approach with the help of the non-immediate quote application and assume that the number of cycles, in million of cycles, for tasks 1 through

9 is given by $(NC_1, \cdots, NC_9) = (1000, 3000, 5000, 4000, 1500, 1500, 1500, 4000, 1000)$. The speed and cost of the six available processors is $(s_1, \cdots, s_6) = (1000, 1000, 2000, 2000, 3000, 4000)$ millions of cycles and $(c_1^c, \cdots, c_6^c) = (.20, .20, .35, .35, .50, .70)$ dollars per second of usage. The steps of the heuristic are described below for a value of $T_{\max} = 10$ sec and are illustrated by Fig. 5. In what follows, T represents an allocation instant. The value of cost is updated based on the allocation performed in the previous allocation instant.

- $T = 0$, cost= 0, $\mathcal{T}^+ = \{t_1\}$, $\mathcal{C}^+ = \{p_1, \cdots, p_6\}$: allocate t_1 to p_1.
- $T = 1$, cost= $1 \times 0.2 = 0.2$, $\mathcal{T}^+ = \{t_2, t_3, t_4\}$, $\mathcal{C}^+ = \{p_1, \cdots, p_6\}$: allocate t_2 to p_2, t_3 to p_3, and t_4 to p_1.
- $T = 3.5$, cost= $0.2 + 2.5 \times 0.35 + 4 \times 0.2 + 3 \times 0.2 = 2.475$, $\mathcal{T}^+ = \{t_6\}$, $\mathcal{C}^+ = \{p_3, \cdots, p_6\}$: allocate t_6 to p_3.
- $T = 4$, cost= $2.475 + 0.75 \times 0.35 = 2.7375$, $\mathcal{T}^+ = \{t_5\}$, $\mathcal{C}^+ = \{p_2, p_4, p_5, p_6\}$: allocate t_5 to p_2.
- $T = 4.25$, cost= $2.7375 + 1.5 \times 0.2 = 3.0375$, $\mathcal{T}^+ = \{\}$, $\mathcal{C}^+ = \{p_3, p_4, p_5, p_6\}$: no task can be allocated.
- $T = 5$, cost= 3.0375, $\mathcal{T}^+ = \{t_7\}$, $\mathcal{C}^+ = \{p_1, p_3, p_4, p_5, p_6\}$: allocate t_7 to p_1.
- $T = 5.5$, cost= $3.0375 + 1.5 \times 0.2 = 3.3375$, $\mathcal{T}^+ = \{\}$, $\mathcal{C}^+ = \{p_2, p_3, p_4, p_5, p_6\}$: no task can be allocated.
- $T = 6.5$, cost= 3.3375, $\mathcal{T}^+ = \{t_8\}$, $\mathcal{C}^+ = \{p_1, \cdots, p_6\}$: allocate t_8 to p_1.
- $T = 10.5 > 10$ ($T_{\max}$). Backtrack to previous allocation instant (see arrow labeled (1) in Fig. 5) and remove p_1 from $\mathcal{C}^+$.
- $T = 6.5$, cost= 3.3375, $\mathcal{T}^+ = \{t_8\}$, $\mathcal{C}^+ = \{p_2, \cdots, p_6\}$: allocate t_8 to p_2.
- $T = 10.5 > 10$ ($T_{\max}$). Backtrack to previous allocation instant (see arrow labeled (2) in Fig. 5) and remove p_2 from $\mathcal{C}^+$.
- $T = 6.5$, cost= 3.3375, $\mathcal{T}^+ = \{t_8\}$, $\mathcal{C}^+ = \{p_3, \cdots, p_6\}$: allocate t_8 to p_3.
- $T = 8.5$, cost = $3.3375 + 2 \times 0.35 = 4.0375$, $\mathcal{T}^+ = \{t_9\}$, $\mathcal{C}^+ = \{p_1, \cdots, p_6\}$: allocate t_9 to p_1.
- $T = 9.5$, cost = $4.0375 + 1 \times 0.2 = 4.2375$, $\mathcal{T}^+ = \{\}$, $\mathcal{C}^+ = \{p_1, \cdots, p_6\}$: all tasks have been allocated. Stop.

The final allocation assigns tasks t_1, t_4, t_7, and t_9 to computing resource p_1, tasks t_2 and t_5 to computing resource p_2, and tasks t_3, t_6, and t_8 to computing resource p_3. The total cost is \$4.2375 and the total execution time is 9.5 seconds.

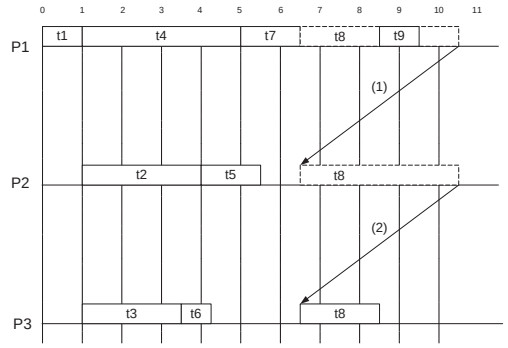

Figure 5. Example of a heuristic schedule.

The extension of this heuristic to the allocation of other resources assumes that are two types of allocations instants: start task allocation instants and task completion allocation instant.

1. *Network resource allocation.* Networks are allocated at task completion allocation instants. If a completing task needs to communicate with any of its successors, network resources are allocated at this time using the following heuristic. If only one network is needed (i.e., the completing task has only one successor), then pick the least expensive network from the network domain $\mathcal{N}^+$. If $n > 1$ networks are needed, then select the n least expensive networks from $\mathcal{N}^+$ and allocate them to the task communication pairs by assigning the pairs with highest number of bits to transmit to the fastest networks.
2. *Service Provider allocation.* SPs are allocated at task start allocation times. If task t uses one or more SPs, then they are allocated by selecting first from the least expensive SPs from $\mathcal{S}^+$.
3. *Data Storage allocation.* Physical data storage resources are allocated at task start time if the task reads from or writes to a data store. As before, the allocation is made by picking the least expensing resource from $\mathcal{D}^+$ first. Differently from the other cases, physical data storage resource allocation is global to the application. Thus, once an allocation from a logical to a physical data storage has been made, it has to be preserved by other tasks that may need the same logical data storage.

The above discussion left out some details. In particular, one needs to determine what happens when a backtrack to an allocation instant occurs. If the backtrack is to a task completion allocation instant, the least expensive network resource is removed from $\mathcal{N}^+$ and the allocation is attempted again. If the backtrack is to a task start instant, then there are three types of resources that can be involved: com-

puting resources, SPs, and physical data storage resources. If the task in case does not use any SP or data storage, then only the compute domain $\mathcal{C}^+$ has to be changed by removing the least expensive compute resource. If the task uses SPs and/or data storage resources, then a determination has to be made if the task is compute-bound, I/O bound, or service-bound. This determination is made based on how much time the task is spending, under the current allocation, in computing, doing I/O, and accessing service providers. Then, the least expensive resource is removed from the domain that has the largest contribution to the task execution time and the allocation is attempted again.

8. Concluding Remarks

Enterprise grid applications use a variety of resources including computing resources, networks, data stores, and service providers. The problem of optimally allocating resources to such applications in a way that minimizes cost while guaranteeing executing time service level agreements was formalized in this paper. This paper provided a framework to optimally allocate resources in grid environments compatible with the more stable grid architecture defined in [6, 7]. Because the problem is NP-hard, a framework for designing heuristics was presented. Our ongoing work consists of designing many such heuristics, implementing, and evaluating them.

Acknowledgements

The work of Daniel Menascé was partially supported by grant number NMA501-03-1-2033 from the National Geospatial-Intelligence Agency. The work of Emiliano Casalicchio was partially funded by the PERF project supported by the Italian MIUR under the FIRB program.

References

[1] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing," *J. Concurrency and Computation: Practice and Experience*, Special issue on Grid computing environments, 2002.

[2] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid," HPC Asia, Beijing, China, May 14-17, 2000, pp. 283–289.

[3] H. Casanova, G. Obertelli, F. Berman, R. Wolski (2000). The apples parameter sweep template: user-level middleware for the grid. *Proc. 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 60. IEEE Computer Society.

[4] I. Foster and C. Kesselman, *The grid: Blueprint for a new Computing infrastructure*, 1999.

[5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *IEEE Computer*, 35(6), 2002.

[6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, The physiology of the grid: an open grid services architecture for distributed systems integration, 2002.

[7] I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science* 2150, 2001.

[8] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," *Proc. Intl. Workshop on Quality of Service*, 1999.

[9] X. He, X.-H. Sun, and G. Laszewski, "A QoS guided scheduling algorithm for grid computing," *Proc. Int'l Workshop on Grid and Cooperative Computing (GCC02)*, Haiman, China, 2002.

[10] O.H. Ibarra and C.E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, 24(2), April 1977, pp. 280–289.

[11] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R.F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," *Proc.* 8^{th} *Heterogeneous Computing Workshop (HCW'99)*, 1999.

[12] D. Menascé, "Mapping Service Level Agreements in Distributed Applications," *IEEE Internet Computing*, 8(5), Sept./Oct. 2004.

[13] D. Menascé, S. Porto, and S. Tripathi, "Static heuristic processor assignment in heterogeneous multiprocessors," *Intl. J. High Speed Computing*, 6(1), 1994.

[14] D. Menascé, D. Saha, S. Porto, V. Almeida, and S. Tripathi, "Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures," *J. Parallel and Distr. Comp.*, 28(1), 1995, pp. 1–18.

[15] A. Othman, P. Dew, K. Djemame, and I. Gourlay, "Adaptive Grid Resource Brokering," *Proc. IEEE Intl. Conf. Cluster Computing*, 2003.

[16] J.M. Schopf and L. Yang, "Using Predicted Variance for Conservative Scheduling on Shared Resources," in *Grid Resource Management*, eds. J. Nabryski, J.M. Schopf, and J. Weglarz, Kluwer Academic, 2003.

[17] X.-H. Sun and M. Wu, "Grid harvest service: A system for long-term,application-level task scheduling," *Proc. Intl' Parallel and Distributed Processing Symposium (IPDPS03)*, 2003, IEEE Computer Society.

[18] M. Wu and X.-H. Sun, "A general self-adaptive task scheduling system for non-dedicated heterogeneous computing," *Proc. IEEE Cluster Computing Conference*, Hong Kong, 2003.

[19] L. Young, S. McCough, S. Newhouse, and J. Darlington, "Scheduling Architecture and Algorithms within the ICENI Grid Middleware," *Proc. UK e-Science All Hands Meeting*, Nothingham, UK, Sept. 2003, pp. 5–12.