A Latency-aware Co-deployment Mechanism for Cloud-based Services

YU KANG[*], ZIBIN ZHENG[*], AND MICHAEL R. LYU[*,+]

[*]*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China*
[+]*School of Computer Science, National University of Defense Technology, Hunan, China*
{*ykang, zbzheng, lyu*}*@cse.cuhk.edu.hk*

***Abstract*—Cloud computing attracts considerable attention from both industry and academic these years. Nowadays, a number of research investigations have been conducted on cloud-based services (e.g., IaaS, PaaS, SaaS, etc.). Deployment of cloud-based services is one of the most important research problems. In cloud computing, multiple services tend to cooperate with each other to accomplish complicated tasks. Deploying these services independently may not lead to good overall performance, since there are a lot of interactions among different services. Making an optimal co-deployment of multiple services is critical for reducing latency of user requests. When deploying highly related services, taking only distribution of users into consideration is not enough, since the deployment of one service would affect others. To attack this challenge, we employ cross service information as well as user locations to build a new model in integer programming formulation. To reduce the computation time of the model, we purpose a sequential model running iteratively to obtain approximate solution. Extensive experiments have been conducted over a large real-world dataset, involving 307 distributed computers in about 40 countries, and 1881 real-world Internet-based services in about 60 countries. The experimental results show the effectiveness of our proposed model. Our real-world dataset is publicly released to promote future research, which also makes our experiments reproducible.**

***Keywords*-cloud; multi-service; deployment; integer programming;**

I. INTRODUCTION

As the coming of the cloud era, we have witnessed the rapid growing of cloud-based services. The tendency of cloud-based services is to deliver computation, software, and data access as services located in data centers distributed over the world [1], [2]. Typically, cloud provides three layers of services namely infrastructure as a service, platform as a service and software as a service (*i.e.*, IaaS, PaaS and SaaS). The cloud-based services are generally deployed on virtual machines (VM) in the data centers (*e.g.*, Amazon EC2). Since there are typically a large number of VMs in a cloud, making optimal deployment of cloud-based services to suitable VMs is an important research problem.

To provide good service performance for users, auto scaling and elastic load balance are widely studied . Currently, different cloud services are typically deployed independently by their own providers. Let's take Facebook and Google as examples. Facebook provides social network service in its own cloud; Google also provides email services and document services in its own cloud. These companies have their own users. They make optimal deployment of their services independently. Cloud-based services enjoy the feature of centralization which means the services are deployed in limited number of data centers and the network distance between the data centers can be measured periodically. This feature makes the deployment of cloud-based services different from deployment of traditional web services. In our previous work [3], a redeployment mechanism has been proposed to make optimal deployment for such kind of independent cloud services, considering the distribution of users and workload of servers.

However, in reality, different cloud-based services may cooperate with each other to complete complicated tasks. For example, when watching a video on YouTube, you can click the share button to share it to Facebook or Twitter by invoking the services provided by Facebook or Twitter; when editing a file on Google Doc you can call Gmail service to shard the document with your colleagues by email; when purchasing commodities on Taobao[1], you can call Alipay[2] service to pay the money (both Taobao and Alipay are affiliated entities of Alibaba Group). With the increasing number of various kinds of cloud-based services, composing multiple services to fulfill user requests becomes more and more common. Notice that multiple services are different from multiple service components in [4] by that a service provides complete function and can be accessed directly by a user while a service component does not.

Compared with making optimal deployment for a single service, making optimal deployment for multiple correlated services is a much more challenging research problem. The situation becomes very complicated when there are tens of services and their deployments affect each other. We could not simply treat the request from other services as the same as those from users since the service VMs of other services are under deployment at the same time and could be migrated to other places. Therefore it is critical to make a global decision for deploying multiple services together, especially for companies (*e.g.*, Google and Alibaba) which host many services or for companies that would like to

[1]http://www.taobao.com
[2]http://www.alipay.com

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.90

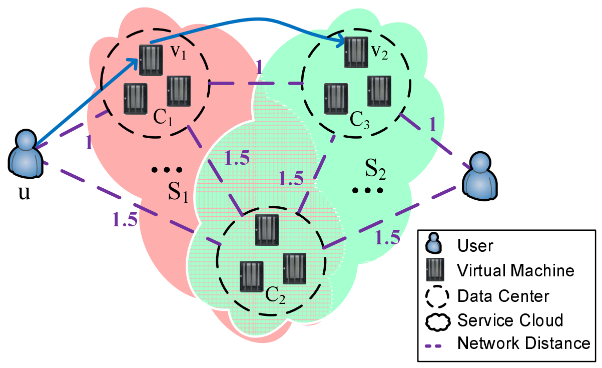

Figure 1. The Framework of Cloud-based Multi-service

cooperate with each other. We call this the co-deployment problem of cloud-based services.

This paper points out the new research problem of multi-service co-deployment in cloud computing and provides comprehensive investigations. For ease of discussion, we assume the cloud-based services are deployed on VMs. Replacing VMs with physical servers does not affect our model. We model the multi-service co-deployment problem formally as an integer programming problem which minimizes the latency of user requests. To evaluate the effectiveness of our proposed model, large-scale real-world experiments are conducted, involving 307 distributed computers in about 40 countries, and 1881 real-world Internet-based services in about 60 countries. The comprehensive experimental results show that our proposed latency-aware co-deployment mechanism can provide much better performance than traditional independent deployment techniques. To make our experiments reproducible, we publicly release[3] our reusable research dataset, which includes about 577,000 accesses of Internet-based services and 94,000 pings of computers over the world.

The rest of this paper is organized as follows. Section II introduces the framework of cloud-based multi-service and the data processing procedure. Section III gives the model of single service deployment. Section IV presents our multi-service co-deployment model. Section V discusses experimental results and Section VII concludes the paper.

II. Framework Cloud-based Multi-service

Figure 1 shows the general framework of cloud-based multi-service. As shown in the figure, multiple services (*e.g.*, S_1 and S_2) are to be deployed in different clouds. Suppose S_1 (*e.g.*, Google Doc service) and S_2 (*e.g.*, Gmail service) are correlated services (*i.e.*, one request of S_1 would involve S_2 and vice versa). There are three data centers. The network distances between these data centers are shown in the figure (the unit is second). Among these data centers, C_2 (*e.g.*, Amazon EC2) allows public access while C_1 and C_3 are private clouds (*e.g.*, provided by Google Doc and Gmail, respectively). In this example, S_1 can be deployed in C_1 and C_2 while S_2 can be deployed in C_2 and C_3.

[3]http://www.zibinzheng.com/cloud2012

Model 1 Single Cloud-based Service Deployment Model

$$\text{minimize} \sum_{\substack{i\in U\\ j\in C}} r_i d_{ij} x_{ij}$$

subject to:

$$\sum_{j\in C} x_{ij} = 1, \qquad \forall i \in U, \qquad (1)$$

$$x_{ij} \le y_j, \qquad \forall i \in U, \forall j \in C, \qquad (2)$$

$$\sum_{j\in C} y_j \le k, \qquad (3)$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in U, j \in C,$$

$$y_j \in \{0,1\}, \qquad \forall j \in C.$$

Network distances between users and data centers are marked in the figure. Firstly, we apply the traditional independent deployment technologies for each service, where the requests from end users and these from other service VMs are not distinguished. It is not hard to see that C_1 and C_3 would be chosen separately for S_1 and S_2. This deployment makes the average latency of requests from both users and other services to be $1s$. If we deploy S_1 (or S_2) in C_2, the average network distance of S_1 (or S_2) would increase to $1.5s$. The deployment decision seems optimal for both S_1 and S_2 when making independent deployment. However, if the providers of S_1 and S_2 cooperate with each other to seek a better deployment solution (e.g., development teams of Google Doc and Gmail try to making co-deployment of these two services together, since there are a lot of invocations between these services and these two teams belong to the same company), the objective becomes to reduce the overall request latency for the users of both services instead of independent services.

For example, a request of S_1 (S_2 similarly) is actually a calling sequence $user \rightarrow S_1 \rightarrow S_2$. To finish this sequence, the latency $2s$ consists two parts, namely $1s$ from user to service VM of S_1 in C_1 and $1s$ from service VM of S_1 in C_1 to service VM of S_2 in C_3. If the deployment problem would be considered globally, deploying both services in C_2 would be a better choice. Therefore the overall request latency of the calling sequence would be reduced to $1.5s$.

III. Independent Deployment of Single Service

First we consider a simple case of deploying a single cloud-based service. Suppose a service is to be deployed on k VMs with that we have obtained the distances between

Table I
ALPHABET OF THE MULTIPLE CLOUD-BASED SERVICES CO-DEPLOYMENT MODEL

Notation	Descriptions
U	Set of users
m	Number of services
C_h	Candidate VMs set of service h
k_h	Number of VMs service h could deploy
r_{hi}	Number of requests for service h from user i
d_{hij}	Distance between user i and j-th VM of service h
r_{iqs}	Number of times that service q is involved in the requests for service s from user i
d_{pqrs}	Distance between p-th VM of service q and r-th VM of service s
x_{hij}	(Indicator) whether user i would request j-th VM for the service h
y_{ipqrs}	(Indicator) for user i whether r-th VM of service s is chosen to respond the requests from p-th VM of service q
z_{hj}	(Indicator) whether j-th VM of service h is chosen

users and service VMs. We apply the p-median model [5] in the integer programming formulation as Model 1.

In Model 1, U and C are the set of users and potential service VMs. r_i is the count of the requests from user i. d_{ij} is the network distance between user i and j-th candidate service VM. x_{ij} and y_j are decision variables. y_j indicates whether j-th service VM is chosen to deploy the service. x_{ij} indicates whether user i would request j-th service VM. This model aims at selecting a set of service VMs to deploy service ($y_j = 1$ if selected). Users i could connect to the closest available service VM. The objective function minimizes the total distance of all requests. Constraint 1 ensures the user would connect to one service VM to fulfill the requests. Constraint 2 ensures the requests of the users would only be processed on the selected service VM. Constraint 3 ensures at most k service VMs are chosen.

The model is in integer programming formulation. We can apply typical techniques to solve the problem very efficiently with an acceptable approximate solution.

IV. CO-DEPLOYMENT OF MULTI-SERVICE

After introducing the single cloud-based service deployment, we move on to the more complicated problem of deploying multiple services. We extend the model of deploying single service. An integer programming model is proposed and heuristic approach is provided to solve the problem. The notations in the model are given in Table I.

A. Multiple Cloud-based Services Co-deployment Model

As we have introduced the motivation of deploying multiple services simultaneously, an all-in-one model is designed. Model 2 optimizes the latencies of both user requests and cross-service requests.

Model 2 decides the co-deployment of service VMs in a candidate set for all services h (z_{hj}=1 if the VM is selected). Users i could connect to the closest available service VM for service h ($x_{hij} = 1$ if connected). The objective function aims at minimizing the total distance of all requests (including requests from all users and other services). While the set of service users and the frequency of user accesses may change over time, we could periodically calculate the model to do some updates. There could be different execution paths for the combination of multiple services. We decoupled the execution path by a sequence of service calls and add it up in r_{iqs}. Therefore all the paths are considered in the model.

The Constraints 4 to 6 are similar to those in single cloud-based service deployment model. The function $sign(x)$ in Equation 4 returns 1 for $x > 0$ and 0 for $x = 0$. Equation 4 constrains user i would connect to one service VM to fulfill the requests if the user has a request of service h. Inequality 5 ensures the requests of the users would only be processed on the selected service VM. Constraint 6 ensures at most k_h service VMs are chosen for service h. The Constraints 7 to 9 are introduced for cross-service requests. Inequality 7 means for user i the cross-service requests from service h to s should be initiated from the j-th VM of service h, which is chosen to serve user i for requests of service h. Inequality 8 constrains the cross-service requests could only be sent to selected service VMs. Constraint 9 ensures one link would be set if and only if there are cross-service requests.

B. Iterative Sequential Co-deployment Algorithm

We have nicely acquired an all-in-one co-deployment model in Section IV-A, whereas, there are too many decision variables. It is infeasible to apply general methods to obtain an approximate solution for the model. Therefore we take the special properties of this problem and get a heuristic approach. Algorithm 1 shows the approach.

Algorithm 1 first generates a random deployment (Line 3 to 6) as there is no information about where to deploy. After deciding a temporary deployment, the algorithm tries to sequentially improve the deployment of each service one by one (Line 11 to 14). It treats the requests from other services the same as these from the users in Line 12. By iteratively doing the improvement procedure in Line 9 to 20, the deployment of all services would converge. The best deployment of the iteration is not necessary appeared at last, so we record the best ever deployment in Line 16 to 19. Since the iterative sub-procedure would fall into a local minimum, we disturb the solution (usually change one or two chosen service VMs) and run n loops to find a good enough co-deployment.

It is worth mentioning that this iterative computing algorithm is different from doing unsupervised single service

Model 2 Cloud-based Multi-service Co-deployment Model

$$\text{minimize} \sum_{\substack{i\in U\\ 1\le h\le m\\ j\in C_h}} r_{hi}d_{hij}x_{hij} + \sum_{\substack{i\in U\\ 1\le q,s\le m, q\neq s\\ p\in C_q, r\in C_s}} r_{iqs}d_{pqrs}y_{ipqrs}$$

subject to:

$$\sum_{j\in C_h} x_{hij} = sign(r_{hi}), \quad 1\le h\le m, \forall i\in U, \tag{4}$$

$$x_{hij} \le z_{hj}, \quad 1\le h\le m, \forall j\in C_h, \forall i\in U, \tag{5}$$

$$\sum_{j\in C_k} z_{hj} \le k_h, \quad 1\le h\le m, \tag{6}$$

$$\sum_{\substack{1\le s\le m\\ s\neq h\\ r\in C_s}} y_{ijhrs} \le x_{hij}, \quad 1\le h\le m, \forall j\in C_h, \forall i\in U, \tag{7}$$

$$y_{ipqrs} \le z_{rs}, \quad 1\le q,s\le m, q\neq s, \forall p\in C_q, \forall r\in C_s, \forall i\in U, \tag{8}$$

$$\sum_{\substack{p\in C_q\\ r\in C_s}} y_{ipqrs} = sign(r_{iqs}), \quad 1\le q,s\le m, q\neq s, \forall i\in U, \tag{9}$$

$$x_{hij}\in\{0,1\}, \quad 1\le h\le m, j\le C_h, \forall i\in U,$$

$$y_{ipqrs}\in\{0,1\}, \quad 1\le q,s\le m, q\neq s, \forall p\in C_q, \forall r\in C_s,$$

$$z_{hj}\in\{0,1\}, \quad 1\le h\le m, \forall j\in C_h.$$

deployment. The goal of this algorithm is to acquire a feasible suboptimal solution for Model 2. It is true that if we does not participate the deploying schedule of all the services, after some time these services would also evolve to an overall balanced suboptimal deployment. We call this a nature evolution approach. There are several disadvantage of nature evolution approach compared to our algorithm. We list three of them.

1) For the nature evolution approach, we have to wait for quite a long time until the deployment performing well globally since redeployment of services would not be done frequently. On contrary, our algorithm can compute the result in a short time.
2) The nature evolution approach would incur many real migrations of service VMs which costs a lot. The migration of one service could cause the migrations of some other services and a chain reaction continues, but the order is random and hard to predict. Our algorithm exploits the changes in a more systematic way and does not trigger any real migrations.
3) Last but the most important reason that our algorithm outperforms the nature evolution approach is we are not necessary to fall into local optimum. We disturb the result in Line 21 and repeat our iterative algorithm to jump out the local minimum, while the nature evolution approach has no choice but to be stuck in the local optimum.

Algorithm 1 Iterative sequential co-deployment algorithm

```
1: $tempS \leftarrow \phi, S \leftarrow \phi$
2: $temp \leftarrow MAX, tempmin \leftarrow MAX, min \leftarrow MAX$
3: for all service $h$ do
4:     Select a set $S_h$ of $k_h$ service VMs randomly among the candidate set $C_h$
5:     $tempS \leftarrow tempS + S_h$
6: end for
7: for $i = 1 \rightarrow n$ do
8:     $tempmin \leftarrow$ Evaluate the solution $tempS$
9:     repeat
10:        $temp \leftarrow tempmin$
11:        for all service $h$ do
12:            Select a set $S'_h$ of $k_h$ service VMs according to Model 1 with decided $tempS$
13:            $tempS \leftarrow tempS - S_h + S'_h$
14:        end for
15:        $tempmin \leftarrow$ Evaluate the solution $tempS$
16:        if $tempmin < min$ then
17:            $min \leftarrow tempmin$
18:            $S \leftarrow tempS$
19:        end if
20:    until $|tempmin - temp| \le \epsilon$
21:    Disturb the solution set $tempS$
22: end for
23: return S
```

V. Experiment and Discussion

We have obtained a large real-world latency data set. Based on the data set we conduct the experiment to evaluate methods of deploying multiple cloud-based services. We compare our algorithm to the independent deployment method. The independent deployment method deploys the service VMs randomly at the beginning. After the logs have been collected, all services redeploy the VMs applying single service deployment model without differentiate requests from users and from other services.

Ilog Cplex 9.0 is applied to give a good enough approximate solution of the integer programming problems.

A. Latency Data Collection

We use a similar assumption as in [3]. Generally, latency of a service request (the request either from user or another service) contains three parts: the Internet delay between the request sponsor and the gateway of cloud data center, the

Intranet delay inside the cloud, and the computing time on the service VM. While inside a data center, usually gigabyte switches are used, the Intranet delay can be ignored. Moreover, assuming two service VMs in the same data center have the same computational capability. Thus their processing time of a service can be viewed as the same. Therefore, the dominant part of the latency is the Internet delay. Under this assumption there is no difference between two VMs in the same data center. We use *distance* between a user u and a data center C to denote the latency between u and *any* VM v in C. Similarly, *distance* between a pair of data centers C_i and C_j represents the latency between *any* pair of VMs v_i and v_j in C_i and C_j separately.

Since the service VMs are running in the cloud, the service providers can collect the latency data of fulfilling every user requests. The calling sequences of services are recorded as well. One log entry contains the user ID, the service ID it calls, a sequence of service IDs involved and a sequence of latencies between every two calling.

For any user u, if u ever requested a service VM v in data center C, the recorded latency between (u, v) is used to represent the distance between u and C. When more than one VM in C is requested by u, we take the average latency. Distance between two pair of data centers is retrieved by the same way. If there exists a latency record from a service VM v_i in data center C_i to v_j in C_j, this value is taken as the distance between C_i and C_j.

We use again the example in Figure 1 to illustrate this. u requests v_1 followed by v_1 requests v_2. The call sequence is recorded together with the latency between the pair (u, v_1) and (v_1, v_2). These two values are regarded as distances between pair (u, C_1) and (C_1, C_2).

After this period of data processing, we obtain two distance matrices. One matrix records the distances between every $(user, datacenter)$ pair. Another matrix records the distances between every pair of data centers.

B. Dataset Description

To obtain real-world response time values of different users and service VMs, we implement a Crawler using Java. Employing our Crawler, we got the addresses of 1,881 openly-accessible services. We deploy shell script codes to 307 distributed computers of PlanetLab, which is a distributed test-bed with hundreds of computers all over the world. During the experiment, each PlanetLab computer pings all the Internet services as well as all the other PlanetLab computers once and records the round trip time (RTT). The ping operations are done simultaneously in a randomized way to avoid overflow on one computer. By this real-world evaluation, we obtain two matrices and publicly release them[4]. One is a 307×1881 matrix containing RTT values between PlanetLab computers and web services (P2W Matrix). The other one is a 307×307 matrix with RTT values between every pair of PlanetLab computers inside (P2P Matrix). Some statistic results of these two matrices are shown in Table II. These two matrices are used as the two matrices described in previous section. All the service VMs are represented by the containing data center in the experiment.

[4]http://www.zibinzheng.com/cloud2012

Table II
DATASET STATISTICS (UNIT: MS)

Statistics	P2W Matrix	P2P Matrix
minimum	0.01	0.08
25th percentile	53.69	58.58
median	118.14	133.76
75th percentile	176.00	188.53
maximum	1604.02	5704.04
average	129.41	138.38

Table III
PARAMETERS USED IN RANDOMIZED LOG GENERATION

Notation	Descriptions	Default
n_h	Number of log entries of service h	1880
ρ_h	Ratio of users use service h to total users	0.2
l	Number of services involved in one service request	5

C. Experiment Parameters

The experiment is conducted on the data set we obtained with randomly generated user logs. The iterative sequential algorithm for solving Model 2 is evaluated. If it is not specially explained the parameters of the model in Table I are set to default values. By the default setting, there are 1881 users, 10 services. Every service would deploy 10 service VMs among a candidate set in 100 data centers. A user of service s would have 5 request logs. One request of a service would involve on average 5 requests of other services. The parameters without default value r_{iqs} together with parameter r_{hi} depend on user logs. We generate user logs randomly. Table III states several parameters related to the randomized log generation.

D. Algorithm Specifics

1) Convergence of Iterative Sequential Procedure: There is a sequential procedure in Algorithm 1 (Line 11 to 14) that does the service deployment one by one. As shown in Figure 2, the procedure would soon converge after 5 iterations. So it is quite safe to limit the iteration number and make the algorithm run faster.

2) Number of Disturbs: In case falling into local minimum, we disturb the result and run several loops to get a better result in Algorithm 1. Figure 3 shows the effective of adding disturbs. The latency value of disturb 0 is the

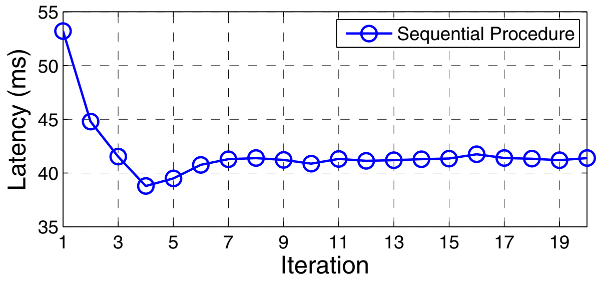

Figure 2. Convergence of Sequential Procedure

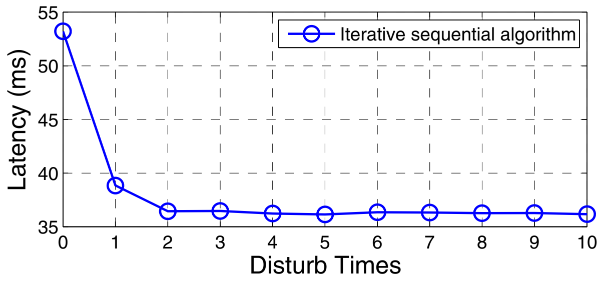

Figure 3. Number of Disturbs

average latency of random deployment. We could see as the figure shows the biggest improvement is made on the first iteration. The disturbing would let the algorithm jump out the local minimum. The result of first several iterations is good enough that less consequent improvements are made.

E. Number of Services

We change the total number of services and conduct the experiment. The result is shown in Figure 4. We can see our algorithm has the improvement about 10% to 20%. Since we set the number of services involved in one request to be the default value, the average latency of one request does not vary a lot. The variations of two curves are caused by the random generation of the candidate set of service VMs of different services in different service number setting.

An observation is that the improvement of our algorithm decreases for big service numbers. It is because with the

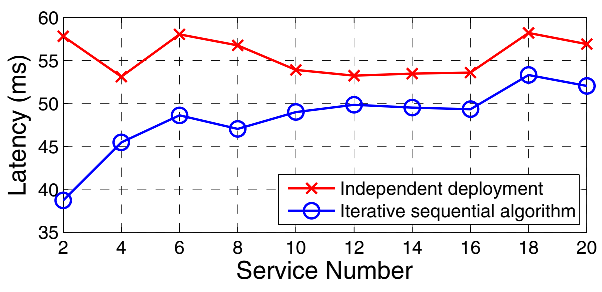

Figure 4. Number of Services

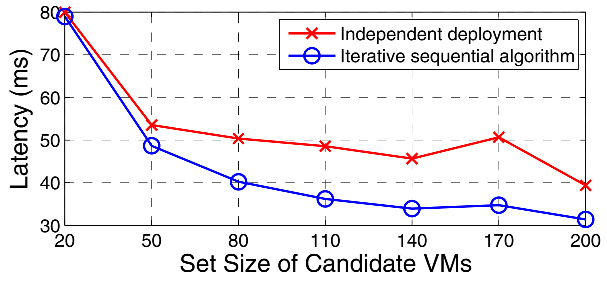

Figure 5. Set Size of Candidate VMs

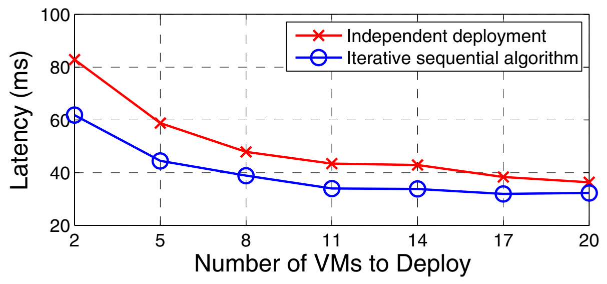

Figure 6. Number of Service VMs to Deploy

growing number of services there are more service VMs selected. Therefore more information is known when deploying one service. Every service could choose overall better service VMs for many users and other service VMs independently. Thus the improvement of our algorithm decreases. While it is more common situation that there are not many services, our algorithm is effective.

F. Number of Service VMs

1) Size of Candidate Set of Service VMs: We set different size of candidate service VM set. The result is shown in Figure 5. There is a decreasing tendency of the curve. Since there are more potential service VMs to choose from the result should be better. The abnormal increasing on set size 170 is caused by randomized initial deployment phase. It is worth mention that our algorithm has greater improvement with bigger set size. The reason is with more candidates to choose from our algorithm could do a much better decision than considering the deployment independently.

2) Number of Service VMs to Deploy: Figure 6 is generated by modifying the number of total service VMs of every service. We can see our algorithm outperforms the independent deployment method by at most 25 percentages. Especially in smaller number of VMs to deploy, we can make a much better decision. If we can deploy many service VMs, independently decision could find several outstanding VMs that perform well. Therefore in these cases our algorithm does not win too many advantages.

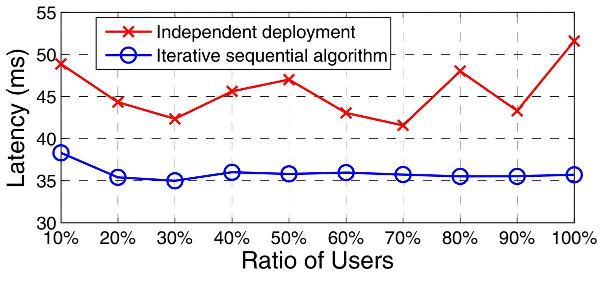

Figure 7. Number of Service Users

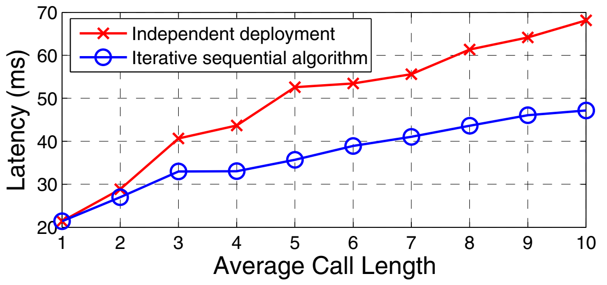

Figure 8. Average Call Length

G. *Services Logs*

We tried different log generation parameters and see the behavior of our algorithm under different use cases.

1) Number of Service Users: In Figure 7 we change the number of users. There are totally 1881 sample users and we change the ratio of users to use one service. For example 10% means we have logs of 188 users that use a specific service. The result shows that the performance of independent deployment varies a lot due to randomization, while our algorithm is quite stable under all situations. Moreover, our algorithm reduces the average latency up to 30%.

2) Average Call Length of Service: We define the call length as the number of service requests involved in one user task. For example, a call length of 5 means a user request of a specific service s would involve 4 consequent requests of other services from the VM of service s. Our algorithm outperforms the independent deployment method especially when the service call length is big. It can also be seen that the increasing curve of our algorithm is more stable.

3) Number of Logs: We modify the number of total logs on randomized generation. The result is shown in Figure 9. Average usage of a user means the average number of requests of a user for a specific service. We can expect the more records of users the more intelligent we can deploy the service. As we can see in the figure, the average latency decreases with more user logs. An observation is that the gap between our algorithm and independent deployment method becomes narrower with the growing number of the

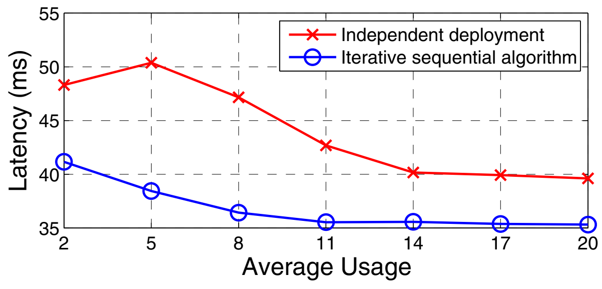

Figure 9. Average Usage

user logs. This could be explained by that there are many user logs providing enough information even for deploying service independently. We can see the tendency of two curves. Our algorithm converges when there are more than 10 average usages, while independent deployment method converges after 14. Therefore our algorithm could do a better deployment when the number of logs is not sufficient.

VI. Related Work

A. *Cloud-Based Service Deployment*

Web server placement is widely studied previously.

Web server replicas placement is studied by Qiu et al. [6]. They use the result of *p-median* model as super-optimal result. The work focuses on using different algorithms to give an approximate solution of the model.

Zhang et al. [7] studied service placement in shared service hosting infrastructures. They formulated a model similar to the general capacitated facility location model. They did not use response time directly but define a new penalty cost.

However, the previous studies are not specially tailored for cloud computing. In the work of Kang et al. [3], the properties of cloud computing are considered. The user experience is highlighted and a general framework of service deployment in cloud environment is proposed.

While these works focus on single service deployment, we proposed the model for multiple services co-deployment.

B. *p-Median Model and Multi-commodity Facility Location*

A series of facility location problems have been well studied in supply chain management field. Melo et al. [8] provide an extensive review. One discrete variation of facility location problem called *p-median* attracts many interests in areas such as web service deployment [6]. Consequently improvements have been done to approximate the solution [9], [10], [11]. Arya et al. [11] provide the currently best known $3+\varepsilon$ approximation of the problem.

The original *p-median* model considered the facility location problem for only one commodity. Our model is more similar to multi-commodity facility location problem.

Pirkul et al. [12] proposed PLANWAR model which is a long existing formulation to the multi-commodity, multi-plant, capacitated facility location problem. Shen [13] modified the cost function and obtain a new model. Cao et al. [14] proposed a variation of *p-median* model for the problem. However, these models do not consider cross-plant transportation. Instead the commodities are regarded as rather independent.

Thanh et al. [15] proposed a very complex dynamic model with about 40 constraints. The model considers a multi-echelon, multi-commodity production-distribution network with deterministic demands. They take the assumption that the production process can be divided into several steps and can be shared between several plants. The production process does not rely on other productions or subroutines. The relation of two commodities is that they can be manufactured/stored in one facility simultaneously. The multi-echelon is divided according to the life cycle of a commodity but not cross-commodity. Therefore this model is quite different from our model also.

VII. Conclusion and Future Work

In this paper, we investigate the latency-aware cloud-based multiple services co-deployment problem. Motivating examples are given to illustrate the problem. We offer the method to show what and how the user logs could be collected. The co-deployment problem is abstracted in a general framework. Integer programming formulation is employed to model the problem and a new heuristic algorithm is given to obtain an approximate solution. Extensive experiments are conducted on hundreds of computers over world. The experiment results show the effectiveness of our model.

In our future work, we would add more constraints to the model. For example, our model in this paper does not limit the number of requests to one service VM. We could add a threshold of requests. The computing time is regarded as constant in this model. Actually the computational capability is related to the workload of the service VM, we would further study the characteristics of VMs and find a function which fits the computing time of VM over different workload. This function could be added as an item in the model to punish users sending requests to heavy load VMs. Thus a better load balance could be maintained.

Acknowledgment

The work described in this paper was fully supported by the Research Grants Council of the Hong Kong, China (Project No. CUHK 415311), the NSFC/RGC Joint Research Scheme sponsored by the Research Grants Council of Hong Kong and the National Natural Science Foundation of China (Project No. N_CUHK405/11), the National Natural Science Foundation of China under Grant No. 61100078, and the National Basic Research Program of China (973) under Grant No. 2011CB302603.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *ACM Communication*, vol. 53, no. 4, pp. 50–58, 2010.

[2] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Proc. of the 1st International Conference on Cloud Computing (CloudCom'09)*, 2009, pp. 626–631.

[3] Y. Kang, Y. Zhou, Z. Zheng, and M. Lyu, "A user experience-based cloud service redeployment mechanism," in *Proc. of the 4th International Conference on Cloud Computing (CLOUD'11)*, july 2011, pp. 227 –234.

[4] Z. Zheng, Y. Zhang, and M. R. Lyu, "CloudRank: A QoS-driven component ranking framework for cloud computing," in *Proc. of the 29th International Symposium Reliable Distributed Systems (SRDS'10)*, 2010, pp. 184–193.

[5] D. M., "Network and discrete location: Models, algorithms and applications," *Journal of the Operational Research Society*, vol. 48, no. 7, pp. 763–763, 1997.

[6] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *Proc. of the 20th IEEE International Conference on Computer Communications (INFOCOM'01)*, vol. 3, 2001, pp. 1587–1596.

[7] Q. Zhang, J. Xiao, E. Grses, M. Karsten, and R. Boutaba, "Dynamic service placement in shared service hosting infrastructures," in *Proc. of the 9th International IFIP TC6 Networking Conference (NETWORKING'10)*, vol. 6091, 2010, pp. 251–264.

[8] M. Melo, S. Nickel, and F. S. da Gama, "Facility location and supply chain management - a review," *European Journal of Operational Research*, vol. 196, no. 2, pp. 401 – 412, 2009.

[9] M. Charikar and S. Guha, "Improved combinatorial algorithms for the facility location and k-median problems," in *Proc. of the 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, vol. 0, 1999, p. 378.

[10] K. Jain, M. Mahdian, and A. Saberi, "A new greedy approach for facility location problems," in *Proc. of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, 2002, pp. 731–740.

[11] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristic for k-median and facility location problems," in *Proc. of the 33rd Annual ACM Symposium on Theory of Computing (SOTC'01)*, 2001, pp. 21–29.

[12] H. Pirkul and V. Jayaraman, "A multi-commodity, multi-plant, capacitated facility location problem: formulation and efficient heuristic solution," *Computers and Operations Research*, vol. 25, pp. 869–878, October 1998.

[13] Z.-J. M. Shen, "A multi-commodity supply chain design problem," *IIE Transactions*, vol. 37, no. 8, pp. 753–762, 2005.

[14] B. Cao and G. Uebe, "An algorithm for solving capacitated multicommodity p-median transportation problems," *The Journal of the Operational Research Society*, vol. 44, no. 3, pp. 259–269, 1993.

[15] P. N. Thanh, N. Bostel, and O. Péton, "A dynamic model for facility location in the design of complex supply chains," *International Journal of Production Economics*, vol. 113, no. 2, pp. 678 – 693, 2008.