Advances in Computer Science and its Applications (ACSA)
Vol. 2, No. 3, 2013, ISSN 2166-2924

www.worldsciencepublisher.org

Towards Proximity-aware Application Deployment in Geo-distributed Clouds[1]

[1]Hangwei Qian and [2] Qixin Wang

[1]VMware Inc., CA, USA
[2] University of Southern California, CA, USA

Email: qianhangwei@gmail.com

Abstract –In this paper, we propose a proximity-aware cloud service selection system to help application providers to deploy their applications in the cloud. Cloud platforms deploy multiple data centers geographically distributed around the world. We argue that proximity plays a very important role to improve the application performance when deploying applications in such geo-distributed cloud platforms. We design and evaluate the system for automatic selection of cloud infrastructures, in which we not only considers the deployment cost, but also takes into account the location of cloud infrastructures, application clients and related applications, and the interaction among application components when selecting the cloud infrastructures. Since the solution space is very prohibitively large, we propose a stepwise application placement algorithm to address the scalability problems. In our simulated experiments, it shows the great performance and high efficiencies of our approach.

Keywords –Cloud Computing, Cloud Selection, Proximity-aware

1. Introduction

In infrastructure as a service (IaaS) clouds, application providers rent the virtual machines (VMs) and pay the cloud providers based on resource usage to get rid of the expensive and tedious infrastructure deployment and focus on their business logic. With the popular adoption of cloud computing paradigm, many cloud providers are emerging in the industry, such as Amazon, Microsoft, Google, Rackspace, HP and Terremark, etc.

Among many factors to take into account when selecting which data centers to deploy the applications, price is the first thing to consider. Different cloud providers offer different billing models to their customers. Moreover, different types of resources, like CPU and network bandwidth, are charged differently. In order to reduce the monetary cost, we not only need to compare the billing policies of different cloud providers, but should also consider the resource requirements of the applications.

At the same time, we should also ensure the high quality of service (QoS) of the applications. While higher capacity and larger bandwidth are certainly the options to be considered to improve the application performance, it is also very important to take into account the proximity when selecting the cloud infrastructures. According to [23], there is strong inverse correlation between network distance and bandwidth during the interaction between the clients and servers. Thus, factoring the proximity into the selection of cloud infrastructures can significantly reduce the client response time and increase the network bandwidth. We consider the following aspects for proximity: *i) distribution of data centers; ii) distribution of Internet clients; iii) dependencies among the components; IV) location of related applications.*

We design a system to facilitate the selection of cloud infrastructures of IaaS cloud platforms. It jointly takes into account the price and proximity when selecting the cloud infrastructures to minimize the monetary cost and meanwhile maintain the high QoS of the applications. We formulate this problem as a multi-objective optimization problem and show that it is NP-hard. In order to solve the scalability issue due to the large number of available data centers and possibly the big number of application components and related applications, we propose an efficient heuristic algorithm to search the solution space, which is dubbed *stepwise application placement*. We discuss it in more details in Section 2.

In summary, we address the cloud infrastructure selection problem in IaaS platforms and make the following contributions:

- We consider both the price and proximity when selecting the cloud infrastructure services.
- We develop an efficient algorithm to solve the scalability issue in the cloud infrastructure selection.
- We evaluate our system and show that our proposed algorithm is very efficient to calculate the placement policy with performance very close to the optimality.

2. Background

[1]**The preliminary version of this paper appeared in the 2013 International Conference on Collaboration Technologies and Systems (CTS 2013) [34]**

2.1 Distribution of data centers

Cloud providers usually deploy multiple data centers distributed geographically around the world, allowing the applications to be able to reach the global footprints according to their business requirements. For example, Amazon deploys data centers at North America, Europe and Asia [19]. According to [21], all the major cloud platforms span multiple states, countries and continents.

2.2 Distribution of Internet clients

Placing the applications in the cloud infrastructures that are close to the clients can greatly reduce the client-perceived response time and increase the traffic bandwidth. In order to effectively manage the huge amount of client information, we apply the network-aware clustering to classify the Internet clients into groups according to their network-aware prefixes that can be obtained from the Border Gateway Protocol (BGP) tables [16]. More detailed discussion is at Section 3.

2.3 Dependencies among the components

Modern applications usually consist of multiple components interacting with each other. For example, it is common practice that web applications have multi-tier architectures, including web servers, application servers and back-end databases. At each tier, there might be multiple replicas for high service reliability and scalability. Furthermore, there are some highly distributed applications that have many more components and provide the services to Internet clients across the world, like the content delivery networks (CDNs) [2] and multiplayer distributed online games [6, 15], etc. When deploying the applications in the cloud platforms, it is very important to consider the proximity among the application components that interact with each other.

2.4 Location of related applications.

We also need to take into account the location of the related applications when selecting the cloud infrastructures for an application. It is not uncommon that applications communicate with each other. For example, the financial systems need to communicate with each other for many critical tasks, like money transfer and credit card payments. Also, hospitals often share patient information and need to communicate with each other. It would be preferable to place the applications close to the location of related applications to reduce the network latency and increase the network bandwidth.

3. Cloud Infrastructure Selection

In this section, we first explain the cloud infrastructure selection problem and show the problem hardness. Then we propose a heuristic algorithm to solve the problem. Finally, we discuss some algorithm variations taking into account other elements, such as component replication and service reliability, etc.

3.1. Environment

As mentioned above, Internet clients are divided into groups, called *client clusters (CCs)*. Clients in the same cluster share the same BGP prefix and are close to each other [16]. They are treated as a single entity when considering the proximity in the selection of the cloud infrastructures. We can mine the client information from the application server logs, which contain not only the client IP addresses, but also other important information, like how frequently the clients access the applications. Alternatively, client information can also be obtained from the logs of the associated authoritative DNS servers (For applications that are newly developed, it might be hard to know the client IP addresses. In this case, application providers can make some estimation at their best effort, for example, obtaining the client information according to their business relationships and characteristics of the applications). In this paper, we assume the client distribution is known.

When selecting the cloud infrastructures, we should know the resource requirements of all application components, such as CPU, memory, network bandwidth and disk space, to calculate the deployment cost. Meanwhile, we also need to know how the application components communicate with each other and with other related applications. For instance, for three-tier applications, when web servers receive requests from clients, they usually need to contact the application servers which further interact with the backend databases. This is different from some existing works which address the resource allocation problem for applications in isolation, such as [28, 33, 22]. In order to get these information, existing application profiling techniques, such as [29, 32, 7], can be applied.

The communication pattern among the components as well as with other related applications is represented in the pattern graph $G=(V,E)$. Each component has a corresponding node in the graph G. In particular, the nodes that accept requests from Internet clients (such as web server component) is referred to as *external nodes*. When one application component C_i interacts with another component C_j , an edge $e(C_i;C_j)$ would be added between the C_i and C_j . For each edge, there would be a weight associated with it, which is the volume of interactions between the two components in order to serve a single client request. The larger the weight is, the more coupled the components are. We assume that the pattern graph is a directed acyclic graph (DAG), since if there is a cycle there, then the clients would never get the response from the applications. In addition, we also simply assume that the pattern graph is a connected graph based on the observation that our modeling and proposed algorithms discussed in the remaining sections can be applied without change for each isolated subgraph. Section 4.4 has more detailed discussion about this.

In addition, it is necessary to know the distances between the cloud infrastructures and the client clusters as well as the distance among the cloud infrastructures

after obtaining the location information of them. In order for that, we can simply allocate a dummy virtual machine at each data center and get the network distances from each data center to all the client clusters as well as the distances among the data centers by running "ping" command from these dummy virtual machines. Since the distance information is relatively very stable, this kind of measurement can be conducted very infrequent, say, every month. Also, the dummy virtual machines are only used for the simple measurement, and do not have much resource consumption. Thus this method is simple and cheap. There are also many existing techniques for measuring or estimating the distance between hosts and can also be applied in our system, like [10] and [12].

3.2. Problem Statement

The pattern graph mentioned above is extended to include the client clusters. Figure 1 shows an example. Each client clusters (only three, CC1, CC2 and CC3 are shown in the figure) and related application (like node F and G) have a node in the graph, dubbed *client cluster node* and *application node* respectively. An edge is added from each client cluster node to the external node (like node A). Also, if a component visits other component ors related applications, an edge is added between them, like the edge $e(A;B)$ and $e(B; F)$. A value is associated with each edge, which is the distance between the nodes. Note that, there is another value on the edges between the components indicating the volume of the interaction between the components, which is not shown in the figure.

Assume there are M components of the application. Each component $COMP_k$ is associated with the resource requirements expressed as {CPU, Memory, Disk, Network}. The external node is denoted as COMPK0 (For simplicity, we assume that applications has only one external component receiving requests from Internet clients. Section 4 discusses the case when components are replicated.) . Also assume that there are N cloud providers, each with d_i data centers. The total number of data centers is $W = \sum_{i=1}^{N} d_i$. Let $COST_{k,i}$ be the cost of deploying component $COMP_k$ at the data center DC_i. P denotes the placement matrix of the components. If component $COMP_k$ is deployed at the data center DC_i, then $P_{k,i} = 1$; otherwise, $P_{k,i} = 0$, for $k = 1, 2, \ldots, M$, and $i = 1, 2, \ldots, W$. Among all the nodes in the graph, the location of client cluster nodes and related application nodes are known and referred to as *fixed nodes*, while others are called *unfixed nodes* (node A, B, C and D in the figure 1 for instance). The goal is the find a matrix P to place unfixed nodes satisfying their resource requirements.

The first objective of the problem is to deploy the unfixed nodes to the cloud infrastructures to minimize the overall deployment cost *COST*, which is:

$$COST = \sum_{k=1}^{M} \sum_{i=1}^{W} COST_{k,i} P_{k,i} \quad (1)$$

where $\sum_{i=1}^{W} P_{k,i} = 1, k = 1, 2, \ldots M.$

Assume there are L client clusters and R related applications. Let the distance between the client cluster CC_l and data center DC_j be $DIST_{l,j,}$ the distance between data center DC_i and data center DC_j be $DIST_{i,j}$ and the distance between the data center DC_i and the related application RAP_r be $DIST_{i,r.}$ Also, let Q_l denote the request rate for the application from the client cluster CC_l, and $V_{k1,k2}$ denote the volume of communication between component $COMP_{k1}$ and $COMP_{k2}$ to serve a single request. The second objective of the problem is to deploy the unfixed nodes to minimize the overall network distance, *DIST*, among all the nodes in the graph for all the requests, which is:

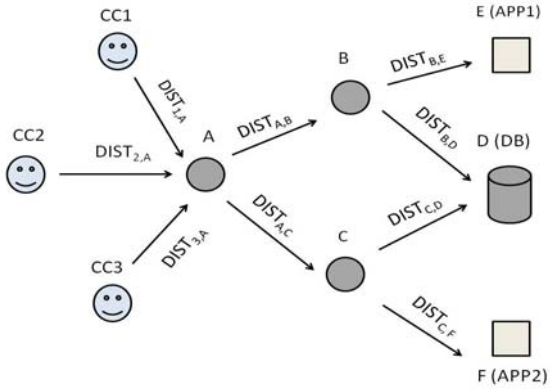

Figure 1 System Graph

$$DIST = \sum_{l=1}^{L} Q_l DIST_{l,k0} + \sum_{l=1}^{L} Q_l \sum_{k1=1}^{M} \sum_{k2=1}^{M} DIST_{k1,k2} V_{k1,k2} + \sum_{l=1}^{L} Q_l \sum_{k=1}^{M} \sum_{r=1}^{R} DIST_{k,r} V_{k,r} \quad (2)$$

where the edge $e(k1, k2) \in E$, and $e(k, r) \in E$. Among them, $\sum_{l=1}^{L} Q_l DIST_{l,k0}$ is the distance from all the clients to the external component; $\sum_{l=1}^{L} Q_l \sum_{k1=1}^{M} \sum_{k2=1}^{M} DIST_{k1,k2} V_{k1,k2}$ is the distance among the components of the application to serve the client requests; and $\sum_{l=1}^{L} Q_l \sum_{k=1}^{M} \sum_{r=1}^{R} DIST_{k,r} V_{k,r}$ is the distance between the components and the related applications.

This is a multi-objective optimization problem: deploying all the components at the cheapest cloud infrastructure would incur smallest cost, but can lead to larger user-perceived response time, which would hurt the business. Our system allows the application providers to make the trade-off according to their recognition of the importance of the deployment cost and proximity by adding two weight parameters: α and β. The multi-objective optimization problem is transferred to the single objective problem to minimize the overhead score η:

$$Minimize \quad \eta = \alpha COST + \beta DIST \tag{3}$$

subject to:

$$\sum_{i=1}^{W} P_{k,i} = 1, P_{k,i} \in \{0,1\} \tag{4}$$

$$\alpha + \beta = 1, \alpha, \beta \in [0,1] \tag{5}$$

$$DIST_{k1,k2} \geq 0 \tag{6}$$

$$V_{k1,k2} \geq 0 \tag{7}$$

$$COST_{k,i} \geq 0 \tag{8}$$

Noticing the binary choice of each element in the placement matrix P, it is a variation of the 0-1 integer linear programming problem, which is NP-hard [5]. The complexity of this problem is $O(W^M)$, which is prohibitively expensive. In this paper, we propose a stepwise application placement algorithm to solve this problem, which is elaborated at section 3.2.

3.2. Stepwise Application Placement

The nodes in the graph are divided into three categories: fixed nodes, semi-fixed nodes and unfixed nodes. Fixed nodes refer to the nodes whose locations are already known, like the client cluster nodes and the related application nodes F and G in figure 1. Semi-fixed nodes are the nodes that are adjacent to at least one fixed node, like the node A, B and D in figure 1. The remaining nodes are unfixed nodes, like node C in figure 1. Let the ξ be the set of fixed nodes, Ψ the set of semi-fixed nodes and θ the set of unfixed nodes. We need to decide where to deploy the corresponding components for nodes in Ψ and θ.

Based on the observation that nodes in Ψ need to be close to the fixed nodes to which they are adjacent, we propose the stepwise component placement algorithm, in which nodes in Ψ are deployed first and only one of them is considered at a time. Whenever the location of a node in Ψ is decided, it is removed from Ψ and put into ξ. Meanwhile its neighboring unfixed nodes in θ becomes semi-fixed nodes and are transferred to set Ψ.

When deciding how to deploy the components for nodes in Ψ, we only care about the location of the fixed nodes that are adjacent to them, as well as the deployment cost. For a node in Ψ, each data center DC_j would obtain a overhead score according to the formula (3) assuming the component of this node is deploy there (in this case, *COST* and *DIST* only count the component of this particular node deployed at DC_j, in stead of the cost and network distance of all components as formula (1) and (2)). Note that, in order for deployment cost and proximity to be comparable, both *COST* and *DIST* are normalized by their maximum value respectively across all data centers respectively before they are combined with the weight α and β to calculate the overhead score. The data center with the smallest overhead score would be selected to deploy the component.

One important issue is that different orders in which we deploy the components of nodes in Ψ can result in significantly different placement policies. Take the system graph in figure 1 for example, both node A and B are in set Ψ. If we select the cloud infrastructure for node A first, then node A would be place close to clients. And node B would be placed depending on the location of both node A and F. But if node B is selected before node A, then the location of node B only relies on the location of the node F, and the location of node A depends on the location of the clients and the node B. In order to address this issue, we prioritize the nodes in Ψ the when deciding which one to deploy first. The priority of each node in Ψ is based on the number of fixed nodes it connects to directly and the volume of communication between them. For example, node A would have higher priority than node B since it connects to large number of client cluster nodes. Nodes with equal priority are deployed at random order.

The details of the stepwise application placement algorithm are as shown in figure 2 and 3.

4. Placement Variations

The step-wise component placement algorithm in Section 3.2 considers only the deployment cost and proximity. While these two aspects are the critical factors to consider in the selection of cloud infrastructures, application providers may consider more in real life. For example, some application might require some or all the components to be in the same VLAN, which is supported by some cloud providers. However, if these components are distributed among different cloud providers, no single

```
1.  Divide and put the nodes into set ξ, ψ and θ;
2.  Get the priority of nodes in ψ and order them decreasingly;
3.  While ψ ≠ ∅
4.      Pick up the first node n in ψ and call deployNode(n);
5.      Remove the node n from ψ and put it in ξ;
6.      For each node in θ
7.          If the node becomes semi-fixed node
8.              Transfer the node from θ to ψ;
9.              Get its priority and reorder the nodes in ψ;
10.         End If
11.     End For
12. End While
```

Fig. 2. Stepwise application placement algorithm.

```
1.  dc_id = -1; cur_score = Integer.max;
2.  For j = 1 to W
3.      Calculate score value tmp assuming n is deployed at
4.      DCj according to formula (3)
5.      If tmp < cur_score
6.          dc_id = j;
7.          cur_score = tmp;
8.      End If
9.  End For
10. Return dc_id;
```

Fig. 3. Algorithm of deployNode(n).

entity (it is very hard if not impossible for cloud providers to coordinate) would be able to configure VLAN for these components. In this case, application providers would choose to deploy the components across

data centers belonging to a single cloud provider. Also, things like reliability credit of cloud providers would also affect the decision making of application providers In addition, some components, like the web server component, play such an important role for the application that it is better to replicate them across multiple locations for high reliability and good performance. In this case, we need to decide how to place the replicas at different data centers (note that, the algorithm described at Section 3.2 would put all the replicas of a component in the same location where the overall cost is smallest.). In this section, we discuss and explain how to integrate these aspects into the stepwise application placement.

4.1 Single Cloud Provider

When application providers want to deploy the applications within a single cloud provider, only very simple modification is needed to integrate this requirement into the stepwise application placement algorithm. Instead of going through all the data centers of all the cloud providers (refer to line 2 of figure 3), it only checks the data centers of a single cloud provider. In other words, when deciding the deployment of the application components, only one cloud provider is considered at a time. After deciding the deployment of the application components in a cloud provider, an overhead score value (which is basically the sum of the final *cur_score* in figure 3 of all the components) is calculated for that cloud provider. After trying all the cloud providers, the one with the smallest overhead score value is selected to deploy the application.

4.2 Cloud Provider Reliability

It is also simple to integrate the reliability credit (assuming the higher the better) of the cloud providers into the stepwise application placement algorithm. We can just extend the formula (3) by introducing the reliability credit, *CREDIT*, of the cloud providers as well as the weight parameter. The extended formula would like follows. Again, *CREDIT* is normalized.

$$Minimize \quad \eta = \alpha COST + \beta DIST + \gamma \frac{1}{CREDIT} \quad (9)$$

where $\alpha + \beta + \gamma = 1, \alpha, \beta, \gamma \in [0,1]$.

4.3 Component Replication

When trying to place the replicas of a component at different data centers, two options can be applied. In the first option, we can simply get the top *b* best candidate data centers and place the replicas there. Here *b* is the preassigned number of replicas of the component. In order to do that, the deployNode(*n*) method at Figure 3 can be called *b* times, excluding the data centers that have already been selected as the candidate data centers each time. When updating the status of its neighboring nodes, the component is marked to be placed at the best candidate data center. One problem with this method is that, only the replica placed at the best data center would impact the placement of the neighboring components. For example, suppose the two replicas of the component *A* in the Figure 1 are placed at the data center DC_1 and DC_2, where DC_1 is the best candidate data center for the component *A*. Then the placement of component *B* depends on the location of DC_1 and the component *F*, not DC_2.

In order to avoid this limitation, as an alternative option, we can put all the replicas as components in the pattern graph, and all replica nodes of the same component have the same connectivity. When calling the deployNode(*n*) method, we also exclude the data centers that have already been placed with replicas of the component. But all the replica components would impact the placement of neighboring components, since all of them are connected to each neighboring component.

4.4 Isolated Components

By now, we assume that the system graph is a connected graph. However, in reality, this might not always be the case. For example, an enterprise application can be deployed at multiple branches, and some branches never interact directly with each other. In order to address this issue, we can model the problem for each isolated subgraph in the same way as in Section 3.2 and apply the stepwise application placement on them respectively. In other words, the components in each subgraph are considered as an ”independent application”.

5. Evaluation

In this section, we discuss the evaluation of our system. We compare the proposed stepwise cloud selection algorithm with two baseline algorithms: random selection and optimal selection. We conduct simulation-driven experiments to measure the performance and the execution time of the three algorithms.

5.1 Environment

In order to evaluate our system, we collect proximity information from real systems and build a cloud model. Also, we generate DAG graphs with varying size to represent the relationship among application components as well as the related applications. In this subsection, we explain then in details.

Cloud model. We extract the IP addresses from the Gnutella peers at the University of Oregon [3] and get about 157803 pingable ones, which are used to mimic the locations of client clusters. Meanwhile, we allocate 20 PlanetLab nodes from the [4] to represent the locations of data centers. From each PlanetLab node, we obtain the network latency to each Gnutella peer by conducting the "ping" command. We get about 100546 IP addresses of Gnutella peers that are pingable from all the PlanetLab nodes. In addition, we get the network latency between each pair of PlanetLab nodes by asking all the PlanetLab nodes to "ping" each other.

In our basic cloud model, there are 100546 client clusters and 20 data centers. We use the network latencies between the 20 PlanetLab nodes and the 100546 Gnutella peers as the proximity information between the data centers and client clusters, and ones among the 20 PlanetLab nodes as the proximity information among the data centers. According to the [16], there are about 400k client clusters around 2000. Thus we believe the 100546 client clusters are representative for the real life. When evaluating the scalability of our system, we randomly generate the network latency between the data centers and client clusters as well as among the data centers when the number of data centers is larger than 20.

DAG generation. We randomly generate DAG graphs to represent the relationship among the application components and related applications based on the tool [1]. We assume that the each component is related to at least one other component and remove the isolated nodes (no associated edges) in the generated DAG graphs.

Also, as mentioned before, we assume that all other application components are accessible in one way or another3. Thus, only the node representing the web server that receives requests from Internet clients in the DAG graph has noninbound edges. In order for that, we randomly select a noninbound node in the generated DAG graph to represent the web server node, and add an edge from it to all other non-inbound nodes.

Baseline algorithms. We compare our proposed stepwise cloud selection algorithm with two other algorithms: random selection and optimal selection. In random selection algorithm, we deploy each application component to a randomly selected data center. This algorithm is very efficient in terms of computation overhead, but may incur high cost and bad performance since it is totally unaware of the deployment cost and the proximity. In the optimal selection algorithm, we check each placement policy and choose the one with the lowest overall score. This algorithm is the optimal tradeoff regarding the deployment cost and performance, but is prohibitively expensive in computation overhead.

Setup. Our experiments are conducted in a 4-core virtual machine running on a host with Intel Xeon(R) 3.20GHz CPU and 6G memory.

5.2 Policy Performance

When comparing the stepwise cloud selection algorithm with the algorithms mentioned above, we want to see how the number of application components and the number of data centers impact the results. We keep one value fixed while increasing the other value in our experiments. For example, in the first set of experiment, the number of the application component increases from 4 to 10, with the number of the data centers constantly to be 10. Also, in the second set of experiments, the number of application component is fixed to be 8, while the number of data centers varies from 10 to 20. The upper bound of the number of application components and data centers in the experiments is determined by the large execution time of the optimal placement algorithm (Section IV-C shows results on this). In addition, in order to see how the weight of the deployment cost and the proximity would affect the results, we conduct these experiments in three different cases, each with weights α-β as 0.3-0.7, 0.5-0.5 and 0.7-0.3 respectively.

Figure 4-9 show the performance of different placement algorithms. We can see that the performance of the stepwise application placement algorithm is much better than the random placement and is very close to the optimality in many cases. Another interesting observation is that the good performance persists regardless of the weights for deployment cost and proximity.

5.3. Scalability

In this section, we evaluate the scalability of different algorithms by showing the execution time. The methodology in the experiments is the same with the Section IV-B. Figure 10 and 15 show the execution time

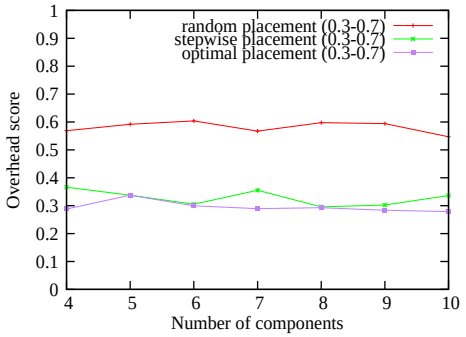

Fig 4. Performance varying component numbers (0.3-0.7)

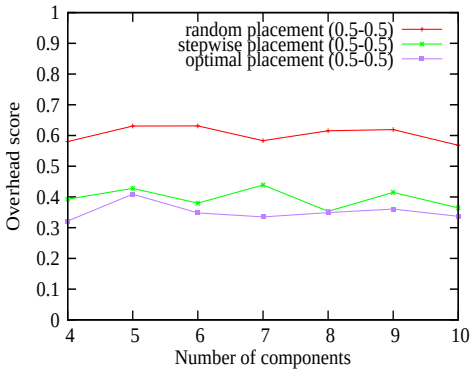

Fig 5. Performance varying component numbers (0.5-0.5)

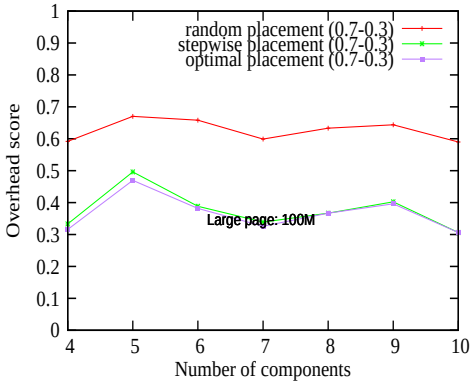

Fig 6. Performance varying component numbers (0.7-0.3)

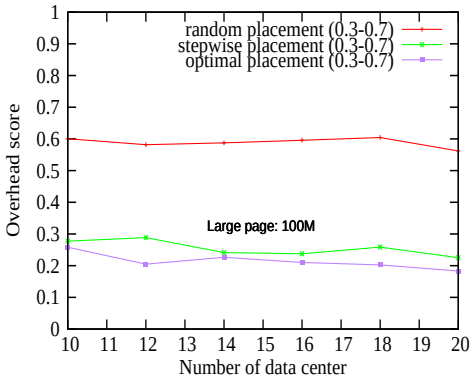

Fig 7. Performance varying DC numbers (0.3-0.7)

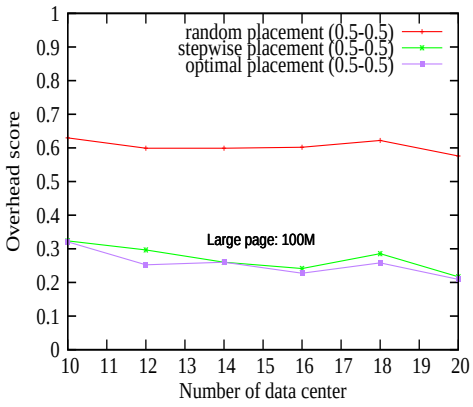

Fig 8. Performance varying DC numbers (0.5-0.5)

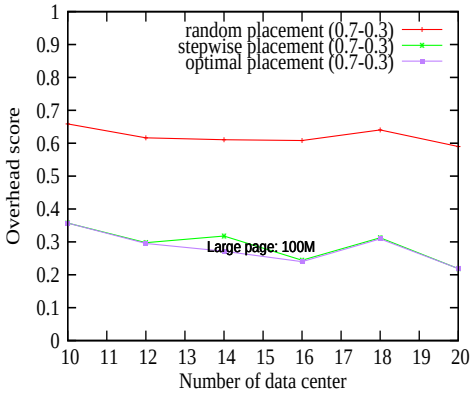

Fig 9. Performance varying DC numbers (0.7-0.3)

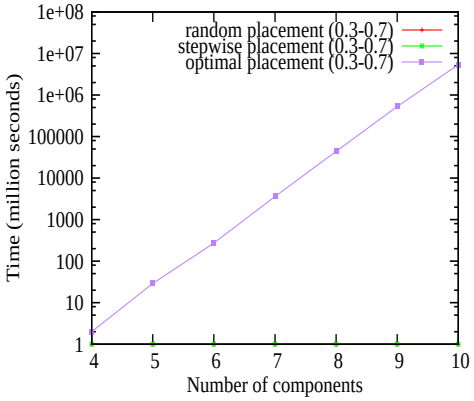

Fig 10. Run time varying component numbers (0.3-0.7)

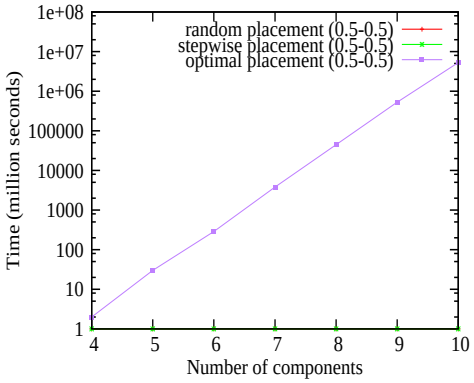

Fig 11. Run time varying component numbers (0.5-0.5)

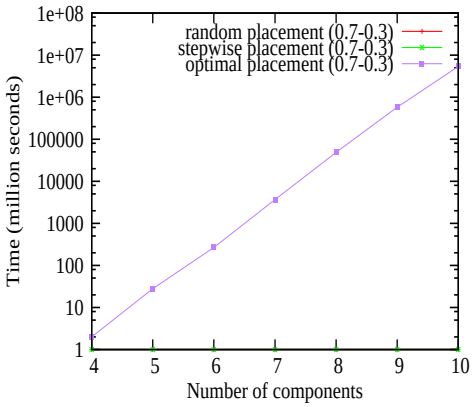

Fig 12. Run time varying component numbers (0.7-0.3)

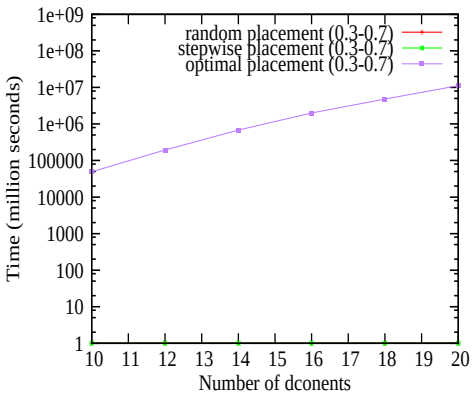

Fig 13. Run time varying DC numbers (0.3-0.7)

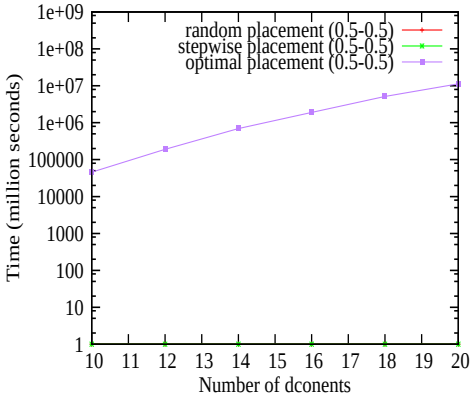

Fig 14. Run time varying DC numbers (0.3-0.7)

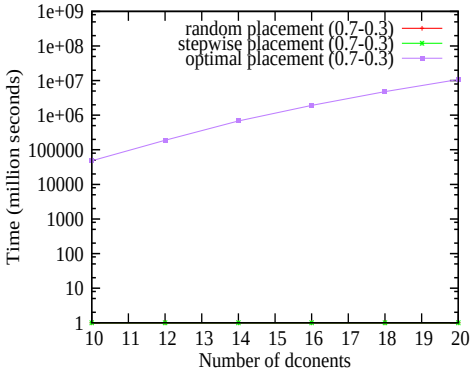

Fig 15. Run time varying DC numbers (0.3-0.7)

of the algorithms. We can see that the execution time of the optimal placement algorithm is exponential (noting the log scale of the figures), while the execution time of the stepwise placement algorithm stays near 1 million second, similar with the random placement algorithm. Thus, the proposed stepwise placement algorithm is very efficient in calculating the placement policies for the selection of cloud infrastructures.

V. RELATED WORK Cloud computing has obtained tremendous attention recently. Among the large volume of works, several of them focus on the comparing and selecting the cloud services. For example, the work in [17, 18] compared the service performance of four major cloud providers to help the customers choose cloud services. In [13], a conceptional framework is proposed to compare the cloud services based on the performance of virtual machines. Unlike their work, we take the proximity into account and aim to build a system to select the cloud infrastructures automatically for customers. In [34], we consider the proximity in the cloud selection and did some preliminary investigation of the problem.

Meanwhile, authors in [24] introduced a mathematical formulation and method of the cloud service selection based on multiple abstract criteria. Different from it, our work intends to develop a real system to automatically selection the infrastructures in IaaS cloud platforms. In [11], an approach utilizing the analytic hierarchy process is proposed to select the services for software as a service cloud. Our work differentiate from them in that we focus on the infrastructure as a service cloud, in which the proximity should be considered explicitly, while in software as a service cloud, only the overall service quality needs to be considered. Finally, many works focus on the storage issues, like storage selection, security, storage backup and deduplication etc., such as [25, 14, 8, 30], and network issues, like [26, 20]. Different from them, this work considers the general applications. The work in [35] proposes a unified approach for the application placement and demand distribution problems in global cloud and shows great promises in the optimization of the cloud platforms.

In addition, to our knowledge, this work is the first one to consider the proximity and interaction between application components and different applications when making the selection of cloud services.

6. Conclusion and future work

In this paper, we design and evaluate our system for cloud service selection in IaaS platforms. We not only consider the deployment cost, but also take into account the location of cloud infrastructures, application clients, related applications as well as the interaction among the application components. Through experiments, we show that our proposed system is very efficient to calculate a placement policy with performance very close to the optimality.

In our future work, it is interesting to investigate more complex methodologies, such as genetic algorithms [9] and neural networks [27, 31], to solve the problem. Also, we would consider the energy-related problems in cloud environment.

References

[1]http://condor.depaul.edu/rjohnson/source/graph_ge.c.
[2]http://en.wikipedia.org/wiki/content_delivery_network.
[3] http://mirage.cs.uoregon.edu/p2p/snapshots.html.
[4] http://www.planet-lab.org.
[5] Integer programming. http://en.wikipedia.org/wiki/integer programming.
[6] Daniel Bauer, Sean Rooney, and Paolo Scotton. Network infrastructure for massively distributed games. In *NetGames*. ACM, 2002.
[7] Peter Desnoyers, Timothy Wood, Prashant Shenoy, Rahul Singh, Sangameshwar Patil, and Harrick Vin. Modellus: Automated modeling of complex internet data center applications. In *Trans. Web*. ACM, 2012.
[8] Fred Douglis, Deepti Bhardwaj, Hangwei Qian, and Philip Shilane. Content-aware load balancing for distributed backup. In *LISA*. USENIX, 2011.
[9] Stephanie Forrest. Genetic algorithms. In *ACM Comput.Surv.* ACM, 1996.
[10] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. Idmaps: a global internet host distance estimation service. In *IEEE/ACM Trans. Netw.* IEEE Press, 2001.
[11] Manish Godse and Shrikant Mulik. An approach for selecting software-as-a-service (saas) product. In *CLOUD*. IEEE, 2009.
[12] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. In *IMW*. ACM, 2002.

[13] Seung-Min Han, Mohammad Mehedi Hassan, Chang Woo Yoon, and Eui-Nam Huh. Efficient service recom mendation system for cloud computing market. In *ICIS*. ACM, 2009.
[14] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *FC*. Springer, 2010.
[15] Rajesh Krishna Balan, Maria Ebling, Paul Castro, and Archan Misra. Matrix: adaptive middleware for dis tributed multiplayer games. In *Middleware*, 2005.
[16] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *ACM SIGCOMM*, 2000.
[17] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *IMC*. ACM, 2010.
[18] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: shopping for a cloud made easy. In *HotCloud*. USENIX, 2010.
[19] Where Amazons Data Centers Are Located. http://www.datacenterknowledge.com/archives/2008/11/18/where-amazons-data-centers-are-located/.
[20] Hangwei Qian, Chenghua Cao, Li Liu, Hualong Zu, Qixin Wang, Menghui Li, and Tao Lin. Exploring the network scale-out in virtualized servers. In *International Conference on Soft Computing and Software Engineer ing*. Advanced Academic Publisher, 2013.
[21] Hangwei Qian and Qian Lv. Proximity-aware Cloud Selection and Virtual Machine Allocation in IaaS Cloud Platforms. In *International Workshop on Internet-based Virtual Computing Environment*. IEEE, 2013.
[22] Hangwei Qian, Elliot Miller, Wei Zhang, Michael Rabinovich, and Craig E. Wills. Agility in virtualized utility computing. In *VTDC*, 2007.
[23] Hangwei Qian, Michael Rabinovich, and Zakaria Al-Qudah. Bringing local dns servers close to their clients. In *GLOBECOM*, 2011.
[24] Zia ur Rehman, Farookh K. Hussain, and Omar K. Hussain. Towards multi-criteria cloud service selection. In *IMIS*. IEEE, 2011.
[25] Arkaitz Ruiz-Alvarez and Marty Humphrey. An auto mated approach to cloud storage service selection. In *ScienceCloud*. ACM, 2011.
[26] Seetharami R. Seelam and Patricia J. Teller. Virtual I/O scheduler: a scheduler of schedulers for performance virtualization. In *VEE*. ACM, 2007.
[27] Alexei N. Skurihin. Neural networks in j. In *Proceedings of the international conference on APL*. ACM, 1992.
[28] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *WWW*, 2007.
[29] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In *SIGMETRICS*. ACM, 2005.
[30] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *FAST*. USENIX, 2012.
[31] Qixin Wang, Chenghua Cao, Menghui Li, Mingyi Gao, and Hualong Zu. A new model based on grey theory and neural network algorithm for evaluation of aids clinical trial. In *Advances in Computational Mathematics and its Applications*, 2013.
[32] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling re source usage of virtualized applications. In *Middleware*. Springer, 2008.
[33] Hangwei Qian, Elliot Miller, Wei Zhang, Michael Rabinovich and Craig E Wills. Agile resource management in a virtualized data center. In *ACM WOSP/SIPEW*, 2010.
[34] Hangwei Qian, Hualong Zu, Chenghua Cao, and Qixin Wang. CSS: Facilitate the cloud service selection in iaas platforms. In CTS, IEEE, 2013.
[35] Hangwei Qian and Michael Rabinovich. Application Placement and Demand Distribution in a Global Elastic Cloud: A Unified Approach. In ICAC, USENIX, 2013.
[36] Qixin Wang, Yang Liu, Xiaochuan Pan (2008), Atmosphere pollutants and mortality rate of respiratory diseases in Beijing, Science of the Total Environment, Vol.391 No.1, pp143-148.
[37] Qixin Wang, Menghui Li, Li Charlie Xia, Ge Wen, Hualong Zu, Mingyi Gao (2013), Genetic Analysis of Differentiation of T-helper lymphocytes, Genetics and Molecular Research, Vol.12, No.2, PP. 972 – 987
[38] Qixin Wang, Menghui Li, Hualong Zu, Mingyi Gao, Chenghua Cao, Li Charlie Xia(2013), A Quantitative Evaluation of Health Care System in US, China, and Sweden, HealthMED, Vol.7, No.4, PP. 1064-1074