2012 IEEE Asia-Pacific Services Computing Conference

Dynamic Resource Arrangement in Cloud Federation

Yisheng Wang, Haopeng Chen
School of Software, Shanghai Jiao Tong University
Shanghai, 200240, China
Email: easonyq@hotmail.com, chen-hp@sjtu.edu.cn

***Abstract*—Cloud Federation is one of the ideal solutions to this random burst traffic problem. It focuses on 'borrowing' computing resources from foreign Clouds when home Cloud is about to overloaded and 'leasing' resources to foreign Clouds when home Cloud is free. Considering crucial importance of business value for a Cloud provider, we need to perform some dynamic, reasonable and simple rules or mechanisms to help Cloud provider making better billing decisions and improving the overall performance in a federation scope. Cloud Federation is implemented by network connections, so in common situation the best candidate Cloud to be federated enjoys a higher speed of connection to the home Cloud. This remains a problem that several Clouds located in a subnet have a high possibility to be federated with each other, thus complicated dependency relations among them will appear. In this paper, we have carried out a series of mechanisms involving dynamic resource arrangement in establishment and deconstruction of a Cloud Federation, in order to sort out these dependency relations which are the potential risk factors to the overall performance.**

***Keywords*-Cloud Federation, dynamic resource arrangement, dependency relation;**

I. INTRODUCTION

Most people cannot distinguish Cloud from other "popular" definitions such as grid computing, distributed computing and parallel computing. All of them share the notion that "we do not compute on local computers, but on centralized facilities operated by third-party compute and storage utilities" [9]. Actually, Cloud Computing puts more emphasis on providing users with computing resources (such as storage, CPU, memory, etc) in Iaas, Paas and Saas mode while Grid Computing is project-oriented and focus on only computing resources in most cases [10] [11]. Although problems like stability of Cloud-app and security of data transmission still remain unsolved, it is widely believed that Cloud will have a promising future and an increasing proportion of online application will be moved to Cloud Platform [20] [21]. Cloud Computing has many advantages that attract increasingly more business users such as low deployment cost, high deployment speed, pay-as-you-go and service flexibility. Modern Cloud platforms like AWS [17] and Amazon EC2 serve as IaaS, Google App Engine [6] and Windows Azure provide users with services in SaaS and PaaS mode. This trend of building up their commercial Cloud platform in many IT companies is also implying this idea.

Cloud Federation is proposed as the next hot-spot in Cloud researching field, but researchers do not reach an agreement in details. Many new terms have then been coined as "Inter-cloud" [16] or "Cross-cloud" [15]. The ultimate goal of Cloud Federation is to enable message transmission and collaboration among Clouds so that resources located in different Cloud platform can be used to serve a single service. Cloud can 'borrow' some virtualized resources from other Clouds. Similarly, free resources can also be 'leased' to other Clouds for business consideration. In federation story, client requests can always be served, even if the target Cloud is already saturated. Home Cloud is able to serve more customers than its original capacity through 'borrowing' resources. And free resources can also be 'leased' outside, which is better than either leaving it running idly or simply shutting it down [7]. But it also due to some problems such as the chaos of dependency relationships, too much resource debris, etc. These can lead to the decrease of resource utilities, so we should try to avoid and solve these problems.

The rest of paper is organized as follows: Section 2 lists some motivations of our work. Section 3 shows our approach about how to deal with the resource leasing in Cloud Federation. Section 4 summarizes some simulation evaluations of the mechanisms. Section 5 introduces related work by others recently. Section 6 summarizes the main contribution of the paper and comments on further research.

II. MOTIVATION

In order to investigate the federation situation more clearly, we made a dependency diagram to show its 'borrowing' relationship among Clouds, in which Clouds are illustrated by a rectangle, sometimes with a unique filling color or pattern. We put a solid arrow from Cloud A to Cloud B if there exists a 'borrowing' relationship from Cloud A to Cloud B, or said, Cloud A uses some part of the resources from Cloud B by redirecting client requests or copying compiled source codes and running remotely. Such diagram is called 'Resource Dependency Flow' in Cloud Federation which is shown in figure 1.

In Resource Dependency Flow Diagram (for short, RDFD), there may exist simultaneous appearances of borrowing and leasing in a Cloud. A more common situation is to form a ring, which we call it 'resource dependency ring'. It would make a negative impact to the overall performance. Our

978-0-7695-4897-5/12 $26.00 © 2012 IEEE
DOI 10.1109/APSCC.2012.75

Fig. 1. Resource Dependency Flow in Cloud Federation

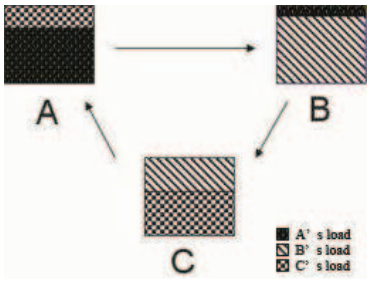

Fig. 2. Resource Dependency Ring in Clouds

mechanisms try to eliminate this situation along with the ring, thus improve performance of running services along with the business benefits for a Cloud provider. Figure 2 shows a common resource dependency ring in Cloud Federation. Here resource dependency ring is displayed intuitively, with several Clouds and circular arrow pointing to each other. All the three Clouds borrowing and leasing resources at the same time.

Cloud Federation is implemented by deploying or migrating services to foreign Cloud. If this service needs data in the process of running, it will make a data fetch request to home Cloud. Note that it is not reasonable to transmit all data to foreign Cloud along with compiled source codes, for three reasons [1]. First, data may be huge enough so that transmitting time becomes unacceptable. Second, it is most the cases that common services only need a small part of data, rather than all, so it is quite time-consuming to transmit much unnecessary data. Third, and the most important one, users perhaps do not trust foreign Cloud enough, thus it is arbitrary and irrational to transmit them without user's approval. Given these, we can easily get the conclusion that time consumption of services on remote nodes is more than that on local nodes due to the time consumption of data transmission, let alone multiple times of transmissions involved in some data-intensive services. To conclude, we need to prevent the simultaneous appearance of borrowing and leasing resources in one Cloud. This is also the main goal of the resource arrangement mechanism performed in this paper. An ideal dependency relation started from Figure 2 is shown in Figure 3.

It is clear that the situation after re-arranging is better than the original one mainly on the fact that only Cloud B need to borrow resources from Cloud D, and both Cloud A and Cloud C run their services locally. The rule we must observe is borrowing and leasing resources should not happen in a Cloud simultaneously.

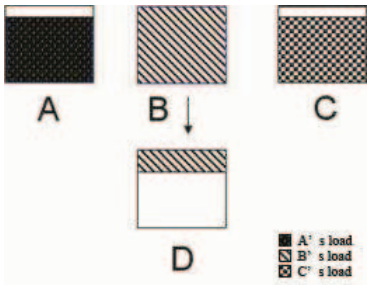

Fig. 3. RDFD after Re-arranging

III. APPROACH

How to deal with the federated resources will be discussed in the following, involving some algorithms and mechanisms. We divide the federation strategies into two parts, dealing with scaling-up and scaling-down respectively. Scaling-up means a Cloud need to scale up by borrowing resources from other Clouds in order to serve extra client requests. In scaling-up part, we focus on when and how to borrow resources from other Clouds, along with some detailed rules and points that we must obey to ensure the performance and rationality of the federated system. Scaling-down means a Cloud has already borrowed some resources from other Clouds, and then some local resources are released. It should move its insourced services from remote Cloud to local, in order to avoid the simultaneous appearance of borrowing and leasing. The effectiveness of this moving action will be discussed in the following sub-section, and the simulation evaluation about this moving will be displayed in Section 4.

A. Scaling-up part

There are varieties of resources in a Cloud, such as CPU, memory, storage, network bandwidth, etc. Dynamic resource arrangement in Cloud Federation will involve all these different types of resources, or some certain combination of these resources. Moreover, resource usage is varying all the times during running period of all services. It is believed that monitoring all these resource usage will be a tough job, because we are able to get information about all the resource usage of each computer, but difficult to gain the exact information of each VM [14]. We should put this problem aside and make an assumption that we are able to get all these information at any time, just like taking photographs to all resources in a Cloud. Moreover, for easiness consideration, 'resources' in our model can be thought as a kind of high level and abstraction of generic resources. It should be admitted that different type of resources varies much. For example, difference of CPU usage in different period of a running service can be very notable while storage usage is relatively stable. Different arrangement towards different type of resources is also important, which we will leave to our future work.

If we take a look at a Cloud in Cloud Federation, we can roughly divided it into three parts, named *OutRent* part, *Local* part and *Free* part respectively.

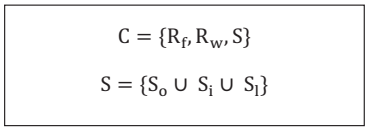

Fig. 4. Cloud Model

OutRent part means resources here are leased to other Clouds. Services from other Clouds can be served here for profits of leasing. Each service has its rent time and rent level which indicates whether it can be moved at anytime. We have designed a configurable property indicating three rent levels and this property should be provided when resource leasing takes place. Level 1 is the lowest level which means this service can be moved to another Cloud at any time without informing the original Cloud provider. Level 2 is similar, but the foreign Cloud should query first whether this service can be moved. Level 3 is the highest level indicating the service cannot be re-located until it is completed. Cloud provider can ask for a different charge towards different rent level, normally higher level charges more. Services running in *OutRent* part need to send data transmission request to federated Cloud if they need data in their running period, and that maybe the dominating drawbacks of their total execution time.

Local part contains resources for services running locally. According to our views *Local* part should be allocated first. Only when *Local* part cannot hold more services should these services be located remotely. Services running locally do not need to transmit data through network connection. Although reading data from Cloud Storage may still be the bottleneck of this service, it is affirmatively faster than running remotely if other conditions keep the same.

Resources in *Free* part are on standby and can be allocated to serve requests at any time. From a business view, *Free* part should be as small as possible because *Free* part does not generate any revenue but asks for standby costs such as the electricity consuming, cooling, etc [12]. It should also be pointed out that reserving *Free* part is essential for a Cloud provider, otherwise the violation of SLA cannot be fixed at a short time, thus leading to a huge SLA compensation. After classifying the resources in a Cloud, we can carry out our model of Cloud, which is represented by 'C' in Figure 4.

Because we are concentrating on the resource arrangement in a Cloud, it is sensible to regard Cloud as the collection of all resources. In the first formula of this model, we use R_f to represent free resources in a Cloud, R_w to represent the total resources in a Cloud and S to stand for all services, including *OutRent* services and *Local* services. Moreover, S can be further divided into three parts, S_o means OutRent services, S_i means insourced services and S_l means *Local* services. It should be noted that insourced services are remote services which running on remote Cloud. But their information is saved locally, for both security and charging reasons. If there are not any OutRent services, then S_o is an empty stack. The same rule also happens on S_l and S_i.

It can be inferred that it is the allocate resource request sent from upper layer which driving the scaling up action. So we should re-write the ***alloc_resource*** method in resource manager layer of a Cloud, which takes the amount of request resources as the only input parameter. The detailed algorithm is shown in algorithm 1.

Algorithm 1: Alloc_Resource

input : R_r Amount of resources needed
output: Resources allocated locally or on remote nodes

```
1  if R_f ≥ R_r then
2      return alloc_local_resource(R_r);
3  end
4  if R_w ≥ R_r then
5      if requester is a Cloud provider then
6          return null;
7      end
8      An empty list MigrateQueue;
       // we cannot use this.N_o == 0 to
          judge whether more services can
          be moved.
9      while (R_f < R_r + MigrateQueue.getTotalRes())
       AND this.hasMoreServicesToBeMigrated() do
10         {s, C} ← findDestination(this)
           // Migration Unit {Src, Dest, s}
              means Service s moving from
              Cloud Src to Cloud Dest
11         MigrateQueue.add({this, C, s});
12     end
13     if R_f < R_r + MigrateQueue.getTotalRes() then
14         while MigrateQueue.hasElement() do
15             migrate(MigrateQueue.pop());
16         end
17         return alloc_local_resource(R_r);
18     end
19     return this.rentRes(R_r);
20 end
21 while this.hasMoreServicesToBeMigrated() do
22     {s, C} = findDestination(this);
23     migrate(new MigrateUnit(this, C, s));
24 end
25 Create empty resource container returnRes;
26 add alloc_local_resource(R_w) to returnRes;
27 add this.rentRes(R_r - R_w) to returnRes;
28 return returnRes;
```

The only input parameter R_r means the amount of resources needed. In our algorithm, we divided all situations into three parts. The first part starts from line 2 to line 4. This is a very simple and easy situation meaning free resource is bigger than requested resource, thus we should allocate local resource directly to upper layer and obey the rule that local resources should be allocated first. The second situation starts from line 4 to line 24 indicating the requested resource is bigger than the left free resource, but still smaller than the total resource.

This situation mostly happens in the peak of the work-load and the value of Cloud Federation reflects here. Firstly, a migrate-queue is created in order to save all the services which needed to be migrated. Then we select some *OutRent* service and find a proper destination Cloud for it. After finding such a pair, we put it into this queue and check again whether free resource is enough to allocate to serve this request. If so, we can migrate all services in migrate-queue and allocate free resource directly (line 19). If not, that means even if we migrated all *OutRent* services outside, we could still not get enough free resource to allocate. Since federation cannot be avoid and context-switch and transmitting of codes will take extra time, it can be easily derived that the number of migrated services should be as less as possible. That is also the reason why we need to create a migrate-queue, rather than migrating services directly. Under this situation, what we should do is to keep all *OutRent* services and send a federation request to other Cloud, in order to locate this new request remotely. We must point out that if the requester is a provider, we should reject this request in this situation. Because this will only add unnecessary dependency relations among Clouds. The rest of algorithm is dealing with the third situation, which serves for the possibility that the requested resource is bigger than the total resource of a Cloud. Although the probability of this situation is little enough for us to ignore it because resources in a Cloud is fairly huge towards a single service, we still add it to our codes for the integrity and robust of the algorithm. In this case, we should release as much *OutRent* services as possible to ensure more part of the big service locating locally. Because of the big size, federation is also inevitable. Actually we can still stick to the rules displayed in the second situation, which says that we should keep the *OutRent* services, but considering the business reasons and the performance of the big service, more local resources should be allocated to serve it.

The function ***findDestination*** at line 7 is implemented quite briefly by invoking two core methods, responsible for selecting a service out of *OutRent* services and finding a destination Cloud to put it. The detailed implementation depends largely on the business strategies of Cloud provider according to client credits, historical trade data, charging price, etc. Each provider should provide the strategy it use to pick out the candidate service.

We should also note that only services with their rent level lower than or equal 2 are able to be migrated. So each time we check if there are any more services which can be migrated we use a function ***this.hasMoreServicesToBeMigrated***(), rather than checking whether this.$|S_o|equals0$.

Another point which needs to be further discussed is that we consider a service as an entirety which cannot be divided further. No matter we decide to run it locally or remotely, it is not recommended to divide it into several pieces and located these pieces in different Clouds. An obvious shortage of this rule is the internal fragmentation, but after knowing the reasons of this rule, its disadvantage can also be accepted. It is known that application today always contains some transactions which need the manager to take some extra care. When it comes to a distributed system, this 'extra care' will become difficult enough to make management cost boost. Consistency and deadlock are two common problems in distributed system, and the division of a service will cause these two problems, thus increasing the cost and complexity of technical management.

B. Scaling-down part

In the previous sub-section we have discussed our algorithm when dealing with allocating resources in a Cloud, or said scaling-up. And we want to perform the most important and only rule in scaling-down period: Moving *OutRent* services to local node as soon as possible.

Scaling-down period happens when some resources are released and *Free* part increased. There are two situations leading to scaling-down period. The first happens when leasing times of *OutRent* services end or the borrower decides to end the renting relationship. The second is the completion of local services. The increasing of free resources means that it is able to run some extra local services now. We should point out that for those services which are allocated remotely, their source codes has already been kept a copy locally. So when moving them to local Cloud, we only need to save the running context in remote Cloud and transmit these context information. Considering the rule about maintaining the priority of local resources, we need to move *OutRent* services to local node and end the renting relationship if allowed. The reasons are listed in the following.

1) Time consumption in data transmission: The implementation of Cloud Federation has decided the fact that services executing on remote nodes need data transmission when it is running. For most services which dealing with querying in database, the amount of data which need to be transmitted is quite huge. Comparing with the speed of reading local data in Cloud, the transmission speed of network connection is much lower. Usually, we have two methods of transmitting data. We can transmitting all data at the beginning or transmitting needed data according to the request sent from services. Both methods will do great harm to the overall performance of Cloud Federation.

Transmitting all data at the beginning can arise two problems. The first and most obvious one is that the data could be very large, large enough to make the transmission unacceptable. It should also be noted that some services do not need to query much data as input, but will generate much data as output. Such cases include the computing services involving multiplication of several vectors. Another problem lies in the consistency of two copies of data. Because the *OutRent* service is transmitted to other Cloud, both home Cloud and foreign Cloud will keep its data copies. In this case, remote copy should be thought as the primary copy, but the extra cost of management of two data copies still adds a strict limit to this method.

Transmission according to requests seems like a good idea because it is consistent to the business model in Clouds: pay-as-you-go. But it still faces the problem of too much transmission times. Network connection needs three times

hand-shake in TCP protocol [2]. Its time consumption can be roughly regarded as a constant. Too much transmission times mean too much time is wasted in making network connection, or more specific, three times hand-shake. But comparing with the last method it is better. So we use this method as the comparison in our simulation evaluation.

2) Business factors: There are generally two paying methods in Clouds: pre-paid and post-paid [8]. Pre-paid means paying before using and post-paid is the opposite. Generally, users may choose post-paid to minimize the wasting time of occupying computing resources. Moreover, Cloud may have a mechanism that once pre-paid rent fee is running out, it will automatically change into post-paid mode. If a service is set to post-paid, moving *OutRent* it to local node as soon as possible can also do contribute to reduce costs because less occupying time is used.

Actually, the decision about moving *OutRent* service to local node should also be made carefully. Although in many cases this rule will help to avoid the simultaneous appearance of borrowing and leasing, it should still be cautious to be executed in the following situations.

1. Services with small-scaled data

If a service does not need large-scaled data as its input or output, its remote execution time will not differ much comparing with its local execution time because time consumption on data transmission through network connection is not so huge. Such services contain scientific computing without much input and output, stateless web-application with little or even no database operations, etc.

2. Services are almost completed

According to our moving strategy shown earlier, time is mainly cost in context-switching and transmitting context information when moving *OutRent* services to local Cloud. But if a service is almost completed, it is recommended to wait for its completion. More specific, if the sum of context-switching time and transmitting context information time is bigger than the rest renting time, moving is unnecessary. Further, because the time consumption of context-switching and transmitting context information is quite stable, it is reasonable to consider it as a constant. In other word, user or Cloud provider can set a threshold to represent this constant. Once the left renting time is smaller than this threshold, moving it back will be rejected by the resource manager layer.

C. Summary

In this section, we have carried out algorithms and mechanisms dealing with scaling-up and scaling-down period respectively. First, this mechanism and algorithm ought to be put in the Cloud itself rather than some third-party central nodes because Cloud Federation should be regarded as an extension functionality of a Cloud. In scaling-up period, we have performed an algorithm to handle three situations happened in resource allocation. The main principle in resource allocation is to keep the integrity of a service and to ensure that local resource should be first allocated. If federation cannot be avoided, keeping *OutRent* services and borrow resources from other Clouds is recommended. In scaling-down period, we stick to a regulation that we need to move *OutRent* service back as soon as possible mainly because of performances' and business' consideration. Two situations are exceptional. They are services with small-scaled data including both input and output, and almost completed services. We will bring out the simulation evaluation of this regulation in scaling-down period in the next section.

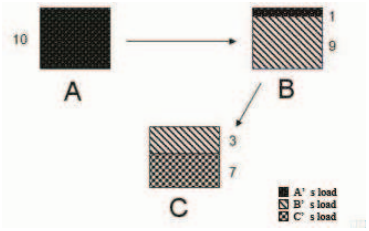

Fig. 5. Initial RDFD

IV. EVALUATION

In this section we want to give out some simulation evaluation of our mechanism proposed in Section 3. Our mechanism contains two parts indicating scaling-up and scaling-down period. We want to demonstrate a case first in order to show how it works, and then give out our simulation evaluation about its improvement.

A. Case Study

Scaling-up part is to deal with the resource allocation decisions to determine whether we should allocate services on remote node. Since no one can predict how much resources next client request asks, it is meaningless for us to weigh it in quantitative way. So our evaluation will focus on a case to show how mechanism works to eliminate resource dependency ring.

Just taking the resource dependency ring described in Figure 2 as an example, we will explain its possible forming process to show it is not just an assumption. We assume the initial situation as illustrated in Figure 5.

We want to make some complement to this initial situation. We assume that all Clouds share a same size, which means they have the same resources, and we marked it 10 units. At first, Cloud A was fully occupied, Cloud B was 90% used and Cloud C was 70% engaged. Then Cloud A received a request which asked for 1 unit of extra resource. So it asked B for leasing. After that, B received a request asking for 3 units of resources, and similar leasing relationship happened from B to C. Thus, the situation illustrated in Figure 5 happened.

After the initial situation, we want to display how our mechanism works. Assume that at some moment a service running in Cloud A occupying 2 units of resources completed and released all its resources. Then Cloud C asked A for 2 units of resources. If Cloud A did not comply with the scaling-down rule, it would possibly arrange these 2 units of resources directly to C, thus a resource dependency ring happened just

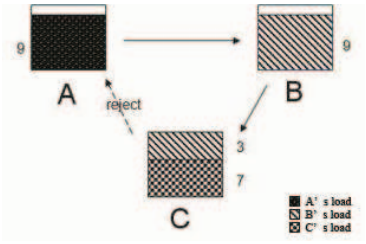

Fig. 6. Initial RDFD

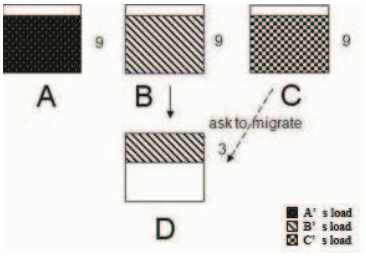

Fig. 7. Final RDFD after Using Mechanisms

the same as illustrated in Figure 2. Otherwise, if it stuck to the scaling-down mechanism, it should first withdraw the service which was running in B and occupied 1 unit of resource just after the 2 units of resources was released. So when C wanted to ask for leasing, Cloud A cannot provide any more resources. It would reject the request sent by C and at the same time clean the leasing relationship from A to B, thus a potential resource dependency ring was eliminated. More intuitive illustration can be seen in Figure 6.

The rejection of leasing resource would not be the whole story. For Cloud B, 1 unit of resource were released and it would also execute the scaling-down mechanism. But unfortunately, in order to assure the integrity of services, nothing would be done in Cloud B. Actually, before C asked A for leasing, it would try to migrate its OutRent services to other Cloud if possible according to the scaling-up algorithm. So C might find some Cloud named D and migrated services with 3 units there. At last, it arranged its new request of service asking for 2 units of resources locally, thus the final situation is illustrated in Figure 7, which is similar to Figure 3 and at the same time assuring the integrity of services.

B. Simulation evaluation

Our comparison is to calculate the difference between keeping OutRent services running on remote node and moving OutRent services back if possible. We will first demonstrate our computing model and explain the rationality of this model.

The first situation is to calculate the original time consumption. It has been already mentioned above that we choose to transmit data as requested, so both the amount of whole data and the number of request times are important factors influencing the final result. Furthermore, after processing input data services will give out some output data, and the proportion of output and input is defined as the 'return back rate'. Thus the formula calculating the original time assumption is as follows.

$$T_{original} = n_p * (\frac{Size_p * (1+R)}{S_{network\_remote}} + T_{c\_remote}) \quad (1)$$

In this formula, T_c stands for the time consumption used for creating network connection. According to the different network condition, we divide it into remote and local situation, corresponding to foreign Cloud and home Cloud. Besides, we divide all data which need to be transmitted in the running period into lots of chunks, once a time when being transmitted, and we assume that each chunk share the same size. Here R means the 'return data rate', n_p means the number of chunks, and $Size_p$ means the size of each chunk. The last two must obey Equation 2.

$$Size_p * n_p = Size_{all} \quad (2)$$

$S_{network}$ represents the speed of transmission data based on network connection, and it can also be divided into two situations similarly to T_c. This formula reveals that the original time consumption relies on the whole size of data, the number of requests and the size of the return data. Because R is determined by the specific workload, so we roughly assume it as a constant.

In the second situation which is recommended, the time consumption is mainly made up with the creation of network connection, transmission of context information and data transmission. When we decide to move it back, we need to save the context information at that time, transmit it back and resume it in order to continue running services locally. After that, data transmission is also needed when running locally, but the speed of network is different. According to the research on the life migration of virtual machine presented in Kemari project [22], time consumption on switching context can roughly be ignored comparing to it on the transmission of context information. So we get the following formula.

$$T_{optimized} = T_{context} + T_{data} \quad (3)$$

$$T_{context} = T_{c\_remote} + \frac{Size_{context}}{S_{network\_remote}} \quad (4)$$

$$T_{data} = n_p * (\frac{Size_p * (1+R)}{S_{network\_local}} + T_{c\_local}) \quad (5)$$

Because we ignore the time consumption on switching context, the final time roughly equals the sum of two parts. The first part is spent for the transmission of context information, and the second part is still for the data transmission, but based on local network connection. In short, the difference between original situation and optimized solution is the different network condition and the time spent on transmitting.

After modeling these two conditions, we want to give out the final result using some real numbers. There are some constants in our model, and we want to give them some simulation values respectively. According to the researches performed in distribution system field [2] [13], creation of network connection based on the Internet takes 100ms to 500ms in most cases, so we set T_{c_remote} to 300ms, and

creation time based on LAN or MAN ranges from 1ms to 10ms, so we set T_{c_local} to 50ms. Similarly, $S_{network\_remote}$ is set to 1Mbps, and $S_{network\_local}$ is set to 10Mbps. R depends on the real workload very much, but for simplicity, we set it to 0.6 as a combination of reading operation and writing operation. We also give three values for $Size_{all}$: 1M, 10M and 100M, three values for n_p: 100, 1,000 and 10,000, and at last, three values for $Size_{context}$: 100K, 1M and 10M. So we get the table shown in Figure 8 after calculation.

The improvement percentage between optimized time consumption and original one ranges around 10% to 20%, which is a real great promotion. So we want to point out here that the results shown in the table is the maximum improvement of each condition. If we think about the calculation formula again, we can find that this calculation is based on the moment when services begins. In most cases we need to make this type of optimization decision in the middle of services' running period, which means the improvement percentage cannot be so great. Besides, it has already been shown that there exists a threshold and it will be meaningless to make optimization when the left running time is less than this threshold. So an important conclusion of this optimization is that the less the running time is left, the less the improvement percentage will be.

V. RELATED WORK

Researches on Cloud Federation Model can be roughly divided into two parts. Their difference mainly lies in the architecture of their implementation. The first view is to add a Cloud Federation Center between users and Clouds which is responsible for the forming and removing of Cloud Federation. This view is proposed by Rajkumar Buyya [18] and it's somewhat similar to the existing SOA architecture. Once some users commit a service to the federation center, it check the service first and then arrange some resource units (or said VMs) to serve for the service. Of course these units may come from different Clouds, and the federation is then formed. In Buyya's InterCloud model, Cloud Broker and Cloud Exchange acts as this center and its advantage mainly lies in that not too many changes will be involved for existing Cloud architecture. It is also obvious that single point failure should be the main shortage for this architecture. Comparing with the first view, more researchers choose to weaken the federation center. In their models there still exists a center node, which can be called a 'directory center'. It is no longer designed to decide the forming and removing of the federation, but mainly focuses on storing information for Clouds and support publish-and-subscribe mode. Both federation model proposed in the research of Cross-Federation by Antonio Celesti [5] and "RESERVOIR" model [19] proposed by IBM share this view. Besides the directory center, there also exists a layer in their models which is responsible for the discovery and communication among Clouds. After the Cloud receives a request, it asks for this layer to allocate remote resources if needs. So this layer sends a query to the directory center for other Clouds who has free resources and prepared to lease. After that, these Clouds communicate with each other, and the directory center is no longer involved in this federation. Although the federation logic is more complicated, the reduction of pressure on the center node and more flexibility that Cloud gain are the key feature against the first model. Moreover, Celesti has proposed a three-phase solution to set up a federation ("discovery, match-making and authentication" [5] [3]) which can be treated as a general method. He also introduced three detailed protocols used in these three phases respectively.

There also exist some other researches emphasizing on some other forms of federation of Cloud. Zhang [23] has presented a Mobile-based Cloud Federation solution named MABOCCF. The collaboration of Cloud is realized by the location movement of mobile phone, which is called "passive" federation. Casola's [4] research mainly focus on the combination and cooperation between Cloud computing and Grid computing, but performance is its dominating drawback. Except for the architecture of Cloud Federation Model, there are also some researches focusing on billing decision and economic model of federation. Goiri [12] has performed the best billing decision under different conditions such as local only, insource, outsource and hybrid respectively. He introduced some parameters and listed a series of inequations to help Cloud providers making their decisions among insourcing, outsourcing and shutting down their computers. More specific, Erik Elmroth [13] has carried out a billing model based on RESERVOIR model [19]. It works as an extension for the RESERVOIR and make up for its economical model.

VI. CONCLUSION

Researches on Cloud Federation have not last for many years and it is thought to be in its infant period. Regardless of the model proposed for Cloud Federation or its detailed implementation, two problems still arise which need all researchers to solve: security and resource allocation. In this paper, we discussed the latter problem and performed a mechanism to deal with the dynamic allocation issues in both scaling-up and scaling-down period. In short, our mechanism can roughly be concluded into these rules:

1. Allocate local resources first because of the performance difference. When there are not enough local resources, migrate *OutRent* resources first and check again. If the left resources are still not enough to hold the new request, then the migration is meaningless, for we should assure the number of services which need to be migrated is as little as possible.

2. Assure the integrity of services. Do not divided a complete service into pieces and allocate it both locally and remotely. The exponential increase in management cost and the complexity in distributed transaction will ruin the whole federation system.

3. Withdraw services running on remote node first unless its left time is lower than a threshold. This will help to improve the overall performance and eliminate the resource dependency ring.

Data / Request / Context		1M	10M	100M	1M	10M	100M	1M	10M	100M
		100	1,000	10,000	100	1,000	10,000	100	1,000	10,000
Original		42.8s	158s	1,310s	312.8s	428s	1,580s	3,012.8s	3,128s	4,280s
Optimized	100K	6.68s	18.2s	133.4s	51.68s	63.2s	178.4s	501.68s	513.2s	628.4s
	1M	7.58s	19.1s	134.3s	52.58s	64.1s	179.3s	502.58s	514.1s	629.3s
	10M	16.58s	28.1s	143.3s	61.58s	73.1s	188.3s	511.58s	523.1s	638.3s

Fig. 8. Simulation Evaluation

Next we are mean to doing some investigations and researches on Cloud Federation and improve our resource allocation mechanism. Out future work plan can be mainly divided into these parts:

1. Investigate and make statistics on different type of Cloud services in order to give out different 'return back rate' (or said R in formula calculating the original and optimized time consumption) according to different services to make our calculation more accurate and convincing.

2. Give out more detailed rules or algorithms in the selection of candidate service which needs to be migrated, and the selection of candidate Cloud which acts as the destination of the migration. It is displayed earlier in this paper that inappropriate selection of services or Cloud will cause the problem of wasting. Although there cannot exists a perfect algorithm that works excellent in all situations, but we try to carry out a better one in order to get a relatively good result in as many situations as possible.

3. Deploy a real Cloud Federation environment and apply our mechanism. We will do some experiments to get real data and calculate its improvement comparing with its original strategy.

REFERENCES

[1] J. G. I. F. I. R. A. Szalay, A. Bunn. The importance of data locality in distributed computing applications. In *NSF Workflow Workshop*, pages 1–2, 2006.

[2] D. J. W. Andrew S. Tanenbaum. *Computer Networks (4th Edition)*. Pearson Education, 2006.

[3] M. V. Antonio Celesti, Francesco Tusa and A. Puliafito. Three-phase cross-cloud federation model: The cloud sso authentication. In *Second International Conference on Advances in Future Internet*, 2010.

[4] V. U. Casola V., Rak M. Identity federation in cloud computing. In *Sixth International Conference on Information Assurance and Security*, pages 253–259. IAS, 2010.

[5] V. M. P. A. Celesti A., Tusa F. How to enhance cloud architectures to enable cross-federation. In *IEEE 3rd International Conference on Cloud Computing*, pages 337–345. CLOUD, 2010.

[6] E. Ciurana. *Developing with Google App Engine*. Firstpress, 2009.

[7] M. K. E. Elnozahy and R. Rajamony. Energy-efficient server clusters. In *2nd Workshop on Power-Aware Computing Systems*, pages 179–196. Cambridge, MA, USA, Feb 2002.

[8] F. G. M. Erik Elmroth. Accounting and billing for federated cloud infrastructures. In *GCC '09. Eighth International Conference*, pages 268–275. Grid and Cooperative Computing(GCC), 2009.

[9] R. I. L. S. Foster I., Yong Zhao. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop*, pages 1–10. GCE, Aug 2008.

[10] S. T. I. Foster, C. Kesselman. The anatomy of the grid:enabling scalable virtual organization. In *The Intl. Jrnl. of High Performance Computing Applications*, pages 200–222, 2001.

[11] S. T. I. Foster, C. Kesselman. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Globus Project*, pages 1–8, 2002.

[12] J. G. Inigo Goiri and J. Torres. Characterizing cloud federation for enhancing providers profit. In *IEEE 3rd International Conference on Cloud Computing*, pages 123–130. CLOUD, Jun 2010.

[13] G. C. Jean Dollimore, Tim Kindberg. *Distributed System: Concept and Design (4th Edition)*. Addison-Wesley, 2005.

[14] D. E. E. Larsson, L.; Henriksson. Scheduling and monitoring of internally structured services in cloud federations. In *Computers and Communications (ISCC)*, pages 173–178. 2011 IEEE Symposium, Aug 2011.

[15] W. Li and L. Ping. Trust model to enhance security and interoperability of cloud environment. In *Cloud Computing*, pages 69–79. CLOUD, Nov 2009.

[16] S. Microsystems. *Take your business to a Higher Level -Sun cloud computing technology scales your infrastructure to take advantage of new business opportunities*. Guide, Apr 2009.

[17] J. Murty. *Programming Amazon Web Services*. OReilly Press, 2008.

[18] R. R. Rajkumar Buyya and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing*. LNCS, 2010.

[19] a. Rochwerger, B.; et. The reservoir model and architecture for open federated cloud computing. In *IBM Journal of Research Development*, pages 1–11. IBM, July 2009.

[20] X. C. X. H. Shuai Zhang, Shufen Zhang. Cloud computing research and development trend. In *Second International Conference*, pages 93–97. ICFN, Oct 2010.

[21] X. H. Shuai Zhang; Xuebin Chen, Shufen Zhang. The comparison between cloud computing and grid computing. In *International Conference Vol: 11*, pages V11–72 – V11–75. Computer Application and System Modeling (ICCASM), Nov 2010.

[22] S. K. Y. Tamura, K. Sato and S. Moriai. Kemari: virtual machine synchronization for fault tolerance. In *USENIX 08 Poster Session*, 2008.

[23] X. Z. Zehua Zhang. Realization of open cloud computing federation based on mobile agent. In *IEEE International Conference, Volume: 3*, pages 642–646. ICIS, 2009.