OmniaScience

Journal of Industrial Engineering and Management

JIEM, 2012 – 5(2): 457-472 – Online ISSN: 2013-0953 – Print ISSN: 2013-8423

http://dx.doi.org/10.3926/jiem.472

A QoS aware services mashup model for cloud computing applications

Yee Ming Chen，Yi Jen Peng

Yuan Ze University，Hsin Sheng College of Medical Care and Management (Taiwan)

chenyeeming@saturn.yzu.edu.tw，pengyijen@gmail.com

Received: March 2012
Accepted: November 2012

Abstract:

Purpose: With the popularity of cloud computing, cloud services have become to be application programming platform where users can create new applications mashup(composing) the functionality offered by others. By composing of distributed, cloud services dynamically to provide more complex tasks, services mashup provides an attractive way for building large-scale Internet applications. One of the challenging issues of cloud services mashup is how to find service paths to route the service instances provider through while meeting the applications' resource requirements so that the QoS constraints are satisfied. However, QoS aware service routing problem is typically NP-hard. The purpose of this paper is to propose a QoS Aware Services Mashup(QASM) model to solve this problem more effectively.

Design/methodology/approach: In this paper, we focus on the QoS aware services selection problem in cloud services mashup, for example, given the user service composition requirements and their QoS constraint descriptions, how to select the required service instances and route the data flows through these instances so that the QoS requirements are satisfied. We design a heuristic algorithm to find service paths to route the data flows through while meeting the applications' resource requirements and specific QoS constraints.

Findings: This study propose a QoS Aware Services Mashup(QASM) model to solve this problem more effectively. Simulations show that QASM can achieve desired QoS assurances as well as load balancing in cloud services environment.

Research limitations/implications: The number of mashup platforms and research works in the survey is limited. Furthermore, mashup platforms are continuously updated, thus some information might be outdated.

Practical implications: It was found that each different cloud service should have distinct business model. The QASM model is a tool for translating cloud computing technology into customer value.

Social implications: The purpose of this study was to explore QoS aspect of cloud computing business model from services provider viewpoint. The cost structure should be continued as economics as clouds are the key driving factor for both services providers and customers.

Originality/value: This paper present a QASM model for providing high performance distributed applications in the cloud computing systems

Keywords: QoS, cloud services, mashup, optimization

1. Introduction

The topic of cloud computing is gaining more and more attention in the service research community. Recently, Cloud technologies are emerging as infrastructure services for provisioning computing and storage resources, and gradually evolving into the general IT resources provisioning (Broberg, Buyya & Tari, 2009). Infrastructure as a cloud service refers to the sharing of service resources services, typically using virtualization technology. Infrastructure-as-a-service (IaaS) is one of the three basic service layers of cloud computing (Figure 1), in which on-demand virtualized computing resources are provided to the customer. Computing and storage resources are sold on-demand with limited by the customers, and consumption is readily scalable to accommodate heterogeneous user requirements. IaaS typically means buying or renting your computer power and disk space from an external service provider. This option allows you access through a private network or over the internet. The service provider maintains the physical computer hardware including CPU processing, memory usage, data storage and network connectivity. These resources can easily be scaled up when on-demand increases, and are typically charged for on a per-pay-use basis. Platform as a Service (PaaS), this service lies on the middle layer, is the encapsulation of a development environment abstraction and the packaging of a payload of services which developers can build and deploy customer application. Service approach (SaaS) is at the highest, most mature layer and features a complete application offered as a service and demand, rather than a traditional, on-premises software.

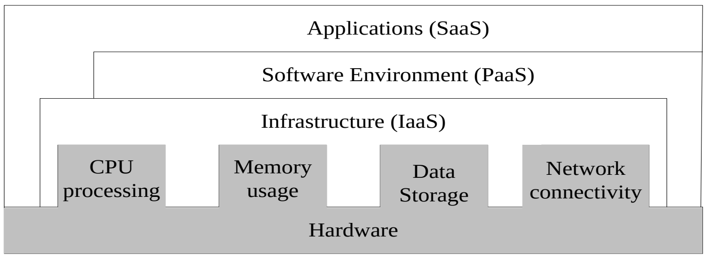


Figure 1. The basic cloud computing service

SaaS made available on a one-to-many basis over a network such as the Internet. The service providers are beginning to leverage their core competencies in so-called service value networks (SVNs) in order to jointly offer complex services (Buyya, Yeo & Venugopal, 2008). Meanwhile, SaaS brings a promising technique to create value-added business applications composed by dynamically selected "Internet of individual services". Analogous to complex products, services will be composed of several modules provided by different supply chain partners. Therefore, research on cloud services brings a promising technique to create value-added business applications composed by dynamically selected individual cloud services. This technology is so-called service mashup. Mashup is a Web-based network resource that composes existing services resources, be it content, data or application functionality, from more than one resource in enterprise environments by empowering the actual end-users to create and adapt individual information centric and situational applications. In internet there are maybe many available web services with various QoS (Quality of Service) providing the same functionality specific to a specific task. So a selection needs to be made. Therefore services mashup have to search for an optimal set of services to construct a composite service and result in a best QoS, under user's QoS constraints and resource requirements. And how to construct the cloud services mashup model is the main research subject of the paper. In this paper, we focus on the QoS aware services selection problem in cloud services mashup, for example, given the user service composition requirements and their QoS constraint descriptions, how to select the required service instances and route the data flows through these instances so that the QoS requirements are satisfied. We design a heuristic algorithm to find service paths to route the data flows through while meeting the applications' resource requirements and specific QoS constraints. The rest of the paper is organized as follows. We introduce the cloud computing system model in Section 2. Section 3presents the design and algorithm of the QoS aware service mashup model, followed by the extensive simulation results, presented in Section 4. Finally, we conclude this paper in Section 5.

2. Cloud Computing System Model

This section describes the cloud computing system model. First, we focus on cloud services mashup with QoS aware for Internet/Intranet-scale cloud computing systems. Then, we present the overview of the QoS aware services mashup.

2.1. QoS aware Services Mashup for Cloud Computing System

The term mashup originates from the practice of mixing song samples from two or more sources to produce a new sound track. Moreover, mashup stems from the emphasis on interactive user participation manner in which they aggregate and composite. Services mashup are websites or applications that combine service from more than one source into an integrated application. Based on the concept of services mashup in the Cloud Services environment, mashup provides flexible and dynamic services with rich experience. This technology also enables a dynamic form of service reusability in contrast to the traditional method of static "cut & paste" reusability. However, since mashup involves the aggregation of another party's instances into some new service or application, we will not deal with the copy right problem. We assume that the service aggregation model always chooses service instances that the user is authorized to use.

The bottom layer of cloud computing system contains the actual Web resources (Figure 2), be they content, data or application functionality. Each Web-based resource can be addressed by a Universal Resource Identifier (URI) giving browsers, mobile devices, and server applications alike accessibility to those resources (i.e. multi-channeling). The resources themselves are sourced via a well-defined public interface, the so-called Application Programming Interface (API) (Cahon, Melab & Talbi*,* 2004). APIs encapsulate the actual implementation as separate from the specification and allow Web-based resources to be loosely coupled. In this sense, the underlying resources are used as core building blocks to compose individual applications on top of existing resources. Being resources based and sourced via public APIs, gadgets (also known as widgets) provide application domain functions or information-specific functions (Chu, Nadiminti, Jin, Venugopal & Buyya*,* 2007). They are responsible for providing graphics, simple and efficient user interaction mechanisms which put a face to the resources and abstract from the technical description (functional and non-functional) of the Web-based resources. By assembling and composing a collection of gadgets stored in a catalogue or repository, knowledge workers are able to define the behavior of the actual application according to their individual needs, creating a composite application as a mashup.

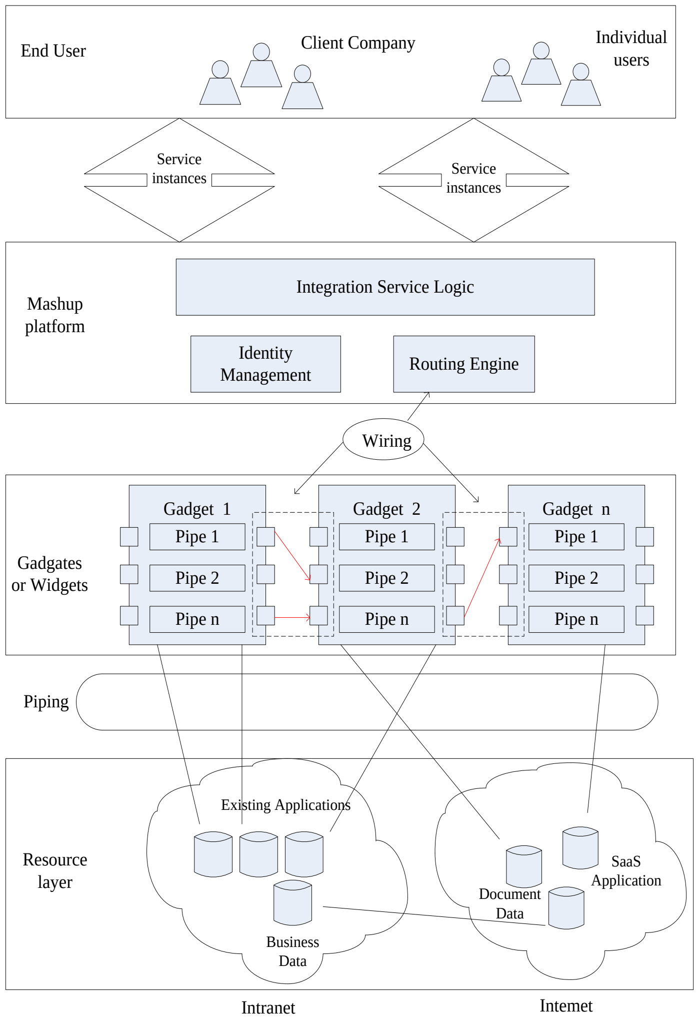


Figure 2. The Services Mashup for Cloud Computing System

As illustrated in the Figure 2, the composition takes place both in the resource layer (piping) and in the gadget (wiring) layer according to the enterprise mashup stack. Consequently, a mashup platform is a Web based tool that allows the creation of mashups by piping resources into gadgets and wiring gadgets together. Users access the services through a mashup platform of their choice. The mashup platforms are connected via APIs to the mashup integration services. To use the services, the users have to identify themselves against the user access control service; this service is connected to a user management service, which controls the users and their settings.

All data coming from the users go through a translation engine to unify the data objects and protocols, so that different mashup platforms can be integrated. The translated data is forwarded to the routing engine, which is the core of them as hub integration services. The routing engine takes care of processing the inputs received from the mashup platforms and forwarding them to the right recipient; the routing is the key part of the on-demand services mashup layer. It tackles the following two issues: (1) The composed service path must be QoS consistent which means that the input QoS requirements of service component must be "satisfied" by the output QoS of its predecessor; (2) If multiple QoS consistent service paths exist, the routing engine should choose the one which has the minimum aggregated resource requirements so that the overall workload of a cloud computing system is minimized. The routing engine can be configured through an API. To simplify this, a gadget could be provided for the end-user. The routing engine is also connected to a message queue via an API. In this paper, we propose a QoS Aware Services Mashup (QASM) model for choosing and composing different cloud services into a service path satisfying the user's quality requirements.

2.2 Overview of the QoS Aware Services Mashup

QoS aware services mashup has drawn much attention in recent years. Some works have been carried out on service selection algorithms for composing services with multiple QoS constraints to find an optimal solution, which is a NP-complete problem (Kirley & Stewart, 2007). Zeng, Benatallah, Ngu et al. (2004) suggested solving this problem by Linear Integer Programming (LIP). Although LIP is an optimal algorithm, its computation time tends to grow exponentially with the size of the problem instance, thus it is limited to use the LIP algorithm in real scenarios, especially in time-critical scenario. Several researchers put forward heuristic algorithms to find a near-optimal solution. Berbner, Spahn, Repp, Heckmann and Steinmetz (2006) presented an algorithm using the result of linear programming relaxation of LIP as the heuristic hint. The heuristic reveals that is extremely fast and outperforms linear programming based solutions with regard to computation time, especially with increasing number of candidate Web services and process tasks.

Currently, many experts and scholars do research on the problem of composed service routing focuses on how to find feasible and optimal paths that satisfy QoS requirements. Based on these forms QoS routing is broadly classified into two categories. MCP Routing (Multiple constrained path) (Wang & Crowcroft, 1996) and MCOP Routing (Multiple constrained optimal path) (Feng, Makki, Pissinou & Doulgeris, 2002). Where In MCP, the target is to find the feasible path satisfying multiple constraints, where as MCOP is a special case of MCP problem in which feasible path is found according to one of the constraints. Then from those optimal path is computed according to other constraint. Restricted Shortest Path (RSP) is a type of MCOP problem (Korkmaz & Krunz, 2001). Among the entire MCP problems RSP has received most attention. In general, MCP and MCOP both are NP-complete in nature that cannot be exactly solved in polynomial time. Hence the objective is to find the technique to reduce the computational complexity. To implement these technique, well known shortest path algorithms e.g. Dijkstra algorithm had been used by most of the researchers(Wang & Crowcroft, 1996; Kirley & Stewart, 2007). In this paper, based on the above ideas, proposing the Dijkstra-like algorithm of modeling and choosing as well as composing different cloud services into a service path satisfying the user's quality requirements.

3. Formal Definition and Problem

In order to describe the problem of services mashup in the cloud computing, we first present the basic definition used in the rest of this paper.

Definition 1: (Abstract Service). Abstract service has function descriptions without implementation and standard service interfaces across different service providers. The user can directly name the requested distributed application, such as video-on-demand (e.g., video server ➔ computation ➔ storage ➔ content ➔ video display). An abstract service is corresponding to a directed acyclic graph (DAG) of workflow tasks.

Definition 2: (Service Instance). Service instances are concrete services published by service providers. They could give the function implementation specified by abstract services. And some service instances may have the same function, but different QoS of all candidate service instances, according to the abstract service path.

Definition 3: (Service Function Graph). Constructing abstract services as a workflow to fulfill user's requirement in functionality will obtain a service function graph. In the service function graph, there are service portals that serve as entrance/exit points.

Definition 4: (QoS aware Services Mashup). Transform the service function graph into a "candidate" graph and add "cost value" to the edges using an integrated metric which composes multiple QoS constraints (e.g. response time, availability) and resource requirements (e.g. CPU ratio and bandwidth ratio). The end user request speciation is represented by $\mathrm{REQ} = \{S_{req}, Q_{req}\}$, where $S_{req}$is a set of services which have to be traversed in a particular order and $Q_{req}$is a set of QoS constraints, the problem of QoS aware services mashup is to compose a service path $p \triangleq s_1 \to s_2 \to \cdots \to s_n$, from s_0(entrance portal) to s_{n+1}(exit portal), such that QoS constraints (equation (1) and (2)), i.e. $RT^{target} \le RT_p$, $A^{target} \le A_p$and also resource requirements (equation (3) and (4)) i.e. $CR_{s_i} \le 1$ and $R_{l_j} \le 1$. The QoS constraints and resource requirements can be defined as follows.

$$RT_p \triangleq \text{response time} = \sum_{i=0}^{n+1} RT_{s_i} \qquad (1)$$

$$A_p \triangleq \text{availability} = \prod_{i=0}^{n+1} A_{s_i} \qquad (2)$$

$$\text{where } A_s = \frac{\text{the amount of time that service is available}}{\text{total time monitored}}$$

$$CR_{s_i} \triangleq \text{CPU Ratio} = \frac{cpu_{s_i}^{required}}{cpu_{s_i}^{available}} \qquad (3)$$

$$\mathrm{BR}_{\mathrm{l_j}} \triangleq \text{Bandwidth Ratio} = \frac{\mathrm{BW}_{\mathrm{l_j}}^{\mathrm{required}}}{\mathrm{BW}_{\mathrm{l_j}}^{\mathrm{available}}} \qquad (4)$$

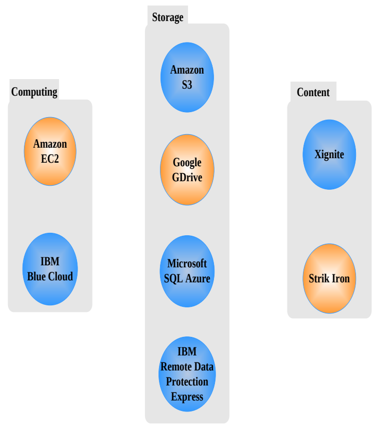
where $\mathrm{l_j}$ is the service link on the edge $\mathrm{l_j} = (\mathrm{s_i}, \mathrm{s_j}) \in \mathrm{E}, \mathrm{s_i}, \mathrm{s_j} \in \mathrm{V}$.

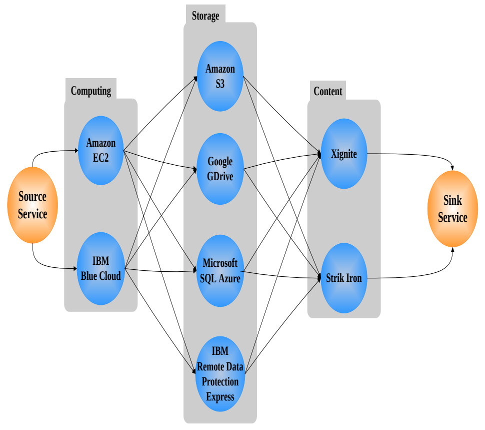
3.1. On-demand services mashup

This section describes the QASM model. First, we present a scenario about finance-on-demand and content retrieval application. Then, we introduce the QASM model for services mashup application.

The scenario consists of a service provider who wants to offer a service mashup that optimizes a given portfolio. This finance service is based on a bundle of sub-services retrieved from decentralized subservice providers (Figure 3(a)). Suppose there are several sub-service providers which are competitors for each type of service such as computing, storage and content. In this scenario, computing is provided by either Amazon EC2 or IBM Blue Cloud (Yu & Lin, 2004). Storage is provided by Amazon S3, Google G Drive, Microsoft SQL Azure, or IBM Remote Data Protection (Foster, Kesselman, Nick & Tuecke, 2004). Finance content can be obtained by either Xignite or Strik Iron which offer services that provide historical and real time finance data such as stock prices, currency rates or various indices. In order to provide the functionality of optimizing the end user's services portfolios, the final result generated by the service mashup is a QoS consistent service path satisfying the end user's QoS constraints and also has the minimum aggregated resource requirements.

The QASM model includes three mapping steps, illustrated by Figure 3 (a). For each end user request, the abstract services first maps(mapping-1) it to a composite service template, and then maps the template to an instantiated service path. Figure 3(b) start from the source service; check the QoS consistency between the current examined service instance and all of its predecessors on the service path. If the QoS constraints (equation (1) and (2)) predecessor satisfies the current examined service, then add a directed edge from the current examined service to the predecessor. We assume that the sink service is set as the end user's QoS requirements.





(a) Services instances for all required abstract services

(b) Add edges between two QoS consistent service instances

Figure 3. Illustration of on-demand finance service

The mapping from the user request to different composite service templates (mapping-1) is constrained by the user's application-specific quality requirements and different pervasive client devices, such as PDAs and cell-phones. If multiple QoS consistent service paths exist, the service composer should choose the one which has the minimum aggregated resource requirements so that the overall workload of a cloud computing system is minimized. In order to guarantee a successful service delivery, the resource requirements of the instantiated service path have to be satisfied. For simplicity, we only consider the CPU resource for end hosts and bandwidth for overlay links. In equation (3), the $\mathrm{cpu}_{s_i}^{\mathrm{required}}$ represents the required CPU resource for running a new s_i process. The $\mathrm{cpu}_{s_i}^{\mathrm{available}}$ r epresents the available CPUresource in the physical hosting environment of s_i.The smaller the CR_{s_i} ,the more advantageous we choose the service instance interms of CPU load balancing because we start the new service process on a lightly loaded host. Similarly, we define the term bandwidth ratio (Equation (4)) for the service link l_j.

The mapping from the composite service template to an instantiated service path (mapping-2) is constrained by QoS constraints (e.g., availability, response time) and resource availability conditions (e.g., computing, network transport).The "cost value" on the edge $l_j = (s_i, s_j) \in E, s_i, s_j \in V$. is defined using Equation (1) ~ (4). Intuitively, the ratio $\frac{w(p)}{R}$ (w represents any of the above attributes and R is the total value) represents the normalized cost of selecting a portion of the service path in terms of one specific factor (e.g., QoS constraints or resource requirements). For each link in the cloud computing system, we also use QoS constraints or resource conditions to represent it QoS attributes. For any path *p* from the entrance point to exit point, the following cost function is used to evaluate the feasible path:

$$C(p) \triangleq \lambda_1 \left(\frac{w_1(p)}{R_1}\right) + \lambda_2 \left(\frac{w_2(p)}{R_2}\right) + \cdots + \lambda_k \left(\frac{w_k(p)}{R_k}\right) \qquad (5)$$

Where $w_i(p)$ is the summation of *i*-th dimension QoS attribute along p, R_i is the total constraint of *i*-th dimension QoS value. $\lambda_i \in [0,1]$ denotes the weight of each QoS attribute which reflects their importance. The path with the minimum cost function value Min(C(p)) has the biggest possibility of being a feasible path. Run the Dijkstra-like algorithm to find the shortest path, which is returned as the result of the QoS aware services mashup, illustrated by Figure 4(thick line).

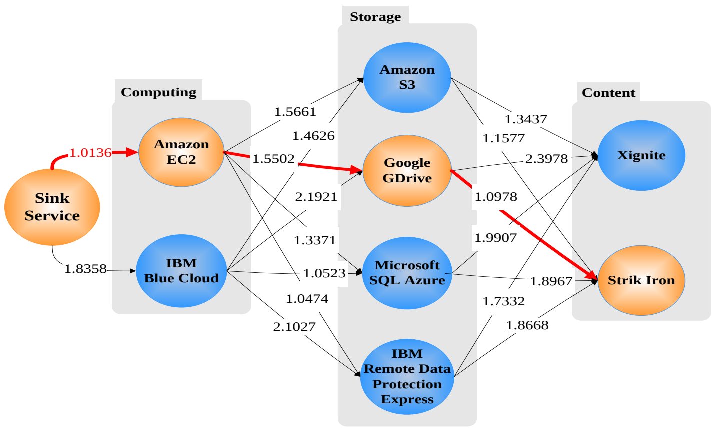


Figure 4. QASM generate the consistent resource shortest service path

Thus, the final result generated by the QASM model is a QoS consistent service path satisfying the end user's QoS constraints and also has the minimum aggregated resource requirements. The concrete computation process of our QASM algorithm is expressed in the following:

```
// initializations
For each node k in DAG do
C[j]=infinity; // Unknown distance from $s_i$ to j
optiPathPrev[j]=null; // Previous node in optimal path from $s_i$
End For
C[$s_i$]=0; // Distance from $s_i$ to $s_i$
Q= the set of all nodes in DAG; //initial no node in the optimal path
// seaching the nodes in the optimal path
While Q is not empty do
 u= the node in Q with the smallest C[];
if C[u]=infinity break; // all remaining nodes are inaccessible
                          from $s_i$
if u=$s_T$ break; // searching has reached the final node
remove u from Q;

//adjust the distance u to the remaining nodes in Q
   For each neighbor node v of u do
```

```
      CTemp=C[u]+C_between(u,v);
       if CTemp<C[v]
          C[v]=CTemp;
          optiPathPrev[v]=u;
       End if
   End For
End While

// reverse output the optimal path
k=$s_T$;
While optiPathPrev[k]<>$s_i$do
   print(optiPathPrev[k]);
   k=optiPathPrev[k];
End While
```

4. Performance Evaluation

In this section, we evaluate the performance of the QASM model by simulation. We first describe our evaluation methodology. Then we present and analyze the simulation results.

We simulated a large-scale cloud computing system of 104service providers. Each service provider is randomly assigned an initial resource attributes = [CPU; bandwidth; availability; response time], ranging from [200;100;0.97;0.05]to [800;1000;1;0.1] units. Different units reflect the heterogeneity in cloud computing systems. During each minute, certain number of end user requests are generated. The user request is represented by any of 40 composite service templates that comprise 2 to 6 services. The metrics we use for evaluating the QoS include the deviation rates of two QoS attributes: availability and response time, respectively. The QoS deviation rate is measured by the ratio of the requests during which QoS deviation happens over the total requests. For each request, the QoS deviation is said to happen if the measured average QoS attribute values (i.e., availability, response time) is worse than that specified in theA^{target} or RT^{target}. The other metric we use to evaluating the load balancing is the throughput. Throughput is defined as the number of setup sessions over total number of all requests. A request is setup if there are enough resources (i.e. CPU, bandwidth) for constructing the service path, ignoring its' QoS conditions. Higher throughput represents better load balancing in the cloud computing system.

Figure 5 and Figure 6 show the simulation results about the deviation rates of two QoS attributes: availability and response time, respectively. In Figure 5, the X axis represents different request rate, calculated by the number of service mashup requests per minute. The range of request rate is selected to reflect different system workload put on the cloud computing system. The Y axis shows the average QoS deviation rate for the availability attribute, achieved by the fixed, random and our QASM algorithm. The fixed algorithm always picks the same service path for a distributed application delivery and chooses the dedicated service providers to instantiate the service path. The fixed algorithm actually represents the conventional client-server systems. The random algorithm randomly chooses service instances to compose the service path.

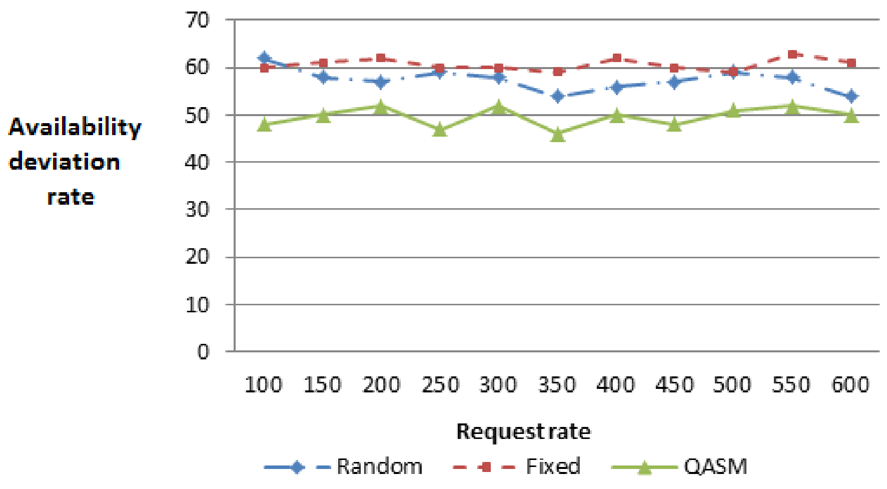


Figure 5. Average availability QoS deviation rate under different system load

In the even workload, we assume each service in the cloud computing system has equal probability being requested by the users. The results of QoS availability deviation rate show that the QASM algorithm achieved much lower than the fixed and random algorithms varying from 100 per minute to 600 per minute. As shown in Figure 5, QASM algorithm as the lowest QoS availability deviation rate, which is about 15% lower than that of random algorithm.

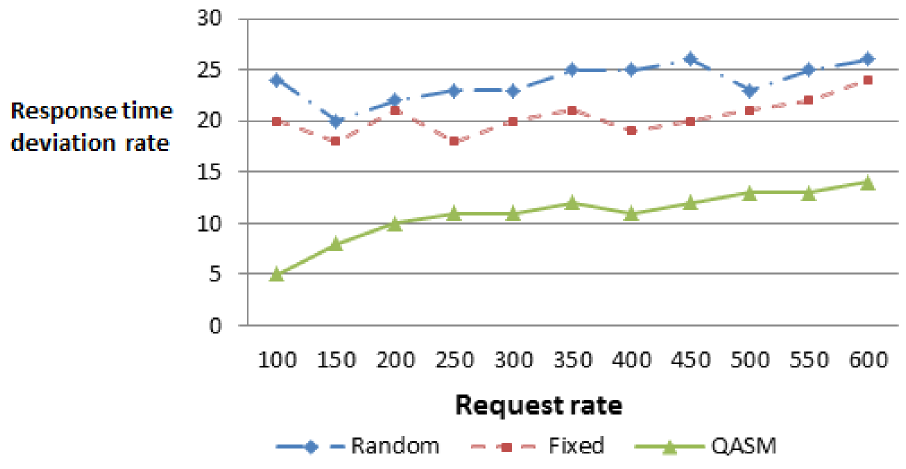


Figure 6. Average response time QoS deviation rate under different system load

Similarly, Figure 6 shows the results of the QoS deviation rate for the response time. Again, the

QASM algorithm achieved much lower response time than fixed and random. Both simulation results prove that QASM tolerates topological variation best and uniformly achieves the highest QoS.

Figure 7 shows the result about throughput. Throughput is a metric of load balancing. We can see from Figure 7 that random algorithm performs the worst in load balancing as it has the lowest throughput. We observe that the throughput of QASM is consistently higher than those of fixed and random. The former may be higher than the latter two as much as 10%and 45%, respectively.

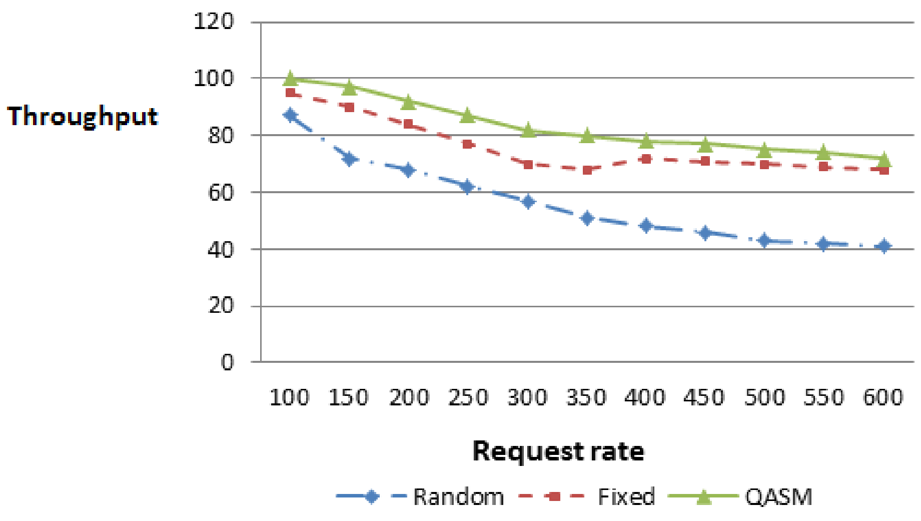


Figure 7. Throughput under different system load (load balancing)

In all of the above experiments, both simulation results prove that QASM tolerates topological variation best and uniformly achieves the highest throughput. The reason is that when QASM selects among candidate service providers for instantiating a service instance, it takes the service providers' average uptimes into account but random and fixed do not.

5. Conclusions

One of the challenging issues of cloud services mashup is how to find service paths and route the data flows through so that the resource requirements and QoS constraints of the applications are satisfied. In this paper, we present a QASM model for providing high performance distributed applications in the cloud computing systems. The major contributions of the paper are as follows: (1) solve two key problems, service composition and peer selection in an integrated service aggregation model; (2) present an on-demand service composition algorithm which generates a quality consistent service path with minimum aggregated resource requirements while satisfying the user's QoS requirements. We have implemented a simulation test and our initial simulation results illustrate the effectiveness of the proposed model and algorithm. In the future, we will implement a prototype of our model and test it in the real Internet environment. We will also investigate how to include other desirable system properties such as stability and fault tolerance into our model.

Acknowledgment

This research work was sponsored by the National Science Council, R.O.C., under project number NSC 101-2221-E-155 -032.

References

Berbner, R., Spahn, M., Repp, N., Heckmann, O., & Steinmetz, R. (2006). Heuristics for QoS-aware Web Service Composition, *Proceedings of the 2006 IEEE International Conference on Web Services* (ICWS06).

Buyya, R., Yeo, C.S., & Venugopal, S. (2008 Sept). Market Oriented Cloud Computing: Vision, Hype, and Reality for delivering IT Services as Computing Utilities. Keynote Paper, in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications* (HPCC 2008), IEEE CS Press, Los Alamitos, CA, USA, 25–27.

Broberg, J., Buyya, R., & Tari, Z. (2009). Metacdn: Harnessing Storage Clouds' for high performance content delivery. *Journal of Network and Computer Applications*, 32(5), 1012–1022. http://dx.doi.org/10.1016/j.jnca.2009.03.004

Cahon, S., Melab, N., & Talbi, E.G. (2004, May). Paradis EO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3), 357–380. http://dx.doi.org/10.1023/B:HEUR.0000026900.92269.ec

Chu, X., Nadiminti, K., Jin, C., Venugopal, S., & Buyya, R. (2007). Aneka: Next-Generation Enterprise Grid Platformfor e-Science and e-Business Applications. *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, IEEE Computer Society Press*, Los Alamitos, CS, 151–159.

Dijkstra, E. (1959). A note on two problems in connection with graphs, *Numerische Mathematik*, l.1 269-271. http://dx.doi.org/10.1007/BF01386390

Foster, I., Kesselman, C., Nick, J.M., & Tuecke, S. (2002). Grid service for distributed system integration, *IEEE Computer*, 35(6), 37-46. http://dx.doi.org/10.1109/MC.2002.1009167

Feng, G., Makki, K., Pissinou, N., & Doulgeris, C. (2002). Heuristic andExact Algorithms for QoS Routing with Multiple Constraints. *IEICETrans. Communications*, E85-B(12), 2838-2850.

Juttner, A., Szviatovszki, B., Mecs, I., & Rajko, Z. (2001). Lagrange relaxation based method for the QoS routing problem. *Proceedings of the IEEE INFOCOM*, 2, 859–868.

Korkmaz, T., & Krunz, M. (2001). A randomized algorithm for finding a path subject to multiple QoS constraints. *Computer Networks*, 36(2-3), 251-268. http://dx.doi.org/10.1016/S1389-1286(00)00209-7

Kirley, M., & Stewart, R. (2007). Multi-objective evolutionary algorithms on complex networks. *Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes Computer Science 4403*, Springer Berlin, Heidelberg, 81–95.

Wang, Z., & Crowcroft, J. (1996). Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14,1228-1234. http://dx.doi.org/10.1109/49.536364

Yu, T., & Lin, K. J. (2004). Service selection algorithms for web services with end-to-end QoS constraints. *IEEE International Conference onE-Commerce Technology (CEC'04),* 129–136.

Zeng, L., Benatallah, B., Ngu, A., et al. (2004). QoS-aware middleware for Web service composition. *IEEE Transactions on Software Engineering*, 5(30), 311–328. http://dx.doi.org/10.1109/TSE.2004.11