Decentralized Self-Adaptation Mechanism for Service Based Applications in Cloud using Spectral Clustering

E. NISHA

SCAD Engineering College, Cheranmahadevi, India.

(Received: March 20, 2014; Accepted: March 30, 2014)

ABSTRACT

Cloud computing, with its promise of (almost) unlimited computation, storage and bandwidth, is increasingly becoming the infrastructure of choice for many organizations. As cloud offerings mature, service-based applications need to dynamically recompose themselves, to self-adapt to changing QoS requirements. In this paper, we present a decentralized mechanism for such self-adaptation, using natural language processing. We use a continuous double-auction to allow applications to decide which services to choose, amongst the many on offer. The proposed scheme exploits concepts derived from graph partitioning, and groups together tasks so as to 1) minimize the time overlapping of the tasks assigned to a given resource and 2) maximize the time overlapping among tasks assigned to different resources. The partitioning is performed using a spectral clustering methodology through normalized cuts. Experimental results show that the proposed algorithm outperforms other scheduling algorithms for different values of the granularity and the load of the task requests.

Key words: Self-Adaptation, Market-Based, Multi-Agent Systems, Spectral Clustering.

INTRODUCTION

Self-adaptation, as a concept, has been around for many years, in several domains like biology, chemistry, logistics, economics etc. Self-adaptivity in computer-based systems is relatively newer. Some of the first references to self-adaptive software systems are from\textsuperscript{1,35,34,31} (where they are referred to, as autonomic systems). By self-adaptivity in software systems, we mean software that monitors itself and the operating environment, and takes appropriate actions when circumstances change. In web-applications, service-oriented architecture has often been used as a mechanism for achieving self-adaptivity\textsuperscript{19}. Web-services allow for dynamic composition, which enables applications to switch services, without going offline. A common instance of using web-services dynamically, is applications living on the cloud, asking for computing power and bandwidth to be scaled up or down, depending on demand. However, one of the cloud's major selling points, operational flexibility is of little use, if applications (or organizations) have to indicate at sign-up time, the kind of services that they intend to use. On Amazon, for instance, a customer
specifies during sign up whether she wants a Hi-CPU instance or a Standard On-Demand instance or a Hi-Memory instance. This assumes that an application is able to forecast its demand for computing and storage resources accurately.

**Our approach**

We would like to create a mechanism that allows multiple applications, constructed across a federation of clouds, to self-adapt. We chose a market-based approach to self-adaptation, not only because it is decentralized, but also due to its easy applicability to the problem domain. Services in the cloud are moving from a fixed-price package to a more flexible, auction-based approach. This enables a self-adaptive application to change the QoS exhibited, by switching to a different Concrete Service.

**Market-Based Control**

Market-Based Control (MBC) essentially involves modeling the system as a marketplace, where self-interested agents use economic strategies to compete for resources. Self-interested competition, along with well-designed utility functions, allow for a decentralized means of decision making. These agents, via their competitive need to get resources, perform a parallel search through the space of decision points. MBC has been used in several contexts, as a mechanism for computing a good solution in a decentralized manner.

**Auctions**

Auctions, specifically Double Auctions (DA), have increasingly been studied in Computer Science, as a mechanism of resource allocation. Daniel Friedman reports on experiments where traders even with imperfect information, consistently achieve highly efficient allocations and prices. The rise of electronic commerce naturally creates a space for efficient exchange of goods, and services. There has been much work on the design space of market-institutions, bidding strategies, agent-based implementations of traders, etc. Gupta *et al.* argue that network management, specifically for QoS issues, must be done using pricing and market dynamics. According to them, the flexibility offered by pricing mechanisms offers benefits of decentralization of control, dynamic load management and effective allocation of priority to different QoS attributes. The continuous-time variant of a DA, called Continuous Double Auction (CDA), is used in stock-markets and commodity exchanges around the world. In a CDA, the market clears continuously.

**Spectral Clustering Algorithm**

We refer to our proposed task allocation scheme as the Spectral Clustering Scheduling (abbreviated SCS) algorithm. An advantage of the SCS algorithm is that it can be used in two different ways. The first is to efficiently schedule tasks on a given number of resources so as to minimize the task time requirement violations. The second is to find the number of resources or virtual machines required to serve the tasks with a given level of task time violations (or with no violations). The latter approach is particularly helpful in a cloud computing environment, where a resource provider should be able to estimate the service-level agreement (SLA) offered to its users for a given number of virtual machines (resources) used.

**General Scheduling Algorithms**

When the tasks' QoS requirements are given in the form of task deadlines, the most well-known scheduling algorithm is the Earliest Deadline First (EDF). Another approach is the Slack Time algorithm, also referred to as Least Laxity First (LLF), where the tasks are selected for execution in order of nondecreasing slack time, defined as the difference between the task's relative deadline and its remaining computational time. A framework for providing hard deadline guarantees in a grid environment, by controlling the rate and burstiness with which users submit jobs to resources, is proposed in. In the case where the infrastructure consist of a cluster of servers, several schedulers have been developed, including Maui, the portable batch system (PBS), and the load sharing facility (LSF). Most of the scheduling algorithms proposed so far for grids provide a best effort (nonguaranteed) service to the submitted tasks and try to optimize a single or multiple metric(s) of interest (for example, task delay, probability to miss a deadline, etc.). Scheduling algorithms also exist that take into account the users' QoS requirements (delay, required number
of CPUs, etc.) and try to serve them in the best possible way\textsuperscript{15}. Fairness in grid scheduling has been considered in\textsuperscript{16-17}, where the proposed algorithms allow for the “fair” sharing of deadlines’ violations among all users. Genetic algorithms have also been widely used for solving NP-complete scheduling problems; see, for example, the works of\textsuperscript{18-21}. Reservation

**Interval Scheduling**

Our work relates to a particular flavor of the scheduling problem, which is called the interval scheduling problem, also known as fixed job scheduling\textsuperscript{33}. In interval scheduling, a list of tasks is given as a set of requested time intervals and the scheduler decides if it can accept or not a task, and, in the second case, assign it to some resource. There are several variants of the problem: One variant assumes an unrestricted number of resources and the objective is to find a minimum cost schedule that does not reject any tasks\textsuperscript{34}. Another variant assumes a fixed number of resources and a predefined profit associated with the successful execution of each task. The objective is to find a maximum-profit schedule in which some, but not necessarily all, of the tasks are assigned to the resources\textsuperscript{3}.

The third case refers to a variant of the aforementioned problems where each task has a set of possible starting times, instead of a single starting time\textsuperscript{35}. Finally, in the online interval scheduling problem tasks are generated one by one and the scheduler has to make a decision for each task before a new task arrives. The objective is to maximize the total length of the intervals assigned to tasks, while ensuring that no pair of tasks assigned to the same resource overlap in time\textsuperscript{36}. In\textsuperscript{32}, the interested reader can find a survey of the interval scheduling literature. In the current paper, we focus on the second type of interval scheduling algorithms. This variant of the problem can be modeled using a min-cost flow formulation. In the case that all jobs have unit weight, greedy algorithms are proposed in\textsuperscript{38-39} for estimating the maximum number of jobs that can be served. Heuristic and exact algorithms have been proposed in\textsuperscript{40} for the case where each job can be executed only on a given subset of machines. The interval scheduling problem can also be formulated using interval graphs. A node in the interval graph corresponds to a task and there is an edge between nodes whose task intervals overlap. Hence, the basic interval scheduling problem is in fact a coloring problem of the interval graph (see [41]). Interval graph scheduling can also be formulated as a graph partitioning problem, where the interval graph is partitioned so as to satisfy a multiobjective criterion. Graph partitioning is a kind of task clustering. Task clustering has been used in the literature for task scheduling in grids. Examples can be found in, where two task routing policies (one static and one adaptive) and six task scheduling policies are examined. Methods with similar goals include workflow allocation, mixed local, and global scheduling.

**Evaluation**

**The Current Scenario**

The current form of service-selection is provider-driven. That is, in all commercial clouds, the cloud provider uses a posted-offer mechanism. A posted-offer is a form of market where the supplier posts a certain price on a take-it-or-leave-it basis. Thus, on Amazon’s elastic cloud compute (EC2), there are several services that are functionally identical, but priced differently. This price differentiation exists due to different QoS being exhibited by these services.

In our case, graph partitioning is performed based on spectral clustering through the use of normalized cuts. The normalized cuts method has advantages over the traditional min-cut methods, as it does not favor the creation of small clusters. To handle this issue, traditional graph partitioning methods, like geometric and multilevel graph partitioning, impose balancing criteria. However, predefining the partition size may be in conflict with the actual data statistics (in our case, the requested start and finish times), where sometimes unbalanced partitions are more appropriate. The use of normalized cuts resolves this issue, by automatically estimating the partition size with respect to the diversity of the tasks’ duration within each partition.

**Qualitative Criteria**

Thus, clobmas must fulfill the following criteria:
1) Allows customers like BizInt to create adaptive applications
2) Generates a higher utilization of services than the posted-offer model currently followed (for Sky-Compute)

Quantitative Criteria
Since, SkyCompute is an ultra-large collection of services, clobmas must be able to scale to large numbers of applications and ConcreteServices. Since there is no public data about the kinds of workflows hosted on commercial clouds, and their corresponding service choices, we made assumptions about the variables involved in dynamic service composition. We make these assumptions based on conversations with performance consultants at Capacitas Inc., and numbers gleaned from the literature review.

Related work
Self-Adaptation
Applications that use dynamic service composition should be able to continuously monitor their current QoS levels and make adjustments when either the demand for QoS changes or the cost constraint changes. The application should thus be able to respond to both internal as well as external stimuli, to trigger a change in its constituent web-services. This change needs to be both timely, as well as correct, i.e., the new set of services should not violate any of the application’s QoS constraints, and the change should happen as fast as possible.

Self-Adaptation in software systems is the achievement of a stable, desirable configuration, in the presence of varying stimuli. These stimuli may be environmental (in the form of workload, failure of external components, etc.) or internal (failure of internal components, changed target states, etc.).

Spectral Clustering
Computational Complexity
The SCS scheduling algorithm consists of two main steps: 1) the eigenvalue decomposition optimization algorithm and 2) the clustering algorithm (k-means in our case). The fastest implementation for the eigenvalue decomposition problem is though the Lanczos method whose complexity is \( O(MN^2) \), where \( M \) is the number of eigenvalues that have to be computed (equal to the number of processors), \( N \) is the number of tasks, and the number of iterations of the algorithm. In our case, takes small values (around 20-40), compared to \( M \) and particularly \( N \). However, for low granularity values, \( M \) is several times smaller than \( N \), making the complexity to be of order \( N^2 \) in that case in practice. The clustering step of the SCS algorithm is implemented using the k-means method, which has complexity \( O(N^2) \) in our case. We present experiments regarding the computational complexity performance of the SCS algorithm, which is compared to that of other scheduling methods. The experiments have been carried out for three examined traces at different granularities values on an Intel Core2 Duo 3.00 GHz. It is clear that as the granularity increases the execution time of the scheduler decreases, since fewer tasks are encountered. Although SCS presents the worst execution time performance, the actual runtime is not so critical even for low granularities values.

CONCLUSION

Future work
Cloud-based service-oriented applications have the potential to self-adapt their QoS, depending on demand. Using a market-based mechanism maps nicely to the real-world situation
of unpredictable change of QoS requirements, costs involved in adaptation and adaptation by competing applications. As the number of possible concrete services increase, the scalability of the self-adaptive mechanism becomes important. We see that the market-based mechanism consists of simple agents, is able to adapt well and yet scales linearly to the number of concrete services. We also see that it is robust in the presence of differences in demand and supply of QoS. Applications implemented as an ASN can thus scale and adapt to the changing business requirements of QoS. We have not modelled complex seller-side behaviour. Specifically, actions like deliberate violation of QoS to free up resources for making Asks with higher prices or mis-reporting of QoS available. Mechanisms like penalties and reputation management can be used to prevent seller agents from behaving dishonestly. The proposed spectral clustering scheduling scheme aims at maximizing processor utilization efficiency, while simultaneously minimizing the tasks’ QoS degradation. Task QoS is specified through the requested start and finish times, while QoS degradation is expressed through a time deviation metric D. Resource assignment is viewed as a normalized cuts spectral graph partitioning problem. The algorithm defines a nonoverlapping measure between tasks, and then uses a matrix representation, the notion of generalized eigenvalues, and the Ky-Fan theorem to perform scheduling (graph partitioning) in the relaxed continuous space. The solution of the continuous relaxation is then rounded to a discrete solution.

REFERENCES

1. Do slas really matter? 1-year case study.


35. Robert Laddaga. Creating robust software


38. Vivek Nallur and Rami Bahsoon. Design of a Market-Based Mechanism for Quality Attribute Tradeoff of Services in the Cloud.

