An Adaptive Process Allocation Scheme in Grid Environment

Tibor Gyires
School of Information Technology
Illinois State University
Normal, Illinois 61790
USA

ABSTRACT

The Grid is an interconnected set of distributed compute servers. An application running on the Grid consists of processes, which can be executed in parallel or in a sequential manner. An application can specify application level and network level Quality of Service parameters including number of processors, memory, special software, network bandwidth, delay, jitter, packet loss, etc. We investigate the question: Which processes are allocated to which compute servers that collectively satisfy the application's resource requirements and optimize performance and cost parameters. We describe a protocol to identify those compute servers that can execute the application with minimal cost and provide the required application level and network level Quality of Service.

Keywords-Grid computing; Quality of Service; resource discovery; heuristic search;

1. INTRODUCTION

The term Grid describes a collection of geographically distributed resources shared by multi-institutional virtual organizations (VOs) in a coordinated manner [1]. A VO is a set of participants that share resources to perform some specific tasks. For instance, the members of an industrial group designing a new industrial unit or member institutions of a multi-national research project form a VO. Resources can be computers' processing power, storage devices, software services, special hardware (microscopes, telescopes, etc), and data. Coordinated resource sharing means that users share multiple resources by establishing and enforcing sharing agreements. We briefly review the Grid architecture using the chart from [1]:

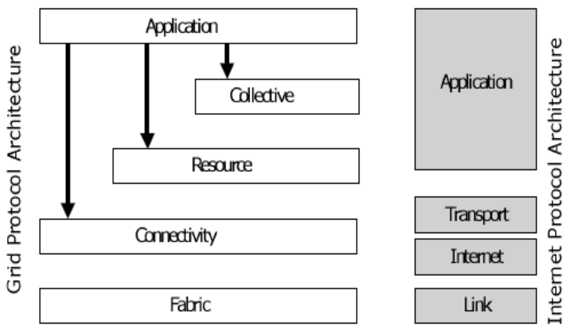


Figure 1. The Grid Architecture [1]

- The Fabric layer encompasses the resources, computers, software services, files, cluster of processors, etc.
- The Connectivity layer specifies the communications, security, and authentication protocols for transactions between resources.
- The Resource layer implements protocols for inquiring the state of resources and negotiates access to them.
- The Collective layer is responsible for the discovery, allocation, coordination, and scheduling of multiple resources.
- The application layer includes the Grid applications of VOs.

The components of the architecture depicted in Figure 1 are implemented in the Globus Toolkit [2] as a result of the open-source project Globus.

Our paper is relevant to the basic services of the Collective layer: Resource discovery, allocation, coordination, and scheduling of resources for the execution of applications. We concentrate on the issue of when an application can be decomposed into subprocesses which can be executed in parallel or sequentially by multiple compute servers of the Grid. Each process has specific application level and network level QoS requirements, such as number of processors, memory, special software, network bandwidth, delay, jitter, packet loss, etc. Collectively, we call the specification of these services Extended QoS (EQoS). Our resource discovery and allocation protocol, discussed in the paper, satisfies the following system properties:

a. In the Grid, services and users can join and leave a VO any time. The changes may not be detected immediately, leading to inconsistency between the registered data in the Grid Information Services [3] and the actual availability of resources. Due to this volatility of the Grid, a user should have the ability to receive the most current information possible on the availability of the requested EQoS without the involvement of a third party that may insert an additional delay in the resource discovery.

b. Resource providers charge a fee for their services. Users specify the EQoS requested for an application. Upon receipt of a request, the resource providers should be able to negotiate the cost of the resources with the users and dynamically readjust their resource allocation in order to satisfy a higher priority request.

Our protocol discovers the most suitable resource providers for the execution of the processes that can grant the requested EQoS with minimal cost. Section 2 describes related works in the literature. Section 3 presents our model. Section 4 specifies our resource discovery protocol in details. We illustrate the protocol in Section 5. A simulation model is presented in Section 6. We conclude our paper in section 7.

2. RELATED WORKS

Many resource discovery systems, like peer-to-peer systems, use names to identify resources [4, 5, and 6]. A well-known resource discovery service, the Domain Name System (DNS), is also based on names of the resources. Web search engines, in addition to names, can locate resources using search criteria. In MDS [7, 8] implemented in the Globus framework, information sources can register with index servers via a registration protocol. Index servers and users can use directory servers to discover resources from the registered information sources. MDS involves a communication phase with a third party server that may not have the current data. The additional phase increases the time of the discovery process. Other systems, although providing faster search algorithms using global resource identification, like in the Plaxton networks, [9] or location-independent names assigned to resources [10], lack the dynamics of the rapidly changing environment of the Grid and do not satisfy the system property a. discussed above.

Resource discovery can also be implemented by resource brokers as well as publisher/subscriber protocols. In publisher/subscriber protocols [11] an application subscribes for an event of a resource. An application-level scheduler could then register for events that match some predicates. By registering for similar events of several resources, the scheduler can select appropriate resources for the application. These protocols do not allow the resource providers to prioritize among subscribers by reallocating resources to satisfy a higher priority request. Users passively wait for resources to become available and resource providers passively wait for incoming requests. This approach is reactive rather than proactive and does not satisfy our criteria a. and b either. Our approach allows the readjustment of resources allocated for other users. That may turn out to be a better overall solution because it triggers a new discovery for more appropriate resources. Our approach is proactive in the sense that a request can alter the resource allocation at a provider to minimize some cost function.

The framework in [22] describes an extension of the Web Services Description Language (WSDL) and the Universal Description and Discovery Integration (UDDI) registry to include QoS properties necessary for the Open Grid Services Architecture's (OGSA) objectives [23]. The framework provides mechanisms for the service requesters to search for services based on application level, middleware level, and network level QoS criteria to provide QoS guarantees for service execution and to enforce these guarantees by implementing service level agreements based on a budget. This framework, similarly to the approach above, does not satisfy our criteria a. and b. either.

The authors of the paper [12] assume that in a VO there are one or more servers, called nodes that store and provide access to resource information. Users send requests to a known node that will respond with the requested resource's description in case the node has the resource; otherwise it forwards the request to another node. Intermediate nodes forward the request until the time-to-live parameter of the request expires or the requested resource is found. If an intermediate node has the information, it sends it back directly to the initiating node. The paper analyzes four protocols including ones that remember past experience of successful resource discovery, such as the number of answers a node replied to a similar request, the largest number of answers a node replied, etc. Our protocol goes beyond the mere discovery of the resources. It combines resource discovery, process allocation, and process execution utilizing statistical data on past performances of compute servers.

Our protocol shows some similarities to the Globus Architecture for Reservation and Allocation GARA [13] in the Globus toolkit. GARA implements mechanisms that enable the coordinated use of reservation and adaptation of resources for a process via support for dynamic feedback among entities involved in resource management decisions. Sensors associated with resources and resource managers permit application-level monitoring of resource state and reservation status, while online control mechanisms enable adaptive control of reservations. The main difference between our protocol and GARA is that our protocol focuses on the whole life span of all processes of an application including the creation, allocation, and execution phase and a feedback mechanism after the execution phase.

Resource Discovery and Allocation

There are several traditional research directions related to process/processor allocation which are applicable in Grid computing. One class of the algorithms assumes that the resource requirements of processes are known in advance. Related algorithms are based on known CPU and memory requirements and the matrix of the amount of traffic between each pair of processes. If the number of processors is smaller than the number of processes, then several processes are assigned to a single processor. These algorithms try to minimize the network traffic.

Another class of algorithms is based on centralized decision making**.** A coordinator maintains a usage table with one entry for each processor. The entries are periodically updated by messages sent from the processors whenever some events happen, such as the availability of some resources, changes in the utilization of CPUs, etc. Processor allocation is based on this table.

According to another categorization, processor allocation strategies can be divided in two broad classes. When a process is created, the local processor makes the decision where to run the process. Once the process is started it stays at that processor. This strategy is non-migratory. In the migratory strategy a process can be transferred to other processor even if it has already been started to execute. Processor allocation algorithms can further be classified if they are deterministic or heuristic, centralized or distributed, optimal or close to optimal.

Our algorithm belongs to another class that models a distributed system as a computerized market economy similar to the algorithms in [17, 18, and 19]. A processor announces some task to execute. Other processors, upon receiving the task announcement, estimate the cost it requires to execute the task, and send the cost estimates back to the processor announcing the task. The task is allocated to that processor, which sent the lowest estimated cost. A processor allocation mechanism tries to optimize some system parameters. Our protocol tries to optimize the overall execution time and the communication cost of providing the requested Extended Quality of Service for process execution.

3. OUR MODEL

A Grid application may arrive at any compute server, which creates a process assigned to a processor for execution. A compute server may have one or more processors, but for simplicity, we assume that a compute server has only one processor. A new process is generated when a running process decides to fork or create a subprocess. Process creation is a recursive procedure; a subprocess may create further subprocesses until the parent process is completed. A process may create subprocesses for several reasons, for example:

- to gain performance by executing subprocesses parallel,
- to improve the efficiency of multiprocessing in a compute server,
- a new, higher priority process arrives,
- a compute server doesn't have the required resources to execute the subprocess, etc.

In all of these cases the parent process may start searching for other compute servers to take over the subprocesses. We characterize the Grid as a stochastic environment owing to the following reasons:

- the system load is unpredictable,
- the kind of processes and the rate of their arrival in the system change in time randomly,
- processor allocation decisions cannot be made in a deterministic way,
- different compute servers have different capabilities and resources,
- the same process can be executed by more than one compute server, and
- there are frequent changes in the load level of a processor.

Each compute server has a knowledge base in the form of process trees describing the processes it knows how to execute. A process tree includes other processes that may not be executed locally. The execution part of a compute server is divided into two segments: the Resource Broker (RB) which manages the local resources and the Cost Estimator (CE) which can estimate the cost of process executions. The CE has knowledge of local conditions, resources required by a process, and local capacities. We assume that a compute server E has the following types of knowledge:

- Process Tree: For each process T, E is able to execute, a Process Tree is maintained. It is a list of subprocesses which have to be completed before completing the process T at the root of the tree. The processes are arranged along an AND tree. The following chart depicts a partial process tree of T1:

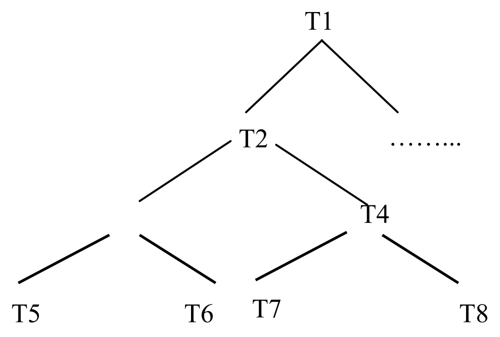


Figure 2. Partial Process Tree Corresponding to Process T1

For each process T in the process tree, compute server E maintains the following data in a Process Directory:

- Names of the compute servers able to execute the process.
- The M-factor for compute server I denoted by M (I) is a measure of the quality and efficiency of I in executing process T based on "j" previous performances of T as calculated by compute server E. Small (close to zero) values of the M-factor indicate high quality and efficiency, and high values indicate that compute server I is a poor performer of process T. A local Grid administrator provides initial values of the M-factor. The calculation of the M factor will be discussed in a subsequent sections.
- Communication cost $d(T,T_1)$: Assuming that process T has a subprocess T_1 and process T is executed by compute server E, furthermore process T_1 is executed by compute server E_1, then the communication cost between compute servers E and $E_1$is denoted by $d(T,T_1)$. It includes the cost of transferring data between E and $E_1$with the requested network level Quality of Service.
- Execution cost m(T (E)): The cost of executing a process T by compute server E. It also includes the cost of providing application level QoS, the cost of integrating the outputs of other compute servers executing T's subprocesses, and the cost of preempting other processes from execution or reassigning resources in order to execute process T.

Successor operator (Γ): The motivation for using the successor operator at compute server E is to obtain all compute servers, which have the potential to execute a certain process. The steps in applying the operator at any compute server E are as follows:

Step 1. E multicasts a message to k qualifying adjacent compute servers with the least M-factor values announcing the request to execute a process T. The EQoS parameters (number of processors, memory, special software, network bandwidth, delay, jitter, packet loss, etc.) are also announced.

Step 2. The recipients of the multicast messages evaluate their own capability to execute the process and compare the origin of T to the ones currently maintained locally. If there is a match, i.e., if the compute server has already been involved in executing the process, it will not respond to the message. Otherwise the compute servers respond to E with an estimated cost associated with the process execution.

(Note: For simplicity, we assume that the multicast messages are sent to adjacent, qualifying compute servers only, since the Extended Quality of Service include QoS provisioning on the communications lines as well. Our protocol can easily be extended for sending multicast messages to all qualifying compute servers, not just adjacent ones.)

The responding compute servers are the successors of compute server E. The operator is used by every compute server along the potential execution path to explore the next candidate compute servers that can be assigned to processes in the process tree. This recursive algorithm of finding the next candidate compute server to form the execution paths of the processes is referred to as 'tree-expansion'. If the compute server and network characteristics change very frequently, then the Γ operator is applied every time a process starts execution. Otherwise, Step 1 and Step 2 could be done in advance and then cached for use when an actual process execution occurs.

We further assume that compute servers periodically send information to their neighbors on the available resources, such as number of processor available, bandwidth provided, special software and hardware, etc. These messages do not contain the fine granularity of the actual state of the resources.

Process Tree

The Process Tree implicitly defines an AND/OR tree. The root of the tree is the initial process assigned to a compute server. The successors of the root are recursively given by the Process Tree of the subprocesses. The processes with the same ascendant are in AND relation. Each process is assigned to at most one compute server. Assuming that more than one compute server can execute a process, the candidate compute servers are in OR relation. A branch of the example AND/OR tree belonging to process T1 of Figure 2 and the candidate compute servers E1 through E9 are shown in the following figure:

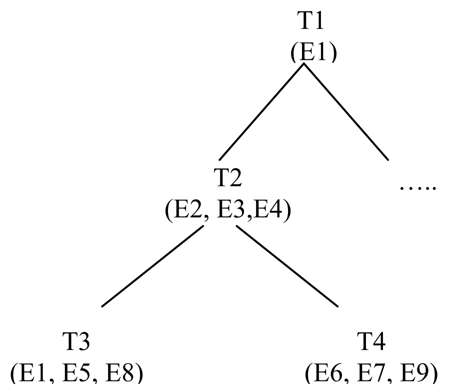


Figure 3. A Partial AND/OR Tree Representing Process T1

The subprocesses at the same level of the tree are in AND relations, the candidate compute servers in parentheses are in OR relations. For instance, subprocess T2 can be executed either E2, or E3, or E4. The compute servers are selected by the search procedure presented later. The recursive definition of a solved node in an AND/OR tree is similar to the definition in [20]:

1. The leaves are solved nodes. These nodes represent the execution of processes without the involvement of further compute servers.
2. If a non-leaf node has OR descendants, then it is solved iff at least one of its descendants is solved.
3. If a non-leaf node has AND descendants, then it is solved iff all of its descendants are solved.

The solution of the original process is then represented by a subtree of solved nodes, called solution tree. A node T (E) of the solution tree represents a process T assigned to compute server E.

The cost P(T (E)) of node T (E) of a solution tree is defined recursively:
- If process T doesn't have any subprocesses, then $P(T(E)) = m(T(E))$.
- If a process T has AND subprocesses T1, T2,…, Tn allocated to compute servers E1, E2,…, En, then
$P(T(E))=\Sigma_i [d(T,T_i) + P(T_i(E_i)) + M(E_i) + m(T(E))]$, i=1,2,..,n.
- If a process T has a subprocess T_j and the compute servers able to execute T_j are E_i, then
$P(T(E))= d(T,T_j) + P(T_j(E_i)) + M(E_i) + m(T(E))\}$, for any i=1,2,..,n,

where $d(T,T_j)$ is the communication cost, $M(E_i)$ is the M factor of compute server E_i, and m(T(E)) is the execution cost of process T by compute server E. Our goal is to find a solution tree with minimal cost. We call such a tree an optimal solution tree.

The search algorithm, implemented in our protocol, is derived from the Simple Recursive Best-First Search (SRBFS) [24] which is an extension of the IDA* [25]. Similarly to SRBFS we define the heuristic function P*(T(E)) as a cost estimate of the optimal solution tree:
- If process T doesn't have any subprocesses, then
$P^*(T(E)) = m^*(T(E))$, where m*(T (E)) is the estimate of the execution cost of process T by compute server E.
- If a process T has AND subprocesses $T_1, T_2,…, T_n$, then
$P^*(T(E)) =\Sigma_i [d^*(T, T_i) + P^*(T_i(E_i)) + M(E_i) + m^*(T(E))]$.
- If a process T has a subprocess T_j and the candidate compute servers are E_i, then
$P^*(T(E))= \min_i\{d^*(T,T_j) + P^*(T_j(E_i)) + M(E_i) + m^*(T(E))\}$, i=1,2,..,n, where $d^*(T,T_j)$ is the estimate of the communication cost related to the execution of processes T and its subprocess Tj.

A new application is submitted to a compute server E_1 along with the Extended Quality of Service parameters required by the application's processes. For reducing the complexity of the examples we don't identify the compute servers in the charts below. If compute server E1 can execute the process T1 and its subprocesses and it is not aware of other candidate compute servers, it starts and completes the execution of the process. Assuming that compute server E1 can execute the process T1 but cannot execute T1' subprocesses, then E1 has to make the decision about the location to execute the second process T2 of the application. It sends a request to k number of qualified processors with the smallest M-values along with an upper bound on the execution cost and the EQoS requirements of process T2. The upper bound is the maximum cost a compute server is willing to "pay" for the execution of the process. It is determined by the user submitting the application or the virtual organization the user belongs to. A compute server is qualified if it has been able to execute similar processes in previous cases. The selection mechanism includes the parameter M, which is a measure of a compute server's inefficiency (as calculated by its predecessor compute server along the process execution tree) based on its previous process executions. M may be different for different processes and is dependent on the specific process to execute. Small (close to zero) values of M indicate high quality and efficiency, and high values indicate that the compute server is unreliable and unstable in executing the process. A compute server's M value is recalculated after each process execution by the adjacent compute server that sent a process execution request to it. After each process execution it is determined how realistic a compute server's cost estimate has been with respect to the recalculated cost. By testing a statistical hypothesis it can be determined if it has been unrealistic. If the corresponding hypothesis H_0 is rejected no process execution request messages will be sent to this compute server in the future for process execution. If a compute server's cost estimate has been realistic, i.e., the corresponding hypothesis H_0 is accepted, the M value is recalculated. The closer the estimated cost is to the actual cost, the smaller is the M value. Subsequent values of M for each compute server are calculated using the statistical sampling method given in the Appendix. The motivation for using M is to enable the search process to learn from past performances and use this knowledge to select the most efficient compute servers for a given process tree.

The compute servers compute the estimated cost of each of the activities associated with execution of T2, such as the allocation or reallocation of resources, preempting existing but lower priority processes, etc. These cost estimates are sent back to E1, which will select the compute server with the smallest cost estimate. The procedure continues recursively. At each recursive step a compute server uses three arguments: The name of the compute server, an upper bound on the execution cost, and a subprocess. Each step expands the execution path by those children through which the estimated execution costs do not exceed the upper bound. (The first step assumes an upper bound of infinity at E1.) Each step returns the estimated cost along the path to a child, replacing parent values with the minimum of the estimated costs via the last children expanded, going backward along the path, until a better cost estimate is reached. Then, the procedure continues along that path. Generally, the upper bound on a child is equal to the minimum of the upper bound on its parent and the current value of its lowest cost sibling. Initially, a compute server is assigned an estimated cost by itself. After a recursive step this cost value will be equal to the minimum estimated cost path to the last child along the expanded subtree. We call it the compute server's stored value after the SRBFS algorithm in [24]. In the charts below the figures at the nodes of the tree denote the cost estimates of the compute servers. The upper bounds are in parentheses. We assume the process tree in Figure 2 and d=1 for simplicity. Compute server E1 announces T_2 to k number of qualified processors with the smallest M-values. Assuming that only three compute servers responded with their estimated cost for T_2 (10, 7, and 6), process T2 is tentatively assigned to the compute server with the smallest estimate, 6. A partial solution tree is shown in Figure 4.a. (For simplicity we also omit the identification of the subprocesses from the example figures.) At each level of the search tree a subprocess is tentatively assigned to a compute server. The compute server, which sent the smallest estimate (6) is selected to continue the search with the upper bound of the next lowest cost estimate 7. Similarly to E_1 it announces the next processes in the process tree, T_3 and T_4, to eligible compute servers with the smallest M-values. Figure 4.b shows the resulting partial solution tree. This path of the search tree exceeds the upper bound 7 and costs more than the other paths (7+5+2=14, 14+1=15), therefore the search stops on this branch of the tree and continues with the compute server with the smallest cost estimate 7 and the upper bound 10, as it is shown in Figure 4.c.

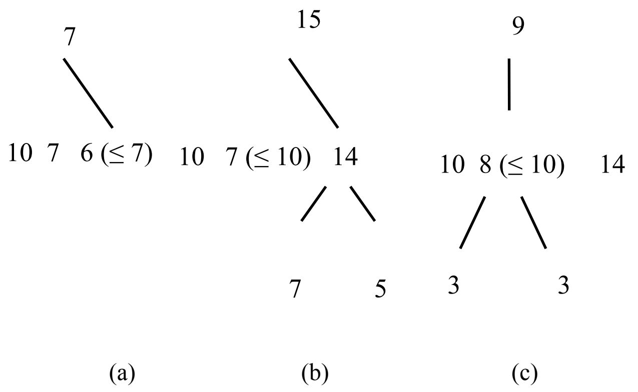


Figure 4. Partial Solution Tree

Since the upper bound 10 has not been exceeded on this branch, the processes T3 and T4 are tentatively assigned to the compute servers with estimated costs 3. The search continues from these compute servers announcing the processes T_5, T_6, and T_7, T_8 to eligible processors with the smallest M-factors. Figure 4.d depicts the new estimated solution tree. The resulting cost estimate (13) exceeds the upper bound 10, hence the search continues with the compute server with cost estimate 10 and upper bound 13. Figure 5.e displays the final estimated tree. Processes T_5, T_6, T_7, T_8 have been tentatively assigned to the compute servers at the leaves of the tree with cost estimates 2, 2, 1, and 1, respectively. The process execution tree is along the return path of the recursive steps:

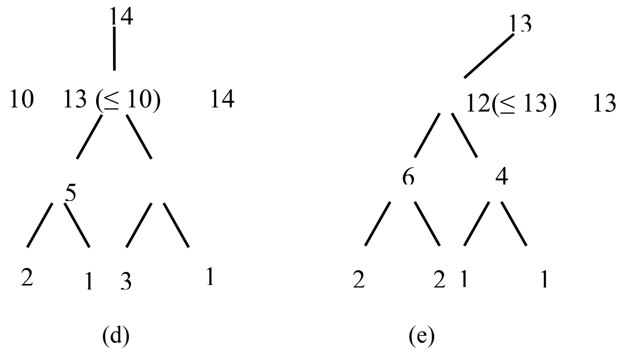


Figure 5. Building a Tentative Solution Tree.

Execution Phase

When the estimated solution tree T* has been found, the compute server E at the root node T(E) of the solution tree activates the actual execution of the subprocesses of T: It sends the message 'Perform' to the selected compute servers that in turn do the same along the solution tree. After executing the processes at the leaves, each compute server transmits the result to its predecessor, which in turn, acts similarly. This procedure continues toward the root of the tree, where the final result is presented to the user. Along with the intermediate results all compute servers in the hierarchy also transmit the actual cost P to their predecessor. Each processor recalculates the M factor of its successors based on P and P*.

4. GRID PROCESS ALLOCATION PROTOCOL (GPAP)

The goal of our process allocation protocol is to find the compute servers that can execute the process tree with the least cost. The process allocation protocol is initiated by the compute server E receiving a new process to execute or discovering that a subprocess cannot be executed locally. Assume that the subprocesses are $T_1, T_2, .., T_n$.

We introduce the lists L(E) at a compute server E, a list of compute servers generated by applying the Γ operator at compute server E sorted by their stored values. Let's denote compute server E's stored value by S(E), then $L(E)=\{(E_1, S(E_1)), (E_2, S(E_2)),.., (E_k, S(E_k))\}$, k = # of children of E, and $S(E_1) \leq S(E_2) \leq ..., \leq S(E_k)$. Whenever Γ is applied at a compute server, the successor compute servers are put in L(E) for later expansion. The protocol can be formulated as a recursive algorithm as follows:

```
GPAP (compute_server E, upper_bound B, process Tj)
1.  if  S(E) > B then return S(E);
2.  If Tj can be executed by E then exit;
3.  Apply the Γ operator to generate E's children;
    if E has no children then return "The process cannot be executed";
4.  Let $S(E_i) = h^*(T_j(E,E_i)$ and construct the list
    $L(E)=\{(E_1, S(E_1)), (E_2, S(E_2)),.., (E_k, S(E_k))\}$;
5.  While($S(E_1) \leq B$)
        $S(E_1)$ = GPAP (compute server $E_1$, upper_bound min{B, $S(E_2)$}, process $T_j$);
        Insert $E_1$ and $S(E_1)$ to L(E);
        $S(E) = \min\{S(E_1), S(E_2),.., S(E_k)\}$, $S(E_m) \in L(E)$, m = 1,2,..,k;
        Tentatively assign Tj to E1.
6.  Return $S(E_1)$
```

Similarly to [24] it can be shown that the process allocation protocol always finds the least-cost execution tree, if the following condition $h^*(T_i(E) \leq h(T_i(E)$ is satisfied for all compute servers in the lists L(E).

5. COMPLEXITY ANALYSIS

The algorithmic complexity of our Grid Process Allocation Protocol is defined by the expected number of sites expanded. The Iterative-Deepening-A* (IDA*) [25] performs a sequence of depth-first searches, pruning branches when their cost exceeds a threshold for the current iteration. The initial threshold is determined by the cost estimate at the root and increases for each iteration of the algorithm. Each subsequent threshold for each iteration is the minimum cost of all values that exceeded the previous threshold. IDA* expands the same number of nodes asymptotically, as A* [20]. It is shown in [25] that IDA* is asymptotically optimal in terms of time for tree searches. The important property of A*, that it always finds the lowest-cost solution path if the heuristic is admissible, also holds for IDA*. Although it is easier to implement than A* (as there are no open and close lists to be managed), it uses a global threshold, that is difficult to maintain in a distributed system.

Our protocol is a modification of the Simple Recursive Best-First Search (SRBFS), which is an extensions of the IDA*. Therefore, the complexity of our algorithm can be derived from the complexity of this algorithm. While iterative-deepening uses a global threshold, SRBFS uses a local cost threshold for each iteration with two parameters: a site and an upper bound on cost. It explores the branch below the node as long as it contains expanded nodes, whose costs do not exceed the upper bound. Each iteration returns the minimum cost of the newly expanded nodes. Although the space complexity of our algorithm is O(db) (similarly to SRBFS), where b is the branching factor and d is the maximum search depth, the worst-case time complexity is $O(b^{2d})$ depending on the cost function. With a monotonic cost function, it finds an optimal solution while expanding fewer nodes than iterative-deepening. The method in SRBFS and in our protocol reduces the space complexity of best-first search from exponential to linear (assuming a constant branching factor). The reason is that the recursive procedure only maintains the path to the best frontier nodes of the explored subtree and the siblings of all nodes along the path. While in IDA* each new iteration regenerates the entire previous tree, our algorithm only explores the branches of sibling nodes on one of the last paths of the most recent iteration. The algorithm increases the time complexity by only a constant factor [24].

Simulation Results

We constructed the following simplified model in the Comnet modeling tool:

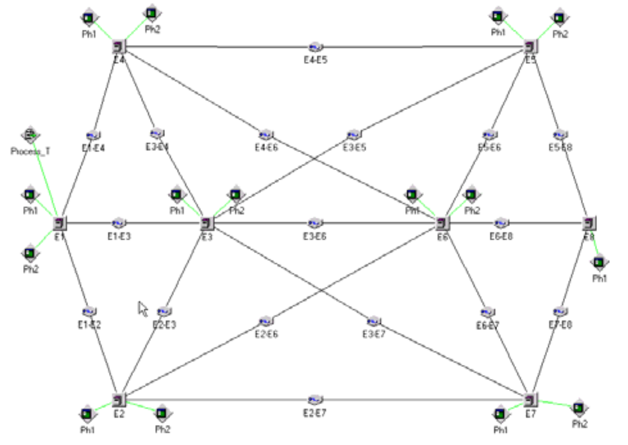
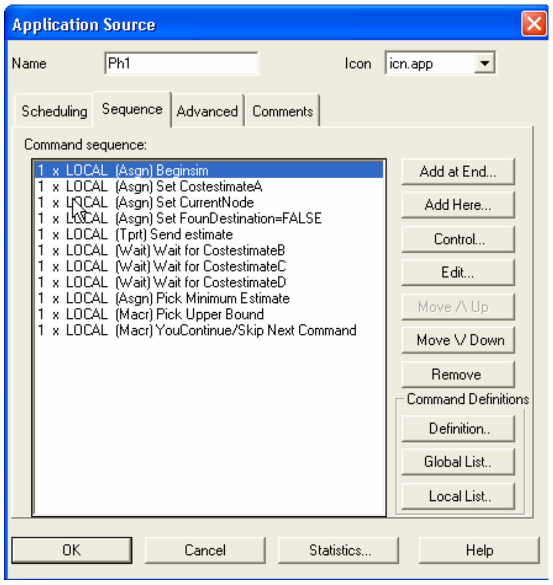


Figure 6. Model of a Grid of Compute Servers

E1-E8 represent the compute server objects connected by high-speed links. The model simulates the assignment of processes

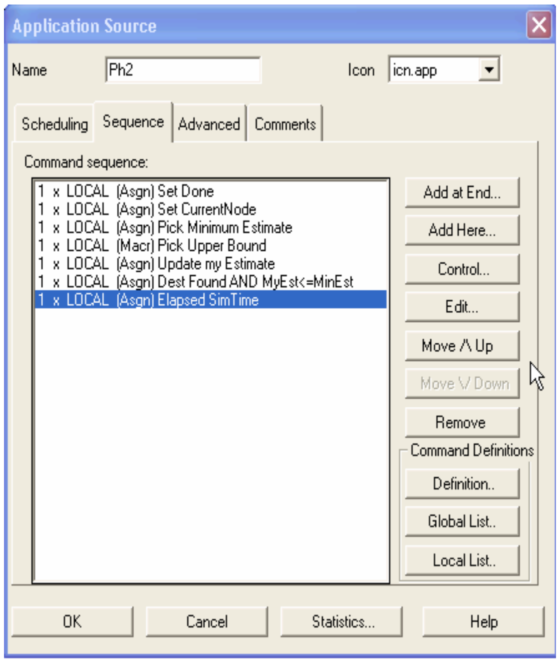
T1, T2, T3, T4, and T5 to compute servers according to the protocol discussed above. Each compute server is connected to two application source objects Ph1 and Ph2 corresponding to the two phases of the process execution. The Ph1 and Ph2 objects are programmed using a simple language as shown below for illustration purposes only:



Name Ph1

Command sequence:

```
1  x  LOCAL  (Asgn) Beginsim
1  x  LOCAL  (Asgn) Set CostestimateA
1  x  LOCAL  (Asgn) Set CurrentNode
1  x  LOCAL  (Asgn) Set FounDestination=FALSE
1  x  LOCAL  (Tprt) Send estimate
1  x  LOCAL  (Wait) Wait for CostestimateB
1  x  LOCAL  (Wait) Wait for CostestimateC
1  x  LOCAL  (Wait) Wait for CostestimateD
1  x  LOCAL  (Asgn) Pick Minimum Estimate
1  x  LOCAL  (Macr) Pick Upper Bound
1  x  LOCAL  (Macr) YouContinue/Skip Next Command
```



Name Ph2

Command sequence:

```
1  x  LOCAL  (Asgn) Set Done
1  x  LOCAL  (Asgn) Set CurrentNode
1  x  LOCAL  (Asgn) Pick Minimum Estimate
1  x  LOCAL  (Macr) Pick Upper Bound
1  x  LOCAL  (Asgn) Update my Estimate
1  x  LOCAL  (Asgn) Dest Found AND MyEst<=MinEst
1  x  LOCAL  (Asgn) Elapsed SimTime
```

In order to measure the performance of the process assignment protocol we collected the following statistic:

- The run time of Ph1 can be considered as a factor representing the protocol's time complexity. The simulation showed that the run time of Ph1 to find the execution paths is very small relative to execution of the process tree Ph2, i.e. the protocol does not add significant overhead to the processing time of the application.
- Another measurement characterizing the protocol's performance is the link utilization. A low link utilization indicates that the protocol doesn't generate excessive traffic on the network. The simulation proved that the protocol requires only insignificant portion of the bandwidth.
- Similarly, the number of messages created during the protocol can be considered as a measure of performance. Low number of messages in the simulation indicates low protocol overhead.

For interested readers the details are available from the author.

6. CONCLUSION

We presented a protocol for resource discovery and process allocation and execution in the Grid of compute servers. An application can specify application level and network level Quality of Service parameters including number of processors, memory, special software, network bandwidth, delay, jitter, packet loss, etc. The protocol assigns processes of Grid applications to compute servers that collectively satisfy the application's resource requirements and minimize the execution time and communication cost We modeled the resource discovery and process assignment as a heuristic search algorithm using a tree structure. The execution of the processes was formulated by the a search for a solution tree. The compute servers calculated the estimated cost of the solution tree as a heuristic function of the search algorithm. After process execution the actual cost of the solution tree could be calculated. Based on statistical measurements of the compute servers' performance in past process executions the protocol could identify the minimal solution tree. The paper also presented the complexity analysis of the algorithm along with a brief discussion of the simulation of the protocol.

APPENDIX

Calculation of the M-factor

Assume that a compute server E sends a request to capable compute servers for executing process T. The compute servers compute the estimated cost P^* based on the activities, procedures, requested Quality of Service and other resources associated with the process execution. Compute server Q cost estimate is calculated as follows: To perform process T compute server Q has to perform the subprocesses T_1, T_2,.., Tn, $n \geq 2$. In the cost estimate Q plans a_j cost for subprocess T_j, where $j = 1,..,n$ Assume that Q's offer is the smallest cost estimate; therefore it is accepted by compute server E. After performing process T the costs of the subtasks T_1, T_2,...,T_n are recalculated. Let these values be: $c_1, c_2,..., c_n$. We can further assume that the

(1) $$c_j, (j = 1,.., n)$$

values are random variables with expectations.

Our goal is to decide whether Q's cost estimate can be accepted with respect to the recalculated cost, or in other words, the hypothesis

(2) $$H_0 : E(c_j) = a_j, (j = 1,.., n)$$

is acceptable or not with respect to the quantities (1).

In order to give a decision procedure for this problem we assume that quantities (1) are normal distributed, independent random variables, i.e,

(3) $$c_j \in N(e_j, \sigma_j^2), (j = 1,.., n)$$

where

$$E(c_j) = e_j, D^2(c_j) = \sigma_j^2, (j = 1,.., n)$$

Based on the central limit theory this assumption is plausible. Namely, each of the random variables (1) can be interpreted as a superposition of independently distributed random variables.

If the hypothesis H_0 fulfills then the random variables

(4) $$Y_j = \frac{(c_j - a_j)}{\sigma_j} \in N(0,1), (j = 1,..., n)$$

form a sample of random variable $Y \in N(0,1)$.

Thus we can use the Student test to accept or to reject the hypothesis H_0. For this reason we use the following notations:

$$\bar{c} = \frac{1}{n}\sum_{j=1}^{n}\frac{cj}{\sigma j} \quad , \quad a = \frac{1}{n}\sum_{j=1}^{n}\frac{aj}{\sigma j}$$

$$(n-1)s_n^2 = \sum_{j=1}^{n}\left(\frac{(c_j - \bar{c})}{\sigma_j}\right)^2$$

From the definition of the random variables (4) we obtain that

$$\bar{Y} = \frac{1}{n}\sum Y_k = \bar{c} - a \in N(0, \frac{1}{n})$$, and

$$(n-1)s_n^2 = \sum_{j=1}^{n}\left(Y_j - \bar{Y}\right)^2 \in \chi_{n-1}^2$$

where χ_{n-1}^2 is the Chi square distribution with degree of freedom $n-1$, and it is well-known that these random variables are independent. Consequently, if the hypothesis H_0 holds, then

$$\bar{Y}\frac{\sqrt{n}}{s_n} \in t_{n-1},$$

where t_{n-1} is the student distribution with degree of freedom $n-1$.

Then, the decision procedure is as follows:

Let $\varepsilon > 0$ be the significance level. Let $h_1(\varepsilon), h_2(\varepsilon)$ be positive values satisfying the equality:

$$.\int_{-h_1(\varepsilon)}^{h_2(\varepsilon)} \mathrm{t}_{\text{n-1}}(x)dx = 1 - \varepsilon$$

where $t_{n-1}(x), x \in R$ is the density function of the Student distribution with degree of freedom $n-1$.

Hypothesis H_0 is accepted on the level ε if

$$-h_1(\varepsilon)\frac{s_n}{\sqrt{n}} \le \bar{Y} \le h_2(\varepsilon)\frac{s_n}{\sqrt{n}}$$

In other words processor Q's cost estimate is acceptable with respect to the recalculation. Otherwise, if

(5) $$\bar{Y} > h_2(\varepsilon)\frac{s_n}{\sqrt{n}}$$

or

(6) $$\bar{Y} < h1(\varepsilon)\frac{s_n}{\sqrt{n}}$$

then Q's preliminary calculation is unacceptable with respect to the recalculation. In the case of (6) we say that the preliminary calculation is strongly underestimated, and in the case (5), it is strongly overestimated on the level.

This method has a disadvantage. Namely, the random variable

$$\bar{Y}\frac{\sqrt{n}}{s_n}$$

depends on the parameters $\sigma_j, (j = 1,.., n)$ and usually these values are unknown. However if

(7) $$\sigma_j = \sigma, (j = 1,.., n)$$

the method is independent of the variances of the random variables $c_j, (j = 1,..., n)$.

The condition (7) is obviously very strong assumption, and can be verified only if we compare the result obtained under condition (7) with the reality.

The difference $a - \bar{c}$ is an appropriate measure of the deviation between the estimated and the recalculated cost. In the case of the acceptance of hypothesis H_0 the

$a - \bar{c}$ value satisfies the inequality:

(8) $$-h_1(\varepsilon)\frac{s_n}{\sqrt{n}} \le (a - \bar{c}) \le h_2(\varepsilon)\frac{s_n}{\sqrt{n}}$$

Let $a_i - c_i$ denote the estimated and recalculated costs in the last ith circuit restoration satisfying inequality (8). Then, for the last j restorations Q's uncertainty related to a process T is calculated by processor E as follows:

(9) $$R(Q) = \sum_{i=1}^{j} | a_i - \bar{c} |, \ a_i \le \bar{c}$$

A compute server's M-factor is recalculated after each process execution by the compute server that sent the request. After each process completion it is determined how realistic a compute server's cost estimate has been with respect to the recalculated cost. If it has been unrealistic, i.e., the hypothesis H_0 above is rejected, no request messages will be sent to this compute server in the future for process execution. If a compute server's cost estimate has been realistic, i.e., the hypothesis H_0 is accepted, its M-factor is recalculated. The closer the estimated cost is to the actual cost, the smaller is the M-factor. The GPAP protocol discussed in this paper can find an optimal solution tree if the cost estimates at each compute server are close lower bounds of the actual costs. It follows from the calculation of the heuristic function P* and the M-factor that it is a higher possibility for a compute server to perform a process, if it can estimate the execution cost close to the actual cost.

References

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of High Performance Computing Applications, 15(3), 2001.

[2] Globus Project. http://www.globus.org.

[3] http://www.gridforum.org.

[4] Gnutella protocol specification, http://www.clip2.com/articles.html.

[5] I. Clarke, O. Sandberg, B. Wiley, and Theodore W. Hong. Freenet: "A Distributed Anonymous Information Storage and Retrieval System," in Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, International Computer Science Institute, 2000.

[6] I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan,"Chord: A Scalable Peertopeer Lookup Service for Internet Applications," SIGCOMM'01, San Diego, California, pp. 149-160, August 2001.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing," in the Proceedings of the IEEE Symposium on High Performance distributed Computing, 2001.

[8] http://www.globus.org/toolkit/docs/4.0/info/key/index.html#id2763255.

[9] C. Plaxton, R. Rajaraman, and W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in ACM Symposium on Parallel Algorithms and Architectures, 1997.

[10] M. Van Steen, P. Homburg, and A. Tanenbaum, "Globe: A Wide-Area Distributed Systems," IEEE Concurrency, pp. 70-78, 1999.

[11] H. Casanova, "Distributed Computing Research Issues in Grid Computing," ACM SIGACT News Distributed Computing Column 8, July, pp. 50-70, 2002.

[12] A. Iamnitchi and I. Foster, "On Fully decentralized Resource Discovery in Grid Environments," In the Proceedings of the International Workshop on Grid Computing, Denver, Colorado, November 2001.

[13] I. Foster and A. Roy, "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation," in Proceedings of the Eight International Workshop on Quality of Service (IWQOS 2000), pp. 181–188, June 2000.

[14] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman. "Grid Service Specification", Argonne National Laboratory, Argonne, IL. Draft3 (7/17/2002).

[15] I. Foster, C. Kesselman, C. Lee, B Lindell, K. Nahrstedt, A. Roy , "A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation", Proceedings of the International Workshop on QoS, pp.27-36, 1999.

[16] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". Downloadable as: http://www.globus.org/research/papers/ogsa.pdf, 2002.

[17] J. Gomoluch and M. Schroeder, "Performance Evaluation of Market-based Resource Allocation for Grid Computing," Concurrency and Computation: Practice and Experience, 00:1–6, 2004.

[18] R. Buyya and S. Vazhkudai, "Compute Power Market: Towards a Market-Oriented Grid," in Proc. of the 1st Int. Conf. on Cluster Computing and the Grid, CCGrid'01. IEEE, 2001.

[19] D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker," Future Generation Computer Systems, (FGCS) Journal, 18(8), pp.1061–1074, October 2002.

[20] N. J. Nilsson, "Principles of Artificial Intelligence, Morgan Kaufmann Publishers, Inc., pp. 99-109, 1980.

[21] Dongyan Xu, Klara Nahrstedt, Duangdao Wichadakul, "QoS-Aware Discovery of Wide-Area Distributed services", Department of Computer Science, University of Illinois at Urbana-Champaign, *f*d-xu,klara,wichadak*g*@cs.uiuc.edu, Technical Report UIUCDCS-R-2000-2189, Nov. 2000

[22] Al-Ali, R.; Rana, O.; Walker, D.; Jha, S. & Sohail, S. "G-QoSM: Grid Service Discovery Using QoS Properties." Computing and Informatics Journal, (4), pp. 363-82.21, 2002.

[23] I. Foster, C. Kesselman, J. Nick, S.Tuecke, "The Physiology of the Grid: AnOpen Grid Services Architecture for Distributed Systems Integration", http://www.globus.org/research/papers/ogsa.pdf, 2002.

[24] R. Korf, "Linear-Space Best-First Search: Summary of Results," Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, California, July 12-16, pp. 533-538, 1992,.

[25] R. Korf, "Iterative-Deepening-A*: An Optimal Admisible Tree Search," Proceedings of the Ninth International Joint Coference on Artificial Intelligence, UCLA, August 18-23, pp. 1034-1036, 1985.