

Quality Aware Service Planning in Computational Grids

Sharath Babu Musunoori
Simula Research Laboratory
1325 Lysaker, Norway
sharath@simula.no

ABSTRACT

As advancements in grid computing continue to support a variety of parallel and distributed systems, the issue of large-scale application scheduling is becoming a key concern. In particular, a class of quality-sensitive applications that requires to fulfil some quality requirements in order to satisfy the user needs. There is a growing need to support these applications as they perform unacceptably if platform resources are scarce or if the deployment is not carefully configured and tuned for the anticipated load. In this paper, we present the notion of application service planning where the application is a composition of service components representing their corresponding functionality, and is configured on the grid environment such that all services involved in the composition get sufficient resources to allow them to deliver at least the minimum required quality to be useful for the application. Addressing such a service planning problem, we summarise a quality deviation model and a learning automaton based solution techniques as a flavour of our research.

Categories and Subject Descriptors

C.2 [Computer-Communication Networks]: Distributed Systems; D.2.8 [Software Engineering]: Metrics—*performance measures*; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling, Sorting and searching*; I.2.6 [Artificial Intelligence]: Learning—*Knowledge acquisition, Parameter learning*

Keywords

Quality of Service; Service Configuration; Grid Computing; Learning Automata; Application Scheduling

1. BACKGROUND AND MOTIVATION

Today's parallel and distributed computing have grown from monolithic super computers to clusters of workstations. With the availability of *grid middleware*, the computing power of

the clusters is supplemented with the capacity to harvest unused *cpu* cycles of other available workstations on the local area network. Thus, the computing power available has grown to eventually reach the boundary of the administrative organization. Internet opens the possibility for tomorrow's applications to run where the resources are best available for its successful completion. Drawing on the advances in grid and web technologies, in particular grid-web services [3], it can be envisioned that a range of distributed applications can be understood as a composition of services. However, achieving acceptable performance in this environment remains a difficult engineering challenge. This is particularly relevant to a broad class of applications that requires the fulfilment of some quality requirements in order to satisfy the user's needs. Such concerns occur not only in real-time and multimedia applications, but also in any application where the user is waiting for outputs. These applications will perform unacceptably if platform resources are scarce or if the deployment is not carefully configured and tuned for the anticipated load. In general, many such applications can provide greater utility by sacrificing quality in one dimension for better quality in another.

In compliance with common component models [4], the term *service composition* refers to a process of composing multiple service components that together deliver the required application functionality. That is any application is equivalent with its service set $S = \{S_1, S_2, \dots, S_{|S|}\}$, where all the services involved in a service composition are taken to be primitive (or atomic) services that can not further be composed of other services. The useful application as experienced by the user is consequently a service graph with corresponding (logical) communication between the services. This distributed application is to run on a set of computing nodes that offer certain physical capacities to the user, e.g. memory, cpu, disk, input/output. A component based *service configuration* in grid computing environment is a mapping of individual service components of a service composition onto the underlying grid resources while satisfying the specified quality requirements of the application. Thus the key problem is the grouping of services into executable units we call *capsules*. There is a one-to-one correspondence between a capsule and a computing node, thus the aggregate resource consumption of the services combined in a capsule can not exceed what is physically offered by the executing node of that capsule.

In order to find a suitable service configuration for the ser-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2nd International Doctoral Symposium on Middleware '05, November 28-December 2, 2005 Grenoble, France Copyright 2005 ACM 1-59593-267-4/05/11... \$5.00

vice composition of the services in \mathbb{S} , several details like application characteristics, size of input and output data, available network bandwidth and other computational resource availability have to be considered. However, any application service composed for a particular platform configuration may be inappropriate for other configurations. In such cases, the application may perform unacceptably even on a standard grid platform with adequate resources. Therefore it is important to separate the functional specification and quality requirements of the application service from the implementation decisions that depend on the deployment environment. Let $|\mathbb{C}|$ be a number of computing nodes of the grid environment, each node C_m has a set of local resources $\mathbb{R}_m = \{R_{m,1}, R_{m,2}, \dots, R_{m,|\mathbb{R}_m|}\}$. Let $\mathbb{Q}_i = \{Q_{i,1}, Q_{i,2}, \dots, Q_{i,|\mathbb{Q}_i|}\}$ be a set of QoS requirements (e.g.: *latency*, *framerate*, and etc.) that determine the quality of a service S_i . To obtain a quality $Q_{i,j}$, the service S_i requires to consume a set of available resources $\mathbb{r}_{C_m, Q_{i,j}} = \{R_{C_m,1}, R_{C_m,2}, \dots, R_{C_m,|\mathbb{r}_{C_m, Q_{i,j}}|}\}$ (e.g.: *cpu*, *mem*, *communication links*) of any grid node C_m . The $\mathbb{r}_{C_m, Q_{i,j}}$ shows an association between resources \mathbb{R}_m of a grid node C_m and a quality requirement $Q_{i,j}$ of a service S_i (i.e., $\mathbb{r}_{C_m, Q_{i,j}} \subseteq \mathbb{R}_m$).

Our research problem includes the application service composition, configuration and reconfiguration in the grid computational environment. We refer to this as *service planning* problem. More concrete description of the service planning problem is presented in the immediately following section. Our research approach including different solution techniques, is described in Section 3.

2. THE PROBLEM DESCRIPTION

Although the use of component technology simplifies the application design by allowing the application developer to compose different functional service components implementing their respective service types, mapping these services on to appropriate grid resources (nodes) while satisfying the specified QoS requirements of overall application is a complex task. In particular when multiple quality objectives of the application service depend on each other to determine the overall application performance. Despite the fact that various computational, communication, and other (sensors, cameras) resources contribute to deliver the required levels of application quality, the scheduler is required to understand the application service context, service user requirements and availability of dynamically changing resources. By accounting all these QoS-aware service configuration requirements, the first fundamental problem we address is that of partitioning: *How to split the application's service set into the number of capsules sub-sets such that all services get sufficient resources to allow them to achieve at least the minimum quality to be useful for the application?* That is, the objective is to partition the service set \mathbb{S} on to $|\mathbb{C}|$ grid nodes.

Let $\Pi(t) = \{S_1, S_2, \dots, S_{|\mathbb{C}|}\}$ be a partition at any instance of time t such that $\bigcup_{i=1}^{|\mathbb{C}|} S_i = \mathbb{S}$ and $S_i \cap S_j = \emptyset$ for all pairs $i \neq j$. Given this partition $\Pi(t)$, any service S_i will find a set of required resources, $\mathbb{r}_{C_m, Q_{i,j}}$ of its host C_m for all quality dimensions j . In other words, $\mathbb{r}_{C_m, S_i} = \{\mathbb{r}_{C_m, Q_{i,1}}, \mathbb{r}_{C_m, Q_{i,2}}, \dots, \mathbb{r}_{C_m, Q_{i,|\mathbb{Q}_i|}}\}$ is set of resources required for a service S_i . Similarly the set $\mathbb{r}_{C_m} = \{\mathbb{r}_{C_m, S_1}, \mathbb{r}_{C_m, S_2}, \dots,$

$\mathbb{r}_{C_m, S_{|\mathbb{S}|}}\}$ be a set of all possible resource capabilities of the node C_m for all services in \mathbb{S} respectively. By combining the predetermined resource capabilities of each grid node C_m , the overall computational capability of the grid environment is captured as:

$$\mathbb{r} = \mathbb{r}(\Pi(t)) = \{\mathbb{r}_{C_1}, \mathbb{r}_{C_2}, \dots, \mathbb{r}_{C_{|\mathbb{C}|}}\} \quad (1)$$

In order to make service configuration decisions for the above defined computational capability of the grid environment, the performance or utility of each service S_i is evaluated as a function of quality:

$$Q_{S_i}(\mathbb{r}(\Pi(t))) = Q_{S_i}(\mathbb{r}_{C_m, S_i}) = \sum_{j=1}^{|\mathbb{Q}_i|} w_{Q_{i,j}} * Q_{S_i, j}(\mathbb{r}_{C_m, Q_{i,j}}) \quad (2)$$

where $\sum_{j=1}^{|\mathbb{Q}_i|} w_{Q_{i,j}} = 1$; and $Q_{S_i, j}(\mathbb{r}_{C_m, Q_{i,j}})$ is a quality function that estimates the quality $Q_{i,j}$ of the service S_i from the available resource set $\mathbb{r}_{C_m, Q_{i,j}}$. Further the quality function of the service S_i in each individual quality dimension $Q_{i,j}$, is defined as:

$$Q_{S_i, j}(\mathbb{r}(\Pi(t))) = Q_{S_i, j}(\mathbb{r}_{C_m, Q_{i,j}}) = \min_{k=1}^{|\mathbb{r}_{C_m, Q_{i,j}}|} q_{S_i, j}(R_{C_m, k}) \quad (3)$$

Where $q_{S_i, j}(R_{C_m, k})$ is a service quality-resource function which estimates service quality $Q_{i,j}$ with respect to a resource $R_{C_m, k}$. The optimal QoS-aware service configuration is the one that simultaneously tries to maximize the quality objective of all services from the given set of available computational and communication resources. That is the quality objective of any service S_i is to:

$$\text{Maximize } Q_{S_i} \quad (4)$$

The optimal service configuration Θ is a partition of the service set \mathbb{S} , $\Theta = \{\Theta_1, \Theta_2, \dots, \Theta_{|\mathbb{C}|}\}$ such that $Q_{S_i, j}^{min} \leq Q_{S_i, j} \leq Q_{S_i, j}^{max}$ for all i and j , $\Theta_i \subseteq \mathbb{S}$, and $\bigcup_{i=1}^{|\mathbb{C}|} \Theta_i = \mathbb{S}$ and $\Theta_i \cap \Theta_j = \emptyset$ for all pairs $i \neq j$. Here $Q_{S_i, j}^{min}$ and $Q_{S_i, j}^{max}$ are minimum and maximum required quality values respectively for the service S_i . The combinatorial explosion ($|\mathbb{C}|^{|\mathbb{S}|}$ in worst case) makes the above service configuration a complex task, and this process of application service configuration has been proven to be NP-hard [6]. In the context of service configuration problem, the objective is to seek a feasible solution instead of an optimal solution. This is because, the computational and communication resources in the grid environment may join and leave very often. A feasible service configuration is the one that tries to achieve at least minimum required quality levels for all services from the given set of resources. That is, any service S_i involved in the feasible service configuration should satisfy the following the condition:

$$Q_{S_i, j} \geq Q_{S_i, j}^{min} \quad \forall Q_{i,j} \in \mathbb{Q}_i \quad (5)$$

However, it is important to note that every feasible service configuration may not guarantee the overall performance of

the application. This is because of the differences between the quality levels of services in the service set, \mathbb{S} . For example in distributed real-time applications like object tracking service [2], a small difference between individual services with acceptable quality may degrade the overall performance of the application. In other words all services in \mathbb{S} must be synchronized. That is:

$$\text{Minimize } |Q_{S_i} - Q_{S_j}| \quad \forall i, j \in [1, |\mathbb{S}|] \quad (6)$$

These service configuration decisions are made based on the characteristics of different resources and their availability at deployment time. However in grid environment, computational and communicational resources may join and leave very often. Therefore the configured application service must not depend on specific resources, but should adapt efficiently to dynamic changes in resource availability. The next subsequent problem we address in this research is service reconfiguration: *How to re-arrange or even adapt the application's service set dynamically when the underlying resources of grid environment change with time, such that all services get sufficient resources to deliver at least the minimum quality to be useful for the application?*

3. OUR APPROACH

Within any service composition, each service S_i is an independent functional component blueprint. The service S_i allows one or more inputs, performs its functional task, and returns one or more casually related outputs. Besides this, each service running in a grid environment consumes a variable amount of resources to produce its specified functionality and associated quality levels. However, the amount of resources used and the associated quality levels delivered by the service, are dependent on the context of the service use. Generally it is important to note that there may exist a large set of functions to establish such a relation between resource usage and delivered qualities in different application service contexts. Among them, exponential functions (e.g., sigmoid function [10]) are widely used to relate the input and output of functional units in dynamic systems (e.g., artificial neural networks). To characterize similar relations in our application service configuration setting, we adapt a service quality-resource function of the following form:

$$q_{S_i,j}(R_{C_m,k}) = 1 - e^{-\text{qualityFactor} * R_{C_m,k}} \quad (7)$$

From the above quality function, the slope of the resource-quality curve is determined by the value of a *qualityFactor*. The *qualityFactor* is a variable that changes with application context, since the services in different application contexts change their service behavior.

3.1 Quality Deviation Model

Having defined the quality objectives of each individual service in the service set \mathbb{S} , the overall quality of the application service composition is estimated as a function of application quality:

$$Q_{\mathbb{S}} = Q_{\mathbb{S}}(r(\Pi(t))) = \sum_{i=1}^{|\mathbb{S}|} w_{S_i} * Q_{S_i}(r(\Pi(t))) \quad (8)$$

where $\sum_{i=1}^{|\mathbb{S}|} w_{S_i} = 1$. From this, the overall objective of the composed application is to maximize the combined quality level of all services, and is presented as:

$$\text{Maximize } Q_{\mathbb{S}}(r(\Pi(t))) \quad (9)$$

In the context of real-time and multimedia application service configuration, the above general weighted sum solution technique (equation 8) derives a set of feasible solutions by finding a convex sum of service quality objectives. However, often it is the case that the generated feasible points in the solution space are pulled over a single or multiple quality objectives while other remaining objectives receive less attention [1]. For example, two objectives of an object tracking application: *maximize framerate* and *minimize latency* depend on each other and provide a trade-off. A simple case like lower latency but lower framerate, may converge into a feasible solution because of higher utility in latency, but this may degrade overall performance of the application. As a result, generated solution points are located into different regions of the objectives. Therefore selection of the right trade-off points in the solution space is not guaranteed.

We argue that the above process of finding trade-off points in the objective space can be improved by setting aspiration levels within the objective space. The term *aspiration* is defined as maximum useful quality a service can achieve in a given application context. This allows the service user to set more realistic quality targets for real-world applications. For example, an object tracking service to track a person walking in a street or in an office environment would not need to process more than 30 frames per second to produce 100 % accurate results. Therefore aspired value for framerate in this service context can be set to 30 fps. We also define the term *actual* as a quality value achieved by the service. The *error* is measured as a distance between the aspired and actual quality levels. A full length discussion of the quality deviation model which proposes to minimize the distance between the achieved and aspired service quality levels, can be found in [7]. By using this, the error $e_{i,j}$ in any quality dimension $Q_{i,j}$ of a service S_i is measured as a distance between aspired and actual quality levels:

$$e_{i,j} = a_{Q_{i,j}} - q_{S_i,j}(R_{C_m,k}) \quad (10)$$

Here $a_{Q_{i,j}}$ is the aspired quality ($Q_{i,j}$) value of the service S_i in a specific application context. The above small change in the quality aspiration levels makes a significant change in service quality estimation while finding a nice compromise between the quality objectives. That is, minimizing the error (instead of maximizing quality itself) gives the following two benefits: first, it provides fine grained trade-off between the quality objectives. Second, by targeting specific levels of a quality, only a required amount of grid resources are considered to be allocated for each service. This allows to

find more valid solution points within the objective space.

The quality deviation model has been verified in the context of a real-time multimedia object tracking application service [7]. From the results, it is observed that when both the computational and communication resources of the grid environment are busy, service quality points tend to come closer to their aspiration points of intersection in the quality space. That is beginning from individual resources of computing nodes of the grid environment to the services involved in the composition, a fine-grained trading between multiple quality dimensions help to derive feasible solutions. However, it also has limitations of finding service configurations with increase in number of quality objectives, services and resources. In particular, the effectiveness of the quality deviation technique used in our service planning process is limited to applications with very few quality objectives [1], since one objective must be determined before the rest of the objectives. This makes it difficult to guarantee a feasible solution.

3.2 Learning Automaton Solution

In order to satisfy the quality objectives of the composed application service and to maintain these quality levels by adapting the dynamically changing grid environment with minimum changes, we focused on finding solution techniques that provide good enough solution, but not necessarily optimal. In this respect, some existing solutions such as integer programming, and branch-and-bound have limitations for solving NP-completeness of combinatorial explosion which occurs in the problem of service planning. By noting the most important limitations of scalability and multiple quality objectives, we adapted more advanced heuristic, learning based solution techniques and recently we proposed to use a *learning automata* based solution to find a useful partition. A learning automata basically proposes an *action* to a stochastic environment that returns a binary feedback (reward or penalty). Through several such iterations the automaton will *learn* which action is most likely to give a positive feedback. The use of learning automata for our problem is motivated by the fact that the fastest known equipartitioning algorithm is based on a fixed structure learning automata [9]. The theory of learning automata have also been successfully used to map the processes of a parallel application onto processing nodes [5], and this problem resembles the one at hand.

The functionality of the Learning Automata (LA) can be described in terms of a sequence of repetitive feedback cycles in which the automaton interacts with the environment. During a cycle, the automaton chooses an action, which triggers a response from the environment, a response that can be either a reward or a penalty. The automaton uses this response and the knowledge acquired in the past actions to determine which is the next action. By learning to choose the optimal action, the automaton adapts itself to the environment. The goal of a learning automaton is to determine the optimal action out of a set of allowable actions, where the optimal action is defined as the action that maximizes the probability of being rewarded.

The beauty of incorporating LA in any particular application domain, is indeed, the elegance of the technology. Es-

entially LA are utilized exactly as one would expect: by interacting with the "environment", i.e. the system from which the LA learn. The details of how this is achieved in the various application domains involve modelling the "actions", transforming the systems's outputs so that they are perceived to be of a reward or penalty flavour. Within the domain of service planning, automaton are represented by services and the environment by a set of capsules. The action is equivalent to choosing a capsule of the grid environment. The environment's feedback is produced based on the output from quality objective functions defined in the section 2.

Using this approach, individual services in the service set interact with the grid environment to make service configuration decisions. Such a service configuration is done in an autonomous fashion to improve the scalability of distributed service planning. The proposed learning automaton based solution [8] establishes an autonomous learning environment for the services. That is, as long as the current mapping or partition is not feasible, the unsatisfied services of the current partition move between the capsules. Here it is important to note that the term *move* does not mean: moving the instance of a service involving communication, but moving a model of the service in computing. During this process all services learn and even unlearn from behavior of the grid computational environment. Such a learning process of each service is done by making changes in its confidence level. That is, any positive feedback (reward) from the grid environment allows a service S_i to increase its confidence, similarly a penalty forces the service to decrease its confidence. To make this happen, the learning algorithm follows a specific strategy. When tested for various sizes of the service set and the grid environment, it is observed that the learning algorithm always finds a feasible solution. More detailed description of the proposed learning automaton based solution can be found else where in [8]. That is, in the best case of a grid computational environment with excess resources, the probability of any service to get rewarded is high and as a result all services eventually gain their maximum quality. In worst case of a grid computational environment with scarce resources, the probability of any service to get penalized is high and as a result all services eventually get satisfied with at least minimum specified quality.

As grid middleware advancing, resource management systems are now able to allocate and manage grid resources on per component basis. These improvements are very important for configuration and reconfiguration of complex real-time systems with large number of software components. By noting these advancements in grid computing, resource managers running on grid nodes monitor changes in the underlying computational and communication resource availability and notify the respective services. Using this information received from the resource managers, services move to other grid nodes with sufficient resources or even adapt the variations in resource availability such that the overall application quality is maintained.

4. CONCLUSION

Giving more emphasis on simplifying application design by allowing to compose different functional service components, the issue of configuring these application services onto the

grid platform has been introduced as a service planning problem. The described service planning problem is proposed for a broad class of applications that requires the fulfillment of some quality requirements in order to satisfy user needs.

Besides the above described limitations on multiple quality dimensions and of centralized service planning, the quality deviation model improves the capability of the resultant feasible solution set. Addressing scalability issues, a fixed structure learning automaton solution has been adopted and successfully applied in the domain of application scheduling and application quality management. In addition to the initial service configuration at deployment time, our ongoing and future work will also investigate more advanced learning strategies to configure and reconfigure the composed application service during runtime.

5. ACKNOWLEDGEMENTS

The author would like to thank Frank Eliassen and Geir Horn for their comments and suggestions during the design and implementation of service planning algorithms.

6. REFERENCES

- [1] I. Das. Applicability of existing continuous methods in determining the pareto set for nonlinear, mixed-integer multicriteria optimization problems. In *Proc. of 8th AIAA Symposium on Multidisciplinary Analysis and Optimization*, CA, September 2000.
- [2] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis. In *Proceedings of the 11th ACM International Conference on Multimedia, ACM MM'03, Berkeley, California, USA*, pages 21–32, November 2003.
- [3] D. Gannon and et al. Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications. *Journal of Cluster Computing, Special Issue on Grid Computing*, July 2002.
- [4] M. Govindaraju and et al. Merging the CCA Component Model with the OGSF Framework. In *Proceedings of CCGrid2003, 3rd International Symposium on Cluster Computing and the Grid*, May 2003.
- [5] G. Horn and B. J. Oommen. A fixed-structure learning automaton solution to the stochastic static mapping problem. In H. J. Siegel, D. A. Bader, and J.-L. Gaudiot, editors, *Proceedings 19th IEEE International Parallel and Distributed Processing Symposium*, pages 297b – 297b, Denver, Colorado, April 2005. IEEE.
- [6] J. Liang and K. Nahrstedt. Service Composition for Advanced Multimedia Applications. In *In Proc. of twelfth Annual Multimedia Computing and Networking (MMCN'05), San Jose, CA*, January 2005.
- [7] S. B. Musunoori and F. Eliassen. Qos-aware application service configuration in a grid environment. *To appear in the 9th IASTED International Conference on Software Engineering and Applications (SEA2005), Phoenix, AZ, USA*, November 2005.
- [8] S. B. Musunoori and G. Horn. A fixed-structure learning automaton solution to the quality aware application service configuration in a grid environment. *To appear in the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005), Phoenix, AZ, USA*, November 2005.
- [9] B. J. Oommen and D. C. Y. Ma. Deterministic learning automata solutions to the equipartitioning problem. *IEEE Transactions on Computers*, 37(1):2–13, Jan. 1988.
- [10] X. Yin, J. Goudriaan, E. A. Lantinga, J. Vos, and H. J. Spiertz. A flexible sigmoid function of determinate growth. *Annals of Botany* 91, 361-371, 2003.