2013 IEEE Seventh International Symposium on Service-Oriented System Engineering

Scaling Service-oriented Applications into Geo-distributed Clouds

Jieming Zhu*†, Zibin Zheng*†, Yangfan Zhou†, and Michael R. Lyu*†‡

*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong
†Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
‡School of Computer Science, National University of Defence Technology, Changsha, China
{jmzhu, zbzheng, yfzhou, lyu}@cse.cuhk.edu.hk

***Abstract*—With the significant prevalence of cloud computing, more and more data centers are built to host and deliver various online services. However, a key challenge faced by service providers is how to scale their applications into geo-distributed data centers to improve application performance as well as minimizing the operational cost. While most existing deployment methods ignore the service dependencies in an application, this paper proposes a general dynamic service deployment framework to bridge this gap, in which a deployment manager and a local scheduler are designed to optimize data center selection and auto-scale the service instances in each data center respectively. More specifically, we formulate the deployment problem across multiple data centers as a compact minimization model, which can be solved efficiently by a genetic algorithm. To evaluate the performance of our approach, extensive experiments are conducted based on a large-scale real-world latency dataset. The experimental results show that our approach substantially outperforms the other existing methods.**

***Keywords*—Dynamic deployment; service-oriented application; geo-distributed clouds; genetic algorithm**

I. Introduction

The cloud computing paradigm has recently gained much popularity for provisioning a pool of computational resources to host and deliver various large-scale online services over the Internet, including search engine, e-commerce, social network, file hosting service, and so on. With the prevalence and benefit of cloud computing, many cloud providers like Amazon, Google and Microsoft have built large data centers in geographically distributed locations to achieve reliability and serve millions of users world-wide. For example, Amazon EC2[1] nowadays provisions cloud services over nine geographically dispersed regions, where service providers have options to deploy their applications in data centers from Virginia, Oregon, California, Ireland, Singapore, Tokyo, Sydney, São Paulo and GovCloud. Moreover, as the ideas of InterCloud and cloud federation [1], [2] become mature, more and more geo-distrbuted data centers will be cooperatively utilized for the purpose of operational cost minimization, traffic load balancing, demand spikes accommodation and catastrophic recovery.

In this context, a key challenge faced by service providers is how to scale their applications across these geographically distributed data centers. That is how to optimize the deployment strategies to take full advantage of the geo-diversity to achieve better performance and minimize the operational cost when serving globally dispersed users. In addition to the conventional resource constraints (*e.g.*, resource capacity, CPU and memory requirements of virtual machines) and on-demand resource assignment of service deployment in a single data center, the dynamic pricing [3] in different data centers and non-negligible time-varying communication latencies between data centers are also needed to take into consideration when deploying applications across multiple data centers. Moreover, the dynamic service demand and geographical distribution of end users [4] further increase the difficulty of this task. As a result, there is an increasingly urgent need for a dynamic and adaptive deployment strategy to scale cloud applications into geo-distributed clouds.

On the other hand, with the attractive features (*e.g.*, high re-usability, fast development and reduced cost) of service-oriented architecture (SOA), many online applications delivered in the cloud are currently designed in the SOA style (namely service-oriented applications), which consist of a number of service components and data components, as well as a workflow. Individual service components and data components are composed together to form a higher-level functionality, where the interaction relationship between these components are defined by the workflow. As an example, Fujitsu, as a provider of ICT-based business solutions for the global market, describes their approach to SaaS (Software-as-a-Service) in [5], which envisions to provision service-oriented online applications not only from single data centers but across multiple geo-distributed data centers, so as to extend functions and services with global expansion.

However, the data sharing and interdependency between services or even between applications will pose new challenges in deploying service-oriented applications into geo-distributed data centers. The deployment should be aware of these dependencies, since the deployment strategy of each service will directly impact the performance (*e.g.*, response time) of the application. In recent literature, although a few work have been conducted to address the challenge of service deployment across multiple data centers, most of them (*e.g.*, [3], [6], [7], [8], [9]) have not taken the service dependencies into consideration, where single services are independently deployed to optimize the corresponding performance and operational cost. Some work (*e.g.*, [10], [11], [12]) has investigated the service dependencies of the SOA application. However, the deployment problem across geo-distributed clouds has not

[1]http://aws.amazon.com/ec2/

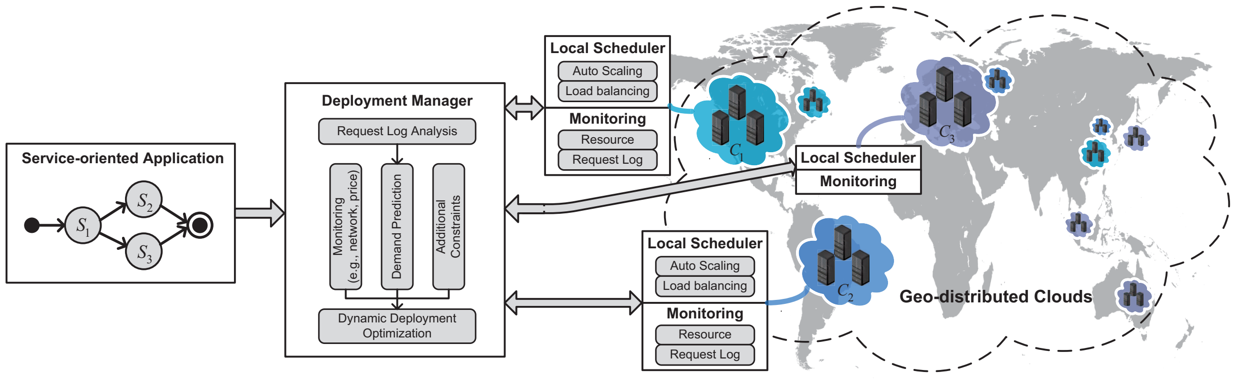

Fig. 1. The Framework of Service-oriented Applications Deployment in Geo-distributed Clouds

yet been addressed. In [13], a service co-deployment model based on integer programming has been proposed to optimize the deployment strategy with potential service dependencies. However, the integer programming based approach suffers from poor scalability, as its complexity grows exponentially with the number of services and candidate data centers.

To tackle these challenges, in this paper, we propose a dynamic service deployment framework to cope with the service-oriented application deployment across geo-distributed clouds. In this framework, we answer two questions on service deployment strategy optimization: 1) *Which data centers should be selected for each service deployment?* 2) *How many service instances should be replicated for each service in a data center?* In particular, we focus on the data center selection problem for each service that takes service dependencies into account, which is formulated as an optimization problem. While the original model is a NP-hard problem, we tailor the genetic algorithm to solve it, which provides a good trade-off between the running time and the quality of the result. To evaluate the performance of our proposed deployment model, we conduct extensive experiments based on a large-scale real-world latency dataset, which includes 307 geo-distributed Planetlab[2] nodes across about 40 countries and 1,881 public Web services in about 60 countries. The experimental results show the effectiveness and efficiency of our model, which substantially outperforms other existing approaches.

In summary, our main contributions are three-fold:

- First, we propose a general framework to address the problem of dynamic service-oriented application deployment in geo-distributed clouds, which optimizes the application performance while keeping minimal operational cost.
- Second, this paper exploits service dependencies to optimize the deployment strategy across a set of candidate data centers. A tailored genetic algorithm is employed to solve our model efficiently.
- Finally, extensive experiments are conducted based on a large-scale real-world latency dataset to evaluate the effectiveness and efficiency of our proposed deployment strategy.

[2]http://www.planet-lab.org

Paper Organization. Section II presents our proposed dynamic deployment framework of service-oriented applications across geo-distributed clouds. Section III formulates the deployment problem and Section IV describes the detailed genetic algorithm to solve this problem. The experimental results are reported in Section V. We discuss the related work in Section VI and finally conclude this paper in Section VII.

II. System Architecture

To address the challenge of scaling service-oriented applications into geo-distributed clouds, we propose a general dynamic deployment framework, as illustrated in Fig. 1, to periodically optimize the deployment strategies of services while taking service dependencies into consideration. The process to deploy service-oriented applications across geo-distributed data centers typically involves the following two phases: 1) deciding the deployment strategy for each service component, *i.e.*, selecting data centers to deploy each service components; and 2) automatically deciding the number of service instances for each service running in the data center. In this paper, we propose a two-level management framework to solve these problems, including a deployment manager in data center level and a local scheduler in service instance level (either physical server or virtual machine). In the high level (*i.e.*, the data center level), the deployment manager is employed to analyze the service dependencies and predict the request demand to decide the number and locations of each service across data centers. In the low level (*i.e.*, the service instance level), the local scheduler is used to automatically scale the service instances according to the workload assigned to this data center.

1) **Deployment Manager:** The deployment manager is a key component in our framework to determine the locations of each service in the geo-distributed clouds. Generally, the service provider makes an initial deployment and then iteratively improve the deployment. Towards this end, the request logs are collected and analysed to capture the user demand, and the network latencies can be measured periodically to adapt to the network dynamics. As such, the service deployment can be optimized periodically to improve the application performance, while service dependencies are considered in addition to the

network condition, cloud service prices, and other additional constraints (*e.g.*, some data components are required to be placed in certain data centers for privacy concern). It is worth mentioning that there is a large body of work demonstrating the effectiveness of network coordinate systems for network performance prediction, which can also be incorporated into our framework to reduce the measurement overhead.

2) **Local Scheduler:** After deploying the services into the selected data centers, each local scheduler will automatically scale service instances according to the dynamic request workload and route the service requests with load balancing. For example, we can employ the auto scaling[3] and elastic load balancing[4] services in Amazon EC2 to implement the local scheduler. Furthermore, the request logs will be collected in each data center (*e.g.*, with the approach proposed in [10]) to facilitate the log analysis in the deployment manager. In each local scheduler, the platform-level resource allocation can be achieved, where the resource constraints of virtual machines are considered.

In this paper, we focus only on the deployment manager component. In other words, we address how to select the candidate data centers for deployment, since the local scheduler problem is well studied in the literature (*e.g.*, [14]) as a resource allocation problem in a single data center.

III. Deployment Model

In this paper, we focus on the minimization of user-perceived application-level latency. Specifically, given a workflow of a service-oriented application, how to deploy each service into multiple data centers to achieve minimal latency while keeping low operational cost. A straightforward approach is to deploy each service into every candidate data centers, so that each user can perceive the minimal latency. However, this straightforward approach is not cost-effective. The minimization of user-perceived latency should also brings operational cost into account.

Application-level latency (*i.e.*, response time) typically denotes the time duration starting with a user request sent out and ending with a response to the user finally received, during which multiple invocations across services are performed. Generally, it can be computed as a summation of three elements: the involved communication delays between user and data centers and also those between data centers, the involved communication delays inside data centers, and the processing time for the service request. The second element, *i.e.*, the communication delays inside data centers, are negligible compared with the other elements, since machines in a data center are all connected by high-speed links. In addition, as the processing time of each service request is only affected by the computing capability of a service instance, we assume each service instance is the same and the total processing time is constant for each application. Consequently, for simplicity, we only study the relationship between the fist element and the service deployment strategy. Note that the processing time can easily be incorporated into our following formulation.

[3]http://aws.amazon.com/autoscaling/

[4]http://aws.amazon.com/elasticloadbalancing/

Model 1 Service Deployment Model across Data Centers

$$\mathbf{min} \quad \sum_{i=1}^{N}\Big(\sum_{j=1}^{M} f_{ij}\cdot \min_{c_j^m\in C_j} d(i,c_j^m) + \sum_{j=1}^{M}\sum_{\substack{k=1\\k\neq j}}^{M} f_{jk}^{i}\cdot \min_{\substack{c_j^m\in C_j\\c_k^n\in C_k}} d(c_j^m,c_k^n)\Big) \tag{1}$$

$s.t.$

$$C_j \subseteq C, \qquad \forall j=1,2,\cdots,M \tag{2}$$

$$|C_j| = K_j, \qquad \forall j=1,2,\cdots,M \tag{3}$$

TABLE I. Notations of Deployment Model

Notations	Descriptions
N	Number of users
M	Number of service components in an application
f_{ij}	Frequency of invocations between user i and service j
c_j^m	The $m\text{-}th$ datacenter deployed by service j
$d(i,c_j^m)$	Latency between user i and datacenter c_j^m
f_{jk}^i	Frequency of invocations between service j and service k for user i
C_j	Deployment strategy of servie j (the seleted data centers for service j)
C	The set of candidate datacenters
K_j	The number of instances of service j

Model 1 shows the constraint minimization formulation of our deployment model. In this model, the objective function aims at minimizing the total latencies of all requests including both user requests and cross-service requests. The notations in the model are summarized in Table I.

The intuition of our model is how to decide the deployment strategy (*i.e.*, the number and locations of selected data centers for each service) so as to minimize the user-perceived latencies. In detail, the frequency f_{ij} and f_{jk}^i indicates the dependencies between user-service requests and cross-service requests, as we jointly consider the sum of their latencies in our objective function. $\min_{c_j^m\in C_j} d(i,c_j^m)$ denotes the lowest network latency that user i can experience when invoking service j. Similarly, $\min_{c_j^m\in C_j, c_k^n\in C_k} d(c_j^m,c_k^n)$ describes the lowest latency between dependent services j and k in the invocations. Moreover, the first constraint guarantees that for each service j the data centers C_j are selected from the whole candidate set C, while the second constraint ensures each service j will be deployed into K_j data centers. K_j keeps a trade-off between the user-perceived latency and the operational cost.

We can see that such a formulation is actually an NP-hard problem (reduced to the set k-cover problem). Next, we will show how it can be solved efficiently with an approximation algorithm, namely, the genetic algorithm (GA).

IV. Genetic Algorithm Design

To solve our deployment model, we employ the genetic algorithm, which is a popular search heuristic originated from the natural genetic systems to solve optimization problems. The basic idea of GA is to survive the fittest. Generally, the genetic algorithm works with a set of genomes called a

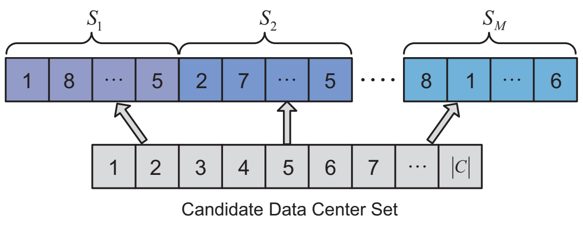

Fig. 2. Genome Encoding of Deployment Strategy

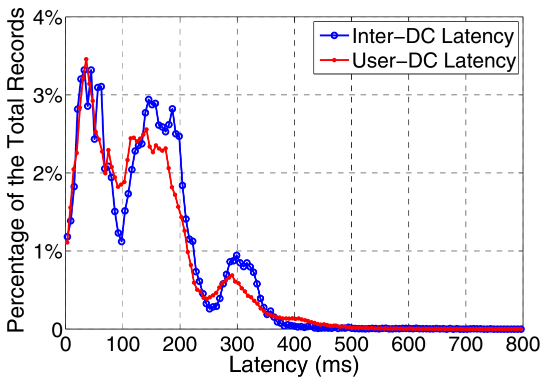

Fig. 3. The Approximate Distributions of Inter-DC Latencies and User-DC Latencies (DC: Data Center)

population, where each genome is a feasible solution encoded with a string of integers. And each genome is associated with a fitness value, which indicates the possibility of survival and reproduction in the next generation for each individual genome. At each generation, the population goes through a set of operations, including selection, crossover, mutation and evaluation, to evolve to the next generation. At the beginning, individuals in the population are selected in pairs as parents to take the crossover operation for each pair. The crossover operation randomly cuts off the original genome and swaps the parts to generate offsprings. Then the offsprings are placed back into the population to replace the weaker individuals. After the crossover operation, each genome will be mutated with some probability to change the genome into a new one. Finally, the evaluation operation is performed to update the fitness value of each genome. This process is repeated until some stop criteria are met (*e.g.*, until the best fitness value remains unchanged for a given number of generations, or the maximum number of generations has been reached).

In this paper, we tailor the genetic algorithm by encoding the deployment strategy as shown in Fig. 2. As we can see, each service j consists of a set of genes with size K_j, which can be selected from the candidate data center set. We jointly consider all the service components in a service-oriented application as a genome, and we can get different genomes by varying the values of the genes, as the indicator of each data center. For the parents selection, we use the *roulette wheel* method, which selects individuals of higher fitness values with higher probability. We use the classical *single-point crossover* operator and *real-value mutation* operator.

V. Experiments

A. Data Description

To evaluate the performance of our deployment model, we collected a large-scale real-world latency dataset. In our experiments, we use the PlanetLab as the hypothetical geo-distributed data centers, which consists thousands of computers distributed all over the world. We first got a list of 588 active PlanetLab nodes via the CoMon2[5] service, since some registered nodes may shut down or lose connection of the Internet. Meanwhile, We obtained 5,800 openly-accessible Web service addresses across over 60 countries by crawling the Internet, which are taken as users dispersed world-wide.

To collect the communication latency data, we use ping messages to measure the round-trip time (RTT) from each PlanetLab node to each Web service. We send 32-byte ping packets continually for ten times and take the average RTT from all replies as the latency between two hosts. The network latencies between pair-wise PlanetLab nodes are obtained in a similar way. The raw data is then post-processed to retain the nodes and Web services that are all reachable. Finally, we are left with 307 PlanetLab nodes and 1,881 Web services. The relatively low yield is partially due to the case that some Internet hosts are ping-unavailable, and also due to the failure of the Internet connection.

In this way, a 307×1881 matrix (denoted as User-DC matrix) of network latencies between 307 data centers and 1,881 users, and a 307×307 matrix (denoted as Inter-DC matrix) of network latencies between 307 data centers are obtained. The approximate distribution of the two latency matrix is illustrated in Fig. 3. In particular, we also evaluate the average latencies of these two matrix, which is $129.4ms$ and $138.4ms$ respectively. Our dataset is also publicly released online[6] for future research.

B. Performance Comparison

For evaluation purpose, we simulate user requests by randomly generating the request logs. By default, we set $N = 1881$, $M = 10$, and $|C| = 100$. For simplicity, we set all the values of K_j equally and the default setting is $K_j = 10$ for each j. A user of a service s would have 5 request logs. One request of a service would involve on average 5 requests of other services. In addition, the collected latency data are used to make the simulations realistic.

In our evaluation, we compare our approach to the following four heuristics for service deployment. The experimental results show that our algorithm substantially outperforms these heuristic methods.

- **Random:** In *random* deployment, the services are deployed randomly in K_j data centers.
- **OneDC:** *OneDC* is proposed in [10], which deploys all the services in one data center with the highest performance of all the candidates. It is commonly employed by many companies due to its simplicity,

[5]http://comon.cs.princeton.edu

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

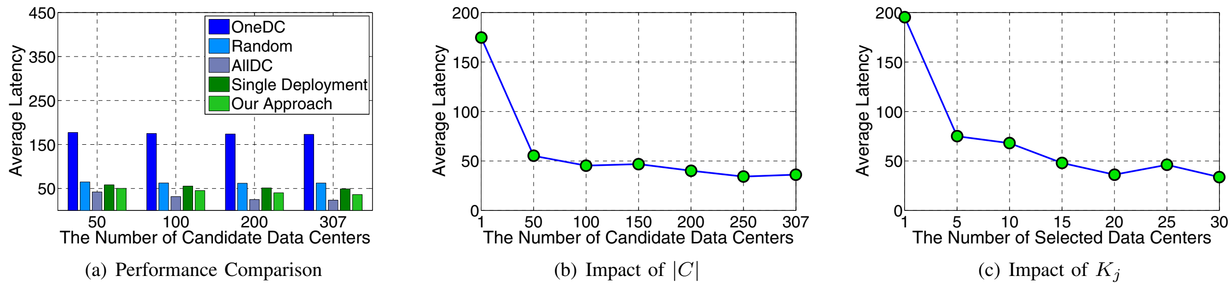

Fig. 4. Performance Evaluation

but the advantages of geo-distributed data centers are not considered.

- **AllDC:** *AllDC* simply deploys each service in every data center. It can be regarded as an extreme baseline, where the user-perceived latency is minimized with the operational cost ignored.
- **Single Deployment:** *Single Deployment* is proposed in [6], which optimizes the deployment strategy independently for each single service. Hence, it does not take service dependencies into account.

We compare the performance of different approaches. The results are shown in Fig. 4(a). We can observe that the OneDC heuristic performs worst, although it is widely employed due to its simplicity. AllDC can achieve the lowest average latency, yet with the the highest cost by selecting all the candidate data centers. Compared with AllDC, our approach obtains competitive performance result while saving cost by about $15\times$.

In the following, we report the experimental results under different situations, where the impact of the number of candidate data centers $|C|$ and the impact of the number of selected data centers K_j are studied. While the parameters, such as average call length, average number of request logs, the number of services, are also evaluated, the results are similar to those reported in the previous work [13] and thus omitted for report here due to the space limit.

*1) **Impact of** $|C|$:* To study the impact of $|C|$, *i.e.*, the number of candidate data centers, we vary it from 1 to 307, and obtain the latency values. The results are shown in Fig. 4(b). We can see a decreasing trend of the curve. But such reduction of latency gets smaller as the number of the candidate data centers increases. As a result, we can improve the application performance by deploying each service across multiple data centers, while the cost can be limited by using only a part of the candidate data centers. The reason is that with more candidates, our approach can find a better deployment strategy by considering service dependencies.

*2) **Impact of** K_j:* For simplicity, we assume the values of K_j are the same for each j. In this experiment, we vary the value of K_j from 1 to 30, to investigate the impact of K_j on user-perceived average latency. The experimental results are shown in Fig. 4(c). We can see that the average latency decrease dramatically with the increasing of K_j (The little fluctuation at $K_j = 25$ may be caused by the random initial deployment). This is because with more data centers selected for service deployment, our approach can take advantage of the geo-diversity of distributed data centers to improve the performance for the dispersed users.

VI. Related Work

A. *Facility Location and Replica Placement Problems*

The service deployment model shares some similar properties with the popular facility location problem (*a.k.a.* the k-center problem), which concerns optimal placement of facilities to minimize transportation costs. An extensive review of this problem can be found in [15]. Some classical algorithms (*e.g.*, k-median model) are also introduced to solve the replica placement problem [16].

Although traditional replica placement algorithms (*e.g.*, [16], [17]) have been extensively investigated in recent literature, these approaches primarily focus on the scenario of content replicas placement in content delivery networks with static topology and thus fail to take the dynamics into consideration in current cloud systems.

B. *Service Deployment*

Recently, much attention has been paid on the service deployment problem in the cloud. Some studies [18], [19], [20], [21] consider the service deployment problem in a single cloud, where services are dynamically replicated and mapped into different servers with various resources to optimize a set of proposed metrics, such as performance, resource utilization and operational cost.

Nowadays, with the growing of data center infrastructures, more research work has been conducted to address the challenge of deploying services across geo-distributed clouds. However, most of the work (*e.g.*, [3], [6], [7], [8], [9]) has not taken the service dependencies into consideration, where individual services are independently deployed to optimize the corresponding performance and operational cost.

Service-oriented applications deployment problem has been widely studied. The work in [22] investigates the underlying deployment mechanism instead of the deployment strategy. Work [18] studied the evolution service deployment in a single cloud, where communication latencies between data centers are not considered. Furthermore, Björkqvist et al. propose the dynamic replication of service component in a single cloud. In contrast, this paper focuses on the data center selection problem. Some studies (*e.g.*, [10], [11], [12]) also consider the service or data interdependency for deployment, but they have not focused on the problem of service-oriented applications deployment across geo-distributed clouds.

The most related work to ours is [12], which identifies the composite service placement problem with service dependencies in the cloud and solve the model by evolutionary algorithms. However, it focuses on the resource optimization from the perspective of cloud providers, and does not characterize the service deployment across geo-distributed data centers. Moreover, each service component is only deployed as one service instance, which is not the case in reality. In [13], the service co-deployment problem is addressed by modeling it as an integer programming formulation. However, the integer programming model fails to scale to large-scale service-oriented applications deployment problem across multiple data centers, since its complexity grows exponentially with the number of services and candidate data centers.

C. Genetic Algorithms

A genetic algorithm is a classical search heuristic which mimics the process of natural evolution and tries to approximate optimal result by a set of operators, such as crossover, mutation and selection. Genetic algorithms are widely used to address service deployment [18], [12], service selection and so on. This paper tailors the genetic algorithm to solve our service deployment model, which can achieve good efficiency.

VII. Conclusion and Future Work

This paper is the first work to address the problem of service-oriented applications deployment in geo-distributed clouds. Towards this end, we propose a dynamic service deployment framework including the deployment manager and the local scheduler, which can optimize the deployment strategy in the data center level and automatically scale the number of service instance in the instance level. More specifically, we focus on the deployment strategy optimization problem which aims at minimizing the user-perceived latency while keeping low operational cost. This problem is formulated as a constraint minimization problem and can be solved efficiently via the genetic algorithm. To evaluate the performance of our deployment model, extensive experiments are conducted based on a real-world latency dataset. It is shown that our approach substantially outperforms other existing methods.

In our future work, we will explore how to improve our deployment model by incorporating the capacity and cost models for each data center. Moreover, more experiments can be conducted in the production geo-distributed clouds (*e.g.*, Amazon EC2) to evaluate the performance of our approach.

Acknowledgment

The work described in this paper was fully supported by the National Basic Research Program of China (973 Project No. 2011CB302603), the National Natural Science Foundation of China (Project Nos. 61100078 and 61100077), the Shenzhen Basic Research Program (Project No. JCYJ20120619153834216, JC201104220300A), and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 415212).

References

[1] I. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit," in *Proc. of IEEE CLOUD*, 2010, pp. 123–130.

[2] X. Lu, H. Wang, J. Wang, J. Xu, and D. Li, "Internet-based virtual computing environment: Beyond the data center as a computer," *Future Generation Comp. Syst.*, vol. 29, no. 1, pp. 309–322, 2013.

[3] Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba, "Dynamic service placement in geographically distributed clouds," in *Proc. of IEEE ICDCS*, 2012, pp. 526–535.

[4] H. Qian, "Proximity-aware cloud selection and virtual machine allocation in iaas cloud platforms," in *Proc. of the 2013 International Workshop on Internet-based Virtual Computing Environment (iVCE)*, 2013.

[5] K. Satake, "Fujitsu's approach to saas in Japan - Fujitsu SaaS platform," *Fujitsu Scientific and Technical Journal*, vol. 45, no. 3, 2009.

[6] Y. Kang, Y. Zhou, Z. Zheng, and M. R. Lyu, "A user experience-based cloud service redeployment mechanism," in *Proc. of IEEE CLOUD*, 2011, pp. 227–234.

[7] M. Steiner, B. G. Gaglianello, V. K. Gurbani, V. Hilt, W. D. Roome, M. Scharf, and T. Voith, "Network-aware service placement in a distributed cloud environment," in *Proc. of ACM SIGCOMM*, 2012, pp. 73–74.

[8] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. of IEEE INFOCOM*, 2012, pp. 963–971.

[9] F. Chang, R. Viswanathan, and T. L. Wood, "Placement in clouds for application-level latency requirements," in *Proc. of IEEE CLOUD*, 2012, pp. 327–335.

[10] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proceedings of USENIX NSDI*, 2010, pp. 2–2.

[11] E. Juhnke, T. Dörnemann, D. Böck, and B. Freisleben, "Multi-objective scheduling of BPEL workflows in geographically distributed clouds," in *Proc. of IEEE CLOUD*, 2011, pp. 412–419.

[12] Z. I. M. Yusoh and M. Tang, "Composite saas placement and resource optimization in cloud computing using evolutionary algorithms," in *Proc. of IEEE CLOUD*, 2012, pp. 590–597.

[13] Y. Kang, Z. Zheng, and M. R. Lyu, "A latency-aware co-deployment mechanism for cloud-based services," in *Proc. of IEEE CLOUD*, 2012, pp. 630–637.

[14] M. Björkqvist, L. Y. Chen, and W. Binder, "Dynamic replication in service-oriented systems," in *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'12)*, 2012, pp. 531–538.

[15] M. T. 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.

[16] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proc. of IEEE INFOCOM*, 2001.

[17] X. Tang and J. Xu, "Qos-aware replica placement for content distribution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 10, pp. 921–932, 2005.

[18] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "Evolutionary deployment optimization for service-oriented clouds," *Softw. Pract. Exper.*, vol. 41, no. 5, pp. 469–493, 2011.

[19] P. Fan, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu, "Topology-aware deployment of scientific applications in cloud computing," in *Pro. of IEEE CLOUD*, 2012, pp. 319–326.

[20] J. Edmondson, A. Gokhale, and D. Schmidt, "Approximation techniques for maintaining real-time deployments informed by user-provided dataflows within a cloud," in *Proc. of IEEE SRDS*, 2012.

[21] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues, "Orchestrating the deployment of computations in the cloud with conductor," in *Proc. of USENIX NSDI*, 2012, pp. 27–27.

[22] S. van der Burg and E. Dolstra, "A self-adaptive deployment framework for service-oriented systems," in *Proc. of 2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'11)*, 2011, pp. 208–217.