Proceedings of The 6th IEEE International Symposium on Service Oriented System Engineering (SOSE 2011)

Evaluating Services on the Cloud using Ontology QoS Model

Guang Chen *, Xiaoying Bai* †, Xiaofei Huang *, Muyang Li *, Lizhu Zhou *
*Department of Computer Science and Technology, TNList, Tsinghua University, Beijing, China
† SKLSDE, Beijing University of Aeronautics and Astronautics, Beijing, China
Email: {chen-g08, li-my08, huangxf10}@mails.tsinghua.edu.cn, {baixy, dcszlz}@tsinghua.edu.cn

***Abstract*—To achieve large scale reuse and economy of scale, software are wrapped into services and migrated to the cloud platform. As a result, the quality of software services are affected by many third party components such as infrastructure services and platform services. In the collaborative framework, the stakeholders may have distinct, even contradictive, quality objectives. It is necessary to build agreements on quality properties, and to evaluate at runtime system's conformance to its service level agreements from different views. The paper proposes a QoS (Quality of Services) model for software services deployed on the cloud. The model identifies QoS concerns from three dimensions: system resource utilization, service performance, and price. Ontology language is used to describe these quality concerns and their properties. Based on the quality model, the paper proposes a method for service satisfaction evaluation from the end users' view point, taking all the three categories of factors into consideration. Experiments are conducted on Amazon EC2 platform using SPECWeb *Support* benchmark. It analyzes how each factor contributes to the end systems' satisfaction for different EC2 instance configurations.**

I. Introduction

Following the Service-Oriented Architecture (SOA), software systems can be built by composing services online based on standard protocols. Cloud provides an open platform for hosting the services using shared resources. Compared with in-house software development, Cloud-based service composition is expected to provide economy of scale. That is, it aims to reduce cost throughout the software lifecycle (including development, deployment, runtime, and maintenance) in the face of tremendous increases in software size and complexity.

To deliver the promises of this novel computing paradigm, quality is of critical importance. However, as systems are dynamically constructed using third-party services, it is hard to ensure the services' observability, controllability and testability. For example, Amazon provides a huge cloud infrastructure EC2 (Elastic Compute Cloud) and web-hosting system AWS (Amazon Web Services) based on EC2 [1]. It promises to keep customers' sites up and running 99.95% of the year, which only allows for 4.4 hours of downtime. Unfortunately, an unexpected crash happens in April, 2011 due to operation mistakes during network reconfiguration [2]. More than 70 organizations are affected including FourSquare, the New York Times, and Reddit, who pay to use AWS to run their websites on EC2. Due to the accident, the performance of these websites are greatly decreased, and some sites were even down for dozens of hours.

Consequently, the Quality of Services (QoS) for cloud and service computing presents new challenging problems, as discussed below.

- **How to define QoS?** Cloud and SOA enable collaborative development. Various stakeholders participate in the platform including infrastructure provider, service provider, service consumer, etc. Each party takes a distinct view of quality objectives. For example, infrastructure providers need to maximize resource utilization by reclaiming unused resources in real-time, service providers want to enhance system performance with reserved resources, and end users prefer acceptable performance with low cost. The QoS requirements from different stakeholders may even conflict with each other. Conventional QoS models do not address issues like economy. Hence, new unified QoS model is necessary so that stakeholders can negotiate to achieve agreements on service levels.
- **How to describe QoS?** QoS requests are usually handled manually. For example, users need to fill in the application forms online when they apply Amazon EC2 services to host applications. The requirements cannot be applied across different clouds and service providers. To support dynamic service composition and provisioning, QoS should be expressed using machine-interpretable language that can be automatically precessed across platforms.
- **How to evaluate QoS?** In general, QoS can be objective or subjective. Objective QoS evaluates system behavior quantitatively using metrics like resource utilization, service availability, request-to-response time, etc. Subjective QoS is usually represented by qualitative measurements like user experience and satisfaction. There lacks effective mapping mechanism between objective and subjective QoS measurements. Furthermore, even within the same category, it is hard to compare between different quality concerns, due to differences in measurement units and semantics. To avoid "apples-to-oranges" comparison problem, QoS evaluation needs to 1) consider the influence and relationships between different quality concerns; 2) normalize data which are defined using different units; 3) transform from objective measurements to subjective measurements.

978-1-4673-0412-2/11/$26.00 ©2011 IEEE

In counter to these problems, the paper proposes a QoS model which considers quality concerns from three dimensions: resource utilization, service performance, and cost. Ontology language is used for describing the QoS model. QoS concepts are defined as ontology classes, and QoS constraints are described using first-order logic. Based on the QoS model, quantitative data can be collected at runtime. Services are evaluated from end users' viewpoint. Users' satisfaction is defined as a weighted sum of quality statistics of QoS concerns from different dimensions. A method called Analytical Hierarchy Process (AHP) is applied for normalizing the data and calculating the relative weight of each quality concern. Users may have different preference on resource, cost and performance. The paper analyzes how such preferences affect satisfaction evaluation.

In recent years, some ontology QoS models were proposed as extensions to current Web Services standards [3], [4], [5], [6], [7], [8], [9], [10]. The objective is to provide an unified conceptual model and knowledge-based language for QoS definition, description and evaluation. Compared with existing approaches, the paper contributes in the following ways:

- For the QoS model, it takes cost into consideration. From the system perspective, quality is usually achieved as a tradeoff between performance and cost.
- It applies AHP method to QoS quantitative analysis. In this way, quality concerns of different units can be normalized to a unified measurement and compared with each other.
- It makes an early attempts to derive qualitative system evaluation (e.g., satisfaction) from quantitative quality measurements (e.g., resource usage, service performance, and price).

A case study is exercised considering four quality properties: response time, throughput, CPU utilization and cost. It deploys SPECWeb benchmark on the EC2 platform and simulates 70 concurrent sessions from 5 clients. Taking differenting weighting profiles, it illustrates the process of the proposed approach.

The rest of the paper is organized as follows. Section II introduces the definition of the QoS model and its ontology representation. Section III presents the method of user satisfaction evaluation based on multi-dimensional quality properties. Section IV reports the case study on the EC2 platform. Finally, section V summarize the paper.

II. Ontology-Based QoS Model

A. QoS Model

The QoS model is defined from two aspects: **QoS Concerns** and**QoS Properties**. **QoS Concerns** define quality requirements from each stakeholder's viewpoint. Particularly, to address the needs of service systems on the cloud, the paper identifies three categories of quality concerns: *resource*, *service* and *price*.

- *Resource* measures the performance of infrastructure services, such as usage of CPU, Memory and Disk, disk read/write rate, I/O and network throughput, etc.
- *Service* measures observable service performance, especially from the end users' experience, such as request-to-response time, throughput, etc.
- *Price* measures the cost of performance following the pay-per-use model for resource renting and service invocation.

Ideally, performance should be directly proportional to cost. But price plans and physical limitations may cause variations to the linear scale. For example, TABLE I shows two price plan for EC2 rented instances. An instance is a package of CPU, memory and storage of different sizes such as *Small*, *Large*, and *Extra Large*. Instances can be rent in two modes: on demand and reserved. The cost is calculated by time, say \$/hour or \$/year. However, the rented instances may not be used all day with full usage of computing capacities. Hence, from users' perspective, the cost for each invocation of the system deviates from ideal scale. Given a rented instance, for the load within its capacity, the cost \$/transaction decreases with increasing load at each hour period [11].

TABLE I: EC2 Pricing

	Reserved Instances		**On-Demand Instances**	
	1 year Term	**3 year Term**	**Linux /UNIX Usage**	**Windows Usage**
$Small$	\$227.50	\$350	\$0.085/hour	\$0.12/hour
$Large$	\$910	\$1400	\$0.34/hour	\$0.48/hour
$ExtraLarge$	\$1820	\$2800	\$0.68/hour	\$0.96/hour

QoS Properties model the properties of QoS concerns. The paper captures three types of properties: *Constraints* of each concern, *Influence* between concerns, and *Weight* of each concern.

- *Constraints* identifies the restriction and limitations of each concerned quality parameter. For example, the response time of a service should below a pre-defined threshold.
- *Influence* identifies how different quality parameters may affect each other. It is defined by two parameters: positive/negative and level. For any two concerned parameters, p_1 and p_2, if p_1 has a positive influence on p_2, $Inf_+(p_1, p_2)$, then p_2 increases (decreases) as p_1 increases (decreases). On the contrary, a negative influence, $Inf_-(p_1, p_2)$, results in reverse change directions. The strength of the influence is measured by levels, say high, medium, and low influence levels.
- *Weight* identifies the contribution of a concerned parameter to the quality of the system. For example, suppose two quality profiles $Q1 =< f_1, c_1 >$ and $Q2 =< f_2, c_2 >$ where f represents performance parameter and c for cost parameter. To compare them, a policy may be defined as follows: when performance is within an acceptable range, cost is the primary decision factor. The degree of its impacts on final decision is measured by *Weight* for quantitative analysis.

B. QoS Ontology Specification

Ontology techniques provide the basis for specifying key concepts, their semantic interconnections, and inference rules in certain domain[12]. Originated from Philosophy, Ontology has been a knowledge representation technique that is widely used in computer science and information science[13]. An ontology is usually defined as a "formal, explicit specification of a shared conceptualization"[14]. It models domain concepts in terms of classes, properties, relationships, constraints, and instances (individuals).

In recent years, several QoS models are proposed using ontology language [3], [4], [5], [6], [7], [8], [9], [10], [15]. For examples, OWL-Q[4] is an upper level ontology that extends OWL-S for QoS-based Web Services description of both requests and offers. The ontology is separated into several facets including *Connecting Facet*, *Basic Facet*, *Metric Facet*, *Function Facet*, *Measurement Directive Fact*, *Schedule Facet*, etc. QoS Ontology Language [5] defines the associations between QoS attributes and their measurements. Each QoS attribute is annotated with *QoSParameters* representing non-functional properties, *Metrics* for measuring parameters, and *QoSImpact* representing how each parameter contributes to the service quality perceived by end users. onQoS ontology[6] supports specification of QoS advertisements by service providers and QoS requirements by service consumers. Tran et al. [10] made a survey of QoS ontology models. They pointed out two needs of QoS model for services: how to model QoS properties and how to support the usage of QoS properties. The former includes *Modular* and *Flexibility*, *Metric*, *Unit*, *Value Type*, *Impact Direction*, *QoS Property Relationships*, *QoS Transformation*, *QoS Dynamism* and *Valid Period*. The latter includes *Effect Level*, *Quality Level*, *Roles*, *Constraints*, *QoS Interdependence*, *Concrete QoS*, *QoS Priority*, *QoS Value Comparison*, *QoS Mandatory* and *QoS Grouping*.

This paper takes the resource utilization and cost into consideration, and proposes the ontology model to support service provisioning on the cloud. As shown in Fig. 1, the key concepts defined in section II-A are represented using ontology class model.

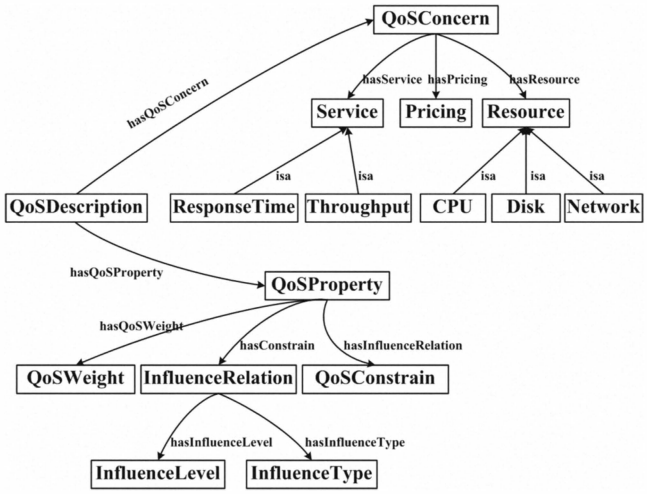


Fig. 1: QoS Ontology Specification

III. Service Satisfaction Evaluation

Based on QoS model, service quality parameters can be collected at runtime. We then evaluate service satisfactions from end users' view. Assume that $P = \{p_i\}$ is the set of parameters of a service S. Service satisfaction $Sat(S)$ is calculated by Equation (1) as a weighted sum of each concerned parameters, where $Sat(p_i)$ is the satisfaction of each parameter p_i and w_i is p_i's weight.

$$Sat(S) = \sum_{i=1}^{n} w_i \times Sat(p_i) \tag{1}$$

A. Parameter Satisfaction

Service quality parameters may be defined in different units and semantics. To have quality concerns comparable with each other, the parameter values are normalized to values within the interval of $[0, 1]$, as an indication of the degree of satisfaction.

Litoiu[16] uses Probability Density Function (PDF) to normalize the parameters. Given a parameter p, suppose its value rages in the interval of $[R_l, R_h]$ where R_l is the lower bound and R_h is the upper bound. The utility function for calculating satisfaction is as follows, where r is the runtime value of p. In case a higher value is preferred, it uses Equation (2); while Equation (3) is used when a lower value is preferred.

$$Sat(p) = \begin{cases} 0, & r < R_l \\ \frac{r - R_l}{R_h - R_l}, & R_l \le r \le R_h \\ 1, & r > R_h \end{cases} \tag{2}$$

$$Sat(p) = \begin{cases} 1, & r < R_l \\ \frac{R_h - r}{R_h - R_l}, & R_l \le r \le R_h \\ 0, & r > R_h \end{cases} \tag{3}$$

Litoiu's utility function assumes that satisfaction follows uniform distribution within the interval. In real cases, users' satisfactions are more like normal distribution. The paper then improves Litoiu's method to calculate satisfaction using normal distributions, as defined by Equation (4) and Equation (5).

$$Sat(p) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{r} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \quad , \quad -\infty \le r \le +\infty \tag{4}$$

$$Sat(p) = \frac{1}{\sqrt{2\pi}\sigma} \int_{r}^{+\infty} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \quad , \quad -\infty \le r \le +\infty \tag{5}$$

Where μ is normal variance, σ is normal mean, computing by Equation (6) Equation (7).

$$\mu = \frac{R_h + R_l}{2} \tag{6}$$

$$\sigma = \frac{R_h - R_l}{6} \tag{7}$$

B. Parameter Weight

We use Analytical Hierarchy Process (AHP) [17] to evaluate the weight of each QoS parameter. AHP is a widely used method for making decisions of complex problem with multiple criteria. It formalizes the decision-making algorithm, and allows for consideration of both qualitative and quantitative decision elements. Essentially, AHP involves interpreting the decision process as a series of one-on-one comparisons, and then synthesizing the results, in the process establishing a clear basis upon which the final decision can be made.

1) Pair-wise Comparisons Matrix: We use one-to-one comparisons of parameters, and build pair-wise comparison matrix (judgment matrix) $\mathbf{B}$.

$$\mathbf{B}= \begin{array}{c} \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{array} \begin{array}{c} \begin{array}{cccc} p_1 & p_2 & \cdots & p_n \end{array} \\ \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{21} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{pmatrix} \end{array} \tag{8}$$

In this matrix, each node $\{p_i|1 \leq i \leq n\}$ is a QoS concerned parameter, and each weight b_{ij} denotes the relative significance of any two parameters p_i and p_j, which satisfies the following equations.

$$b_{ii} = 1(i = 1, 2, \cdots, n) \tag{9}$$

$$b_{ij} = \frac{1}{b_{ji}}(i, j = 1, 2, \cdots, n) \tag{10}$$

2) Calculating Eigenvector: Based on the pair-wise matrix, the relative weight of each parameter is defined as the eigenvector $\mathbf{W}$ of $\mathbf{B}$.

$$\mathbf{BW} = \lambda_{\max}\mathbf{W} \tag{11}$$

$\lambda_{\max}$ can be calculated by the following 5 steps.

1) Normalize each column in the judgment matrix.

$$\overline{b}_{ij} = \frac{b_{ij}}{\sum_{k=1}^{n} b_{kj}}(i, j = 1, 2, \cdots, n) \tag{12}$$

2) Sum up the values of each row in the judgment matrix.

$$\overline{W}_i = \sum_{j=1}^{n} \overline{b}_{ij}, (i = 1, 2, \cdots, n) \tag{13}$$

3) Calculate the eigenvector.

$$W_i = \frac{\overline{W}_i}{\sum_{j=1}^{n} \overline{W}_j}, (i = 1, 2, \cdots, n) \tag{14}$$

4) Compute the principle eigenvalue.

$$\lambda_{\max} = \sum_{i=1}^{n} \frac{(\mathbf{BW})_i}{nW_i} \tag{15}$$

5) Check Consistency. We need to check the consistency of pair-wise comparisons matrix, using consistency index CI computed by Equation (16). It is obvious that judgment error increases while the order of matrix n increases. Decision makers judge consistency using consistency ratio CR as Equation (17), where RI is the consistency index defined by TABLE II. In general, the decision is considered consistent if $CR \leq 0.10$. Otherwise, the comparison matrix $\mathbf{B}$ must be readjusted until it reaches consistency.

$$CI = \frac{\lambda_{\max} - n}{n - 1} \tag{16}$$

$$CR = \frac{CI}{RI} \tag{17}$$

TABLE II: The Mean Random Consistency Index

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RI	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45

3) Hierarchical Comparison: Multi-dimensional criteria are always decomposed hierarchically from higher level generic evaluation goals to low level easier controllable factors. In this research, each QoS concerned parameter can be decomposed into sub-parameters, forming a tree-like structure. To compare parameters from different branches and levels, we further define two types of weights: local weight and global weight. The local weights represent relative importance of the nodes within a group of siblings with respect to their parent. The siblings at each level are compared and computed locally through pair-wise comparison matrix. The global weights are obtained by multiplying local weights of the siblings by their parent's global weights, as shown in Equation (18).

$$w_i^{global} = w_i^{local} \times w_{Parent(p_i)}^{global}, \quad (i = 1, 2, , n) \tag{18}$$

Here,

- w_i^{global} is the global weight of parameter p_i.
- w_i^{local} is the local weight of parameter p_i.
- $Parent(p_i)$ is the parent parameter of p_i.
- $w_{Parent(p_i)}^{global}$ is the global weight of $Parent(p_i)$.

IV. Case Study

A case study is conducted on real cloud with benchmark systems.

A. Experiment Setup

We use Amazon Elastic Compute Cloud (Amazon EC2)[18] for experiments. EC2 provides instances of three sizes, as shown in TABLE III on the next page to support different computing capabilities.

We use SPECweb2009[19] benchmark system as the service deployed on EC2. SPECweb2009 is a product of the SPEC (Standard Performance Evaluation Corporation) series. It is composed of four components: *Web Server*, *BeSim* backend simulator, *Prime Client* and *Client*. In this case study,

TABLE III: EC2 Instances

Instances	Compute Unit	Virtual Core	Memory	Storage	Pricing
Extra Large Instance (m1.xlarge)	8 EC2 Compute Units	4 cores	15 GB	1690 GB instance storage	$0.68 per hour
Large Instance (m1.large)	4 EC2 Compute Units	2 cores	7.5 GB	850 GB instance storage	$0.34 per hour
Micro Instance (t1.micro)	Up to 2 EC2 Compute Units	1 Core	613 MB	EBS storage only	$0.02 per hour

SPECWeb Web server is deployed to three instances of size *t1.micro*, *m1.large* and *m1.xlarge* respectively. We simulate 70 concurrent user sessions from 5 *Client* deployed on *t1.micro* instances, using *Support* benchmark workload. Each workload run include three iterations. Each iteration runs 30 minutes for measurement period.

Quality concerns are defined by 4 parameters, including *Aggregate Bit Rate*(p_5), and *Response Time*(p_6) for service performance, *CPU Usage*(p_7) for resource utilization, and *Pricing*(p_4) for cost.

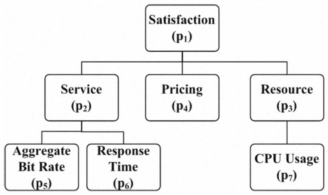


Fig. 2: QoS Parameters

B. Results

TABLE IV and TABLE V show the definition of judgment matrix and the calculated local weight and global weight of each parameter. The relative importance of b_{23}, b_{24}, b_{34} are defined as variables of x, y, and z respectively. For simplicity, the two parameters of service performance are considered equally important, hence $b_{56} = b_{65} = 1$. To ensure consistency of the judgement matrix, we take $z = y/x$ for simplicity in this case study.

TABLE IV: Judgment Matrix I

	p_2	p_3	p_4	w_i^{local}	w_i^{global}
p_2	1	x	y	$xy/(xy+x+y)$	$xy/(xy+x+y)$
p_3	$1/x$	1	y/x	$y/(xy+x+y)$	$y/(xy+x+y)$
p_4	$1/y$	x/y	1	$x/(xy+x+y)$	$x/(xy+x+y)$

TABLE V: Judgment Matrix II

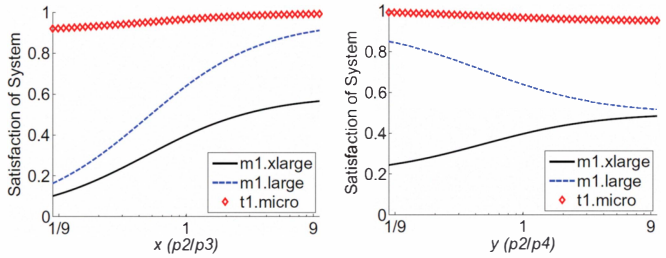
	p_5	p_6	w_i^{local}	w_i^{global}
p_5	1	1	1/2	$xy/(2xy+2x+2y)$
p_6	1	1	1/2	$xy/(2xy+2x+2y)$

TABLE VI shows the experiments results and computed satisfaction of each parameter. In general, it is expected that the system can make full use of the rented resources, deliver good enough performance, and is paid economically. With light load experiment, the instances of three sizes can all provide enough computing capabilities, and service performance are almost the same to each other. However, CPU usage and cost differ a lot. Hence, when evaluating each parameter individually, micro instance is the most satisfied one of the three. The satisfaction of each parameter is compared among the three instances as listed in the last row of the table. Fig. 3 and Fig. 4 shows the experiment results of service satisfaction with changing x and y values. Compared among the three instances (Fig. 3),

- As the weight of resource decreases, that is, with increasing ignorance of resource usage, all of the three instances' satisfaction increase. But the satisfaction of xlarge instance can only achieve about a half of that of micro instance.
- As the weight of pricing decreases, that is, with increasing ignorance of cost, the satisfaction of large instance decreases while that of xlarge instance increases until they reach almost the same.

For each individual instance Fig. 4 on the following page, it shows clearly that:

- The satisfaction of xlarge instance drops greatly with increasing attentions to cost and resource usage.
- The satisfaction of large instance is sensitive to weight resource utilization parameter.
- The satisfaction of micro instance is always above 0.9.



(a) x-Sat (b) y-Sat

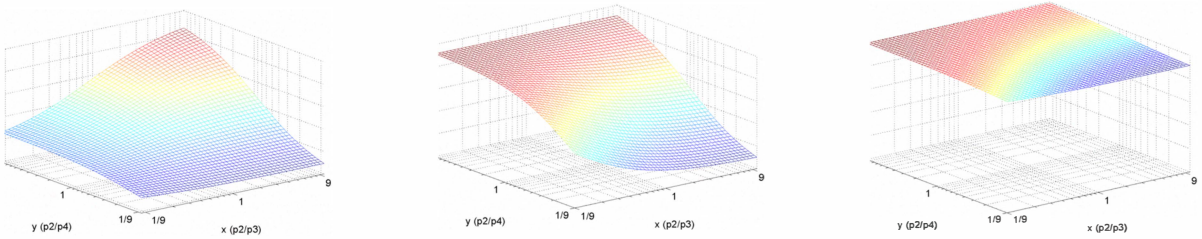
Fig. 3: Comparison of Three Instances

V. Summary

With the new paradigm, software systems are built by composing services that are remotely hosted on cloud platforms following a pay-as-you-go pricing model. Conventional QoS definition, description, usage and evaluation methods need to be adapted to address new issues like scalability, cost, and multi-dimensional evaluation. The paper proposes

TABLE VI: Satisfaction of QoS Properties

		Aggregate Bit Rate (p_5)	**Response Time (p_6)**	**CPU Usage (p_7)**	**Pricing (p_4)**
p_i	m1.xlarge	97.225KB/s	5.699s	1.849%	$0.68
	m1.large	97.135KB/s	5.692s	4.101%	$0.34
	t1.micro	97.066KB/s	5.723s	50.292%	$0.02
$Range$	R_l	0KB/s	5s	0%	$0.1
	R_h	100KB/s	12s	70%	$1
Desired		R_h	R_l	R_h	R_l
$Sat(p_i)$	m1.xlarge	0.998	0.992	0.002	0.193
	m1.large	0.998	0.992	0.004	0.919
	t1.micro	0.998	0.991	0.905	1.000
$Sat_{xlarge} : Sat_{large} : Sat_{micro}$		1 : 1 : 1	1 : 1 : 1	1 : 2 : 450	19 : 92 : 100



(a) xlarge instance (b) large instance (c) micro instance

Fig. 4: Satisfaction Trend

a QoS model to evaluate from three dimensions of resource utilization, service performance and cost. It uses ontology language to describe the quality concepts and their properties such as constraints, impacts and weight. The satisfaction of the system is defined as a weighted sum of parameter satisfactions. A case study is exercised on EC2 platform using SPECWeb benchmark. It shows how the selection of parameter weight affects satisfaction evaluation.

ACKNOWLEDGMENT

This project is supported by National Science Foundation China (No. 61073003), National Basic Research Program of China (No. 2011CB302505), and the Open Fund of the State Key Laboratory of Software Development Environment (No. SKLSDE-2009KF-2-0X).

REFERENCES

[1] Amazon web services. [Online]. Available: http://aws.amazon.com/
[2] "News Briefs," *Computer*, vol. 44, pp. 18–20, 2011.
[3] C. Zhou, L.-T. Chia, and B.-S. Lee, "DAML-QoS ontology for web services," in *IEEE International Conference on Web Services*, 2004, pp. 472– 479.
[4] K. Kritikos and D. Plexousakis, "Semantic QoS metric matching," in *4th European Conference on Web Services*. IEEE Computer Society, 2006, pp. 265–274.
[5] I. V. Papaioannou, D. T. Tsesmetzis, I. G. Roussaki, and M. E. Anagnostou, "A QoS ontology language for web-services," in *International Conference on Advanced Information Networking and Applications (AINA)*, vol. 1, 2006, pp. 101–106.
[6] E. Giallonardo and E. Zimeo, "More semantics in QoS matching," in *IEEE International Conference on ervice-Oriented Computing and Applications (SOCA)*, 2007, pp. 163–171.
[7] G. F. Tondello and F. Siqueira, "The QoS-MO ontology for semantic qos modeling," in *ACM symposium on Applied computing*. Fortaleza, Ceara, Brazil: ACM, 2008, pp. 2336–2340.
[8] L. Shuyu and Z. Juan, "The WSMO-QoS semantic web service discovery framework," in *International Conference on Computational Intelligence and Software Engineering (CiSE)*, 2009, pp. 1–5.
[9] G. Dobson, R. Lock, and I. Sommerville, "QoSOnt: a QoS ontology for service-centric systems," in *EUROMICRO Conference*, 2005, pp. 80–87.
[10] V. X. Tran and H. Tsuji, "A survey and analysis on semantics in QoS for web services," in *International Conference on Advanced Information Networking and Applications (AINA)*, 2009, pp. 379–385.
[11] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow? towards a benchmark for the cloud," in *Proceedings of the Second International Workshop on Testing Database Systems*, 2009, pp. 9:1–9:6.
[12] T. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, pp. 199–220, 1993.
[13] M. Singh and M. Huhns, *Service-oriented computing: semantics, processes, agents*. John Wiley & Sons Inc, 2005.
[14] T. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," in *International Journal of Human-Computer Studies*, vol. 43, no. 5-6. Kluwer Academic Publishers, 1993, pp. 907–928.
[15] V. X. Tran, "WS-QoSOnto: A QoS ontology for web services," in *S IEEE International Symposium on ervice-Oriented System Engineering (SOSE)*, 2008, pp. 233–238.
[16] M. Litoiu, "A performance engineering method for web applications," in *12th IEEE International Symposium on Web Systems Evolution (WSE)*, 2010, pp. 101–109.
[17] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, 1990.
[18] Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2/
[19] SPECweb2009. [Online]. Available: http://www.spec.org/web2009/