Real Time Elastic Cloud Management for Limited Resources

Sijin He, Li Guo, Yike Guo
Department of Computing,
Imperial College London,
London, United Kingdom
E-mail: {sh107, liguo, yg}@doc.ic.ac.uk

***Abstract*— An Infrastructure-as-a-Service (IaaS) provider is usually assumed to own a large data centre with significant computational resources. For a small or medium sized Internet Data Centre (IDC), offering cloud computing service is a nature of business model but there are technical barriers which need to be resolved. One of the key issues is ineffective resource management given such an IDC usually has only limited resource. In this paper, we propose an efficient resource management solution specially designed for helping small and medium sized IaaS cloud providers to better utilise their hardware resources with minimum operational cost. Such an optimised resource utilisation is achieved by a well-designed underlying hardware infrastructure, an efficient resource scheduling algorithm and a set of migrating operations of VMs.**

Keywords- Cloud Computing; IaaS; Resource Scheduling; VM Migration

I. INTRODUCTION

Driven by the rapid growth of the demand for efficient and economical computational power, cloud computing, in the last few years, has led the world into a new era. By applying virtualisation technology, it enables Internet Data Center (IDC) to provide virtual computational services (computing power, storage, and networks) so that users are able to consume them over internet as utilities.

Many cloud providers such as Amazon EC2 [1], GoGrid [2], Salesforce.com [3] have made their great success, which encourages more conventional IDCs to get involved into this business. However, unlike those giant providers, conventional IDC providers, especially those small and medium sized ones still find that it is difficult to answer some of the critical questions for their cloud operations. They, in particular, are short of proper knowledge for better resource utilisation in order to reduce their operational costs which is crucial to their business success as they normally only have limited resources. Moreover, it becomes possible that a cloud federation can be built where several IDCs can aggregate their resources with a resource-profit sharing agreement to provide uniform services to users.

Resource utilisation in clouds is an important issue that affects not only the IDC's business operations but also the performance of the cloud experienced by its clients and the prices paid by them. There are many existing works that relate to the resource scheduling optimisation. They try to solve the issue by adopting scheduling algorithms which are applied before the resources are given to the users. From providers' perspective, how their hardware is utilised is also determined at the users' request submission time as shown in Figure 1. We call it a pre-scheduling algorithm.

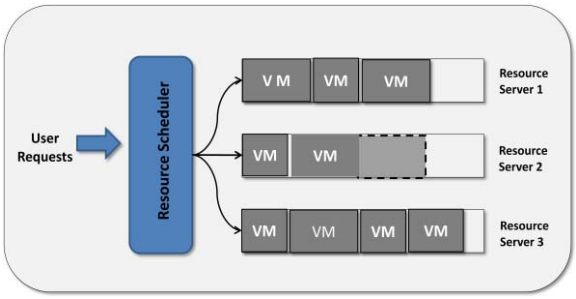

Figure 1. Pre-scheduling Algorithm for Cloud Resource Allocation

We believe that such an approach has the following drawbacks:

- There are always some unused resources left on the resource servers due to the unpredictable sizes of incoming user requests. Without accurate historical data, it's hard to schedule resource allocation properly.
- Memory balloon and CPU virtualisation are widely supported features of virtualisation technology. It enables users to configure their virtual machines [4] (VMs) memory size and CPU number at operation time, which makes the scheduler work less efficient. For instance, at a request time, a user may submit his resource requirement with two CPUs and 2GB memory but he could change the memory size to 4GB later on. If many users on a same resource server are trying to execute the same operation, some of their requests may fail due to the unavailability of the physical resources.

To address the problem, in this paper, we propose an efficient resource management solution aiming to help small and medium sized cloud providers to better utilise their hardware resources with minimum operational cost. Such an optimised resource utilisation is achieved by a well-designed underlying hardware infrastructure, a real time resource scheduling algorithm, and a set of migrating operations of VMs.

978-0-7695-4460-1/11 $26.00 © 2011 IEEE
DOI 10.1109/CLOUD.2011.47

The rest of the paper is organised as follows: Section II reviews the related work of existing virtualisation techniques, VM migration methods and applications, and cloud resource scheduling algorithms and applications. Section III describes a real time resource reallocation algorithm for better resource utilisation of bounded resources. Section IV presents an ideal cloud system infrastructure for resource reallocation. System design and implementation are presented in Section V, and we conclude the paper and suggest some future work in Section VI.

II. Related Work

A. Virtualisation Techniques

Virtualisation in modern computing has been implemented using different approaches. Two significant techniques that have been heavily deployed in cloud computing are full virtualisation and paravirtualisation. Full virtualisation [5] provides a complete VM enabling unmodified guest operating system (guest OS) to run in isolation. It provides flexibility to run different versions of different operating systems, however, the guest OS does not know that it is being virtualised that would lead to some performance issues. Paravirtualisation [6] provides a complete but specialised VM to each guest OS allowing modified guests to run in isolation. It provides a lightweight and near native speed, and allows the guest OS to cooperate with hypervisor to improve performance. However, this technology is only limited to open source guest OS.

Among various virtualisation systems nowadays, Xen, Kernel Virtual Machine (KVM) and VMware are popular due to they have a combination of features that make them uniquely well suited for many important applications. Xen [5] is one of a few Linux hypervisors that support both full virtualisation and paravirtualisation. It allows several guest OS running on domains (VMs) to execute on the same computer hardware concurrently. The hypervisor has direct access to the hardware and also allows those VMs to run on top of it. Each guest OS uses a pre-configured share of a physical machine. A privileged domain called Domain0 is a bare-bone OS that actually controls physical hardware and create, configure, migrate, or terminate other VMs. KVM [7] is a newcomer of the virtualisation system. It not only can create and run multiple virtual machines, but also supports full virtualisation. It is a modification to the Linux kernel that actually makes Linux into a hypervisor on inserting a KVM module. One of the most interesting KVM operation is that each guest OS running on it is actually executed in user space of the host system. This approach makes each guest OS look like a normal process to the underlying host kernel. VMware [8] is the first company to offer commercial virtualisation technology. It offers a hypervisor called ESXi server [9] that also supports full virtualisation. Paravirtualisation can also be supported by using VMI [8].

B. VM Migration

One of the most prominent features of these virtualisation systems is that they all support Live Migration [10] which allows for the transfer of a VM from one physical machine to another, with little or no downtime of the services hosted by the VM. It transfers the working state and memory of a VM across the network, while they are running. This has been already a built-in feature for both Xen and KVM. Recently, VMware has also added Live Migration, called VMotion [11]. Other architectures including Microsoft's Hyper V and Advanced Micro Devices' AMD-V also support this feature.

VM migrations have been deployed for many different purposes in cloud computing industry and academia. Wood et.al [12] presents a system that automates the task of monitoring and detecting hotspots, determining a new mapping of physical to virtual resources and initiating the necessary migrations. Voorslurys et.al [13] evaluate the effects of live migration of virtual machines on the performance of applications running inside Xen VMs. Grit et.al [14] explores architectural and algorithmic issues for resource management policy, such as they investigate the number of migrations required when the broker and a site scheduler use conflicting policies, and also show how migration is a useful mechanism to resolve conflicts between the separate provisioning and placement policies. Zhao and Figueiredo [15] provide a model that can characterize the VM migration process and predict its performance, based on a comprehensive experimental analysis.

C. Cloud Resource Scheduling

Cloud resource scheduling is a very important feature in cloud system infrastructure. Many excellent applications have been developed, for example, Librato Silverline [16] is a Software-as-a-Service for fine-grained monitoring and management of application workloads, and maximizing server resource utilization in clouds.

Researchers in cloud computing have contributed great efforts to the cloud resource scheduling. Abdelzaher et.al [17] describes performance control of a Web server using classical feedback control theory to achieve overload protection performance guarantees and service differentiation in the presence of load unpredictability. Assuncao et al. [18] evaluate the cost of six scheduling strategies used by an organization that operates a cluster managed by virtual machine technology and seeks to utilize resources from a remote IaaS providers in order to reduce the response time of user requests. Stage et al. [19] describes an optimization formulation that determines an optimal schedule of virtual machines, which needs to satisfy the demand of a particular application. The algorithm can be used to select the cost minimal offering of different hosting providers, but also to control the allocation and de-allocation of virtual machines with an IaaS provider over time.

Capacity planning has also been employed in the provisioning of cloud infrastructures. This is a powerful

tool for managing the quality of service (QoS) to clients. Allspaw [20] and Menasce and Almeida [21] both present the basic concepts, techniques and tools for capacity planning. Garg [22] uses these techniques in provisioning Web Services Applications. Lopes et.al [23] proposes a model that can be used for evaluating the gains that can be achieved with the planning of the infrastructure capacity in a number of scenarios.

III. Real Time Resource Reallocation for Better Resource Utilisation

As explained earlier in Section I, because of the elasticity of cloud resource provision, only applying a pre-scheduling approach cannot optimally improve the resource utilisation. Some cloud providers are tackling this problem using resource reservation approach. They require users to setup a lower and an upper limit of the requested resource and use these limits for their resource scheduling. However, for the worst case, this would cause huge amount of resource wasting. During the operation time, users may scale their resource down to the lower limit in order to save cost. Although the unused resources are released, cloud providers cannot reallocate them for other purposes as they need to guarantee that the user will get his upper limit when he resumes his resource setup.

We can clearly see that to fully utilise the cloud resources without affecting users' normal operations, run time resource reallocation is required. There are two cases that occur in a cloud provider's day-to-day operation where resource reallocation needs to be carried out:

- *Use Case One*: We assume the cloud provider only has two resource servers (Server 1 and 2) and each has 20GB memory, running five VMs in total. Server one serves for VM 1 to VM 4, each with 2GB memory and VM 5 with 12GB memory is running on Server Two. If a user submits a request for 16GB memory, there is not sufficient memory for it on either server and therefore resource reallocation needs to be applied. The use case is depicted in Figure 2.

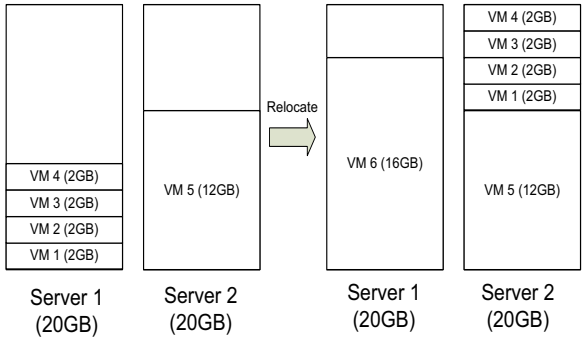

Figure 2. Resource Reallocation for Resource Fragment Collection

- *Use Case Two*: Multiple VMs are located on a same resource server trying to extend their resources to larger scales. As a result, the sum of the extension exceeds the capacity of the underlying server; resource hence needs to be reallocated. For example, we assume that there are just two resource servers in an IDC. They both have memory capacity of 6GB. Two VMs on resource server 1 occupy 2GB memory and three VMs on resource server 2 occupy 4GB memory as shown in Figure 3. VM 1 and VM 2 on resource server 2 are both required to scale up their memory to 3GB. However, scaling up the memory for both VM 1 and VM 2 at the same time will cause the occupied memory reaching 7 GB which exceeds the 6 GB memory capacity on the resource server 2. The only possible solution is to move either VM 1 or VM 2 to the resource server 1 where there is enough free memory space (4GB) for either VM 1 or VM 2 to scale up its memory.

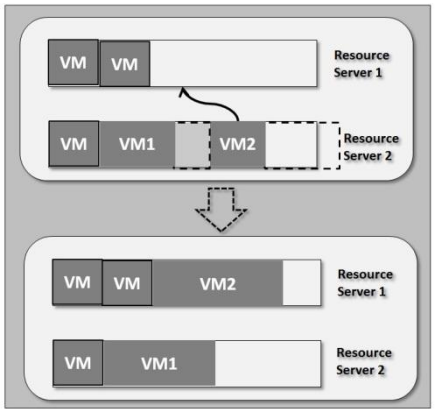

Figure 3. Resource Reallocation for VM adjustment

Both use cases require VMs to be moved around in order to improve the resource utilisation. Fortunately VM migration provides us a good foundation for such dynamic resource management. We describe how the resource reallocation should be carried out using Real Time Resource Reallocation algorithm which is specifically designed for a specific type of cloud system infrastructure that is presented in Section IV. This algorithm helps us to determine the best suitable node for a VM operation (such as creation, migration and resizing, etc.) which is submitted by a user under the use cases described earlier.

A. Properties for Real Time Resource Reallocation

The first step is to define the properties and describe each property in a resource server (node). Suppose there are n resource nodes, each node i contains a set of m properties $P_i = \{v_1, v_2, \dots, v_j, \dots, v_m\}$, each property is associated with a weight w_{ij} and a preference f_j, such that $w_{1j} + w_{2j} + \cdots + w_{ij} + \cdots + w_{nj} = 1$ and $f_1 + f_2 + \cdots + f_j + \cdots + f_m = 1$. The following describes some possible properties of a node:

1) Available memory for resource reallocation

The following formula calculates the available memory for resource reallocation in a node:

$$AM = UM + \sum MSVM \qquad (1)$$

where AM is the available memory for resource reallocation in the node, UM is the unused memory in the node and $MSVM$ represents the memory occupied by a shut-down VM in the node. The equation 1 calculates the available memory for resource reallocation in the node by adding the unused memory to the total occupied memory of all shut-down VMs in the node

Availability of CPUs and hard disks will not be taken into account as the CPUs are time-sharing and VMs are located in the shared storage that will be explained later in Section IV.

2) Hypervisors

Different hypervisors can be supported, such as Xen, KVM, and VMware ESXi server, etc.

3) The purpose of the node

Different purposes can be defined, such as computing, hosting, backup, or any other purposes.

4) Hardware

Hardware properties can also be taken into account, such as physical devices, hardware speeds, and network speeds etc.

5) Others

Any other properties, such as name of the node, age of the node, etc., can also be added.

B. Matching Properties with a query

The following algorithm is to find the most suitable node for resource reallocation where a query $Q = \{r_1, r_2, \dots, r_j, \dots, r_m\}$ is required to be the input. Each r_j in Q is a requirement corresponding to a property v_j in P. The algorithm returns a list of suitable nodes ($SNset$) by matching every property in every node with a given query.

```
1  for(∀j: r_j ∈ Q)
2      c := 0
3      for(∀i: v_j ∈ P_i )
4          if(r_j = null)
5              w_ij := 1
6          else
7              if(match(v_j, r_j))
8                  c := c + 1
9              else
10                 w_ij := 0
11     for(∀i: w_ij ∈ P_i ∧ isAssign(w_ij))
12         w_ij := 1/c
13 for(∀i: w_ij ∈ P_i )
14     N_i := w_i1 × w_i2 × ... × w_in
15 SNset := {}
16 for(∀i: N_i ∧ (N_i > 0))
17     addElement(SNset, N_i)
```

The above algorithm starts from the first requirement in the query (line 1), initialises the counter c to be 0 (line 2), and iterates through the first property for all nodes (line 3). If a requirement is not stated in the query, the weights of the corresponding properties in the nodes change to 1 (line 4 and 5). If the requirement matches the corresponding property in the corresponding node, the counter will be incremented by 1 (line 7 to 8). Otherwise, the weight of the corresponding property in the corresponding node changes to 0 (line 10). The weight of the corresponding property in a node that has not yet been assigned a value will be updated by using the formula $1/c$ (line 11 and 12). When a requirement matches a property of a node, it means that it is a binary relation operation that can normally be a string or numerical comparison. A value N_i for each node i is calculated by multiplying all weights of property in node i (line 13 and 14). Every non-zero N_i is added to the suitable node set, i.e. $SNset$ (line 15 to 17).

For example, suppose there is a query $Q = \{2GB, Xen\}$ which indicates a Xen VM with 2GB memory is required to create. There are three nodes, each with two properties, the available memory for resource reallocation and the type of hypervisor using in nodes: node one with the property $\{4GB, KVM\}$, node two with the property $\{1GB, Xen\}$, node three with the property $\{4GB, Xen\}$. After we compare the available memory property for every node with the memory requirement in the query, node one and three are matched. We then compare the type of hypervisor property for every node with the required type of hypervisor in the query, node two and three are matched. We then obtain $N_1 = 0$, $N_2 = 0$ and $N_3 = \frac{1}{4}$. Therefore, the node three is added to the suitable node set.

C. Choosing the Most Suitable Node

If there is more than one node in the suitable node set, we have to decide which node in the set suits the best for a client. Therefore, service differentiation is another important factor that has to be considered. It means clients may have different priorities due to different user levels. The system needs to give preferential attention to more important clients. The preference f for every property in a node is manually defined that helps the system to choose the most appropriate node from the set of the suitable nodes. The greater the value of the preference, the more preferable property it is. Different clients have different preferable values for each property depending on the service differentiation.

Suppose that the suitable node set $SNset$ has n nodes and a preference set $F = \{f_1, f_2, \dots, f_j, \dots, f_m\}$ where $f_1 + f_2 + \cdots + f_j + \cdots + f_m = 1$. Each preference in the set corresponds to a property in a node. The algorithm for choosing the most suitable node is stated in TABLE I.

TABLE I. ALGORITHM FOR CHOOSING THE MOST SUITABLE NODE

1. Rank the preference set in a descending order.
2. Choose the most preferable property from the suitable node set, i.e. choose the one with highest value in the set.
3. Find all corresponding values of the chosen property from $SNset$.
4. Find the maximum value from those values. If there are more than one maximum values, go back to step 2 and start with the next most preferable property.
5. Otherwise, exit the algorithm and choose the node which has the maximum value of the chosen property as the most suitable node.

For example, A VIP client needs to create a VM on a more powerful server. We only consider two properties in the server: hard disk speed and CPU speed, each with the preference value 0.3 and 0.7 respectively. Suppose there are two nodes in the suitable node set, Node one has the property $\{4200\,rpm, 3.3\,GHz\}$ and node two has the property $\{7200\,rpm, 3.3\,GHz\}$. We first compare the most preferable properties between two nodes: the CPU speeds of two nodes are both 3.3GHz, which mean that there are more than one maximum values in that property. We then continue to compare the second most preferable property: node two has a greater hard disk speed than that of node one. Therefore, node two is chosen to be the most suitable node for creating a VM. If a non-VIP client needs to create a new VM, node one might be chosen as it is less powerful, leaving the powerful node for VIP clients. The strategy of selecting a suitable node for a specific type of clients depends on the service differentiation policy which is completely defined by the providers.

IV. SYSTEM INFRASTRUCTURE FOR REAL TIME RESOURCE REALLOCATION

In practice, VM migration is a crucial issue that needs to be resolved in the cloud computing industry. Depending on the underlying network speed, VM migration takes really a considerable long time. According to our experience, migrating a VM that has 2GB Memory and 20GB hard disk over a gigabytes network requires at least 30-40 seconds. This is not feasible in real business operation. The situation certainly can be improved by optical fibre enabled network infrastructure but for small and medium sized cloud providers, this would require them to restructure their current network setup as most of them are Ethernet enabled. Even with high speed local network connection, moving large quantities of data around would jam the whole network and largely reduce system performance.

A. Shared Storage Cloud System Infrastructure

To further tackle this issue, we should reduce the data amount transferred in order to improve the migration speed. Looking into the data transferred, the majority of data is from the VM disk which is normally 10 or 100 times larger than the memory size.

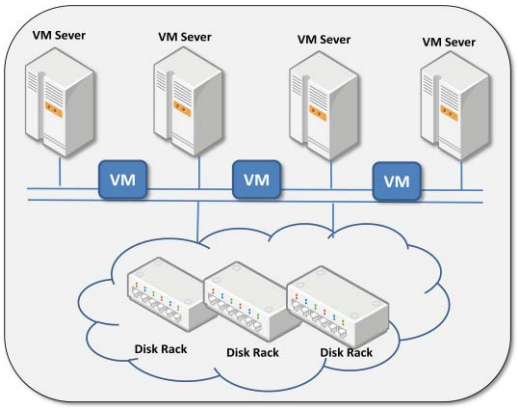

Figure 4. Cloud System Infrastructure

We believe that in order to efficiently support run time resource reallocation and manage the cloud resource, the basic cloud system infrastructure should be set up in a similar way as illustrated in Figure 4. It shows that all the VM servers are using shared storage space where all the VM base images (VM disks) are stored. VMs on each resource server access to their data on networked storage space. VM servers in this case only contribute CPU and memory resources.

B. Live and Static Migrations

With the shared storage system infrastructure, VM migration no longer needs to involve the base image file but only memory data or even just a configuration file. This would largely improve the VM migration speed to a few seconds with the same configuration mentioned earlier. The following describes two types of migration techniques, live and static migrations, which are based on this shared storage system infrastructure.

- *Live migration*: VM live migration transfers the VM working state and memory while they are running from the source server to the target server. The benefit of this type is that there is no intervention of VM operation during the resource reallocation process. Users would not even notice that their VMs are being moving around. Live VM migration still generates network traffic depending on the VM's memory occupation.

- *Static migration*: Static migration is another approach for VM migration, which means that a VM is shut-down and a configuration file is sent from a source server to a target server. The same VM on the target server can be started by using the same configuration file.

It is important to identify the advantages and disadvantages for both migrations in order to determine

the best migration method for resource reallocation. The comparisons between them are shown in TABLE II.

TABLE II. COMPARISONS BETWEEN LIVE AND STATIC MIGRATION

Properties	Live Migration	Static Migration
Transfer Speed	Few seconds or minutes	negligible
Migration between different platforms	No	Yes
Length of downtime	Minimum 60ms in Xen [10]	Always down

Due to the underlying shared storage system infrastructure, the transfer speed is significantly improved for both migrations. The transfer speed of the static migration is an advantage when comparing that of the live migration. This is because time taken for the static migration is negligible. It only needs to transfer a tiny configuration file (normally 1 to 2 kb) across the network, while the live migration takes a few seconds or minutes depending on size of memory and network bandwidth. Transferring the memory would cause an increase in network traffic.

Another advantage of the static migration worth to mention is that it allows VM migration between different hypervisors while the live migration between two different hypervisors cannot be easily implemented.

One advantage of the live migration is that it allows VM operation during migration with an extremely short downtime (minimum 60ms in Xen [10]). However, high frequency VM migration is not recommended as some unexpected and inevitable problems may occur during migrations, such as network failure, server failure etc. .

Based on the comparisons stated above, resource reallocation involves multiple migration operations, using just the live migration is not a good approach. Therefore, we believe our real time resource reallocation algorithm using the static migration is the most efficient and reliable way for resource reallocation. The live migration can only be deployed when no static migration can be executed.

C. VM shut-down Stimulation

As we mentioned earlier in this section, the static migration is more efficient than the live migration for small and medium sized cloud providers. Shutting down a VM when it is not used would be beneficial for better resource utilisation. Actually, according to recent reports, more than 60% of VMs are of idle status during most of time. In order to encourage clients' to shut down their VMs, providers have to give some types of incentives to their clients, such as discounts, cashback or extended use time etc.

Hence, we propose a mathematical model for calculating the incentive.

$$d = a - ae^{-\lambda t} \qquad (2)$$

where a is the maximum incentive that can be given by a provider, t is the duration of a shut-sown VM , d is the calculated incentive obtained by a client, λ is a constant that can control the rate of convergence towards the limit a. The value λ can be adjusted by providers according to their needs. Smaller the value of λ, slower the incentive d converges to the limit a, i.e. taking more time to obtain the maximum incentive, or vice versa.

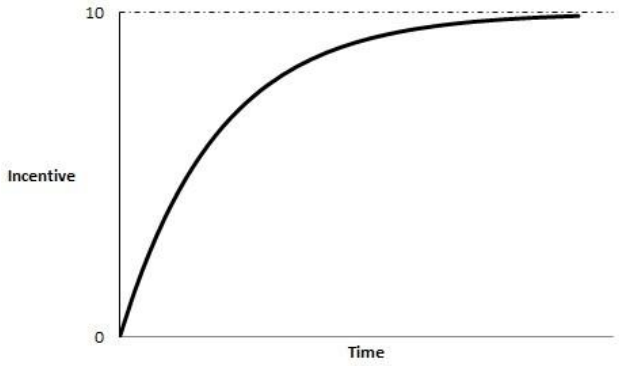

Figure 5. A Mathematical Model for Incentive Calculation

The graph of the equation 2 is presented in Figure 5. It models a relative fast rate of increase in the incentive obtained when the time increases at the beginning. This would encourage clients to shut down any unused VMs. At a certain point approaching to the maximum incentive, the rate decreases as the time increases, this makes clients take longer time to reach the maximum incentive so as to encourage them to keep their VMs shut down.

V. IMPLEMENTATION

A. Implementation for Resource Rescheduling System

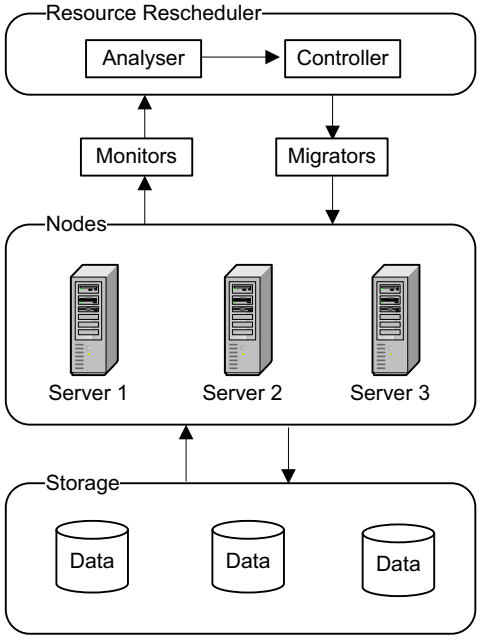

Figure 6. Resource Rescheduling System

The resource rescheduling system we have implemented is based on the shared storage system infrastructure as depicted in Figure 6. Resource severs only contributes their CPU and memory resources. Each resource server has a monitor that is used to continuously track and acquire resource usage metrics, such as memory utilisation level, CPU utilisation level, etc., from the resource server. When a resource reallocation event is triggered (use case one and two in Section III occurs), the monitor instantly pulls real time resource usage metrics out from a resource server and sends them to the analyser via web service. The analyser is the most important module in this system, as it is specifically designed for finding the most suitable migration plan. The real time resource scheduling algorithm is applied in this module. Once the most suitable node for a VM operation (such as creation, migration, resizing, etc.) is found using the algorithm, other VMs that can be moved using static migration are planned to be moved away to other resource servers which have enough free memory space for them to migrate in order to leave enough free memory space in the most suitable node for the VM operation. If no suitable node is found, live migrations of VMs become necessary. Hence, some traditional capacity planning methods can be applied to this situation. Once the analyser creates the migration plan, the controller will dispatch VM migration tasks to specific resource servers according to the plan. A migrator is a VM migration module on a server that is used to execute VM migrating operations received from the controller.

B. *Implemation of VM Migration*

Some popular hypervisors, such as Xen, KVM, and VMware, etc., have fully implemented both live and static migrations in their recent versions. However, an automatic process for cross-platform migration still has not been implemented yet, for example, migration between KVM, Xen, and VMware, etc. Therefore, we have implemented an automatic process for such migrations. Since the images are logically the same for each hypervisor, the only real difference between them is the configuration file that tells the underlying hypervisor what configuration it has available to it. We use Libvirt [24] as a main tool to modify XML configuration files of migrating VMs during the migration process where Libvirt is a toolkit that interacts with the virtualization capabilities implemented on top of different hypervisors. The implementation has been tested in real world deployment for providing Cloud services by several medium sized IDCs.

VI. Conclusion and Future Work

In this paper, we proposed a more efficient resource management solution in order to help cloud providers to improve their cloud resource utilisation. We argued that the existing pre-scheduling algorithm based approaches are not suitable for cloud resource management due to the elasticity feature of cloud computing. Two use cases are given to explain why real time resource reallocation is required. Based on the arguments, a resource reallocation algorithm for bounded resources is provided which is based on VMs' migration. We then analysed the performance issues of current the VM migration technology and proposed a shared storage system infrastructure to tackle the problem as well as a static VM migration approach. At the last, an economic intensive model is presented to encourage cloud users to shut down their VMs to improve the resource reallocation performance.

Since we have integrated the system into Imperial College Cloud (IC Cloud) [25] for campus Cloud provision, the immediate future work is that we are going to use this method to schedule resources among our campus cloud where each department's machines are aggregated to provide shared resource. This campus cloud is very similar with the IDC based Cloud federation when each department can be viewed as an IDC. Thus, we are planning to integrate it to commercially operated cloud infrastructures at the same time. It becomes very interesting to investigate the cross IDC resource migration where the provider-level resource/profit sharing agreement will be taken into account. Another work is to do further algorithm tuning by collecting data from the deployed platform. In addition to that, we will analyse the operational cost improvement of our optimised resource utilisation comparing non-optimised ones. It will be interesting to further study cost-sensitive scheduling algorithms where we may balance the optimal resource schedule with the cost of its realisation.

References

[1] E.C. Amazon, Amazon elastic compute cloud. Retrieved Feb 10 (2009).

[2] S.P.D. Hosting, gogrid Cloud Hosting. Available on the WWW, 2009.

[3] M. Jones, and R. Hayes, Salesforce. com: The Official Guide. (2005).

[4] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, Enforcing performance isolation across virtual machines in Xen, Springer-Verlag New York, Inc., 2006, pp. 342-362.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, Xen and the art of virtualization, ACM, 2003, pp. 164-177.

[6] S. Crosby, and D. Brown, The virtualization reality. Queue 4 (2006) 34-41.

[7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, kvm: the Linux virtual machine monitor, 2007, pp. 225–230.

[8] J. Lo, VMware and CPU Virtualization Technology. World Wide Web electronic publication (2005).

[9] R. Braddock, Creating an Edubuntu virtual machine thorough VMware ESXi. (2009).

[10] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, Live migration of virtual machines, USENIX Association, 2005, pp. 273-286.

[11] M. Nelson, B.H. Lim, and G. Hutchins, Fast transparent migration for virtual machines, 2005, pp. 391–394.

[12] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, Black-box and Gray-box Strategies for Virtual Machine Migration.

[13] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation.

[14] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration.

[15] M. Zhao, and R.J. Figueiredo, Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources.

[16] L. Silverline, https://silverline.librato.com/.

[17] T. Abdelzaher, K. Shin, and N. Bhatti, Performance guarantees for web server end-systems: A control-theoretical approach. Parallel and Distributed Systems, IEEE Transactions on 13 (2002) 80-96.

[18] M.D. De Assunção, A. Di Costanzo, and R. Buyya, Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, ACM, 2009, pp. 141-150.

[19] A. Stage, T. Setzer, and M. Bichler, Automated capacity management and selection of infrastructure-as-a-service providers, IEEE, 2009, pp. 20-23.

[20] J. Allspaw, The art of capacity planning, O'Reilly Media, Inc., 2008.

[21] D.A. Menasce, and V.A.F. Almeida, Capacity Planning for Web Services: metrics, models, and methods, Prentice Hall Upper Saddle River, NJ, 2002.

[22] R. Garg, H. Saran, R.S. Randhawa, and M. Singh, A SLA framework for QoS provisioning and dynamic capacity allocation, Citeseer, 2002, pp. 129-137.

[23] R. Lopes, F. Brasileiro, and P. Maciel, Business-driven capacity planning of a cloud-based it infrastructure for the execution of Web applications, IEEE, 2010, pp. 1-8.

[24] B. Victoria, Creating and Controlling KVM Guests using libvirt, University of Victoria, 2009.

[25] L. Guo, Y. Guo, and X. Tian, IC Cloud: A Design Space for Composable Cloud Computing, IEEE, 2010, pp. 394-401.