Developing Resource Consolidation Frameworks for Moldable Virtual Machines in Clouds

Ligang He[1], Deqing Zou[2], Zhang Zhang[2], Chao Chen[1], Hai Jin[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 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 "moldable" 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 moldable 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 [13][14] has been attracting lots of attention recently. Cloud computing is driven by economies of scale, in which various services (such as Platform-as-a-service, Software-as-a-service and Infrastructure-as-a-Service) are delivered on demand to external customers. The advent of virtualization technology [1][2][3] 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. As the Clustering technology is a popular approach to building a reliable, scalable and cost-effective computing platform for scientific and e-business/commerce applications, 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 [15], Nimbus [16] and so on, can facilitate the system managers and customers to deploy, manage and utilize VCs.

Power consumptions have become a crucial concern in Cloud platforms due to the contradiction between the ever-increasing scale of Cloud platforms and the energy shortage in the modern society. Therefore, reducing power consumptions or conducting "green computing" have become a popular research topic nowadays. Different power saving strategies have been proposed in the literature [27][28][29][30][34][35]. A popular approach among them strives to consolidate VMs to a fewer number of hosting nodes [34][35]. The work in this paper falls into this scope. Most existing consolidation methods proposed in literature regard VMs as "rigid" during consolidation, i.e., VMs' resource capacities remain unchanged. Different from the work in literature, this paper treats VMs as "moldable" during consolidation. 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 "moldable" 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 moldable 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 is reconfigured from the current state to the new one. During the reconfiguration, various VM operations may be performed, including VM creation, VM deletion, VM migration as well as changing a VM's resource capacities. The reconfiguration time represents management overhead and should be minimized. Another contribution of this paper is to formalize a cost model to capture the overhead of a reconfiguration plan, and then develop a reconfiguration algorithm to transit the Cloud to the optimized system state with the low overhead.

Reference [43] presents the earlier version of this work. Compared with [43], this paper introduces new contributions as follows: i) an extension to the related work, ii) detailed discussions of case studies, iii) conducting more simulation experiments, iii) conducting real-life experiments on a cluster.

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 effectiveness of the developed consolidation framework. 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. For example, the virtualized resources that comprise a EUCALYPTUS Cloud are exposed and managed by the Cloud Controller (CLC) whose resource services perform system-wide arbitration of resource allocations and monitor both system components and virtual resources [15]. However, the main objective of the Cloud middleware in literature is to provide the convenient management layer for the system managers and external customers to manage, deploy and utilize the underlying Cloud platform. The optimization strategies in terms of performance (e.g., resource utilization) and quality (e.g., QoS and energy consumption), especially the tradeoff between them, are typically limited and could be further strengthened.

Server consolidation is a way to enhance Cloud middleware by improving resource utilization and reducing energy consumption [34][36]. The work in [36] addresses the issue of using the minimal number of nodes to handle the workload in a cluster consisting of multiple Virtual Clusters. The work develops the "squeeze" and "release" measures to dynamically redistribute the workloads in the cluster according to the workload level in each individual node. The workload redistribution is achieved by a sequence of live VM migrations. The idle nodes in the cluster can then be switched off to save energy. However, the work in [36] only considers a single resource: CPU. In our work, the design of the consolidation strategy is generic and can accommodate multiple types of resource. Moreover, the resource capability allocated to a VM remains unchanged in [36]. In this paper, the resource allocations to VMs may be adjusted to further improve the resource utilizations.

The work in [40] proposes a dynamic optimization model for power and performance management of virtualized clusters. The work applies the mixed integer programming technique to find the minimized power consumptions in virtualized clusters. The objective of the work in [40] is similar as that in this paper. However, the work in [40] only considers a single type of resource, CPU. Moreover, in order to model the problem using mixed integer programming technique, it assumes that the CPU speed is selected from a set of discrete figures. In this paper, we consider multiple types of resource (e.g., CPU and Memory), and the resource capability allocated to a VM can be any figure that is no more than the total capacity of the physical node. Therefore, the problem that this paper aims to address cannot be modeled by the mixed integer programming technique.

The work in [41] proposes a rule-based knowledge management approach to addressing the resource allocation problem for Cloud infrastructure. The rule-based approach sets a target value for resource utilization, and adjusts the allocated resources when the utilization deviates from the target value by a certain threshold. The work in [41] aims to adjust resource allocations so as to achieve SLA (Service Level Agreement) and high resource utilization with low adjustment overheads. The SLA and the resource utilization introduced in [41] are oriented toward a single VM. However, the work presented in this paper endeavors to reduce the number of nodes (i.e., resource consumption) used to host all VMs while maintaining the QoS of all VCs. The QoS requirement in this paper is for a VC as a whole (not an individual VM), and the resource consumption also refers to the consumptions made by all VMs collectively. Therefore, the rule-based approach in [41] cannot be used to solve the problem in this paper either. Another difference between the work in [41] and this work is that in [41], resource allocations will be adjusted only when SLA cannot be met. In this paper, however, even if the current resource allocation can satisfy the QoS, resource allocations may still be adjusted if the VMs can be consolidated into a smaller number of nodes.

The work in [34] 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 "moldable", 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.

As discussed above, the methods presented in literature, including the mixed integer programming technique [40], the rule-based knowledge management approach [41], and knapsack modeling method [34], cannot be applied to address the problems in this paper. In this work, we apply the Genetic Algorithm (GA) to optimize the resource consumptions by moldable VMs, because the crossover operation of the GA can help look for a better way of packing VMs into physical nodes while the mutation operation can be used to adjust the resource capacity allocated to VMs (the details of GA will be presented in Section IV).

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. Various workload management methods, including the control theory [6], SLA-Driven models [8][9], queuing models [10], Lease Scheduling [12] and so on, have been proposed. Many workload managers adopt a two-level management mechanism. The most notable example is Eucalyptus [15][42]. In Eucalyptus, a Cloud Controller acts as a top-level manager responsible for managing multiple clusters in the Cloud while a Cluster Controller sits in each cluster, managing the nodes in the local cluster. There are also other examples of two-level management mechanism in literature. For example, Maestro-VC [8] 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. In [9], a Client Manager is a top level of manager and manages the client's task execution, while the semantic scheduler works as a lower-level manager and allocates physical resources to each task. The work in [30] organizes the VMs into a multilayer rings. Each layer has a leader to balance the workload among the nodes belonging to this layer.

Workload management components mainly focus on designing request scheduling strategies given the Cloud settings. Server consolidation proposed in our work 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).

Our consolidation scheme requires that the virtualization system is able to specify resource utilization consumed by each VM. The requirement can be realized by the advance of the virtualization technology. In a typical virtualized system, the resources such as processors, memory and network interfaces can be assigned to and reclaimed from a VM according to demands [1]. Moreover, the VM technology allows the system to specify the CPU percentage and memory size utilized by each VM. For example, Xen [2] and VMware [3] provide a ballooning driver to dynamically adjust host memory allocation among VMs, and allow the VMM (Virtual Machine Monitor) to dynamically adjust VCPU (Virtual CPU) capability of a VM. Although it is more challenging to accurately specify utilization of other resources, such as I/O, there has been active research work in this area [39].

The consolidation scheme presented in this paper needs to know the performance model of running requests in a VM, i.e., being able to predict the response time of the requests running on a VM given the VM's resource capability. Various methodologies have been proposed to address this issue [38][39]. For instance, the work in [39] used layered queuing network to model the response time of a request in a multi-tiered web service hosted in VM environments, while hardware resources (e.g., CPU and disk) are modeled as processor sharing (PS) queues. The work in [38] modeled the contention of visible resources (e.g., CPU, memory, I/O) and invisible resources (e.g., shared cache, shared memory bandwidth) as well as the overheads of the VM hypervisor implementation. Our work utilizes the methods in [39] to obtain the performance model required by our consolidation scheme.

Our consolidation scheme also needs to know the time cost of VM operations, such as VM deletion, creation, migration and resource capacity adjustment. Various studies in literature [34][39] have presented the methods to measure the time spent in executing these VM operations, and also established the relation between the cost and other VM attributes. For example, the VM migration cost is investigated in [34]. The work first experimentally measures the cost and duration of a single VM migration, and then develops a model to estimate the costs of a set of correlated VM migrations in a Virtual Cluster. The work also established the relation between migration cost of a VM and it memory size. The work in [39] measured the costs of conducting VM deletion, creation, resource capacity adjustments. Our work makes use of these methods to obtain the execution times of the VM operations involved in reconfiguration.

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 desired QoS is expressed as the total service rate of all VMs in a VC cannot be less than a certain figure. 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 also often adopted in literature [8][9][12][30][42], in which the most notable example is Eucalyptus [15][42].

Each node has at most one VM of each VC. The reason for limiting this is because the consolidation framework in this paper can adjust the resource capacity allocated to VMs. Therefore, if there is the need to map two VMs from the same VC to the same physical node, it is very likely that we can increase the resource allocation of one VM (according to the performance model) so that the upsized VM has the same processing capability as the total processing capability of these two VMs. 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 physical nodes are homogeneous. $minc_{ij}$ and $maxc_{ij}$ is normalized as a percentage of the total resource capacity in a physical node. It is straightforward to extend our work to a heterogeneous platform.

A VC's LM can use the existing VM management strategy in literature to create VMs in physical nodes [30], and use existing request scheduling strategy to determine a suitable VM for running an incoming request [30]. 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 server consolidation procedure will be invoked when necessary (the invocation timing will be discussed in Section IV). After the server consolidation is completed, the consolidation procedure will inform LMs 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.

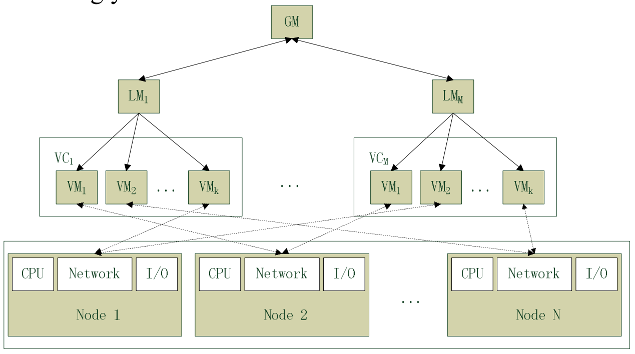

Figure 1. The hierarchy of the Cloud System

IV. THE GENETIC ALGORITHM

In this work, a VM may consume multiple types of resources. For example, the VC serving CPU-intensive requests will mainly consume CPU cycles while the VC processing I/O-intensive requests needs to consume a large amount of I/O capacity. It is a NP-hard problem to optimize the consumptions of multiple types of resources. 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 literature [30], 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

$$sc_{ij} < c_{kj}$$

3) The total spare resource capabilities across all used physical nodes are greater than the capacities required by the new VM, i.e.,

For $\forall j$ ($1 \le j \le R$), the following inequality holds, where β is no less than one and used to control the level of spare capability in the Cloud that can trigger the GA. The bigger β is, less frequently the GA will be invoked and to greater extent the resources will be consolidated. The value of β is determined empirically.

$$\sum_{i=1}^{N} sc_{ij} \ge \beta \times c_{kj}$$

The motivation of invoking the GA is to converge the spare capacity to as few number of nodes as possible so that we can create the new VM in one of the active nodes, and therefore avoid waking up an inactive node.

If the arrival rates of requests decrease, the resource capacity allocated to VMs will become excessive. The deployed request scheduler will re-distribute the requests among VMs. If this redistribution causes a VM becomes idle, the VM may be deleted. If all VMs in a node are deleted, the node can be switched off or enter the sleep mode to save energy. So the decrease in the requests' arrival rates will not trigger the invocation of the GA. But note the deletion of VMs will generate spare resource capacity in nodes.

Typically, a GA needs to encode the evolving solutions, and then perform 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 Operation

Given a generation of solutions, represented in the *S* array, the crossover and mutation operations will be performed in the GA to generate the next generation of solution. The crossover operation takes as input two solutions, called parent solutions, in the current generation of solution set and generates two children solutions using the following method.

Assume there are *M* VCs in the Cloud, VC_1, VC_2, ..., VC_M. The resource capacity allocated to VC_j is recorded in $S[*, j, *]$, which is a two dimensional array. 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).

The validity check will be performed in both children solutions to ensure that the total resource capacity allocated to the VMs in a node is not more than the physical resource capacity of that node. This constraint can be expressed as follows, where *R* is the number of resource types.

For $\forall i, k$: $1 \le i \le N$, $1 \le k \le R$, $$\sum_{j=1}^{M} S[i, j, k] \le 1 \quad (1)$$

If there exists *q* ($1 \le q \le N$) such that Eq.1 does not hold, then the crossover operation is not performed for node n_q.

The crossover operation is designed to consolidate VMs into a smaller number of nodes without adjusting the resource capacities allocated to VMs.

B. The Mutation Operation

After the crossover operation, the mutation operation is then performed over the generated children solutions, aiming to adjust the resource capacity allocated to the VMs to further converge the spare capabilities to a smaller number of nodes. In the mutation operation, the quantity of an element in the matrix $S[i, j, k]$ will be adjusted. So the GA needs to determine which element is adjusted (i.e., determining index *i, j, k*). The procedure is as follows.

i) Determining index i, j, k

A *VC* is first randomly selected (*j* is determined). Then a node is selected (*i* is determined) with the probability proportional to the capacity of the major resource type consumed by the selected *VC* in that node (for the VC which serves CPU-intensive requests, the major resource type is CPU). After that, a resource type is selected (*k* is determined) with the major resource type having a higher probability of being selected than other resource types. The ratio probability of selecting the major resource type to other resource types are set to be *R* and 1, respectively (*R* is the number of resource types in the system). Assume VC_i is selected, VC_i consumes three types of resources and resource r_1 is the major resource type. Then the relative selection probability for resource r_1, r_2, and r_3 is 3:1:1. The reason of giving a higher probability to select the major resource type is because the major resource type has greater impact on the VM's processing capability.

ii) Adjusting resource capacities

After determining *i*, *j* and *k*, $S[i, j, k]$ 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). After the adjustment, the GA calculates the increased service rate in node n_i, denoted as $\Delta\mu$. $\Delta\mu$ is the service rate that can be reduced in another VM, say VM_{qj} ($q \ne i$) and the desired QoS for VC_j can still be satisfied. If there is a VM whose current service rate is less than $\Delta\mu$, then the resources allocated to that VM can be reclaimed. If such a VM does not exist, the capacity of resource r_k allocated to VM_{kj} is reduced by a quantity calculated from the performance model. VM_{qj} is not randomly selected, but with the probability proportional to $1/S[q, j, k]$. The reason for this is that the node with less capacity being allocated to its VM will have higher probability of being selected.

C. The Fitness Function

The fitness function in a GA is used to evaluate the quality of the solutions. In this work, the fitness function is constructed as follows.

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}. Resource fragmentation can be reduced when the spare capacities converge to a smaller number of nodes. Therefore, 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. For example, we prefer the case where a node has balanced spare CPU cycles and memory, rather than the case where there is a large amount of spare memory but fully utilized CPU in one node, while a large amount of spare CPU cycles but fully utilized memory in another. 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.

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

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

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 $\overline{sc_i^s}$, which is partitioned into six bands. Each of the first five bands spans 20% of $\overline{sc_i^s}$ and the weight function falls into the last band 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, VM deletion, VM migration as well as changing a VM's resource capacities. This section analyzes the transition time and presents a cost model for the Cloud reconfiguration. A reconfiguration algorithm is then presented to determine the reconfiguration plan that has low transition time, therefore imposes the low overhead.

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 node n_i is allocated to host VC_j (i.e., VM_{ij}) before running the GA, but is not allocated to host VC_j after. In this case, there are two options to transit the current system 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. m_{ij} can be obtained by monitoring the status of the queue in the VM.

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 when reconfiguring a node, 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 physical 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.

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

If n_i migrates a VM to another node, the reconfiguration procedure will check whether Eq.8 holds for n_i's mapping destination node. If Eq.8 holds, then 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.

When Eq.8 holds for n_i, the reconfiguration procedure of n_i is outlined in 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.5.3 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 end of Subsection V.B.2 mentioned the situation where n_i tries to migrate a VM to a mapping node, but Eq.8 may or may not hold for that node. 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.5.3 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.5.3 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.5.3 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.5.3 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.5.3 does not hold for n_i, the resources will be released until Eq.5.3 satisfies. Then Algorithm 1 is called to reconfigure the node (Step 16).

Algorithm 3. Reconfiguration(n_i)

```
1.  if Eq.5.3 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.5.3 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.5.3 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.5.3 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. Reconfiguring the Cloud

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

```
1.      $\Omega$ = the set of all nodes in the Cloud;
2.      while ($\Omega \neq \Phi$)
3.          Obtain node $n_i$ ($1 \leq i \leq N$) from $\Omega$;
4.          Call Reconfiguration ($n_i$);
5.          $\Omega = \Omega - n_i$;
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 nodes 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 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. A case study is illustrated in Fig.2. Assume there are four VCs (VC_j, $1 \leq j \leq 4$) in the Cloud. Each node has at most four VMs, each belonging to a different VC. Assume n_1 has all four VMs and n_2 has VM_{21} (the VM belonging to VC_2) and VM_{24} (the VM belonging to VC_4) before running the GA. After running the GA, n_1 has VM_{11} and VM_{14} while n_2 has VM_{22} and VM_{23}. The resource capacity of VM_{11} after running the GA is less than that before running the GA, while the resource capacity of VM_{14} after running the GA is greater than that before. The VM distributions in node n_1 and n_2 before and after running the GA can be encoded as in Table I, where 1^- and 1^+ stand for the reduced and increased capacity, respectively, compared with before running the GA.

TABLE I. **VM MAPPING BEFORE AND AFTER RUNNING THE GA**

	n_1	n_2
Before	1 1 1 1	1 0 0 1
After	1^- 0 0 1^+	0 1 1 0

The case study further assumes Eq.8 does not hold for n_1.. Therefore n_1 has to release the resources capacity of VM_{11}, VM_{12} and VM_{13} before VM_{14} can increase its resource capacity. Assume in the current reconfiguration plan, VM_{12} is to be deleted and VM_{13} to be migrated to n_2 (assume n_2 is the mapping destination node). Assume Eq.8 does not hold for n_2 either, and VM_{21} has to migrate to n_3 and VM_{24} migrate to n_4 before VM_{22} and VM_{23} can be created in n_2.

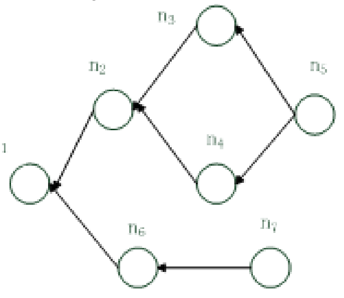

Figure 2. A case study of node dependency during the reconfiguration

Based on the above assumptions, VM operations in n_1 and n_2 as well as their dependencies are as follows. According to the precedence of releasing resources, the sequence of the VM operations in n_1 is: 1) VM_{11} reduces its resource capacity; 2) VM_{12} is deleted; 3) VM_{13} is migrated to n_2. Operation 1) can occur anytime. Operation 2) can occur only after the existing requests in the VM have been completed. Operation 3) depends on VM_{21} and VM_{24} releasing resources, since the creation of VM_{23} can only be performed after VM_{21} and VM_{24} have been migrated. The migrations of VM_{21} and VM_{24} further depend on resource releasing operations in n_3 and n_4, respectively. The chain of dependencies continues until Eq.8 holds for n_5, which means that the VMs in n_3 and n_4 can migrates to n_5 freely and do not depend on other VM operations in n_5. The dependencies between the VM operations in different nodes can also be modeled as a DAG graph. The dependencies between VM operations in n_1 and n_2 can be illustrated in Fig.3.

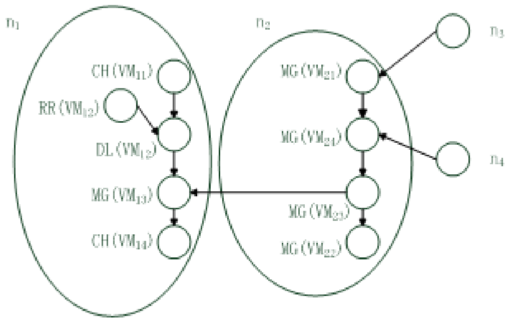

Figure 3. The dependencies between the VM operations in n_1 and n_2

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 these DAG graphs need to be computed. The time of the longest critical path is the transition time of the reconfiguration plan for the whole 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

In this section, we first present the results of the simulation experiments to show the effectiveness of the GA and the Cloud reconfiguration method presented in this paper, and then we present the experimental results of deploying the implementation of CFMV on a real 16-node cluster.

A. Simulation Experiments

A discrete event simulator has been developed to evaluate 1) the performance of the developed GA in consolidating resources, 2) the time spent by the GA in obtaining the optimized system state, and 3) 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 there are three types of VMs: 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%].

N is the number of physical nodes in the cluster, *M* is the number of virtual clusters in the Cloud, *f* is the percentage of the spare capability in a node.

The initial VM-to-node mapping is generated in the following manner.

i) Set the number of VMs in a node is *b* (b is set to be 3 in the simulation experiments, unless otherwise stated);

ii) Use the resource selection ranges above to generate $\lfloor b*N/3 \rfloor$ computation-intensive VMs, $\lfloor b*N/3 \rfloor$ for memory-intensive VMs, and $(b*N - 2*\lfloor b*N/3 \rfloor)$ I/O-intensive VMs;

iii) Calculate the average size of the VCs (i.e., the number of VMs in a VC) as $b*N/M$;

iv) Use the first fit algorithm [34] to generate the initial VM-to-node mapping, i.e, for VM_{ij}, search the nodes starting from n_1, if the node has enough capacity (after deducting the *f* spare capability) to accommodate VM_{ij}, then map VM_{ij} to the node.

GA takes as input the initial system state generated as above and calculates an optimized state.

Other experimental settings are detailed in individual experiments.

Representative times in the literature [34][39] were assumed in our simulation experiments. 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 [34][39]. The migration time in our experiments is in the range of 10 to 32 seconds.

1) *Performance of the GA*

a) *Impact of the Number of Physical Nodes*

Fig.4 shows the number of nodes saved as the GA progresses. In the experiments in Fig.4, the number of nodes with active VMs varies from 50 to 200. The experiments aim to investigate the time that the GA needs to find an 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.4, 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. For example, when the number of nodes is 50, the optimized solution is almost reached after the GA runs for about 13 seconds, while it takes the GA about 53 seconds to reach the optimized solution when the number of nodes is 200.

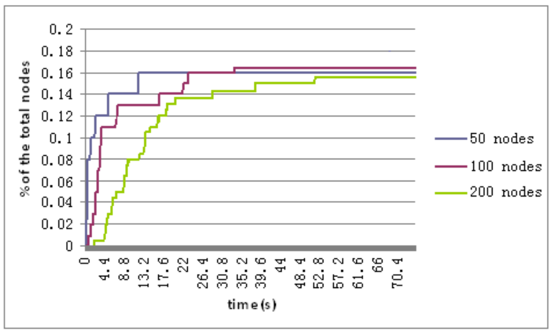

Figure 4. The quantity of nodes saved as the GA progresses

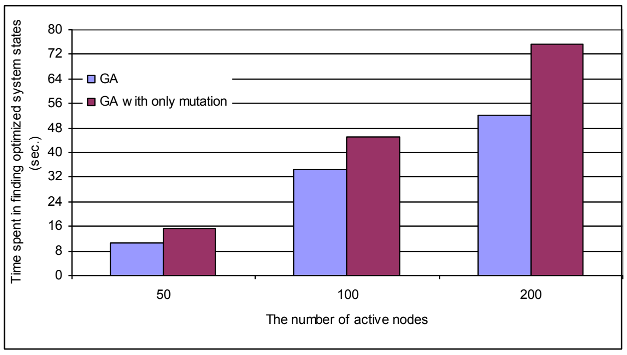

Figure 5. Resource consolidation by the GA with only mutation operations; the experimental settings are the same as in Fig.4.

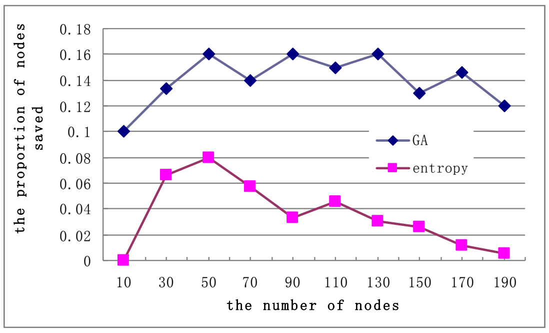

Figure 6. The comparison between the GA and entropy; the average free resource capacity is 20%

In the GA, the crossover operation is used to consolidate resources without adjusting the resource capacities allocated to VMs, while the mutation operation is performed to further converge the spare capabilities to a smaller number of nodes. In order to give insights into the effects of these two operations, we conducted the experiments in which the resources were consolidated by the GA performing only mutation operations. In these experiments, we found that although the GA only performed mutation operations, it could still find optimized system states. However, the GA performing only mutation operations had to spend much longer time to reach the optimized states. Fig.5 compares the time spent by the GA performing both crossover and mutation operations with the time by the GA only performing mutations. The experimental settings are the same as in Fig.4. As can be seen from this figure, under all experimental settings, the time spent by the GA performing only mutations is significantly longer. These results can be explained as follows. The crossover operation essentially tries to find a better way to pack the VMs into physical nodes. Without the crossover operation, although mutation operation can also eventually achieve the same effect by adjusting resource capacities of individual VMs, the process would be much longer. This is because the mutation operation is performed on a single VM each time, while the crossover operation is performed on a set of VMs.

Fig.6 compares the GA developed in this work with the Entropy consolidation scheme presented in [34]. 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 "moldable" 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.

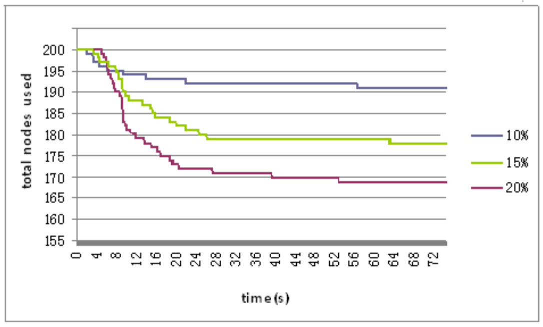

Figure 7. The impact of free resource capacity in nodes on the performance of GA; the initial number of nodes used are 200; the number of the Virtual Clusters in the system is 30;

b) *Impact of Free Capacity*

Fig.7 demonstrates how the GA performs under different level of free capacity in the physical nodes. In the experiments presented in Fig.7, the number of the VCs in the system is 30 while the free capacity of the resources in each node varies from 10% to 20%. It can be observed from this figure that the number of nodes used to host the VCs decreases as the level of free capacity increases. This result demonstrates the effectiveness of the developed GA in exploiting the free resources to consolidate the VMs into a smaller number of nodes. It can also be seen from the figure that although the time that the GA spends to approach the optimal solution increases as the level of free capacity increases, the increase is moderate (not as big as when the number of nodes increases). When the level of free capacity increases from 10% to 20%, the time the GA takes to almost reach the optimized solution increases from 22 seconds to 27 seconds. This result suggests that the level of free capacity in the nodes does not have big impact on the running time of the GA.

c) *Impact of the Number of VCs*

Fig.8 shows how the GA performs under different number of VCs. In this figure, the total number of VMs in the Cloud is fixed to be 600, while the number of VCs varies from 20 to 40. When the number of VCs in the Cloud is 20, 30 and 40, the average number of VMs that each VC has is 30, 20 and 15, respectively. As seen from this figure, the number of nodes used to host the VCs decreases in all cases as the GA progresses, which is to be expected. It can also be observed that the time that the GA spends to approach the optimized solution becomes longer as the number of VCs increases. When there are 20 and 40 VCs, for example, the GA takes 17 and 52 seconds, respectively, to achieve the near-optimal solution. Another observation is that although the free resource capacity is 20% in all cases, the final number of consumed nodes calculated by the GA is different under different number of VCs. As observed from the figure, the number of consumed nodes decreases as the number of VCs in the Cloud increases. This result can be explained as follows. According to the experimental settings in the figure, when there are more VCs, the granularity of a VC in terms of the number of VMs is smaller. Therefore, the GA has more opportunities to consolidate the VCs into a smaller number of nodes. This result shows that the number of VCs has the mixed impact on the GA's performance. When more VCs are hosted, a longer time may be taken to reach the optimized solution, but potentially more resources may be saved. This gives the insight into how to determine a suitable number of VCs in a Cloud, given a certain number of underlying physical nodes.

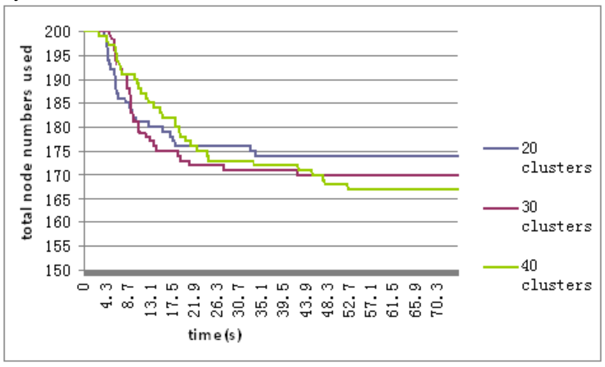

Figure 8. The impact of the number of VCs on the performance of the GA; the number of physical nodes is 200; the average level of free resource capacity in nodes in 20%.

2) *Performance of the Cloud Reconfiguration*

Fig.9 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%.

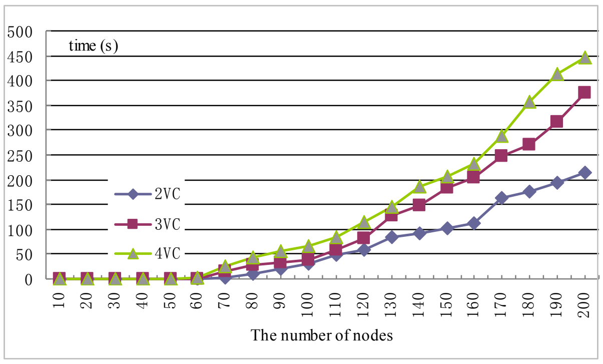

Figure 9. the execution time of the reconfiguration algorithm for different number of nodes and VCs

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).

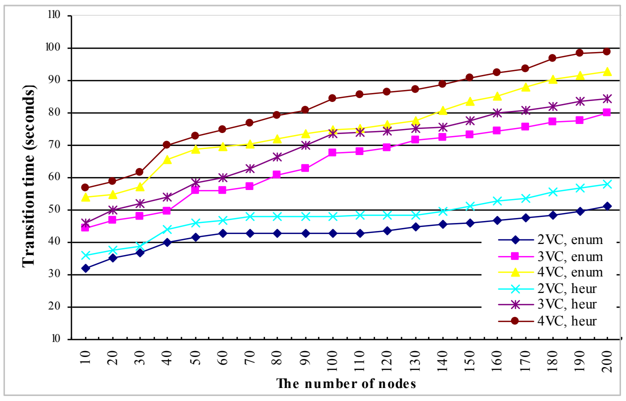

Figure 10. the transition time of the optimal reconfiguration plan obtained by the enumeration approach as well as the sub-optimal plan obtained by the heuristic approach

Fig.10 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. Further observations and analysis show that the number of nodes and the number of VCs have different impact on the transition time of the reconfiguration plan. In the enumeration approach, when the number of nodes increases from 10 to 200, the transition time increases by 20.4 seconds from 32 to 52.4 seconds, while the transition time increases by 22 seconds from 32 to 54 seconds when the number of VCs only increases by 2 from 2 to 4. This result can be explained as follows. When the number of VCs increases, the number of VMs in a node will also increase. Therefore, more VM operations will be performed in a node during the Cloud reconfiguration. Since the VM operations in the same node can only be performed in sequence, the time spent by the VM operations in a node will increase substantially. On the contrary, when the number of nodes increases, the VM operations in different nodes may be performed in parallel unless they have dependencies as analyzed in subsection V.C.

It can also be observed from Fig.10 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 fairly good reconfiguration plan.

B. *Experiments on a Real Cluster*

We implemented the resource Consolidation Framework for Moldable VMs (CFMV) developed in this paper. The framework is implemented by extending an existing consolidation manager, called Entropy, in literature [34]. In the implementation, new functions are added in the source code of Entropy to calculate a better system state (i.e., the VM-to-node mapping and the resource capacity allocated to each VM) and an optimized reconfiguration plan according to the methods presented in this paper. Entropy provides the mechanism to perform the actual VM operations in physical machines, including VM deletion, VM creation and VM migrations. CFMV implements the codes to perform the operation of adjusting resource capacities allocated to a VM, since Entropy only deals with rigid VMs. In CFMV, the newly added functions of finding an optimized reconfiguration plan interface with the mechanism of performing actual VM operations and instruct the mechanism to reconfigure the Cloud to the system state calculated by CFMV.

In the experiments, CFMV is deployed on a cluster. The cluster consists of 16 physical machines. Each machine has a Pentium-4 3.2GHz CPU and 2GB of memory. The machines run on a 1Gbps Ethernet network. The 16 machines run the VMs using Xen 4.0.1.

A VM hosts a RUBiS benchmark [10]. RUBiS is an auction site prototype modelled after eBay.com [10]. It handles the external transaction requests. In this experiment, the "browse only" transactions are used. Up to 4 RUBiS applications, called RUBiS$_1$ to RUBiS$_4$, are hosted in the cluster, and the VMs hosting RUBiS$_i$ form the Virtual Cluster i (VC$_i$). In the experiments, we only consider two types of resource: CPU and memory.

The work in [10] uses Layered the Queuing Network (LQN) models to construct the performance models for the RUBiS benchmark. These performance models are adopted in the experiment. According to the performance model, the service rate of a VM with certain resource capacities can be obtained. In the experiments, the desired QoS delivered by VC_i is expressed as the total service rate of all VMs in the

VC cannot be less than qos_i. qos_i is calculated as λ_i/ρ, where λ_i is the arrival rate of the requests for RUBiS$_i$, and ρ is the desired resource utilization of the system, which is set to be 80% in the experiments. On the cluster, we also benchmarked the costs of VM operations that may be performed during the Cloud reconfiguration, including VM creation, VM deletion, VM migration and VM adjustment.

One additional machine is used to emulate the clients and generate the transaction requests. During the experiment duration, the requests are submitted to the cluster following the Poisson process. In order to evaluate the consolidation ability of the framework, the initial resource allocation is conducted for given arrival rates of requests for RUBiS$_1$-RUBiS$_4$ in the following manner: i) make sure that there is f percentage of spare capacity in each physical node, ii) allocate 10% of a node's resource capacity to run Dom0, iii) the remaining resource capacity is evenly divided among the VCs whose Qualities of Service have not been satisfied yet.

The architecture in Figure 1 is adopted in the experiments. In the implementation of CFMV, the Global Manager (GM) and the Local Managers (LM) are situated in the head node of the cluster. They work in the following fashion.

- LM$_i$ (the Local Manager of VC$_i$) i) calls XenAPI to obtain the resource capacities allocated to the VMs in VC$_i$, ii) uses the performance model in [10] to calculate the service rate of each VM for the RUBiS application (s_{ij} denotes the service rate of VM$_{ij}$), iii) search the nodes in the order from node 1 to node 16 to find such a set of VMs that the VMs' total service rates is no less than qos_i, iv) for a given arrival rate of the requests for VC$_i$, calculates the proportion of the requests dispatched to VM$_{ij}$ (denoted as α_{ij}), which is proportional to s_{ij}, i.e., $\alpha_{ij}/s_{ij}= \alpha_{ik}/s_{ik}$.
- GM i) collects the system state of VC$_i$ from LM$_i$, ii) records the number of physical nodes with active VMs (i.e., the VMs that serve the requests), iii) invokes CFMV to find an optimized system state and an optimized reconfiguration plan, iv) reconfigures the Cloud, and v) records the number of the physical nodes with active VMs after Cloud reconfiguration.

For comparison, Entropy is also used to consolidate the resources in the experiments.

Fig.11 shows the number of nodes saved by CFMV and Entropy, in which the spare capability of each node is set as 30%. It can be seen from this figure that CFMV can save more nodes than Entropy after consolidation. This is because CFMV allows the VMs to adjust the resource capacities, and therefore offers more flexibility to "squeeze" the VMs into a smaller number of nodes.

In order to evaluate the impact of the spare capacity on the number of saved nodes, we also conducted the experiments in which the spare capacity in each node is set as 10%. The results are shown in Fig.12. As can be observed from this figure, there are no nodes being saved except when the arrival rate is 400 under CFMV. This is because the spare capacity in each node is small and the total spare capacity in the cluster is not big enough to free physical nodes. This result suggests that the spare capacity in the nodes impacts on the number of nodes that can be saved by consolidation. The less spare capacity in the active nodes in the cluster, the less number of nodes can be saved.

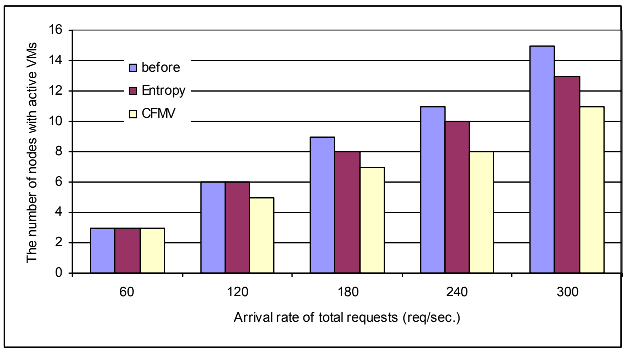

Figure 11. The number of nodes saved under CFMV and Entropy under different arrival rates of requests; the proportions of the requests for RUBiS1-RUBiS4 are 10%, 20%, 30% and 40%, the spare resource capacity in each node is 30%.

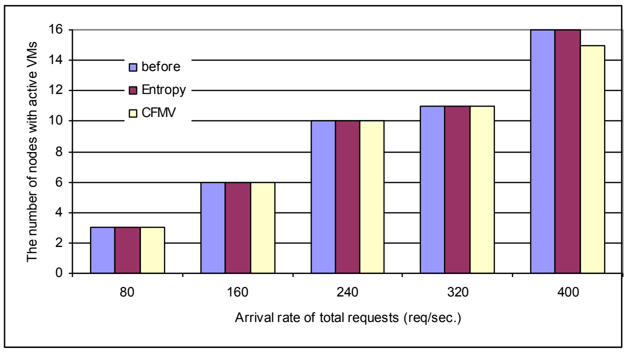

Figure 12. The number of nodes saved under CFMV and Entropy under different arrival rates of requests; the proportions of the requests for RUBiS1-RUBiS4 are 10%, 20%, 30% and 40%, the spare resource capacity in each node is 10%.

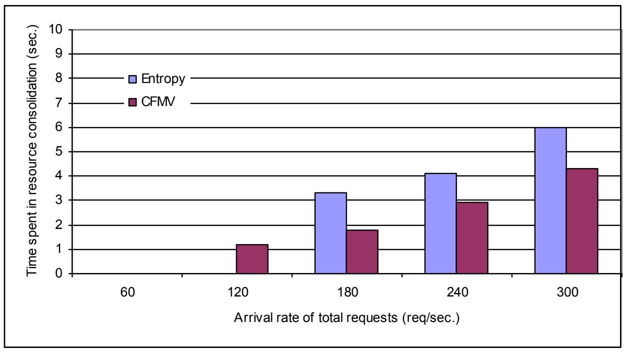

Figure 13. Time spent in resource consolidation by CFMV and Entropy; the experimental settings are the same as in Fig.11; if the resources cannot be consolidated, the time data are not shown.

Fig.13 shows the time spent by CFMV and Entropy to complete the consolidation computations in the experimental settings in Fig.11. In the figure, if no nodes can be saved, the time data are not depicted. As can be seen from the figure, CFMV spends less time in completing consolidation than Entropy. This may be because CFMV can adjust the VMs' resource capacity to help consolidate resources, and therefore it can reach the optimized system state more quickly than Entropy, because Entropy may have to try more possibilities to see if the VMs can be packed into a smaller number of nodes.

Fig.14 shows the reconfiguration cost (i.e., the transition time from the current system state to the state calculated by the consolidation framework) in the experimental settings in Fig.11. Again if no nodes can be saved, the data are not depicted in the figure. It can be seen from this figure that the reconfiguration cost in CFMV is less than that in Entropy. This can be explained as follows. Entropy only uses VM migrations to reconfigure the Cloud, while CFMV may delete and create VMs if the VM migration operation is likely to incur higher transition time.

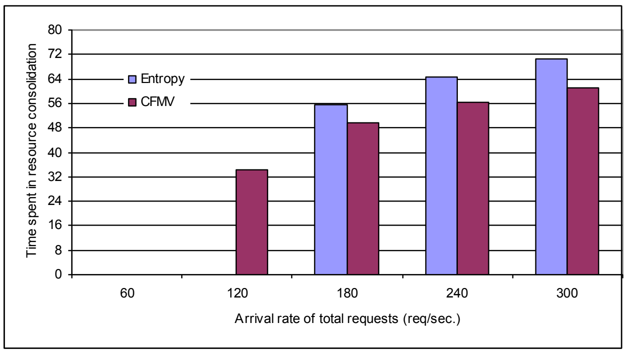

Figure 14. Reconfiguration cost under CFMV and Entropy; the experimental settings are the same as in Fig.11; if the resources cannot be consolidated, the data are not shown.

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.

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] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In Proceedings of the 5th Symposium on Operating System Design and Implementation, pages: 181-194. Boston, MA, Dec. 2002.

[6] P. Padala, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, and K. G. Shin. Adaptive control of virtualized resources in utility computing environments. In Proceedings of the European Conference on Computer Systems(EuroSys), pages: 289-302, 2007

[7] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Shin. Automated Control of Multiple Virtualized Resources. In Proceedings of the European conference on Computer systems (EuroSys), pages: 13-26, Mar 2009.

[8] 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.

[9] 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.

[10] 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.

[11] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. IEEE International Conference on Autonomic Computing, pages: 5-14, 2006.

[12] 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.

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

[14] 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.

[15] 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.

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

[17] K. Keahey, T. Freeman. Contextualization: Providing One-click Virtual Clusters. In Proceedings of the 2008 4th IEEE International Conference on eScience, pages: 301-308, 2008.

[18] Apache Tashi, http://incubator.apache.org/tashi/

[19] Enomaly: Elastic/Cloud Computing Platform, http://www.enomalism.com.

[20] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, and et al. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. IBM Journal of Research & Development, Volume 53, Number 4, 2009.

[21] P. Ruth, P. McGachey, and D. Xu. VioCluster: Virtualization for dynamic computational domains. In Proceedings of the IEEE International Conference on Cluster Computing (Cluster), 2005.

[22] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, and et al. From Virtualized Resources to Virtual Computing Grids: The In-VIGO System. In Future Generation Computer Systems, pages: 896-909, June 2005.

[23] I. Krsul, A. Ganguly, J. Zhang. VMPlants: Providing and Managing Virtual Machine. Execution Environments for Grid Computing. In Proceedings of the 2004 ACM/IEEE conference on Supercomputing, page: 7, 2004.

[24] 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.

[25] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An Extensible Framework for Managing Clusters of Virtual Machines. In

Proceedings of the USENIX Large Installation System Administration Conference (LISA), 2007.

[26] 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.

[27] R. Nathuji, K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), pages: 265-278, October 2007.

[28] R. Nathuji and K. Schwan. VPM Tokens: Virtual Machine-Aware Power Budgeting in Datacenters. In Proceedings of the ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC), Boston, Massachusetts, USA June 2008, pages: 119-128, June 23–27, 2008.

[29] H. Chen, H. Jin, Z. Shao, K. Yu, K. Tian. ClientVisor: leverage COTS OS functionalities for power management in virtualized desktop environment. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE), pages: 131-140, 2009.

[30] 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 Conferenc on Cluster Computing (Cluster 2008), IEEE Computer Society, Japan, pages: 13- 22, Sept. 29 2008-Oct. 1 2008.

[31] S. Kumar, V. Talwar, P. Ranganathan, R. Nathuji, and K. Schwan. M-Channels and M-Brokers: Coordinated Management in Virtualized Systems. In Proceedings of the Workshop on Managed Many-Core Systems (MMCS, in conjunction with HPDC), June 2008.

[32] H. Amur, R. Nathuji, M. Ghosh, K. Schwan, and H. S. Lee. IdlePower: Application-Aware Management of Processor Idle States. In Proceedings of the Workshop on Managed Many-Core Systems (MMCS, in conjunction with HPDC), June 2008.

[33] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (ASPLOS), pages: 48-59, 2008.

[34] 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

[35] 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

[36] L. Hu, H. Jin, X. Liao, X. Xiong, H. Liu, "Magnet: A Novel Scheduling Policy for Power Reduction in Cluster with Virtual Machines", 2008 IEEE International Conference on Cluster Computing

[37] L. Kleinrock, Queueing system, John Wiley & Sons, 1975.

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

[39] 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

[40] V. Petrucci, O. Loques, D. Mosse "A dynamic optimization model for power and performance management of virtualized clusters", In: e-Energy '10. pp. 225-233. ACM, New York, NY, USA (2010)

[41] M. Maurer, I. Brandic, and R. Sakellariou, "Enacting SLAs in Clouds Using Rules", Euro-Par 2011, Bordeaux August 29th - September 2nd 2011, France

[42] http://en.wikipedia.org/wiki/Eucalyptus_(computing)

[43] L. He, D. Zou, Z. Zhang, K. Yang, H. Jin and S. Jarvis, "Optimizing Resource Consumptions in Clouds", *The 12th IEEE/ACM International Conference on Grid Computing (Grid 2011)*, 2011