Abstract

Cloud computing is an emerging computation paradigm. To support successful cloud computing, service oriented architecture (SOA) should play a major role. Due to the nature of widely distributed service providers in clouds, the service performance could be impacted when the network traffic is congested. This can be a major barrier for tasks with real-time requirements. In clouds, this problem can be solved by migrating services to different platforms such that the communication cost can be minimized. In this paper, we consider the problem of service selection and migration in clouds. We develop a framework to facilitate service migration and design a cost model and the decision algorithm to determine the tradeoffs on service selection and migration.

1 Introduction

Cloud computing is an emerging computation paradigm with the goal of freeing up users from the management of hardware, software, and data resources and shifting these burdens to cloud service providers [1]. Cloud computing has drawn the attention of major industrial companies, scientific communities, as well as end user groups. The Clouds provide a large pool of resources, including high power computing platforms, common devices, storages, data centers, and software services. It also provides management to these resources such that users can access them ubiquitously and without incurring performance problems.

At the same time, service oriented architecture (SOA) has been a popular framework in many application domains. The enabling technologies in SOA allow services to be discovered, composed, and executed [2][3][4][5]. Based on these technologies, services can be rapidly composed and the composite service can be deployed to achieve the desired goal. To support successful cloud computing, SOA should play a major role with ever increasing importance. All the hardware, software, and data resources can be wrapped as services in clouds. When an end-user wishes to accomplish a certain task, a composition service can be employed to discover the needed resources and compose them to provide the desired functionality and quality to the end-user.

Currently, many Web services have been deployed by different organizations that are widely distributed over the Internet. These are mostly software services running on fixed hardware resources. When composing multiple services for a system, it is likely that some selected software services are hosted at widely distributed sites. This brings potential performance problems. Sending a service request along with a large quantity of input data across the wide area network can be costly. It increases the network traffic and raises the potential of unexpected delays due to network congestions. This can be a major barrier for applications that have real-time requirements. For example, a commander may dynamically assemble a command and control application that involves a large number of web services, such as many data services based on continuous input from the remote sensors, image processing services, information fusion services, etc. to assist his/her decision making. Communication among two data processing services may involve a large amount of data and may result in delays due to network congestions. Such delays can affect the timeliness of the decision and cause costly consequences.

In SOA research, there have been some works considering the communication overhead in service composition. In [5], service selection based on the goal of minimizing communication cost has been considered. However, if there are a limited number of services to choose from, it may be difficult to significantly reduce the communication latency. In cloud environment, this problem can be solved by considering service migration. One of major advances in cloud environment is that computing hardware resources and their management utilities are all provided as services. The widely distributed computing resources can be used to host migrated services to potentially minimize the communication cost. However, not all services can be migrated. Services based on hardware resources are less flexible and cannot be migrated (not in the cyber world). When the services involve common hardware devices, the devices, even though non-migratable, are likely to be all over the place. Thus, it is possible to select one that can result in minimized communication cost. When a service involves specialized hardware, then it cannot be migrated. Services can potentially be migrated, but the migration costs and gains have to be evaluated to ensure net performance gains.

In this paper, we consider the problem of service selection and migration in a cloud. We develop a framework to facilitate service migration and design a cost model and the decision algorithm to determine the tradeoffs on service selection and migration. The important issues addressed in this paper include:
1. It is necessary to consider the infrastructure support in the cloud to achieve service migration. The computation resources (computer platforms) in the cloud need to be able to support execution of dynamically migrated services. We develop a virtual machine environment and corresponding infrastructure to provide such support.

2. It is also essential to have a strong decision support to help determine whether to migrate some services and where to place them. The consideration involves the service migration cost, consistency maintenance cost, and the communication cost gains due to migration. We develop a cost model to correctly capture these costs and help determine the tradeoffs in service selection and migration in clouds. Then, we use a genetic algorithm to search the decision space and make service selection and migration decisions based on the cost tradeoffs.

The rest of the paper is organized as follows: the overview of the cloud model is discussed in Section 2. It presents all entities in the cloud model. In Section 3, we discuss workflow for service composition. Section 4 presents service selection and migration decision process, including constraints on the selection of concrete services and computing platforms. Section 5 summarizes the paper.

2 Cloud Model for Supporting Service Migration

We consider the cloud environment consisting of general computing platforms and services. The services can be software and data and some other non-migratable services. The general computing platform (GCP) is also a service. It is specifically categorized in the system due to its potential in providing the environment for hosting software and data services. Let \( G = \{ g_i \mid 1 \leq i \leq N \} \) denote the set of \( N \) GCPs in the system. Also, let \( S = \{ s_i \mid \text{for all} i \} \) denote the set of services besides the GCPs. \( s_i \) can be a software, data, or non-migratable service. \( G \) and \( S \) are dynamic sets that can change over time. We assume that each service has a resident GCP (RGCP) and we use \( RGCP(s_i) \) to refer to the resident platform for service \( s_i \).

Each GCP in the cloud uses the Virtual Machine (VM) technology to provide a uniform execution platform so various services can be migrated dynamically and executed at any GCPs. Virtual machine not only provides desired execution platforms for various services, but also achieves a certain degree of security/protection. It prevents the services running on a GCP from attacking the GCP or interfering other web services running on the same GCP.

To support service migration, it is also necessary to provide some management services, including the service migration decision (SMD) services, the service migration management (SMM) services, and the certificate authority (CA) services. To differentiate, we call the regular web services the application services and these services the infrastructure services. The SMD services provide algorithms for making migration decision for application services. It maintains needed information such as the GCPs’ and other cloud resources’ characteristics, overlay network topology, service access patterns, etc. and uses them for decision making. The SMM services maintain information of all services, including their migrated and replicated copies to help with discovery of near-by services to facilitate service composition. It can be viewed as an extension of the UDDI registry, but provides more advanced functionalities. Executing migrated services on GCP platforms may pose security risks to the GCP platform as well as to the services. Thus, it is necessary to have proper infrastructure supports for mutual authentication and access control among GCPs and services. We use distributed CA services to achieve this goal.

Figure 1 illustrates the service based cloud environment. In the cloud, application services and infrastructure services are hosted by GCPs. Cloud may incorporate other physical devices wrapped as services. Clients interact with the cloud to compose services for their tasks while the infrastructure services working transparently to discover and select the most suitable services and replicate the migratable ones if needed to minimize the communication cost and satisfy real time constraints. Client can be end users, enterprise customers, or software applications.

2.1 DHT Based Distributed SMM Services

In conventional SOA environment, information regarding services, including their names, interfaces, semantics, properties, etc., are maintained by UDDI (Universal Description Discovery and Integration). However, UDDI lacks the distributed concept and does not consider multiple service replicas. Thus, it is not suitable in cloud environment. We design a distributed SMM infrastructure using a distributed hash table (DHT).

SMM services are hosted by selected GCPs on strategic locations to facilitate fast accesses. These SMM services form a “P2PSMM ring” (as shown by the dotted links in Figure 1). The P2PSMM ring is based on the one-hop routing concept [6] where each SMM keeps the full routing table. Since GCPs on the clouds are generally reliable and widely distributed, the routing table change frequency is low. Thus, the routing table maintenance cost is not a major
concern. New SMM services may be created and existing ones may be removed based on their access patterns. Multiple hash functions are used for fault tolerance and improved access performance.

Each SMM maintains a GSP table and a service table. The GSP table contains all the GSPs and their corresponding information. The service table is a partial table, only contains the services that are hashed to the SMM based on the P2PSSM ring infrastructure. The names of the application service are used as the primary key in the P2PSSM ring. However, client search requests are generally based on keywords. Since keywords space can be huge, we use the Hilbert Space Filling Curve (HSFC) [7] technique to index the keywords. Each service is associated with a set of keywords. The keywords form a multidimensional keyword space where services are points in the space and the keywords are the coordinates. The HSFC algorithm converts the multidimensional keyword space to one dimensional index space, a HSFC index. The HSFC index is then mapped to the P2PSSM ring to facilitate efficient search.

In the SMM service table, each entry for a service is similar to the entry in the UDDI registry. However, more information is needed on SMM to support service migration and replication. Each service entry in SMM maintains the information of all replicas as well as their hosting GCPs. One of the GCPs is marked as the RGCP for the service. For each service, the SMM maintains service name, service definition, service type, service size, average request and response message sizes, and service replica information. Service type specifies whether the service is migratable.

For each service replica, SMM maintains its replica type, hosting GCP, guaranteed service latency, etc. Replica type is used to indicate a master entry or a replica entry. The master entry is the one hosted by the RGCP. We assume that a service replica has a limited lifetime. A TTL (Time To Live) field is defined for each service replica. When the TTL expires, the service replica is removed from the hosting GCP and the entry is deleted from the table. The TTL value for the master entry is set to 0, indicating that it never expires. It is also necessary to maintain the communication costs among services. However, it is impossible to maintain such costs for all pairs of services, thus each service instance maintains the communication costs to other services that it frequently interacts with. Figure 2 shows the information maintained in the service table.

In the GCP table, an entry corresponds to a GCP. It maintains the GCP IP address, its resource statistics, availability, and potential accounting information. Also, the longitude and latitude coordinates for each GCP is maintained. It can be used to calculate the geographical distance between GCPs and estimate the communication cost between services they host when the communication cost between the services is not available. Figure 3 shows the information maintained in the GCP table.

<table>
<thead>
<tr>
<th>Entry Fields</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Name</td>
<td>The name of the service.</td>
</tr>
<tr>
<td>Service Type</td>
<td>The type is defined according to access pattern of the service. For non-migratable service, type = 0; for migratable service, type = 1.</td>
</tr>
<tr>
<td>Service Definition</td>
<td>WSDL definition of the service.</td>
</tr>
<tr>
<td>Service Size</td>
<td>The size of the service (code or data).</td>
</tr>
<tr>
<td>Message Size</td>
<td>It stores the average request/response message sizes of the service invocation. The message size information is stored in the callee’s service entry. If the callee is a data service, the update message size is also stored.</td>
</tr>
<tr>
<td>TTL(Time To Live)</td>
<td>TTL defines the valid period of a service replica. After the duration of TTL, the entry will be deleted from the registry. For master entry, ttl=0.</td>
</tr>
<tr>
<td>Hosting GCP</td>
<td>The GCP hosting the service.</td>
</tr>
<tr>
<td>Service Latency</td>
<td>The bounded average service time on the hosting GCP.</td>
</tr>
<tr>
<td>Communication Cost</td>
<td>It stores the communication latency between some frequently interacting service pairs.</td>
</tr>
</tbody>
</table>

Figure 2. Service table entry description.

<table>
<thead>
<tr>
<th>Entry Fields</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>IP address and domain name of the GCP.</td>
</tr>
<tr>
<td>Resource Statistics</td>
<td>Resource usage information (CPU, memory, disk, network, VM) of the GCP.</td>
</tr>
<tr>
<td>Physical Location</td>
<td>The longitude and latitude coordinates for the GCP.</td>
</tr>
<tr>
<td>Availability</td>
<td>The ratio of the expected value of the uptime of the GCP to the aggregate of the expected values of up and down time of the GCP.</td>
</tr>
<tr>
<td>Service Charge</td>
<td>Service charge for using the GCP.</td>
</tr>
</tbody>
</table>

Figure 3. GCP table entry description.

2.2 CA Services and Security Protocols

Security is a critical issue in the cloud environment, especially when considering service migration. A malicious service may try to compromise the GCP or try to perform a DoS attack by consuming all the resources of the GCP. Also, one service from a GCP may exploit the security loopholes of another GCP and compromise the privacy and/or integrity of other services from other GCPs. On the other hand, a GCP may try to compromise the services it hosts.

Some of the security problems can be resolved using VM technology on GCP. However, many of the security issues still remain. Thus, it is necessary to build an authentication framework to control the accesses between GCPs and
services. We use distributed certificate authorities (CAs) to provide mutual authentications.

For a service $s$, we consider its residence $RGCP(s)$ as the owner. When the SMD service decides to migrate a service $s$ to a GCP $g$, it informs $RGCP(s)$ and $g$ to perform mutual authentication. $RGCP(s)$ and $g$ send their certificates to each other and contact their CAs to validate the certificates. During the authentication process, $RGCP(s)$ verifies the identity and trust of $g$ and the privileges for $s$ to use $g$’s resources are negotiated. When the migration decision is confirmed, both $RGCP(s)$ and $g$ inform SMD to continue with the migration process.

### 2.3 Virtual Machine Environment on GCPs

We use VM technologies [8] to support service hosting and service execution on GCPs. VM technologies support the setup of different operating systems on the same hardware with each VM being isolated from each other. Thus, the desired execution environments for specific services can be easily set up statically or dynamically and different services running on the same GCP are well protected from each other. The popular VM systems, VMware and Xen, also allow virtual machines being created without excessive overhead. Since Xen is an open source software and its commands (xm) can easily create, pause, and shutdown VMs, we use Xen for realizing the VM environment on GCPs.

Figure 4 shows the basic design of the VM environment on GCPs. A GCP can host multiple VMs and each VM can host one application or infrastructure service. To support service migration and secure service execution, each VM also runs a set of GCP management components, including Migration Manager, Security Manager, and Resource Monitor. When the SMD decides a service should migrate, it informs the Migration Manager to wrap the service and transfer it from a source GCP to a destination GCP. Service migration might pose security risks. Security Manager interacts with CAs and performs service validation, authentication, and authorization. Also the Security Manager responds to authentication requests issued by services from other VMs. Since VM isolates multiple execution environments and supports the ability to run multiple software stacks with different security levels, we use VM to enforce fine-grained access control to services’ and local GCP resources. Service Migration does not always generate performance gain. Migration decision needs to consider the resource usages of VMs, such as CPU, memory, and disk usages of VMs. The Resource Monitor is used to monitor the resource usages of the VM in real time. It periodically reports the resource usage information to the SMM to facilitate migration decision making.

---

**Figure 4. GCP environment using VM.**

### 3 Workflow Model for Service Composition

Workflow is a common model for web service composition. We consider that a workflow is composed of a series of abstract services and these abstract services can be grounded to concrete services [9]. OWL-S (Web Ontology Language for Services) provides flow control constructs to specify various service compositions in a workflow, including Sequence, Choice, If-Then-Else, Iterate, etc.

**Figure 5. A workflow example.**

Let $W$ denote a workflow, where $W = (AS, E)$, $AS$ is the set of abstract services in $W$, and $E$ is the set of directed and labeled edges in $W$. $AS = \{as_1, \ldots, as_n\}$, where $|W|$ is the number of the abstract services in $W$. The edge represents the invocation dependency between two abstract services. The invocation probability property is associated with each edge. Let $ipr(e_{ij})$ denote the invocation probability for $as_i$ to invoke $as_j$ when $as_i$ is executed. If $as_j$ is a data service, the invocation probability with read operation, $rpr(e_{ij})$, and the invocation probability with update operation, $upr(e_{ij})$, properties are also associated with the edge. We have $ipr(e_{ij}) = rpr(e_{ij}) + upr(e_{ij})$, where $as_j$ is a data service. Figure 5 shows an example workflow. We can see that the workflow starts from $as_1$. It flows to $as_2$ with two branches. The flow goes to $as_3$ and $as_4$ with 30% and 50% probability, respectively. Note that the flow may stop at $as_2$ with a 20% probability.

Each abstract service can be grounded by a set of concrete services. Let $CS(as_i) = \{s_{ij} \mid 1 \leq j \leq N_i\}$ denote the set of candidate concrete services for an abstract service $as_i$, where $N_i$ is the number of concrete services for $as_i$. SMD contacts SMM to obtain $CS(as_i)$ for all $as_i$ in $W$. When there are multiple replicas of the same concrete service, then only one of them is selected and placed in $CS(as_i)$ (we select the one hosted by RGCP).

Our goal is to select the appropriate concrete services for grounding the abstract services in a workflow so that the total service latencies and communication costs between services can be minimized and the real-time constraints for
the workflow can be satisfied. Also, when appropriate, some migratable concrete services can be migrated to appropriate GCPs to further reduce the communication cost.

4 SMD Service and Decision Process

SMD services are responsible for making service selection and migration decisions. It takes a workflow $W$ and the corresponding real-time requirements from a client as input. Based on $W$, SMD first interacts with SMM to identify the concrete services for each abstract service. Then, SMD selects concrete services to ground the abstract services and determines whether to migrate or replicate the concrete services on selected GCPs. Let $cp$ denote a configuration solution to the service selection and migration decision problem. Let $sel(as_i)$ denote the index of the concrete service in $CS(as_i)$ that is selected for grounding $as_i$ in $cp$, where $1 \leq sel(as_i) \leq N_i$. Also, let $loc(sel(as_i))$ denote the index of the GCP that hosts the concrete service, where $1 \leq loc(sel(as_i)) \leq N$. In other words, in a configuration solution $cp$, $as_i$ in $W$ is grounded to concrete service $s_i, sel(as_i)$ and $s_i, sel(as_i)$ is selected to be hosted on GCP $g_{loc(sel(as_i))}$. In the genetic algorithm, the configuration solution $cp$ can be mapped to an individual directly as shown in Figure 6.

![Figure 6. An individual representing a configuration solution.](image)

We use the genetic algorithm to search for the optimal configuration solution, i.e., to determine the best settings of $sel(as_i)$, $1 \leq sel(as_i) \leq N_i$, and $loc(sel(as_i))$, $1 \leq loc(sel(as_i)) \leq N$. In the following subsections, we define the constraints and the fitness function for the genetic algorithm.

4.1 Constraints on the Selection of Concrete Services and GCPs

For each $as_i$, if it is non-migratable, then $g_{loc(sel(as_i))}$ has to be $RGCP(s_i, sel(as_i))$. If it is migratable, then either an existing GCP or a new GCP can be chosen to host $s_i, sel(as_i)$. Due to the real-time requirements, it is necessary to have admission controls by both the GCPs and the services. For a given configuration solution $cp$, it is necessary to check whether the configuration would violate the admission constraints. The admission constraint verification procedure is shown in Figure 7.

The algorithm takes $s_i, sel(as_i)$ and $g_{loc(sel(as_i))}$ as inputs. It first invokes search_SMM function to look up the service table on SMM. If service $s_i, sel(as_i)$ is already on $g_{loc(sel(as_i))}$, then search_SMM function returns the average service time guaranteed by $s_i, sel(as_i)$.

Next, SMM checks whether $s_i, sel(as_i)$ can admit $W$. If so, then $sg$ is returned. Otherwise, the algorithm returns 0. If service $s_i, sel(as_i)$ is not on $g_{loc(sel(as_i))}$, then $s_i, sel(as_i)$ needs to migrate to $g_{loc(sel(as_i))}$. So, the algorithm checks if the $g_{loc(sel(as_i))}$ has enough resources to create a new VM. If yes, then $g_{loc(sel(as_i))}$ computes and returns the guaranteed average service time for $s_i, sel(as_i)$ on $g_{loc(sel(as_i))}$. Otherwise, the algorithm returns 0. Note that when the return value is 0, it means the admission constraints are violated and the configuration solution should be rejected from the population. If the configuration solution is valid (positive return value), then the returned value is the guaranteed service response latency, which is used in the fitness value computation.

![Figure 7. Admission constraint verification procedure.](image)

4.2 Fitness Function

The fitness of a solution $cp, fit(cp)$, is defined by

1. $T(cp)$, the total response latencies of the selected concrete services $s_i, sel(as_i)$ executing on the selected GCP $g_{loc(sel(as_i))}$;

2. $C(cp)$, the total communication costs between the concrete services $s_i, sel(as_i)$ on $g_{loc(sel(as_i))}$;

3. $M(cp)$, the migration cost for all $s_i, sel(as_i)$ on $g_{loc(sel(as_i))}$ yet and needs to be migrated;

4. $U(cp)$, the consistency maintenance cost for all $s_i, sel(as_i)$, if $s_i, sel(as_i)$ is a data service and needs to be migrated.

We have $fit(cp) = T(cp) + C(cp) + M(cp) + U(cp)$. 

138
The service time of $s_{i,sel(\text{AS}_k)}$ on $g_{loc(sel(\text{AS}_k))}$ is obtained in the admission constraints verification step. We add the service times of all selected concrete services on the selected GCPs together to obtain $T(cp)$.

Let $k(g_m,g_n)$ denote the estimated communication latency between GCPs $g_m$ and $g_n$. In the SMM, the communication latencies between some frequently interacting service pairs are stored. If the communication latency between some service pair is not available, then we can use the longitude and latitude coordinates stored in the GCP table to estimate the communication latency between the services. In [9], experiment studies have been conducted to derive the correlation between geographical distance and communication latency and the estimation function is $k(g_m,g_n)=0.4+0.3d(g_m,g_n)^{0.735}$, where $d(g_m,g_n)$ is the geographical distance between $g_m$ and $g_n$. Let $psz$ be the packet size. If $s_{m,sel(\text{AS}_k)}$ invokes $s_{n,sel(\text{AS}_k)}$ (which can be determined by traversing $W$), then SMD checks with the SMM to obtain the average request message size, $msq( s_{n,sel(\text{AS}_k)} )$, and the average response message size, $msr( s_{n,sel(\text{AS}_k)} )$. Note that the message sizes are stored in the callee’s service entry. SMD also checks the workflow $W$ to get the access frequency information, $ipr(e_{mn})$. The communication cost between the concrete services $s_{m,sel(\text{AS}_k)} \oplus g_{loc(sel(\text{AS}_k))}$ and $s_{n,sel(\text{AS}_k)} @ g_{loc(sel(\text{AS}_k))}$ is $ipr(e_{mn}) \times (msq(s_{n,sel(\text{AS}_k)})/psz) + (msr(s_{n,sel(\text{AS}_k)})/psz) \times k(s_{m,sel(\text{AS}_k)}, s_{n,sel(\text{AS}_k)}, g_{loc(sel(\text{AS}_k))})$.

$C(cp)$ is obtained by adding the communication costs of all interacting service pairs together.

Now we compute $M(cp)$. Let $M(s_{i,sel(\text{AS}_k)})$ be the migration cost for $s_{i,sel(\text{AS}_k)}$. If $g_{loc(sel(\text{AS}_k))}$ already hosts $s_{i,sel(\text{AS}_k)}$, then $M(s_{i,sel(\text{AS}_k)}) = 0$. Otherwise, SMD checks with the SMM to obtain the size of service $s_{i,sel(\text{AS}_k)}$, $size(s_{i,sel(\text{AS}_k)})$. We have $M(s_{i,sel(\text{AS}_k)}) = [size(s_{i,sel(\text{AS}_k)})/psz] \times k(s_{i,sel(\text{AS}_k)}, RGCP(s_{i,sel(\text{AS}_k)}))$. $M(cp)$ is obtained by adding the migration costs of all services together.

For a data service, it is necessary to maintain consistency among all its replicas. Here, we consider a simplified model and assume that the residence GCP (RGCP) for the data service is the primary copy of the data. We only consider the cost for propagating the updates from a replica to the primary copy. If software service $s_{m,sel(\text{AS}_k)}$ invokes data service $s_{n,sel(\text{AS}_k)}$ and updates its data, then the updates on $s_{n,sel(\text{AS}_k)}$ need to be propagated to $RGCP(s_{n,sel(\text{AS}_k)})$. Let $U(s_{n,sel(\text{AS}_k)})$ be the consistency maintenance cost for $s_{n,sel(\text{AS}_k)}$. If $s_{n,sel(\text{AS}_k)}$ is already hosted by $RGCP(s_{n,sel(\text{AS}_k)})$, then $U(s_{n,sel(\text{AS}_k)}) = 0$. Otherwise, SMD checks with the SMM to obtain the average size of the update messages, $msu(s_{n,sel(\text{AS}_k)})$. Also the access frequency $ipr(e_{mn})$ and the update ratio $upr(e_{mn})$ (the percentage of the requests being the update requests) can be obtained from the workflow $W$. we have $U(s_{n,sel(\text{AS}_k)})=(\sum_{m \in W} ipr(e_{mn}) \times upr(e_{mn}) \times [msu(s_{n,sel(\text{AS}_k)})/psz] \times k(s_{m,sel(\text{AS}_k)}, RGCP(s_{n,sel(\text{AS}_k)})))$

$U(cp)$ can be obtained by adding the consistency maintenance costs of all data services together.

5 Conclusion

We have discussed the problem of service selection and migration in a cloud. We have developed a framework to facilitate service migration and designed a cost analysis model to determine the tradeoffs on service selection and migration. The genetic algorithm is used to find the optimal or near-optimal service migration decisions.

6 References