Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

2013 International Conference on Computational Science

Dynamic QoS Optimization Architecture for Cloud-based DDDAS

Tao Chen[a*], Rami Bahsoon[a], Georgios Theodoropoulos[b]

[a]School of Computer Science, University of Birmingham, B15 2TT, United Kingdom
[b]IBM, Dublin Research Lab, Ireland

Abstract

An emerging class of Dynamic Data Driven application systems heavily depends on cloud and Big Data. We refer to this class of DDDAS as cloud-based DDDAS. Despite the growing interest in marrying DDDAS with the cloud, there is a general lack for architectural frameworks explicating the cloud requirements, which can support cloud-based DDDAS. Given the unpredictable, dynamic and on-demand nature of the cloud, cloud-based DDDAS requires novel approaches for dynamic Quality of Service (QoS) optimization. This is important for providing timely and reliable predictions and for ensuring higher dependability in the solution, as it would be unrealistic to assume that optimal QoS can be achieved at design time. We propose a decentralized architectural style for cloud-based DDDAS, where dynamic QoS optimization is in the heart of the symbiotic adaptation. The architecture leverages on the classical DDDAS primitives to reach a refined decentralized style suited for the dynamic requirements of the cloud. We formulate the QoS optimization problem as a dynamic multi-objective optimization problem. We use a scenario to exemplify and evaluate the effectiveness of the style.

Keywords: Cloud Computing; QoS Optimization; QoS Sensitivity; Multi-objective Optimization; Distributed Simulation

1. Introduction

The Dynamic Data-Driven Applications Systems (DDDAS) [2] paradigm has been widely applied in specific areas like manufacturing process controls, resource management, weather and climate prediction, traffic management, systems engineering, civil engineering, geological exploration, social and behavioral modeling, cognitive measurement, and bio-sensing. These projects are essentially data assimilation/data fusion applications, where the focus is on how data can steer the execution, measurement, simulation, and feedback control in real time for typical real life data-intensive systems. The key concept of DDDAS paradigm is to combine a simulation system and a physical system synergistically in a closed feedback loop. In doing so, the physical system can be influenced or controlled by the simulation system whereas the simulation system can consolidate itself by monitoring the state of physical system. Consequently, such info-symbiotic and bidirectional model is a promising solution for improving the predictive capabilities.

* Corresponding author.
E-mail address: txc919@cs.bham.ac.uk (Tao Chen), r.bahsoon@cs.bham.ac.uk (Rami Bahsoon)

Cloud computing enables dynamic scalability and on demand provision of software and hardware resources. With the emergent of the cloud, DDDAS systems have been heavily benefiting from cloud, its computational infrastructure and Big Data [12]. We refer to this class of DDDAS as cloud-based DDDAS. Current DDDAS contributions have focused on the applications and have not explicitly addressed the cloud software engineering specifics governing the development and evolution of this class of applications. Despite the growing interest in DDDAS and the move to the cloud, there is a general lack for architectural frameworks explicating the cloud and can support the runtime adaptation and evolution for dependable cloud-based DDDAS. In the cloud model, it would be possible to deploy cloud-based DDDAS as Software-as-a-Service. The cloud model can facilitate variants of the service, each coming with different Quality of Service (QoS) requirements. Examples of these QoS may include response time, throughput, availability, security, and so forth. As the cloud environment tends to be dynamic and unpredictable, QoS fluctuations of cloud-based DDDAS affecting its reliability and effectiveness. The promises of maintaining QoS are reflected in Service Level Agreement (SLA). A SLA is a binding contract between the end user and application. These promises can be enforced by using the special on-demand feature of cloud. However, cloud-based DDDAS may suffer from cases of over-provision (SLA is satisfied, but provisions are left as idle leading to unnecessary costs) and under-provision (provisions are insufficient for guarantee the agreed QoS in SLA). These could result in operational, legal, and financial hazards. The dynamic nature of cloud and the consequences of over-/under-provision lead to the requirements of self-adaptive QoS optimization solutions. Because the cloud tends to be highly dynamic and elastic in nature, adaptively optimizing QoS for DDDAS-cloud based application is a complex and challenging problem. We argue that the classical centralized DDDAS architectural style is limited in scale and can not cope with dynamic changes in QoS in the cloud. We call for novel decentralized architectural style, which have the dynamic QoS optimization central to adaptation

The novel contribution of this paper is a decentralized architectural style for cloud-based DDDAS service. The style is refined in a way to facilitate the dynamic and self-adaptive QoS optimization for this category of services. The architecture consists of numbers of *simulators*, each of which are attached to each replica of services, those *simulators* are designed to optimize for QoS per-service and consolidate themselves with the most up-to-dated QoS model. In particular, each *simulator* collects data from multiple services and interacts with others in a peer-to-peer manner, but the continuous modeling of QoS and QoS optimization decision making are done locally. By separating the intensive modeling and optimization processes, it is possible to prevent bottleneck of centralized decision making. As a result, we are able to adopt sophisticated metaheuristic decision makers towards optimal trade-off decision for QoS. The novel perspective is that each of the simulator leverages on the classical DDDAS, where it acts with a symbiotic and bidirectional model. Such info-symbiotic and bidirectional model is a promising approach to fulfill the requirements for self-adaptive QoS optimization in the cloud. We report on the design of this solution and its basic elements. We empirically evaluate our solution via a case study. To the best of our knowledge, we are the first to propose a decentralized DDDAS architectural style for cloud-based systems, where dynamic QoS optimization is in the heart of the symbiotic adaptation.

In the rest of the paper, section 2 models the problem. The architecture and details of its components are proposed in Section 3. Section 4 presents a case study using our approach. Section 5 reviews the related work. Section 6 concludes the paper and discusses future research directions.

2. Cloud-based QoS Optimization

2.1. Preliminary

In the cloud environment, QoS tends to be sensitive to two types of primitives. These are ***Environmental Primitive (EP)*** *and* ***Control Primitive (CP)***. CP can either be software or hardware supporting QoS provision. *Software CPs* are software tactics, such as the number of threads in thread pool and its life time, the number of connections in database connection pool, security and load balancing policies etc. *Hardware CPs* are computational resources, such as CPU, memory and bandwidth. These software and hardware CPs are offered at the Platform-as-a Service [9] and Infrastructure-as-a Service [10] respectively. We related EP to highly dynamic scenarios, which can significantly influence the QoS but the provider cannot manage and control their behavior. Examples include unbounded workload and unpredictable bound received data etc. If the provider would be able to control the presence of these scenarios, such primitives can be then considered as CPs. For given QoS attributes with the budget and capacity constraints, CPs can be scaled accordingly to improve QoS. *Our objective is to optimize these QoS attributes by provisioning the best combination of control primitives, while minimizing the cost of these primitives.* The challenge of self-adaptive QoS optimization encompasses the following dynamics:

Dynamic QoS modeling: Given a set of primitives, it is imperative to have a QoS model that capable to predict the achieved QoS with respect to its sensitivity to primitives. By sensitivity, we are specifically interested in answering these related questions: (i) *Which* primitives correlate with the QoS provision? (ii) *When* these primitives correlate with the QoS? (iii) *How* the uncertainty of QoS provision can be apportioned and be sensitive to these primitives? The QoS model and analysis of its sensitivity can assist in determining the sufficient provisions of CPs to achieve certain QoS objectives. The concerns of *which*, *when* and *how* primitives correlate with the QoS tend to fluctuate during runtime and thereby, result in dynamic QoS sensitivity.

Dynamic conflicting objectives: QoSs may be sensitive to the same CP and thus create the chance for conflicting. In particular, QoS optimization involves multiple non-combinable and possibly conflicting objectives (e.g, throughput versus cost, replica consistency versus performance and security versus performance etc). Such conflict requires dynamic resolution for the involved tradeoffs on the fly, especially when the related SLA/QoS requirements could subject to change in runtime. In addition, conflicts can be *inter services* (i.e. tradeoffs between QoS of multiple services) and *intra service* (i.e. conflicts and tradeoffs for a specific service). As a result, the decision making during optimization involves various constraints and significant trade-offs, which need to be handled at runtime. Existing approaches are either based on single objective [4,5,6] or assume static and limited number of objectives and constraints [7,8]; henceforth, they tend to have limited adaptivity and online dynamics for finding the best trade-offs in the cloud.

By interlinking all the aforementioned concerns, we argue that dynamic and self-adaptive QoS optimization problem for cloud-based DDDAS can be formulated and resolved as a *Dynamic Multi-objectives Optimization Problem*. The process incorporates dynamic tradeoffs decision making, where the dynamics are attributed to continuous changes in the objective function, their degree of conflict and constraints.

2.2. Modeling the problem

We start modeling the problem by presenting our assumptions and how the QoS optimization problem can be formulated as a dynamic multi-objective optimization problem. We assume that cloud-based DDDAS is formed of one or more independent or composable services. These systems are hosted on the cloud infrastructure where resources are shared via Virtual Machines (VMs). It is possible to host multiple systems on the same VM. For the sake of simplicity, in this work we assume one-to-one relationship between an system instance and a VM instance.

A cloud-based DDDAS, composed of concrete services $\{S_1, S_2, \ldots S_i\}$ may have multiple replicas deployed to different VMs. A replica of an application running on a VM is assumed to have its services replicas running on the same VM. In this work, we refer the replicas of concrete service as service-instances; the *jth* service-instance of the *ith* concrete service is denoted by S_{ij}. The primitives, which a QoS is sensitive to are called *relevant primitives* of the said QoS. Different service-instances may reside on different VM instances, therefore their QoSs could have heterogeneous relevant primitives and Physical Machine (PM) capacity. In this context, we consider fine-grained, per-service QoS optimization. More precisely, we tend to dynamically optimize for

QoS in relation to the service-instances, through considering the possible conflicts and various fluctuation in given scenarios. As the application is composed of one or more concrete service and their associated instances, QoS optimization on each concrete service (and their instances) would result in emergent optimization on the whole system. SLA negotiation is an important but out of scope topic for this paper, thus as SLA evolves, we assume that changes are reflected, negotiated and approved.

To model the QoS optimization problem, we first abstracting the QoS model for each service-instance. Formally, the QoS sensitivity model of the *kth* QoS of a service-instance S_{ij} can be formulated as:

$$QoS_k^{ij} = f\left(CP_1^{11}, \ldots CP_a^{xy}, EP_1^{11}, \ldots EP_b^{mn}\right) \quad s.t. \quad CP_a^{xy}, EP_b^{mn} \in SP_k^{ij} \tag{1}$$

where QoS_k^{ij} is the *kth* QoS of S_{ij}, f is the QoS objective function, which subjects to change dynamically. The objective of formula (1) is to minimize or maximize the achieved QoS. We denote SP_k^{ij} as the set of relevant primitives of QoS_k^{ij}. CP_a^{xy} and EP_b^{mn} denote the *ath* CP of service-instance S_{xy} and the *bth* EP of service-instance S_{mn} respectively. The entries in SP_k^{ij} are selected from the primitives that associated with S_{ij} and other related services. In particular, a primitive of a service-instance is considered as an entry if fluctuation of the said primitive positively or negatively interfere QoS_k^{ij}. Certain CP provisions can be partitioned to each service-instance (e.g. per-service database connection), whereas others (i.e. CPU, memory) are subject to be used in a sharing way. By sharing we refer to the amount of provision is accessed by multiple service-instances in a competitive manner. In particular, the sharing CP demands of those related service-instances are measured as an identical value. When SP_k^{ij} involves sharing CPs, the redundant column entries should be merged as they are referring to the same CP.

The *kth* CP of service-instance S_{ij} may be provisioned with certain cost, therefore the total costs model for S_{ij} with *n* CP types is represented as:

$$Cost^{ij} = \sum_{k=1}^{n} g\left(CP_k^{ij}, P_k^{ij}\right) \tag{2}$$

where g is the predefined, unify cost function for each type of CP. P_k^{ij} denotes the corresponding price of the *kth* CP for service-instance S_{ij}, in this work, we assume that the price of each CP type is fixed for all consumers and their service-instances. The objective of formula (2) is to minimize the cost. If multiple service-instances are sharing the same CP provision, the cost of such CP is equally proportioned to each of those instances.

At this stage, any group of QoS or cost models that are sensitive to the same CPs (regardless if such CP is shared) implies that their objectives could be potentially conflicting to certain degree, thereby at this stage, our goal is to continuously optimize every group of possible non-combinable and conflicting objectives by determining the best combination of CP provisions. More formally, for every group of conflicting objectives, the problem can be formulated as the following multi-objectives optimization problem:

$$Max\,/\,Min\left(o_{11,}\, o_{12.} \,..\, o_{ij}\right) \tag{3}$$

The o_{ij} denotes the vector of objectives for a service-instance S_{ij}, formally expressed as:

$$o_{ij} = \langle QoS_{1,}^{ij} QoS_2^{ij} \ldots QoS_k^{ij}, Cost^{ij} \rangle \tag{4}$$

whether an objective in formula (4) is to maximize or minimize depends on the nature of that objective. In particular, these objectives are subject to:

$$\left(\forall\, QoS_k^{ij} \in o_{ij}\right) \geq SLA_k^{ij} \tag{5}$$

$$Cost^{ij} \leq Budget^{ij} \tag{6}$$

$$\left(\forall\, CP_a^{ij} \in QoS_k^{ij}\right) \leq Capacity_a \tag{7}$$

where formula (5) states that any QoSs should meet its minimum requirement in SLA. (6) denotes the total cost of each service-instance should not exceed its budget. Finally, (7) represents any CPs that correlate with the QoS should not exceed the fixed capacity of underlying hardware or software.

3. Data-Driven Architecture for Adaptive QoS Optimization in Cloud

In this section, we describe the decentralized DDDAS-inspired architecture to solve the problem. We also specify the techniques that were designed to support the components in such architecture.

3.1. Decentralized DDDAS based architecture

As shown in Figure 1, service-instances in the cloud are running with symbiotic, distributed and decentralized *simulators*. More precisely, to prevent bottleneck of centralized control, each service-instance is attached with a dedicated simulator instance, which is linked to the VM where the service-instance is deployed. Those *simulators* collect online data from the monitors and analyze the state of service-instances; they aim at providing more accurate predictions of the QoS in relation to its primitives and optimizing QoS tradeoffs. To achieve these goals, our DDDAS based architecture consists of two independent inner loops within the global feedback control. The first inner loop periodically updates QoS models by dynamically capturing QoS sensitivity of the attached service-instance and detecting changes in constraints based on sensing data (step 1-4.1). The second inner loop applies iterative, metahueristic-based optimization to search the best combination of CP provisions, and simultaneously take into consideration all objectives that potentially conflicted with those of the attached service-instance (step 5). Specifically, our architecture optimizes QoS via the following steps:

Step 1: *Data sensor* collects data from the underlying service, platform and infrastructure managers. This data includes the currently achieved QoS, EP of service and demand of CP (both software and hardware CPs), as well as the agreed constraints in SLA. In particular, *data sensors* should sense all likely relevant primitives, from the attached service-instance and even other related service-instances (see section 3.3 for details).
Step 2: *Primitives selector* analyses all historical data from data sensors; its goal is to determine *which* and *when* primitives correlate with a QoS.
Step 3: Once the relevant primitives for a QoS are selected, the *QoS objective function trainer* applies machine learning techniques to determine *how* primitives correlate with QoS, by training the objective functions for each QoS of the corresponding service-instance.
Step 4.1: The steps from 1-3 are run periodically to ensure dynamic QoS sensitivity can be fully captured.
Step 4.2: At a given sampling interval, data is sensed to determine under or over provision states. The *QoS objective function trainer* could then triggers the optimization process with the trained QoS model. Due to the dynamic nature of the optimization, this step may be repeated when the optimization is running for evaluating solutions with the most up-to-dated QoS model and SLA constraints. Such process can be achieved without restarting the entire optimization because of the nature of the used metahueristic techniques.
Step 5: Based on the QoS model, it is possible to optimize QoS by proactively preventing under- or over-provision states. In particular, the *QoS optimizer* iteratively search for better combination of CP provision by looking ahead the predicted QoS for next interval, while considering the conflicting objective and constraints. The process terminates when it reaches its maximum number of iterations.
Step 6: The *QoS optimizer* feedbacks to the Platform- and Infrastructure-a Service cloud provider for provisioning the decided CP. Hence, the service and application can be scaled up/down or in/out accordingly.

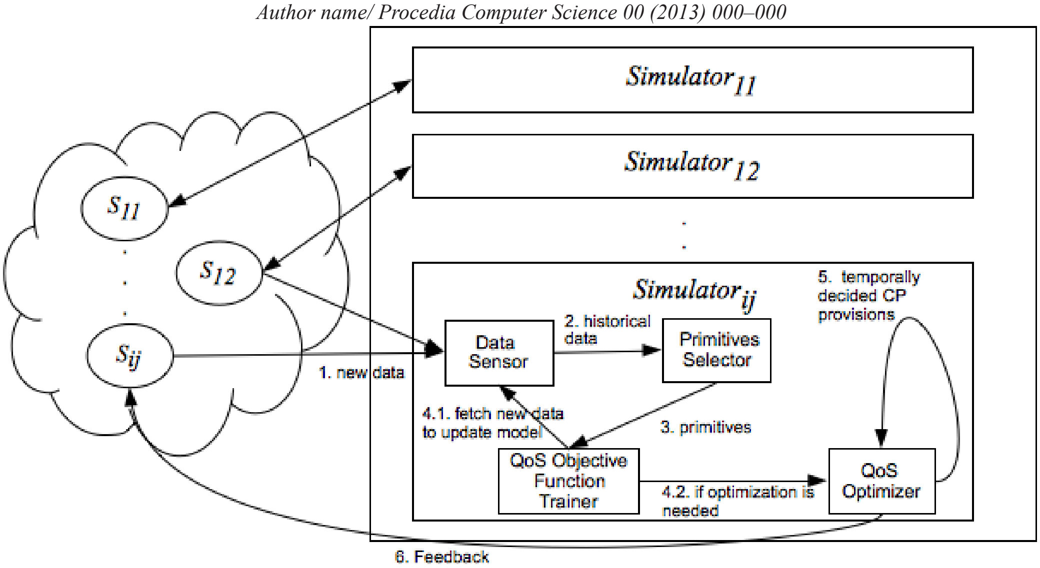

Fig. 1. Cloud Architecture incorporating DDDAS

It is clear that the info-symbiotic DDDAS paradigm is able to adaptively optimize QoS of cloud-based service-instances via the *simulators*. On the other hand, DDDAS can further consolidate the *simulators* in terms of modeling QoS and its sensitivity and detecting the changes of SLA constraints through continuous sensing the service-instances. In the next sections, we describe the components of Figure 1.

3.2. Data sensor

The *Data sensors* are designed to collect QoS values for a given service-instance, and they are also responsible for collecting data related to the likely relevant primitives (see section 3.3 for details), SLA and capacity constraints from the said service-instance and other related service-instances. In this work, we assume that the measurement features are offered by the Platform-as-a Service facilities and Infrastructure-as-a Service hypervisors.

3.3. Primitives selector

To provide a solution for formula (3), we must have QoS models in formula (1) that are capable to dynamically determine how QoS objectives can be achieved and their degree of conflict. In such context, identifying the SP_k^{ij} for certain QoS_k^{ij} is difficult, especially SP_k^{ij} may be continuously changing due to the dynamic nature of cloud. *Primitives selector* is responsible for determining *which* and *when* certain primitives correlate with the QoS at runtime. More precisely, this can be achieved by applying techniques like *symmetric uncertainty* [1], which represents mutual dependency between two random variables. A symmetric uncertainty value of 1 means two variable are closely correlated whereas a value of 0 implies that they are independent.

For each QoS of a service-instance S_{ij}, the likely relevant primitives could be selected from two sets of services: firstly, the service-instances on the same VM that S_{ij} belongs to (include itself) and secondly, service-instances that functionally required by S_{ij}. We observe that the primitives from those service-instances are most likely to result in non-zero symmetric uncertainty value with S_{ij}'s QoS.

3.4. QoS objective function trainer

Recall from formula (1), once the SP_k^{ij} is defined by the *primitives selector*, our next goal is to determine *how* those primitives correlate with QoS_k^{ij}, taking all relevant primitives as inputs. *QoS objective function trainer* is responsible for training QoS objective function for its corresponding service-instance. In particular, we promote the use of machine learning techniques, for example, Artificial Neural Network [13] and Auto-Regressive Moving Average with eXogenous inputs model [14] to model such objective function. It has been proven that they are capable to produce accurate model without knowing internal structure of the service and underlying infrastructure [15,17]. As a result, those modeling techniques are superior to closed-form models (e.g. queuing network) as they do not rely on fixed assumptions of QoS sensitivity. In particular, the data that feed into those models should be normalized values, which can be calculated as the ratio between current value at interval t and the biggest ever value through the entire time series. We conduct the training with normalized values because the original measurements have arbitrary magnitude, which obfuscates the sensitivity of output to inputs and consequently the accuracy of model. To cope with dynamics, the function f shall be continuously trained with the newly-measured values. For improving prediction accuracy, it is possible to apply multiple machine learning techniques simultaneously, in which case the resulting model with the least percentage error would be used for certain point in time.

QoS objective function trainer is also responsible for determining whether to trigger the optimization process based on the sensed data of its corresponding service-instance. More concretely, once the attached service-instance is under/over provision, the trained QoS models are then passed to the *QoS optimizer*.

3.5. QoS optimizer

Objectives could be in conflict if their models are sensitive to the same CP. For each attached service-instance, *QoS optimizer* locally identifies the potential *intra* service conflicting objectives. However, this is insufficient as conflicting objectives could occur *inter* services. To cope with this problem, *simulators* need to continuously interact with each others in a peer-to-peer manner. More precisely, each *QoS optimizer* acquires the constraints, QoS and cost models from the service-instances, which are 1) directly or indirectly interact with the attached service-instance; 2) deployed on the same VM as the attached service-instance and those interacted service-instances; 3) from other VMs but sharing the same CP with the attached service-instance (i.e. a global control of load balancing policy for all instances of a service). In doing so, we can identify all the potential conflicting objectives and they can be optimized on one *simulator*. To prevent duplicate optimization, the *QoS optimizer* also needs to continuously make sure that an objective is not currently being optimized within another optimization process; otherwise, the current optimization should be aborted. This solution is potentially scalable since different *simulators* could trigger numbers of optimizations in parallel, as long as their objectives do not lead to any conflicts.

Due to the dynamic QoS sensitivity and conflicting objectives involved in cloud based QoS optimization, greedily optimize QoS is usually very time and resource consuming, therefore it is generally acceptable to optimize QoS to a “good enough” level. In our *QoS optimizer*, we endorse the use of metaheuristic techniques like Ant Colony Optimization [16] to solve the dynamic multi-objective optimization problem in formula (3) as such approach could efficiently reach sub-optimal solutions under multiple potentially conflicting objectives [7,8]. In addition, they are capable to cope with dynamic changes of objectives' constraints and their degree of conflict even during the search process [8]. For each group of potential conflicting objectives, the metaheuristic approach consists of many iterations for constructing the final pareto set, each iteration is a heuristic searching process based on the given models and what-if scenario to predict if the objectives can be achieved. These models could be based on formula (2) and (1) where (2) is a predefined cost model whereas (1) is the dynamically learned QoS model. As a result, the goal of our *QoS optimizer* is to determine the pareto optimality that consists of the best combination of CP provisions for optimizing objectives in (3), considering constraints in formula (5)-(7).

To cope with dynamics, all solutions are archived and they are re-evaluated when the QoS models are updated. Because of the heuristic search, such goal can be achieved without the needs to restart the entire search process. The metaheuristic concludes when it reaches the maximum number of iterations. At the final stage, the entire pareto set can be sorted based on non-dominated ranking and crowding distance [3], and then the pareto optimal solution is selected as the ultimate trade-offs decision.

4. Applicability

We describe our architecture through a scenario to demonstrate how QoS of cloud-based DDDAS can be optimized in a self-adaptive manner. Consider an organization, which owns an DDDAS-based geographic analysis application named geobay. Such application provides two important services: 1) Calculating and tracking the geographical locations of mobile devices (S_1) 2) Predicting the next geographical movement of mobile devices (S_2), which is more intensive to resource as it requires more computation. The QoS requirements for these two services are: security and throughput for S_1; throughput for S_2. Suppose the organization wish to move to cloud by integrating their application with a Platform-as-a Service provider, named BppEngine. The infrastructure level resources are leased from the Infrastructure-as-a Service provider Bmazon. Suppose the Platform-as-a Service provider provide two software CPs: level of secure constraints and number of database connection. Suppose again, the Infrastructure-as-a Service provider offer CPU and memory as hardware CPs. The EP involved in this scenario is the workload of service. Note that in this case, we refer the throughput as completed request per second, whereas workload denotes the actual incoming request per second regardless whether they are completed or not. We assume that by default, geobay is deployed as two replicas on two VMs with default provisions. More precisely, S_{11} and S_{21} are deployed on the same VM whereas S_{12} and S_{22} are placed on another VM. In such context, the organization would have different QoS requirements and budgets for S_1 and S_2 , all of the above CPs are leased on certain prices with their own capacity, those constraints are shown in Table 1. Note that certain constraints (i.e. budget and throughput) needs to be translated to each service-instance (e.g. if the throughput for S_1 is 100 req/sec then it would be 50 req/sec for S_{11}). Both providers support a simple cost function *g* for each type of CP as: amount of provision times supplying price.

Upon the initial deployment of services, the consumer or third party middleware provides the measurement function and information regarding the required services of geobay to our *data sensors*. In this case, all the services in geobay are standalone services.

The first close loop (steps 1-4.1) formed by *data sensor, primitives selector* and *QoS objective function trainer* periodically produces QoS model for each service-instance at each interval, even when no optimization is needed. Such QoS model is a simulation of the most up-to-dated state of the service; henceforth, it is possible to look ahead to the next interval and predict if it is likely to cause any over- or under- provision during the optimization process. Suppose at the *ith* interval, QoS modeling learns that security of S_{11} is sensitive to the level of secure constraints for S_{11} only. Because S_2 tends to consume large amount of resources, the throughput of S_{11} could be sensitive to CPU, memory of the VM as well as numbers of database connection and workload for both S_{11} and S_{21}. In the mean time, the throughput of S_{21} could be sensitive to CPU, memory of the VM as well as numbers of database connection and workload of itself only. Now, suppose the actual throughput of S_{11} drop below 50 req/s, the *QoS objective function trainer* of S_{11} then triggers the *QoS optimizer* for optimization using metaheuristic (step 5). Objectives that do not lead to conflict are separated into different optimization processes, each requiring its own metaheuristic. This case would lead to two groups of process: Firstly, objectives which are related to S_{11} and S_{21}; and secondly, objectives which are associated with S_{12} and S_{22}. The objectives for S_{11} and S_{21} are potentially conflicting in an inter and intra manner since their QoSs are sensitive to the same CPs (throughput of S_{11} and S_{21} versus their total cost, security of S_{11} versus its total cost and throughput of S_{11} versus throughput of S_{21}). As a result, the optimization objectives that should be considered in formula (3) are: security, throughput and total cost of S_{11} as well as the throughput and total cost of S_{12}. The *QoS optimizer* of S_{11} leverages the trade-off estimated by the QoS and cost model, and eventually concludes with an optimal trade-off solution.

An example of the final CP provisions and the optimized QoSs for S_{11} and S_{21} are shown in Table 1. Our DDDAS based solution is able to adaptively assist the organization to make good leverage among the objectives of security and throughput with reasonable cost for each of geobay's service (and their instances).

Table 1. Example of provisioned CPs and QoSs after optimization (p/h = per hour)

QoS	Provisions of CPs	SLA for (5)	Budget for (6)	CP capacity for (7)	Price for (2)	Total Cost
S_{11} security=0.82 unit	Level of secure constraints for S_{11}=3	0.7 unit	$3.5 p/h	CPU= 2.88GHz memory=2GB level of secure constraints=5 database connection=100	CPU=$0.32 per GHz p/h memory=$0.11 per GB p/h level of secure constraints =$0.52 per level p/h database connection=$0.07 per 20 connection p/h	$2.302 p/h
S_{11} throughput=63 req/sec	CPU=1.33GHz, memory=1.22GB, database connection for S_{11}=82, database connection for S_{21} =51	50 req/sec				
S_{21} throughput=42 req/sec	CPU=1.33GHz, memory=1.22GB, database connection for S_{21}=51	40 req/sec	$1.5 p/h			$0.459 p/h

5. Related Work

Approaches proposed for QoS optimization have been using static rule-based mathematical model, which relies on assumptions that a single, optimal solution would be always discovered. In particular, [5] argues that the problem of finding the optimal VM allocation for QoS optimization can be formulated as the mixed integer linear optimization problem, which is solved by a heuristic approach. [4] views the QoS driven resource provisioning in cloud as the mixed integer non-linear programming problem and proposes solution based on force-directed search algorithm. [11] identifies the importance of software CPs when managing QoS and proposes a nested double feedback loop for realizing the management process; one for software CPs and one for hardware CPs. However, unlike the applied DDDAS in this paper, those approaches rely on single directional model, which means the adaptive controller has limited sensitivity to the changed state of systems. In addition, they assume fixed and closed-form QoS model. In the truly dynamic cloud, assumptions on fixed QoS sensitivity and closed-form QoS models are infeasible. Similar to the concept of DDDAS, [6] also proposes a bidirectional control loop, in which the state of the system is retrieved and stored along with the action that enables the system to reach such state in a knowledge database, and the system can be influenced by selecting the proper action in future iteration. Nevertheless, their architecture is centralized, which tends to cause high latency when the number of service increases. Machine learning techniques have been studied for managing QoS [15,17], however their treatments of *which* and *when* primitives correlate with QoS have been static. In addition, they do not consider software CPs. Our approach resolve all those concerns and we assume fine-grained QoS with *simulators* to dynamically model QoS and its sensitivity based on DDDAS concept.

Another broad of approaches take conflicting objective into account and they look at evolutionary heuristic techniques. Particularly, [7] posits that QoS optimization should be done upon service deployment, and they propose genetic algorithm based solution for searching the optimal resource plan with consideration of four objectives. [8] describes a successful use of ant colony optimization for finding optimal service workflow for optimizing QoSs to meet their requirements. However, their approaches rely on fixed assumptions of objective and degree of conflict. Our approach considers more than four objectives and adopts online metaheuristic using DDDAS, which adaptively cope with those dynamics and making good trade-off decisions.

6. Conclusion

In this work, we have proposed a decentralized architectural style for cloud-based DDDAS, where dynamic QoS optimization is in the heart of the symbiotic adaptation. The architecture leverages on the classical DDDAS primitives to reach a refined decentralized style suited for the dynamic requirements of the cloud. We have formulated the QoS optimization problem as a dynamic multi-objective optimization problem and described the principles for architecting the solution. The proposed architecture is exemplified using a case study. In future work, we intend to simulate the behavior of more sophisticated composite cloud-based DDDAS services in real settings. We will also report on scalability and elasticity of the approach in terms of execution time and as when compared to a centralized variant.

References

[1] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann: Los Altos, CA, 2005.

[2] Darema, F., C. Douglas, and A. Patra. 2010. The Power of Dynamic Data Driven Applications Systems, *Multi-Agency Workshop on Info-Symbiotics/DDDAS, Air Force Office of Scientific Research and National Science Foundation*, Washington DC, 2011.

[3] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan, A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II. I*n Proc. Parallel Problem Solving From Nature VI Conf.* 2000, pp. 849–858.

[4] H. Goudarzi and M. Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. *In Proc. of the IEEE Cloud*. 2011.

[5] V. Cardellini et al. SLA-aware Resource Management for Application Service Providers in the Cloud*. In Proceedings of the 1st International Symposium on Network Cloud Computing and Applications*, 2011.

[6] V. C. Emeakaroha, M. A. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. D. Rose. Towards autonomic detection of sla violations in cloud infrastructures. *Future Generation Computer Systems*, 2011.

[7] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. Evolutionary Deployment Optimization for Service-Oriented Clouds. *Software: Practice and Experience,* 2011, 41(5), pp.469-493.

[8] H. Liu, D. Xu and H. Miao. Ant Colony Optimization Based Service flow Scheduling with Various QoS Requirements in Cloud Computing. *In Proceedings of the 1st International Symposium on Software and Network Engineering*, 2011.

[9] Google App Engine, http://code.google.com/appengine/

[10] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/

[11] Y. Zhang, G. Huang, X. Liu, and H. Mei. Integrating Resource Consumption and Allocation for Infrastructure Resources on-Demand. *In Proceedings of 3rd IEEE International Conference on Cloud Computing*, 2010.

[12] Darema, F. Keynote in CyberBridge, http://www.cse.psu.edu/conferences/cb2012/fdarema.html

[13] W. S. Sarle. Neural networks and statistical models, 1994.

[14] G.Box, G.M. Jenkins and G.C. Reinsel. Time Series Analysis: Forecasting and Control, third edition. Prentice-Hall, 1994.

[15] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao and K. Dutta, Modeling Virtualized Applications using Machine Learning Techniques, *in Proceedings of the 8th International Conference on Virtual Execution Environments*, London, UK, 2011, pp. 3-14.

[16] M. Dorigo et al. The ant system: Optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics, 26(1), 1996.

[17] Q. Zhu and G. Agrawal, Resource provisioning with budget constraints for adaptive applications in cloud environments, *in Proceedings of the19th ACM International Symposium on High Performance Distributed Computin*g (HPDC'10), 2010.