Optimizing Resource Consumptions in Clouds

Ligang He[1], Deqing Zou[2], Zhang Zhang[2], Hai Jin[2], Kai Yang[2] and Stephen A. Jarvis[1]
1. Department of Computer Science, University of Warwick, Coventry, United Kingdom
2. School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
liganghe@dcs.warwick.ac.uk, Deqingzou@hust.edu.cn

Abstract*—This paper considers the scenario where multiple clusters of Virtual Machines (i.e., termed as Virtual Clusters) are hosted in a Cloud system consisting of a cluster of physical nodes. Multiple Virtual Clusters (VCs) cohabit in the physical cluster, with each VC offering a particular type of service for the incoming requests. In this context, VM consolidation, which strives to use a minimal number of nodes to accommodate all VMs in the system, plays an important role in saving resource consumption. Most existing consolidation methods proposed in the literature regard VMs as "rigid" during consolidation, i.e., VMs' resource capacities remain unchanged. In VC environments, QoS is usually delivered by a VC as a single entity. Therefore, there is no reason why VMs' resource capacity cannot be adjusted as long as the whole VC is still able to maintain the desired QoS. Treating VMs as being "mouldable" during consolidation may be able to further consolidate VMs into an even fewer number of nodes. This paper investigates this issue and develops a Genetic Algorithm (GA) to consolidate mouldable VMs. The GA is able to evolve an optimized system state, which represents the VM-to-node mapping and the resource capacity allocated to each VM. After the new system state is calculated by the GA, the Cloud will transit from the current system state to the new one. The transition time represents overhead and should be minimized. In this paper, a cost model is formalized to capture the transition overhead, and a reconfiguration algorithm is developed to transit the Cloud to the optimized system state at the low transition overhead. Experiments have been conducted in this paper to evaluate the performance of the GA and the reconfiguration algorithm.

Keywords-virtualization; Cluster; Cloud

I. INTRODUCTION

Cloud computing [10][11] has been attracting lots of attention recently. The advent of virtualization technology [1][2][3][4] provides dynamic resource partition within a single physical node, while the VM migration enables the on-demand and fine-grained resource provisions in multiple-node environments. Therefore, a virtualization-based Cloud computing platform offers excellent capability and flexibility to meet customer's changing demands. A virtualization-based Cloud platform often creates multiple Virtual Clusters (VCs) in a physical cluster and each VC consists of multiple VMs located in different physical nodes. A VC then forms a service deployment or application execution environment for external customers. Some popular Cloud middleware, such as EUCALYPTUS [12], Nimbus [13] and so on, can facilitate the system managers and customers to deploy, manage and utilize VCs.

Reducing power consumptions or conducting "green computing" has become a popular research topic nowadays. Different power saving strategies have been proposed in the literature [16][18]. A popular approach among them strives to consolidate VMs to a fewer number of hosting nodes [17][18]. The work in this paper falls into this scope. Most existing consolidation methods proposed in the literature regard VMs as "rigid" during consolidation, i.e., VMs' resource capacities remain unchanged. Different from the work in the literature, this paper treats VMs as "mouldable". This is because in VC environments, QoS is usually delivered by a VC as a single entity. Therefore, as long as the whole VC can still maintain the desired QoS, there is no reason why VMs' resource capacity cannot be adjusted. Treating VMs as "mouldable" may be able to consolidate VMs into an even fewer number of nodes. This paper investigates this issue and develops a Genetic Algorithm (GA) to consolidate mouldable VMs. In a virtualization-based Cloud, two fundamental attributes of the system state are VM-to-node mapping and the resource capacity allocated to each VM. The developed GA performs the crossover and mutation operation on system states and is able to generate an optimized state. Moreover, the design of this GA is not limited to a particular type of resource, but is capable of consolidating multiple types of resource.

After the optimized system state is calculated, the Cloud needs to be reconfigured from the current state to the optimized one. During the reconfiguration, various VM operations may be performed, including VM creation (CR), VM deletion (DL), VM migration (MG) as well as changing a VM's resource capacities (CH). The reconfiguration time represents management overhead and therefore should be minimized. Another contribution of this paper is to formalize a cost model to capture the overhead of a reconfiguration plan, and then this paper develops a reconfiguration algorithm to transit the Cloud to the optimized system state with the low overhead.

The remainder of this paper is organized as follows. Section II discusses the related work. Section III presents the Cloud architecture and workload models considered in this paper. The GA is presented in Section IV to consolidate VMs. Section V presents the cost model to capture the transition overhead, and develops an algorithm to reconfigure the Cloud with the low overhead. Experiments are conducted in Section VI to evaluate the performance of the GA and the reconfiguration algorithm. Finally, Section VII concludes this paper.

II. Related Work

Existing Cloud middleware normally provides resource management components to meet the resource demands from the Cloud users [12]. Server consolidation is a way to enhance Cloud middleware by improving resource utilization and reducing energy consumption [16][17]. The work in [17] develops a server consolidation scheme, called Entropy. Entropy strives to find the minimal number of nodes that can host a set of VMs, given the physical configurations of the system and resource requirements of the VMs. The objective is formalized as a multiple knapsack problem, which is then solved using a dynamic programming approach. In the implementation, a one-minute time window is set for the knapsack problem solver to find the solution. The solution obtained at the end of the one-minute time space is the new state (new VM-to-node mapping). Similarly to the work presented in this paper, the work searches for an optimal reconfiguration plan to minimize the cost of transiting the system from the current state to the new one. Our work differs from Entropy in the following two major aspects. First of all, the VMs considered in Entropy have to be rigid since the server consolidation is modeled as a knapsack problem. In our work, the VMs are "mouldable", which exposes a new dimension that should be addressed when designing consolidation strategies. Second, although Entropy also tries to find an optimal reconfiguration plan, only VM migrations are performed in the reconfiguration and the reconfiguration procedure is again modeled as a knapsack problem. In our work, however, various VM operations, including VM migration, VM deletion, VM creation and resource capability adjustment, may be performed in the reconfiguration as long as the reconfiguration cost can be further reduced without jeopardizing QoS. Therefore, the cost model for the reconfiguration procedure in our paper is much more complicated and the reconfiguration cannot be modeled as a knapsack problem anymore. The experiments are presented in Section VI to compare our work with Entropy in term of saving nodes.

Server consolidation components normally function below the Cloud middleware. Research work has also been carried out to develop workload management mechanisms sitting on top of the Cloud middleware to improve performance [5-9]. Many workload managers adopt a two-level scheduling mechanism. For example, Maestro-VC adopts a two-level scheduling mechanism based on Virtual Clusters, including a Virtual Cluster scheduler running on the front-end node and a local scheduler inside a virtual cluster, to improve the resource utilization [14]. Workload management components mainly focus on designing request scheduling strategies given the Cloud settings. Server consolidation compliments these workload management components. It can work underneath the Cloud middleware and be conducted transparently from external clients to further improve system-oriented performance (such as resource utilization) while maintaining the client-oriented performance (such as QoS).

III. System hierarchy and workload models

The consolidation scheme proposed in this paper assumes that the Cloud adopts the architecture illustrated in Fig.1. Multiple VCs, denoted as VC_1, VC_2, …, and VC_M, coexist in the Cloud system. The Cloud system consists of a cluster of N physical nodes, n_1, n_2, …n_N. Creating a VM needs to consume R types of resources, r_1, r_2, …r_R, in a node. Each VC hosts a particular type of service, service certain types of incoming request. The Cloud system aims to maintain a steady level of Quality of Service (QoS) delivered by every VC. The definition of the QoS depends on the requirement of the workload manager in the system. For example, the QoS can be expressed as the proportion of a type of requests whose response time is longer than x is no more than y, or the total service rate of a VC must exceeds z. No matter what QoS definition is used, QoS is always a function of the VMs' processing capabilities in a VC, which eventually depends on the resource capacity allocated to the VMs and can be determined using perform models. There are two levels of managers in the Cloud system: Local Manager (LM) and Global Manager (GM). Every VC has its LM, while there is only one GM in the Cloud. The GM dispatches the requests, as they arrive, to the LMs of the corresponding VCs. A VC's LM further dispatches the requests, as they are received, to individual VMs in the VC, where the requests are placed in the VM's local waiting queue and executed on the First-Come-First-Served (FCFS) basis. This two-level workload management framework is often adopted in literature [16].

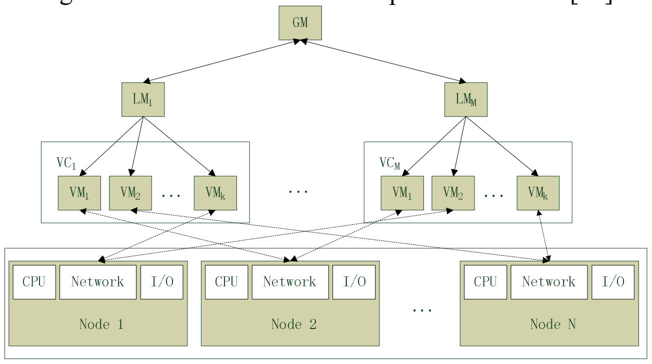

Figure 1. The hierarchy of the Cloud System

Each node has at most one VM of each VC. VM_{ij} denotes VC_j's VM in node n_i. Assume the capacity of resource r_i allocated to a VM in VC_j have to be in the range of [$minc_{ij}$, $maxc_{ij}$]. $minc_{ij}$ is the minimal requirement for resource r_j when generating a VM for VC_i, and $maxc_{ij}$ is the capacity beyond which the VM will not gain further performance improvement. For example, minimal memory requirement for generating a VM in VC_j is 50 Megabytes, while the VM will not benefit further by allocating more than 1 Gigabytes of memory. We assume that the nodes are homogeneous. $minc_{ij}$ and $maxc_{ij}$ is normalized as a percentage of the total resource capacity in a node. It is straightforward to extend our work to a heterogeneous platform.

A VC's LM can use existing request scheduling strategies to determine a suitable VM for running an incoming request [15][16]. The server consolidation scheme is deployed in GM and works with the VM management strategy and the request scheduling strategy in LMs to achieve optimized performance for the Cloud. The consolidation procedure will be invoked when necessary (the invocation timing will be discussed in Section IV). After the consolidation is completed, LMs will be informed of the new system state, i.e., VM-to-node mapping and resource capacities allocated to each VM. LMs can then adjust the dispatching of requests to VMs accordingly.

IV. THE GENETIC ALGORITHM

In this work, a Genetic Algorithm (GA) has been designed and implemented in this work to compute the optimized system state, i.e., VM-to-node mapping and the resource capacity allocated to each VM, so as to optimize resource consumptions in the Cloud. The GA can work with the existing request schedulers in the literature [16], which is deployed in the GM and LMs.

The increase in the arrival rates of the incoming requests may cause the current VMs in the VC cannot satisfy the desired QoS level, and therefore a new VM needs to be created with desired resource capacity. The invocation of the GA will be triggered if the following situations occur, which are termed as *resource fragmentation*: 1) there are spare resource capabilities in active nodes. An active node is a node in which the VMs are serving requests. Denoting the spare capability of resource r_j in node n_i as sc_{ij}; 2) the spare resource capabilities in every node are less than the capacity requirements of the new VM in VC_k, denoted as c_{kj}, (i.e., for $\forall i$ ($1 \le i \le N$), there exists such j ($1 \le j \le R$), so that $sc_{ij} < c_{kj}$); and 3) The total spare resource capabilities across all used physical nodes are greater than the capacities required by the new VM.

Typically, a GA needs to encode the evolving solutions, and then performs the crossover and the mutation operation on the encoded solutions. Moreover, a fitness function needs to be defined to guide the evolution direction of the solutions. In this work, the solution that the GA strives to optimize is the system state, which consists of two aspects: the VM-to-node mapping and the resource capacity allocated to each VM. In this work, a system state is represented using a three dimensional array, S. An element $S[i, j, k]$ in the array is the percentage of total capacity of resource r_k in node n_i that is allocated to VM_{ij} of VC_j. The rest of this subsection discusses the crossover and mutation operation as well as the fitness function developed in this work.

A. *The Crossover and Mutation Operation*

Assume that the resource capacity allocated to VC_j in two parent solutions are $S_1[*, j, *]$ and $S_2[*, j, *]$ ($1 \le j \le M$), respectively. In the crossover operation, a *VC* index p is randomly selected from the range of 1 to M, and then both of the two parent solutions are partitioned into two portions at the position of the index p. Subsequently, the crossover operation merges the head (and tail) portion of parent solution 1 with the tail (and head) portion of parent solution 2, and generates child solution 1 (and 2). In the mutation operation, the quantity of an element in the matrix $S[i, j, k]$ is adjusted in the following way. $S[i, j, k]$ is increased by a quantity randomly chosen from [0, min($maxc_{jk}$–$S[i, j, k]$, sc_{ik})] (sc_{ik} is the spare capability of resource r_k in node n_i). Then depending on the QoS defined in the workload manager, the resource capacity allocated to another VM, VM_{qj} ($q \ne i$), can be reduced by a quantity calculated from the performance model of the running requests in this type of VMs.

B. *The Fitness Function*

Assume the number of active nodes is N and the spare capacity of a type of resource r_k in node n_i is sc_{ik}. The standard deviation of the variables, sc_{ik} ($1 \le i \le N$), can reflect the convergence level r_k's spare capability across N nodes. The bigger the standard deviation is, the higher convergence level.

Since multiple types of resources are taken into account in this work, it is desired that the spare capacity of different types of resources converges to the same node. The standard deviation of the variables, sc_{ik} ($1 \le k \le R$), can reflect to what extent there are balanced spare capacities across different types of resource in node n_i. The smaller the standard deviation is, the more balanced capabilities. The standard deviation of the variables, sc_{ik} ($1 \le k \le R$), in node n_i can be calculated using Eq.2, where $\overline{sc_i^s}$ is the average of sc_{ik} ($1 \le k \le R$) and can be calculated in Eq.3.

$$\sigma_i = \sqrt{\frac{\sum_{k=1}^{R} (sc_{ik} - \overline{sc_i^s})^2}{R}} \tag{2}$$

$$\overline{sc_i^s} = \frac{\sum_{k=1}^{R} sc_{ik}}{R} \tag{3}$$

The above two factors are combined together to construct the fitness function for the GA, which is shown in Eq.4, where $\overline{sc_k^a}$ is the average of sc_{ik} ($1 \le i \le N$) for resource r_k over N active nodes and can be calculated in Eq.5. In Eq.4, $W(\sigma_i, \overline{sc_i^s})$ is a weight function and used to calculate the weighted sum of the deviation of resource r_k's spare capacity in multiple nodes. The weight is determined based on the relation between σ_i and $\overline{sc_i^s}$, and is partitioned into six bands. Each of the first five bands spans 20% of $\overline{sc_i^s}$ and the last band is applied when σ_i is greater than $\overline{sc_i^s}$.

$$\sum_{k=1}^{R} \sum_{i=1}^{N} \frac{(sc_{ik} - \overline{sc_k^a})^2}{W(\sigma_i, \overline{sc_i^s})} \tag{4}$$

$$\overline{sc_k^a} = \frac{\sum_{i=1}^{N} sc_{ik}}{N} \tag{5}$$

$$W(\sigma_i, \overline{sc_i^s}) = \begin{cases} w_i & (i-1) \times 20\% \times \overline{sc_i^s} \le \sigma_i < i \times 20\% \times \overline{sc_i^s} \\ w_0 & \sigma_i \ge \overline{sc_i^s} \end{cases} \tag{6}$$

After a population of solutions is generated, each solution is evaluated by calculating the fitness function. The solution with higher value of the fitness function has higher probability to be selected to generate the next generation of solutions. The GA is stopped after it runs for a predefined time duration or the solutions have stabilized (i.e., does not improve over a certain number of generations).

V. Reconfigurating Virtual Clusters

Assume S and S' are the matrixes representing the system states before and after running the GA, respectively. The Cloud system needs to reconfigure the Virtual Clusters by transiting the system state from S to S'. During the transition, various VM operations will be performed, such as VM creation (CR), VM deletion (DL), VM migration (MG) as well as changing a VM's resource capacities (CH). This section analyzes the transition time and presents a cost model for the Cloud reconfiguration. An algorithm is also developed to reconfigurate the Cloud.

A. Categorizing Changes in System States

The differences between $S[i, j, k]$ and $S'[i, j, k]$ can be categorized into the following cases, which will be handled in different ways.

*Case 1: Both S[i, j, *] and S'[i, j, *] are non-zero, but have different values*: this means that the resource capacity allocated to VM_{ij} needs to be adjusted. It can be further divided into two subcases: 1.1) *S[i, j, *] is greater than S'[i, j, *]*, which means that VM_{ij} needs to reduce its resource capacity, and 1.2) *S'[i, j, *] is greater than S[i, j, *]*, which means that VM_{ij} needs to increase its resource capacity*;*

*Case 2: S[i, j, *] is non-zero, while S'[i, j, *] is zero*: this means that n_i is allocated to run VC_j before running the GA, but is not allocated to run VC_j after. In this case, there are two options to transit the current state to the new one: 2.1) *Deleting VM_{ij}*, and 2.2) *Migrating VM_{ij}* to another node which is allocated to run VC_j after running the GA;

*Case 3: S[i, j, *] is zero, while S'[i, j, *] is non-zero*: this case is opposite to case 2). In this case, the system can either 3.1) *create VM_{ij}*, or 3.2) *accept the migration of VC_j's VM from another node*.

B. Transiting System States

1) VM Operations during the Transition

$DL(VM_{ij})$, $CR(VM_{ij})$, $CH(VM_{ij})$ denote the time spent in completing deletion, creation, and capacity adjustment operation for VM_{ij}, respectively. $MG(VM_{ij}, n_k)$ denotes the time needed to migrate VM_{ij} from node n_i to n_k ($i\neq k$). Note one difference between VM deletion and VM migration. A VM can be deleted only after the existing requests scheduled to run on the VM have been completed, while the VM can continue the service during live migration. The average time needed for VM_{ij} to complete the existing requests, denoted as $RR(VM_{ij})$, can be calculated as follows.

Assume that the number of requests in VM_{ij} is m_{ij}, including the request running in the VM and the requests waiting in the VM's local queue. Assume the average execution time of a request is e, and the request which is running in the VM has been running for the duration of e_0. Then $RR(VM_{ij})$ can be calculated in Eq.7

$$RR(VM_{ij})=(m_{ij}-1)\times e + (e-e_0) \tag{7}$$

These four types of VM operations can be divided into two broad categories: 1) resource releasing operation, including deleting a VM, migrating a VM to another node, and reducing the resource capacity allocated to a VM; 2) resource allocation operation, including creating a VM, accepting the migration of a VM from another node, and increasing the resource capacity allocated to a VM. When both categories of VM operations need to be performed when reconfiguring a node, careful considerations have to be given to the execution order of the VM operations, because the node may not have enough resource capacity so that resource releasing operations have to be performed first before resource allocation operations can be conducted. Therefore, there may be execution dependencies among VM operations. Below, we first discuss the condition under which there are no execution dependencies among VM operations, and then analyze how to perform VM operations when the condition is or is not satisfied. We also analyze the time spent in completing these operations.

2) Performing VM Operations without Dependency

If total resource capacities of the VMs in a node do not exceed the total resource capacity of the node at any time point during the transition, the VM operations in the same node do not have dependency. This condition can be formalized in Eq.8.

$$\text{For } \forall k\text{: } 1\le k\le R,\ \sum_{j=1}^{M}\max(S[i,j,k],S'[i,j,k])\le 1 \tag{8}$$

We now analyze the time spent in completing the VM operations in a node when Eq.8 holds. The existence of VM migrations will complicate the analysis. Therefore, we first consider the case where there is no VM migration in the reconfiguration of the node, and then extend the analysis to incorporate migration operations.

i) time for reconfiguring a node without VM migrations

The transition time for reconfiguring node n_i, denoted as $TR(n_i)$, can be calculated using Eq.9, where S_{dl}, S_{cr} and S_{ch} denote the set of VMs in node n_i that are deleted, created and adjust their resource allocations during the reconfiguration, respectively; The second term in the equation (i.e. the term within the *min* operator) reflects the reality that the activities of creating a new VM and adjusting a VM's resource capacity can be conducted at the same time as executing the existing requests in the VMs to be deleted.

$$TR(n_i)=\sum_{VM_{ij}\in S_{dl}}DL(VM_{ij})+\min\{\sum_{VM_{ij}\in S_{cr}}CR(VM_{ij})+\sum_{VM_{ij}\in S_{ch}}CH(VM_{ij}),\max\{RR(VM_{ij})\mid VM_{ij}\in S_{dl}\}\} \tag{9}$$

ii) time for reconfiguring a node with VM migrations

If the reconfiguration of node n_i involves VM migrations, including n_i migrating a VM to another node and n_i accepting a VM migrated from another node, we introduce a concept of *mapping node* of n_i, which is further divided into *mapping destination node*, which is the node that the VM in n_i migrates to, and *mapping source node*, which is the node that migrates a VM to n_i. When handling VM migrations in n_i, n_i's mapping node will be first identified as follows. If the following two conditions are satisfied, node n_q ($i\neq q$) can be a mapping destination node of node n_i, and node n_i is called n_q's mapping source node.

- $\exists k$, $1\le k\le R$, $S[i, j, k] > 0$, but for $\forall k$, $1\le k\le R$, $S'[i, j, k]=0$
- For $\forall k$, $1\le k\le R$, $S[q, j, k]=0$, but $\exists k$, $1\le k\le R$, $S[q, j, k] > 0$

Note that a node can have multiple mapping nodes. Which mapping node is finally selected by the reconfiguration procedure will have impact on the transition time. If n_i accepts a VM migration from n_q during the reconfiguration of n_i, then the time for reconfiguring n_i can be calculated from Eq.10, where S_{mg} denotes the set of VMs in node n_i that are migrated from another node.

$$TR(n_i) = \sum_{VM_{ij}\in S_{dl}} DL(VM_{ij}) + \min\{ \sum_{VM_{ij}\in S_{mg}} MG(VM_{ij}, n_k) + \sum_{VM_{ij}\in S_{cr}} CR(VM_{ij}) + \sum_{VM_{ij}\in S_{ch}} CH(VM_{ij}), \max\{RR(VM_{ij}) \mid VM_{ij} \in S_{dl}\}\} \quad (10)$$

If n_i migrates a VM to another node, the reconfiguration procedure will check whether Eq.8 holds for n_i's mapping node. If Eq.8 holds, the VM can be migrated to that node at any time point. Otherwise, the VM migration will be handled in a different way, which is presented in Section V.B.3.

If Eq.8 holds for n_i, n_i is reconfigured using Algorithm 1. Step 5-7 of the algorithm deal with VM migrations, which will be discussed in detail in Section V.B.3. Note that Case 3 is not handled in this algorithm. The reason for this will be explained when Algorithm 4 is introduced.

Algorithm 1. Reconfiguration_without_dependency(n_i);

```
1.  for each $VM_{ij}$ in Case 1 do
2.      Adjust the resource capacity of $VM_{ij}$;
3.  for each $VM_{ij}$ in Case 2.1 do
4.      Delete $VM_{ij}$;
5.  for each $VM_{ij}$ in Case 2.2 do
6.      Find a mapping destination node, $n_k$, for the VM,
          denoted as $VM_{ij}$;
7.      Call migration($VM_{ij}$, $n_k$);
8.  return;
```

3) Performing VM Operations with Dependency

If Eq.8 does not hold, there must be at least one VM in the node which releases resources during the reconfiguration. Otherwise, if all VMs in the node only acquire resources and Eq.8 does not hold, the resource capacity allocated to the VMs in the node will exceed the node's physical resource capacity after the reconfiguration.

A VM can release resources by the following three possible operations during the reconfiguration:

- Reducing the resource capacity allocated to the VM,
- Deleting the VM,
- Migrating the VM to another node.

If there are multiple VMs releasing their resources in node n_i, the reconfiguration procedure will release resources in the above precedence, until Eq.8 satisfies, where $S[i,j,k]$ is now the current resource capacity allocated to the VMs in the node after the resources have been released so far. Once Eq.8 holds, it indicates the remaining reconfiguration process can be conducted using the way discussed in Subsection V.B.2. If multiple VMs perform the same type of resource releasing operations, a VM with the greatest amount of capacity to be released will be selected.

The procedure of migrating VM_{ij} to node n_k is outlined in Algorithm 2, which is called in Algorithm 1 (Step 7). The algorithm will first check whether Eq.8 holds for the mapping node (Step 1). If it holds, the VM migrates to the mapping node straightway (Step 21). If it does not hold, the VM migration operation does have dependency and some resource releasing operations have to be completed in the listed precedence (Step 2-19) until Eq. 8 holds. Under this circumstance, the algorithm becomes an iterative procedure (Step 16) and resource releasing operations will be performed in a chain of nodes.

Algorithm 2. migration(VM_{ij}, n_k) //migrating VM_{ij} to n_k

```
1.  if Eq.8 does not hold for $n_k$ then //with dependency
2.      for each $VM_{kj}$ in Case 1.1 do
3.          reduce $VM_{kj}$'s resource allocations and update
            $S[k, j, *]$ accordingly;
4.          if Eq.8 holds then
5.              migrate $VM_{ij}$ to $n_k$ and update $S[k, j, *]$
                and $S[i, j, *]$ accordingly;
6.              return;
7.      end for
8.      for each $VM_{kj}$ in Case 2.1 do
9.          delete $VM_{kj}$ and update $S[k, j, *]$ accordingly;
10.         if Eq.8 holds then
11.             Migrate $VM_{ij}$ to $n_k$ and update $S[k, j, *]$
                and $S[i, j, *]$ accordingly;
12.             return;
13.     end for
14.     for each $VM_{kj}$ in Case 2.2 do
15.         Obtain a mapping node, $n_q$;
16.         Call migration($VM_{kj}$, $n_q$);
17.         If Eq.8 holds then
18.         Migrate $VM_{ij}$ to $n_k$ and update $S[k, j, *]$ and $S[i,
            j, *]$ accordingly;
19.         return;
20.     else //without dependency
21.         migrate $VM_{ij}$ to $n_k$ and update $S[k, j, *]$ and $S[i,
            j, *]$ accordingly;
22.         return;
```

Algorithm 3 is used to reconfigure node n_i. In the algorithm, if Eq.8 does not hold for n_i, the resources will be released until Eq.8 satisfies. Then Algorithm 1 is called to reconfigure the node (Step 16).

Algorithm 3. Reconfiguration(n_i)

```
1.  if Eq.8 does not hold then //with dependency
2.      for each $VM_{ij}$ in Case 1.1 do
3.          reduce $VM_{ij}$'s resource allocations and update
                $S[i, j, *]$ accordingly;
4.          if Eq.8 holds then break;
5.      end for
6.      for each $VM_{ij}$ in Case 2.1 do
7.          delete $VM_{ij}$ and update $S[i, j, *]$ accordingly;
8.          if Eq.8 holds then break;
9.      end for
10.     for each $VM_{ij}$ in Case 2.2 do
11.         Obtaining the mapping node, $n_k$;
12.         Call migration($VM_{ij}$, $n_k$);
13.         if Eq.8 holds then break;
14.     End for
15. End if
16. call Reconfiguration_without_dependency($n_i$);
17. return;
```

Algorithm 4 is used to construct the reconfiguration plan for the Cloud. Note that the VMs in Case 3 are handled in this algorithm (Step 7-9) by creating the VMs. This is because when migrating VM_{ij} from node n_i to the mapping node n_k, Case 3 has been handled for VM_{kj} in node n_k (Case 3.1). Therefore, when Algorithm 4 completes Step 2-6, the VMs that are left unattended are those which were not used as the mapping destination nodes for VM migrations. The only option to deal with those VMs now is to create them.

Algorithm 4. Reconfiguration the Cloud

Input: $S[i, j, k]$

```
1.  Ω = the set of all nodes in the Cloud;
2.  while (Ω ≠ Φ)
3.      Obtain node $n_i$ ($1 \le i \le N$) from Ω;
4.      Call Reconfiguration ($n_i$);
5.      Ω = Ω − $n_i$;
6.  end while
7.  for each node, $n_i$ do,
8.      for each $VM_{ij}$ in Case 3 that has not been handled do
9.          Create $VM_{ij}$;
```

C. Calculating Transition Time

A DAG graph can be constructed based on the dependencies between the VM operations as well as between source and mapping destination nodes. As can be seen in Algorithm 3, if Eq.8 does not hold for n_i, the VM operations have to be performed in a particular order, which causes the dependency between VM operations. Also in Algorithm 2, if migration(VM_{kj}, n_q) is further invoked during the execution of migration(VM_{ij}, n_k), then there is the dependency between node n_q and n_k. This is because n_k depends on n_q releasing resources before a VM in n_k can migrate to n_q.

In this paper, a DAG graph is also used to model the dependency between nodes. In the DAG graph, a node represents a physical node, and an arc from node n_i to n_k represents a VM migrating from n_k to n_i.

If the VM operations in all nodes form a single DAG, calculating the transition time of the reconfiguration plan for the Cloud can be transformed to compute the critical path in the DAG. The VM operations involved in reconfiguring the Cloud may also form several disjoint DAG graphs. In this case, the critical paths of all DAG graphs are computed. The time for the longest path is the transition time of the reconfiguration plan for the Cloud since the VM operations in different DAG graphs can be performed in parallel.

There can be different reconfiguration plans and different plans may have different transition times. The uncertainty comes from two aspects: 1) which of the two VM operations, deletion or migration, should be performed for a VM in Case 2, and 2) if a VM is to be migrated and it has multiple mapping destination nodes, which node should be selected to migrate the VM to. More specifically, before invoking Algorithm 3, we need to decide for all VMs in Case 2, which VMs should be classified into Case 2.1 (relating to Step 6 of Algorithm 3) and Case 2.2 (relating to Step 10). Moreover, in Step 11, the system needs to determine which mapping node should be selected. The objective is to obtain a reconfiguration plan which has the low transition cost. We now present the strategies to find such a plan.

An approach to obtaining the optimal reconfiguration plan is to enumerate all possibilities for each VM falling into Case 2, i.e., to calculate the transition cost for both Case 2.1 and Case 2.2. If there are k VMs which fall into Case 2, then there are 2^k combinations of delete/migration choices and the transition cost for each combination needs to be calculated. After determining to migrate a VM, another uncertainty is that the VM may have multiple mapping nodes. Suppose VM_{ij} has p_j mapping nodes. We need to enumerate all possibilities and calculate the transition cost p_j times for migrating a VM to each of its p_j mapping nodes. Each possibility corresponds to a DAG. Therefore, the enumeration approach will examine all these different DAGs. The DAG with the shortest critical path represents the optimal reconfiguration plan. Apparently, the time complexity of the enumeration approach is very high. We developed a heuristic approach to obtain a sub-optimal reconfiguration plan quickly. The strategies used in the heuristic approach are as follows.

a) determining deletion or migration:

$D(VM_{ij})$ denotes the time the system has to wait for completing the deletion of VM_{ij}. As discussed in subsection V.B.1, $D(VM_{ij})=DL(VM_{ij})+RR(VM_{ij})$. If the following two conditions are satisfied, VM_{ij} is migrated. Otherwise, VM_{ij} is deleted. i) VM_{ij} has at least one mapping node such that migrating VM_{ij} to that node will not trigger other VM deletion or migration operations; ii) For all mapping nodes satisfying the first condition, there exists such a node, n_k, that $D(VM_{ij}) > MR(VM_{ij}, n_k)$

The two conditions try to compare the time involved in deleting and migrating a VM. Before invoking Algorithm 3 in subsection V.B.3, these two conditions will be applied to determine whether a VM in Case 2 should be handled as Case 2.1 (Step 6) or Case 2.2 (Step 10)

b) determining the mapping node

If a VM is to be migrated and there are multiple mapping destination nodes which satisfy condition ii), then Step 11 of Algorithm 3 will select the node which offers the shortest migration time $MR(VM_{ij}, n_k)$.

VI. Experimental Studies

A discrete event simulator has been developed to evaluate 1) the performance of the developed GA in consolidating resources, and 2) the transition time of the reconfiguration plan obtained by the enumeration approach and the heuristic approach.

In the experiments, three types of resources are simulated: CPU, memory and I/O, and three types of VMs are created: CPU-intensive, Memory-intensive, and I/O-intensive VMs. A VC consists of the same type of VMs. For the CPU-intensive VMs, the required CPU utilisation is selected from the range of [30%, 60%], while their memory and I/O utilisation are selected from the range of [1%, 15%]. The selection range represents [$minc_{ij}$, $maxc_{ij}$] discussed in Section II. Similarly, for the memory-intensive VMs, the allocated memory is selected from the range of [30%, 60%], while their CPU and I/O utilisation are selected from the range of [1%, 15%]. For the I/O-intensive VMs, the required I/O utilisation is selected from [30%, 60%], while their CPU and the memory utilisation are from the range of [1%, 15%].

In the experiments, the VMs are first generated in physical nodes according to the above method. A node is not fully utilized and will have a certain level of spare resource capacity. The service rate of requests of each VM is calculated using the performance model in [8]. The workload manager in [16] is used in the experiments. The arrival rate of the incoming requests for each VC is determined so that the VCs' QoS can be satisfied. The average execution time of each type of requests is set to be 5 seconds, and the QoS of each VC is defined as 90% of the requests' response time is no longer than 10 seconds. A VC's workload manager (LM) dispatches the requests to VMs, and therefore the requests' arrival rate for each VM can be determined. Then the developed GA is applied to consolidate VMs so that the spare resource capacity in nodes can converge to a smaller number of nodes. After the GA obtains the optimized system state, the reconfiguration plan is constructed to transfer the Cloud from the current state to the optimized one.

The average time for deleting and creating a VM is 20 and 14 seconds, respectively. The migration time depends on the size of VM image and the number of active VMs in the mapping nodes [17][5]. The migration time in our experiments is in the range of 10 to 32 seconds.

Fig.2 shows the number of nodes saved as the GA progresses. In the experiments in Fig.2, the number of nodes varies from 50 to 200. The experiments aim to investigate the time that the GA needs to find a optimized system state, and also investigate how many nodes the GA can save by converging spare resource capacities. The free capacity of each type of resource in the nodes is selected randomly from the range [10%, 30%] with the average of 20%. The number of the VMs in a physical node is 3. The number of the VCs in the system is 30. As can be observed from Fig.2, the percentage of nodes saved increases as the GA runs for longer, as to be expected. Further observations show that under all three cases, the number of nodes saved increases sharply after the GA starts running. It suggests the GA implemented in this paper is very effective in evolving optimized states. When the GA runs for longer, the increasing trend tides off. This is because that the VM-to-node mapping and resource allocations calculated by the GA approaches the optimal solutions. Moreover, by observing the difference of the curve trends under different number of nodes, it can be seen that as the number of nodes increases, it takes the GA longer to approach the optimized state.

Fig.3 compares the GA developed in this work with the Entropy consolidation scheme presented in [17]. In the experiment results presented in Fig.3. It can be seen from this figure that the GA clearly outperforms Entropy in all cases. This is because the VMs' resource allocations in Entropy remain unchanged, while the GA developed in this paper employs the mutation operation to adjust the VMs' resource allocations. This flexibility makes the VMs "mouldable" and therefore is able to squeeze VMs more tightly into fewer nodes. It can also been observed from this figure that there is no clear increasing or decreasing trend in terms of the proportion of nodes saved as the number nodes increases in our consolidation scheme. This suggests that the number of nodes does not have much impact on the GA's capability of saving nodes.

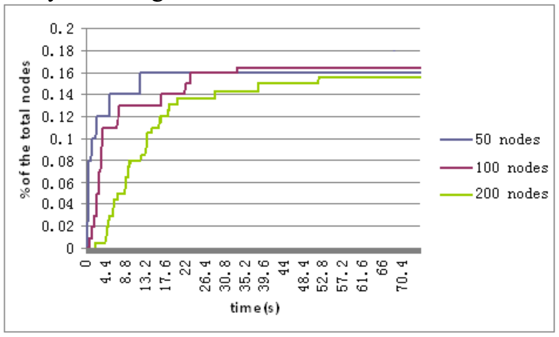

Figure 2. The quantity of nodes saved as the GA progresses

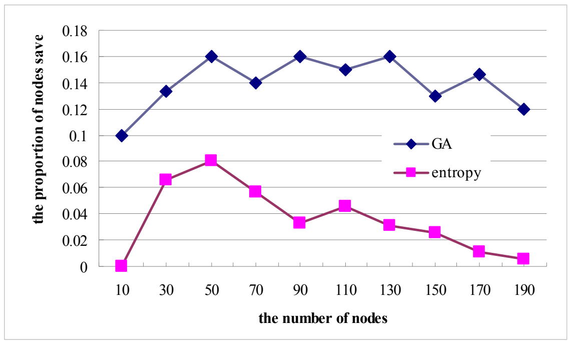

Figure 3. The comparison between the GA and entropy; the average free resource capacity is 20%

Fig.4 shows the time it takes for the enumeration approach to find the optimal reconfiguration plan under different number of nodes and different number of VCs. The optimized system states are computed by the GA. The average spare capacity in nodes is 15%. It can be seen from this figure that the time increases as the number of nodes increases and also as the number of VCs increases. When the number of nodes is 200 and the number of VCs is 4, the time is 450 seconds, which is unbearable in real systems. That is why a heuristic approach is necessary to quickly find the sub-optimal reconfiguration plan for the large scale of systems. Our experiments show that the time spent by the heuristic approach designed in this paper is negligible (less than 2 seconds even when the number of nodes is 200).

Fig.5 shows the transition time of the optimal reconfiguration plan obtained by the enumeration approach as well as the sub-optimal plan by the heuristic approach. As can be seen from this figure, the transition time increases in all cases as the number of nodes increases and also as the number of VCs increases. It can also be observed from Fig.5 that the difference in the transition time between the enumeration approach and the heuristic approach is not prominent. According to our experiment data, when the number of VC is 2, 3 and 4, the average difference in transition time between these two approaches is 4.9, 4.6 and

6.6 seconds. The results suggest that the developed heuristic approach can efficiently find a good reconfiguration plan.

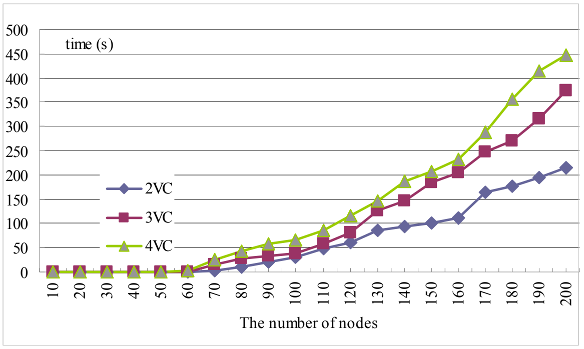

Figure 4. the execution time of the reconfiguration algorithm for different number of nodes and VCs

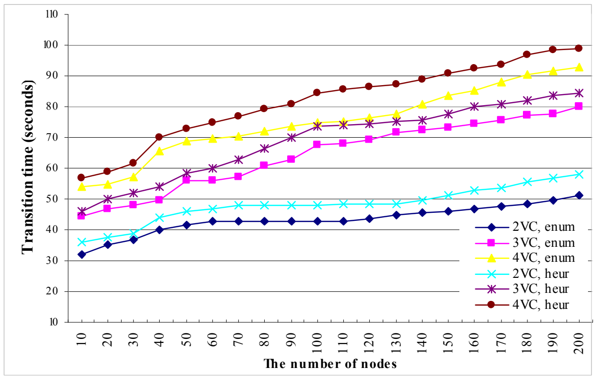

Figure 5. the transition time of the optimal reconfiguration plan obtained by the enumeration approach as well as the sub-optimal plan obtained by the heuristic approach

VII. Conclusions

This paper aims to optimize the resource consumptions in the cluster-based Cloud systems. The Cloud system hosts multiple Virtual Clusters to server different types of incoming requests. A GA is developed to compute the optimized system state and consolidate resources. A Cloud reconfiguration algorithm is then developed to transfer the Cloud from the current state to the optimized one computed by the GA. In the experiments, the performance of the GA and the reconfiguration algorithm is evaluated and the developed scheme is also compared with a consolidation scheme developed in literature.

VIII. Acknowledgement

This work is supported by the Research Project Grant of the Leverhulme Trust (Grant No. RPG-101), National Base Research Program of China (2007CB310900) and National Science Foundation of China (60973038).

References

[1] S. Nanda, T. Chiueh. A survey on virtualization technologies. Technical Report, TR-179, Stony Brook University, Feb 2005. http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf

[2] P. Barham, B. Dragovic, K. Fraser, and et al. Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating Systems Principles, pages: 164-177, ACM Press, 2003.

[3] VMware Infrastructure:"Resource Management with VMware DRS". VMware Whitepaper 2006.

[4] W. Zhao and Z. Wang. Dynamic Memory Balancing for Virtual Machines. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), ACM Press, pages 21- 30, March 11-13, 2009, Washington, DC, USA..

[5] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, C. Pu, "mistral--Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures", ICDCS 2010: 62-73

[6] I. Cunha, J. Almeida, V. Almeida, and M. Santos. Self-adaptive capacity management for multitier virtualized environments. In Proceedings 10th Symposium on Integrated Network Management, pages: 129-138, 2007.

[7] J. Ejarque, M. D. Palol, I. Goiri, F. Julia, R. M. Gui-tart, J. Badia, and J. Torres. SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers. In Proceedings of the 4th IEEE Inter-national Conference on eScience(eScience 2008), Indi-anapolis, Indiana, USA, pages: 8-15, Dec 2008.

[8] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating Adaptation Policies for Multi-Tier Applications in Consolidated Server Environments. IEEE International Conference on Autonomic Computing, pages: 23-32, 2008.

[9] B. Sotomayor, K. Keahey, I. Foster. Combining Batch Execution and Leasing Using Virtual Machines. Proceedings of the 17th international symposium on High performance distributed computing, pages: 87-96, 2008.

[10] M. Armbrust, A. Fox, R. Griffith, and et al. Above the Clouds: A berkeley view of Cloud computing. Technical Report, February 10, 2009.

[11] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented Cloud computing: Vision, hype, and reality for delivering it services as computing utilities. CoRR, (abs/0808.3558), 2008.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. The eucalyptus open-source Cloud-computing system. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pages: 124-131, 2009.

[13] Nimbus, http://www.nimbusproject.org.

[14] N. Kiyanclar, G. A. Koenig, and W. Yurcik. Maestro-VC: On-Demand Secure Cluster Computing Using Virtualization. In 7th LCI International Conference on Linux Clusters, 2006.

[15] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun. Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center. In 9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid), pages: 148-155, May 18-21, 2009.

[16] L. Hu, H. Jin, X. Liao, X. Xiong, H. Liu. Magnet: A Novel Scheduling Policy for Power Reduction in Cluster with Virtual Machines. In Proceeding of 2008 IEEE International Conference on Cluster Computing (Cluster 2008), IEEE Computer Society, Japan, pages: 13- 22, Sept. 29 2008-Oct. 1 2008.

[17] F. Hermenier, X. Lorca, J. Menaud, G. Muller, J. Lawall, "Entropy: a consolidation manager for clusters", Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp. 41-50, 2009

[18] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, "Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center", 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009

[19] O. Tickoo, R. Iyer, R. Illikkal, D. Newell, "Modelling Virtual Machine Performance: Challenges and Approaches", ACM SIGMETRICS Performance Evaluation Review, 37(3), 2009.