Chapter 13
An Infrastructure-as-a-Service Cloud: On-Demand Resource Provisioning

Weijia Song
Peking University, China

Zhen Xiao
Peking University, China

ABSTRACT

Cloud computing allows business customers to elastically scale up and down their resource usage based on needs. This feature eliminates the dilemma of planning IT infrastructures for Cloud users, where under-provisioning compromises service quality while over-provisioning wastes investment as well as electricity. It offers virtually infinite resource. It also made the desirable "pay as you go" accounting model possible. The above touted gains in the Cloud model come from on-demand resource provisioning technology. In this chapter, the authors elaborate on such technologies incorporated in a real IaaS system to exemplify how Cloud elasticity is implemented. It involves the resource provisioning technologies in hypervisor, Virtual Machine (VM) migration scheduler and VM replication. The authors also investigate the load prediction algorithm for its significant impacts on resource allocation.

1. INTRODUCTION

Cloud elasticity refers to the ability of Cloud infrastructure to dynamically make resource provision for Internet applications and services, according to their real time requirements. That feature of Cloud Computing has several appealing implications. It eliminates the dilemma of planning IT infrastructures for Cloud users, where under-provisioning compromises service quality while over-provisioning wastes investment as well as electricity. It offers virtually infinite resource to Cloud users (Armbrust et al., 2009). It has also made the desirable "pay as you go" accounting model possible.

Planning new IT infrastructure for growing demands of Internet applications is complicated. It calls for successful prediction on how appli-

DOI: 10.4018/978-1-4666-2854-0.ch013

cation loadings would change in the future and is particularly hard for Start-Ups since market response is not clear in advance. When it knocks against flash-crowd, self-maintained servers may fail to satisfy the need of surging requests. On the contrary, over provisioning caused by optimistic prediction leaves the server under-utilized, and consequently causes waste in energy and excess investment in fixed asset. In Cloud environment, however, application maintainers need not worry about such problems since the Cloud resource allocation automatically scales up and down on changing load, and the users are billed accordingly.

Sometimes, particularly in data mining applications, a user may require a large number of servers for a short period. It is hard to satisfy such requirement if it were not for Cloud computing. A successful example is, "The Washington Post uses Amazon EC2 to turn Hillary Clinton's White House schedule—17,481 non-searchable PDF pages—into a searchable database within 24 hours." ("AWS Case Study: Washington Post," n.d.). In colleges, researchers may have similar requirements when processing huge amount of experiment data.

There are different approaches to Cloud elasticity depending on how the Cloud infrastructure is constructed and what types of applications running over it. In the next sections, we are going to introduce some popular technologies adopted nowadays Cloud Services. Then we start from basic components of a Cloud infrastructure to explain our own work that handling Cloud elasticity in a real IaaS service. In the end, we will point out, in our perspective, the trend of Cloud elastic technologies.

2. BACKGROUND

Traditionally, Cloud services are categorized into Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides virtual machines to Cloud users. IaaS users are responsible for application development, deployment and management. PaaS take the burden of application management by providing development tools and deployment platform. SaaS model is actually "old wine in new bottles" for conventional Internet applications.

In an IaaS system, virtual machines are generally overcommitted to physical servers to maximize profit from hardware investment and cut down power budget. Cloud elasticity in that environment addresses the challenge of resource provisioning for dynamic load of virtual machines. For example, if physical server cannot satisfy the resource requirements of its virtual machines, some of them are going to be migrated to other servers so that application performance is assured.

It is hard for application developers to predict the user load. In PaaS systems, user applications are managed by Cloud infrastructure to relive developers of the difficult of deployment, so that they can concentrate on application function. Generally, the applications deployed in PaaS are developed by designated program language, development tool and libraries and encapsulated in managed execution engines. An execution engine is a sandbox allocated with a share of CPU resource. Execution engines have uniform management interface for life cycle control and performance monitoring. Elasticity mechanism dynamically adjusts the number of execution engines belonging to an application to suit its load.

The situation in SaaS is similar to that in PaaS. A SaaS service could be built upon a PaaS service to indirectly utilize its elasticity mechanism. Some large SaaS services choose to implement dedicated elasticity mechanism for application specific optimizations (Chen et al., 2008) (Chase et al., 2001). Here we just talk about stateless computing resource. The discussion of data storage technologies like Google File System (Ghemawat, Gobioff, and Leung, 2003) and Big Table (Chang et al., 2008) belongs to another dedicated field out of scope of this chapter.

This chapter focuses on the Cloud elasticity technologies adopted in IaaS systems. We are going to illustrate the dynamic resource provisioning mechanisms and policies in the PKU Cloud, a real IaaS system that supporting research in Computer Department of Peking University.

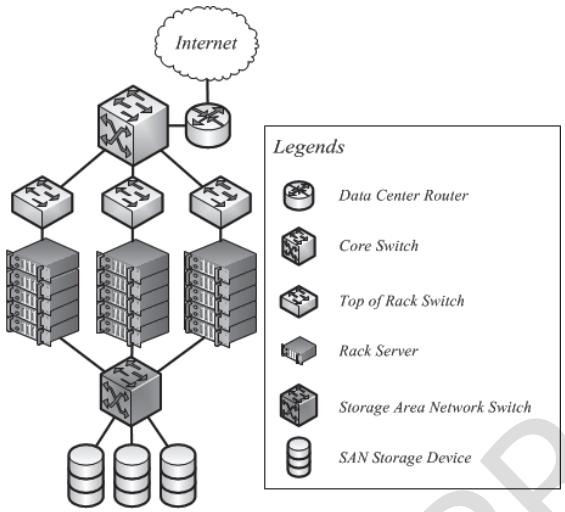
The PKU Cloud adopts typical structure of Cloud infrastructure depicted in Figure 1. Its fundamental part is a virtualized data center hosting several tens of blade servers. The Xen hypervisor virtualize each physical server into several virtual machines. Based on the hypervisors and the virtual machines, the PKU Cloud incorporates peripheral services such as storage, security, and management to provide an integrated solution. Although innovative network structures like FatTree (Al-Fares et al., 2008) and VL2(Greenberg et al., 2009) are emerging, conventional tree structure is the most widely adopted one for data center network. The conventional tree structured data center network is composed of a router, several core switches, and many top of rack switches. In practice, a data center generally has more than one router and core switches for fail-safe or performance purpose. Data center network connect servers to each other and to the outside Internet. Storage Area Network (SAN) connects the servers with back end centralized storage devices.

The PKU Cloud implements elasticity at three levels. Consider a scenario where the load of an application keeps rising. Virtual machines containing the application components begin asking for more CPU and memory from the hypervisor layer. Idle resources of the physical servers are assembled and put into use by the hypervisors. Most of the time, virtual machines are not fully loaded. Physical servers usually have some resource reservation for transient load pulse, even though they are sometimes over-committed. However, the application load keeps going up. Resource reservation in some physical servers is consequently exhausted. In this case, hypervisors has to balance resource allocation among the virtual machines sharing the same physical server to avoid unacceptable degradation of Quality of Service (QoS). In the next section, we are going to discuss in details the resource scheduling technologies at the hypervisor level.

Live migration (Clark et al., 2005) allows a virtual machine to be migrated from one physical server to another, without interrupting the application running in that virtual machine. Ideally, layout of virtual machines on physical servers can be dynamically adjusted with live migration to a state that Service Level Agreement (SLA) is always satisfied as long as there are idle resources available in any physical server in the system. This feature is appealing in case of flash crowd. When application load may get high enough that no matter how the hypervisor allocates the resources to virtual machines, there are always some applications that cannot get enough resource to achieve acceptable performance regarding to SLA. Live migration, however, incurs network overhead because it involves transferring the memory image along with other states of a virtual machine from one server to another. When network resource is busy, using live migration may make the situation worse. In addition, migration may last for an uncertain period of time depending on network traffic; therefore it is unwise to incorporate migration for a transient overload. Due to the limitations of live migration, the aforementioned ideal state is hard to achieve. In the section "elasticity with live migration", we are going to elaborate on how the PKU Cloud uses live migration for resource provisioning.

Performance scalability with service replication has been adopted in web applications (Chase et al., 2001) long before. Stateless tiers of an application, like web front end and thinking logic are replicated in case of flash crowd. An application level switch is responsible for redirecting incoming requests to the right place. In virtualized environment, the mechanism stays the same except that application tiers are encapsulated in virtual machines. But the applications nowadays are growing more and more complicated. On one

Figure 1. Architecture of a virtualized data center



hand, they occupied large amount of memory even in idle. On the other hand, the process of starting or shutting down an application typically lasts too long to react to flash crowd in time. In the section "elasticity in Internet applications", we address those issues with the PKU Cloud approaches to Cloud elasticity at the application level.

The nature of resource scheduling is to provision resources reasonably to applications in the future. If the future load could be known beforehand, an offline algorithm can calculate an ideal resource scheduling solution so that all application requirements are satisfied if possible and that the performance of the Cloud infrastructure is maximized. However, it is usually impossible to know the load in advance. In this case, prediction algorithm, making estimation of a random variable according to its history and other factors, helps understand the trend of how the load would change overtime. Prediction is a well-studied topic in fields such as stock market prediction and weather forecasting. In the field of Cloud computing, many research works and systems have already incorporated that technology. In section "load prediction", we are going to introduce two kinds of load prediction algorithm.

3 RESOURCE SCHEDULING IN HYPERVISOR

Generally, physical servers are overcommitted by virtual machines. For example, a CPU intensive and a memory intensive virtual machine are put together to share a server to improve utilization. Combining virtual machines with alternate peak time takes effect likewise. When load changes, the hypervisor is responsible for adjusting resource allocation among the virtual machines. Three types of resources are usually considered: CPU, memory, and I/O resource. We are going to look into the each kind of resource respectively to understand how elasticity is realized at the hypervisor level. We assume you have basic knowledge of virtualization in this chapter or you can refer to the design of Xen hypervisor (Barham et al., 2003).

3.1 Virtual CPU Scheduler

In Operating System, process scheduling is a well-explored area. OS processes are multiplexed on CPU cores in an elastic time-division manner. Process scheduler incorporates priority and time slice for fair allocation of CPU time as well as

maximum utilization. A process can use more CPU time than its share as long as idle CPU times are available. A greedy process, however, is constrained to its fair share so that other processes are not affected negatively. In a virtualized environment, the mapping of OS processes to physical CPU cores is indirect. Firstly, a process is scheduled onto a Virtual CPU (VCPU) of its containing virtual machine by the process scheduler in the guest operating system. When the VCPU acquires CPU time slice, the process is executed. Scheduling VCPUs onto physical CPU cores is performed by VCPU scheduler in the hypervisor. VCPU scheduler and process scheduler have common objectives such as fairness and performance.

All physical servers in the PKU Cloud are configured to use Credit ("Credit Based Scheduler", 2007), the default VCPU scheduler of Xen hypervisor. In Credit, each virtual machine is associated with two properties, a weight and a cap. The weight tells the proportion of CPU time a virtual machine should have relative to each other; while cap tells the upper limit. We use a two-tuple <weight, cap> to represent a virtual machine with its weight and cap. Consider two virtual machines, A <100, 80%> and B <100, 60%>, which share a server. If the CPU load in both of them exceeds 50% of the capacity of the server, they get 50% each. Otherwise, unused CPU time of one virtual machine can be utilized by the other. For example, A can get up to 80% if B uses only 10%, leaving the other 10% CPU time unused. On the contrary, B can get up to 60%.

3.2 Memory Allocation among Virtual Machines

Memory resource is quite different from CPU resource for its usage is almost independent of application load. A sophisticated application may require a large amount of memory even if the load is low (Karve et al., 2006). The hypervisor needs special measurement for memory requirement. This is important because allocating and reclaiming memory among virtual machines involves modifying the page table and invalidating the Translation Lookaside Buffer (TLB) which incurs overhead. Sometimes, for security reason, a memory page should be flushed (i.e., zeroed out) before being allocated. Otherwise, data of the virtual machine that page belonged to is leaked to the one getting it.

The PKU Cloud adopts Ballooning (Waldspurger, 2002) technology to share memory between Virtual Machines on the same physical server. When memory is scarce, for example, a new virtual machine being created, unused memory can be squeezed out of other virtual machines and reallocated to the one where it is required. That is achieved by a "balloon" process devised in each guest OS. It is under the control of the hypervisor. When the hypervisor squeezes memory from a virtual machine, it sends a "deflate" request to the balloon process in that virtual machine. Then, the ballooning process requests specified amount of memory from OS. On a successful operation, the balloon process pins the memory it has acquired (so that corresponding pages are never swapped out), picks the corresponding page frames off the page table and hands the frames to the hypervisor. Inversely, when the hypervisor returns memory, the balloon process executes an "inflate" operation. It hooks those page frames to the page table again, unpins the memory and frees it back to OS. The memory is like air that flows from one balloon to another. For this reason, the technology is called "ballooning". Although ballooning provides a mechanism of sharing memory, it is a hard problem to find out how much memory a virtual machine requires. It is similar to the classical problem of determining the size of the process working set. The current solution in the PKU Cloud is Self-ballooning (Magenheimer, 2008). It is a daemon residing in Linux guest OS, periodically reading the value of "Commited_AS" item in the "/proc/meminfo" system file as a coarse estimation on

future memory usage. According to the estimation, it actively adjusts the memory usage with the balloon driver. That approach can be improved, since we found by experiment that the estimation is too simple to exclude aged page cache. VMware ESX server adopts a random page sampling technique (Waldspurger, 2002) to measure the working set of a virtual machine: during each measurement period, a random subset of pages in the VM's (pseudo) physical memory are invalidated by the hypervisor so that a subsequent access will result in a trap. This allows the hypervisor to collect statistics on how much memory is being actively used by the guest OS. The default sampling rate in ESX is 100 random pages every 30 second. Experiment result in that work showed a smooth and tight track of the actual memory usage. In another work (Wood et al., 2007), swap activities are used as the signal of memory shortage. Once abrupt increase of swap activities of a virtual machine is detected, memory is inflated by a step of 32 megabyte. In practice, that method is so lazy that the application performance is compromised.

Transcendent Memory (TMEM) (Magenheimer, 2009) takes another approach. As we know, page cache used to accelerate the file access tends to eat up memory. While some virtual machine is suffering from the memory shortage, others may occupy a lot in the page cache for infrequently used files. To avoid that unfairness, TMEM maintains a shared memory pool in the hypervisor for centralized management of page cache. The pool is virtualized so that each virtual machine can have one or more virtual pools for convenience. A virtual pool can be ephemeral or persistent. Data put into an ephemeral pool may be forgotten because of memory shortage. But the availability of data put into a persistent pool is guaranteed. To use TMEM, the page cache and swap implementation in guest OS are extended with two operations, precache and preswap, respectively. Before a clean page in the page cache is reclaimed, its data is written to an ephemeral pool. When the page is read next time, the OS first tries to read it from the ephemeral pool. If the read operation succeeds, which means the data has fortunately survived, a time-consuming disk I/O operation is saved. Since the precache operation works like page cache without using memory dedicated to a virtual machine, a virtual machine only needs to occupy a small amount of memory for OS kernel and application code. Sometimes, a virtual machine needs to swap out some pages. Before the data of a page is eventually swapped out to the disk, the OS tries to put it to a persistent memory pool. If the operation succeeds, data of that page is retrieved from the pool the next time when it is accessed. Again, a disk I/O operation is saved.

TMEM evades the difficulty of working set measurement. But it is more complicated to implement than ballooning. To the time of this writing, TMEM has been implemented in the Xen hypervisor. We plan to update Xen hypervisor in PKU Cloud to support the TMEM.

3.3 Scheduling I/O Resources

Network and disk are two essential I/O resources that are closely related to the application performance. I/O virtualization involves the sharing of a set of network interface cards (NIC) and the disk of a physical server among virtual machines running over it. Hypervisor is responsible for creation of virtual devices exposed to the guest OS and the multiplexing of real I/O devices. In order to understand how I/O resource is scheduled in a virtualized environment, we look into a typical implementation, the split driver model in the Xen hypervisor.

There is a privileged virtual machine, called domain 0, on the Xen hypervisor. It contains all the drivers required to manipulate real devices like gigabyte NICs or IDE disks. Common virtual machines, called domain U, do not have such drivers. They indirectly access the real devices by a mechanism called the split driver model. Each virtual device corresponds to a split driver

composed of a front-end and a back-end driver in the domain U and the domain 0, respectively. Both ends are tied together by Xen hypervisor's communication mechanism including shared memory and event channel. The front-end driver accepts requests from the OS in domain U, while the back-end driver handed these requests to a real driver. On receiving data, the back-end driver will notify the front-end driver about new arrivals in the shared memory. I/O resource Schedulers stand between the back-end driver and the real driver. They decide when and in which order the requests are handled.

Although the PKU Cloud follow the traditional I/O scheme and works fine so far, but the intricate characteristics of I/O systems make fairness and isolation quite difficult. Here we name a few to show the tip of the iceberg.

Accessing disk data involves slow mechanical movements. It is hard to determine how long such an operation will take since the associated head position is unknown beforehand. Not to mention Redundant Array of Independent Disks (RAID) or hybrid storage systems composed of Solid-State Drive (SSD) and traditional hard disk. Thereby traditional disk I/O scheduling algorithms like Completely Fair Queuing (CFQ) (Love, 2004) pursuing fairness in request numbers cannot achieve good fairness and isolation. Gulati pointed out that there is a tradeoff between performance and fairness (Gulati et al. 2007). VIOS (Seelam and Teller, 2007) improves by fairly allocating disk time to virtual machines. AutoControl (Padala et al., 2009) extends such fairness to application level.

A virtual disk may be an image file in the file system, a logic unit in the iSCSI storage or a disk partition on a local hard disk. But the I/O scheduler in the guest OS is unaware of that detail. It is generally optimized for a single exclusive hard disk incorporating technics like anticipatory reading and reordering requests for a shorter head movement. At the hypervisor level, however, the I/O requests from different guest OSes are scheduled again. Mutual interference of the two schedulers may result in an awkward situation. In the XenServer, a commercial version of Xen hypervisor, the noop scheduler using simple FIFO algorithm, is adopted by default at the hypervisor level. Another possibility is to leave that task to the storage system.

Scheduling network resource seems simpler than scheduling disk I/O if only bandwidth of NICs is taken into account. Sometimes, however, the links among switches and routers in data center network may become critical resources due to a hot application or malware. Fairness and isolation at this level is quite difficult. Seawall (Shieh et al., 2011) introduces a sophisticated approach to this problem. It is an end-to-end solution without a central coordinator. Only modification to the hypervisor software is required.

I/O virtualization is a fast changing field. New technologies keep emerging. Our discussion is however limited due to the lack of space.

4 CLOUD ELASTICITY WITH LIVE MIGRATION

Resource scheduling at the hypervisor level realizes limited elasticity owing to the fixed capacity of a physical server. Migration of virtual machines breaks up the limitation by utilizing resources from other servers. Figure 2 sketches how it works. Consider a data center with three physical servers: A, B, and C. At the beginning, as shown in the upper left part of Figure 2, server A and B are running five virtual machines each. They are reasonably loaded (tagged with "WARM"). Server C is standing by to save electricity. When workload grows, server A cannot afford the aggregate resource requirements of its five virtual machines (tagged with "HOT"). We say that server A is a hotspot. A virtual machine scheduler (not shown in Figure 2) detects the hotspot and thereby initiates a migration schedule that migrating two virtual machines from server A to C. After the migration finishes, as shown in the lower right

part of Figure 2, the hotspot is resolved. Resource demands of applications are consequently satisfied. Assume that the workload begins to shrink now. Resource utilization of server A and C drops dramatically. The utilization of server A and C (tagged with "COLD") is too low to be power efficiently. The virtual machine scheduler therefore initiates another migration schedule that dynamically consolidates virtual machines on under-utilized servers together. The released server C again enters low power state. Now, the servers return to their original states.

Some important details are left out in the above scenario for abstraction. As we mentioned in the BACKGROUND section, live migration incurs network overhead and takes some time. Only persistent change in resource demands deserves migration. The virtual machine scheduler (VM scheduler) incorporates load prediction technics, which we will describe later, to differentiate stable workload change from transient fluctuation. Depending on the estimation of future workload, it needs to decide which virtual machine to migrate away and to where. It will be demonstrated later that solving the problem is an NP-hard problem. Only heuristics algorithms are pragmatic. The rest of the section will first illustrate the widely adopted architecture for VM schedulers and then focus on the migration policies. For convenience, the acronym VM and PM are used in the following text to denote a virtual machine and a physical server/machine respectively.

The VM scheduler of the PKU Cloud is shown in Figure 3. This centralized architecture is also adopted in other VM schedulers such as Sandipiper (Wood et al., 2007), Harmony (Singh, Korupolu, and Mohapatra, 2008) and Usher (McNett et al., 2007). Each PM runs a hypervisor with a Node Manager. The node manager collects from the hypervisor the real-time usage statistics of resources for each virtual machine on that PM. The statistics collected at each PM are forwarded to the VM scheduler. The VM Scheduler is invoked periodically and receives from the Node Manager the resource demand history of VMs, the capacity and the load history of PMs, and the current layout of VMs on PMs.

The VM scheduler has several components. The Load Predictor predicts the future resource demands of VMs and the future load of PMs based on past statistics. The load of a PM is computed by aggregating the resource usage of its VMs. The Node Manager at each node first attempts to satisfy the new demands locally by the resource allocation mechanism in the hypervisor. The Scheduling Algorithm module detects if the resource utilization of any PM is above the hot threshold (i.e., a hot spot). Then it decides on

Figure 2. Elasticity with live migration

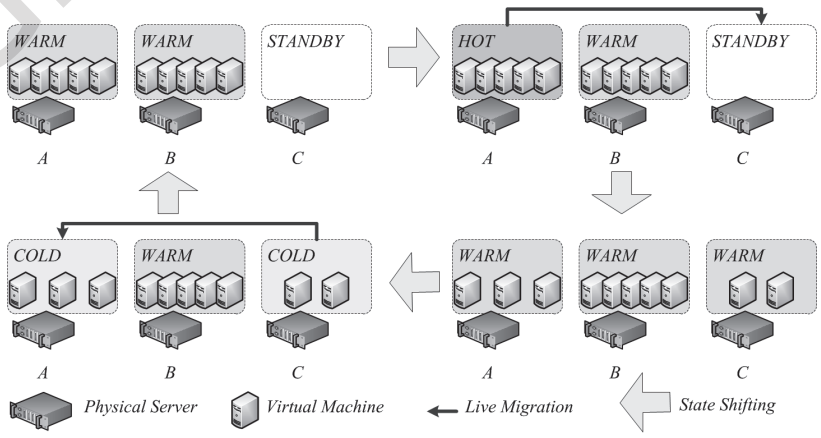
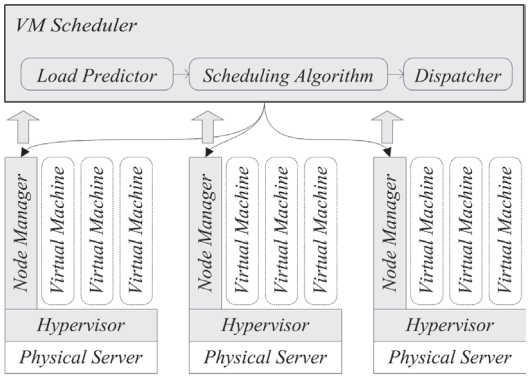


Figure 3. Architecture of a VM Scheduler



which VMs running on hot spots should be migrated away to reduce their load. It also identifies under-utilized PMs and decides whether they should be released by migrating their VMs away so that they can be turned off to save energy. The Scheduling Algorithm finally compiles a migration list of VMs and passes it to the Dispatcher, who in turn contacts the Node Manager for the execution of the planned migration.

Many algorithms can be incorporated in this framework. One category borrows solutions from well-studied theoretic models (Bobroff, Kochut, and Beaty, 2007) (Xu and Li, 2011). The other use pragmatic heuristics which model expected scheduling objectives (Wood et al., 2007) (Singh, Korupolu, and Mohapatra, 2008). The PKU Cloud incorporated two algorithms. The online bin-packing algorithm belongs to the first category, while the Skewness algorithm belongs to the latter one.

4.1 Online Bin-Packing Algorithm

The classical bin packing problem consists of packing a series of items with sizes in the interval (0, 1] into a minimum number of bins with capacity one. We can model VM scheduling as the bin packing problem where each PM is a bin and each VM is an item to be packed. Resource provision is implicitly assured by the rule that the size of items (resource requirement) is less than the bin size (resource allocation), while over provisioning is avoid by the objective of using a minimum number of bins. We assume that all PMs are homogeneous with unit capacity. We normalize the resource demands of VMs to be a fraction of that capacity. For example, if a VM requires 20% of the physical memory of the underlying PM, then it corresponds to an item with size 0.2. If other resource types such as CPU and I/O are considered, the size of an item can be represented by a vector whose elements are normalized demands of the VM corresponding to the resource types. That variation is called vector bin-packing.

The bin-packing problem is well-known to be NP-hard. Although it has been studied extensively in the literature, pure theoretic solutions do not work well in data center environments. Offline algorithms can achieve a performance very close to the optimal algorithm, but they assume the entire sequence of items to be packed is known in advance (Garey and Johnson, 1985). More intricately, in data center environment, the size of an item is changing. Rerunning an offline algorithm in each round of scheduling, however, incurs too many migrations to be practical. Online algorithms that pack incoming items incrementally make no attempt to minimize the movements of already packed items because the overhead of migration is hard to model in the framework. When applied to VM scheduling, they need modifications to constrain the frequency of migration.

We have designed a practical online bin-packing algorithm (Xiao et al., 2010) that applied to an IaaS Cloud service which supports over 200 people in a lab of Peking University. We get this core algorithm by extending an existing one-dimensional online bin-packing algorithm (Gambosi, Postiglione, and Talamo, 2000) with the approximation ratio as low as 3/2, which represents a quite good performance for an online algorithm. In the original algorithm, items are categorized into four types: Tiny (T), Small (S), Large (L), Big (B) items. Item size of the four types falls in

intervals (0, 1/3], (1/3, 1/2], (1/2, 2/3], (2/3, 1], respectively. There are seven combinations of items in a bin:

- A B-bin has only one B-item.
- An L-bin has only one L item.
- An LT-bin has only one L-item and a certain number of T-items.
- An S-bin has only one S-item.
- An SS-bin has only two S-items.
- An LS-bin has only one S-item and one L-item.
- A T-bin has only a certain number of T-items. It is called unfilled if the available space is no less than 1/3. Otherwise, it is called filled.

On arrival of a new item, an insert operation is executed that keeps the following three rules for bin usage:

- At any time, only six types of bins: B-bin, L-bin, LT-bin, S-bin, SS-bin, T-bin are allowed in the system.
- At any time, if there exists a T-bin, then there is no L-bin, and the available space in any LT-bin is less than 1/3.
- At any time, there are at most one S-bin and at most one unfilled T-bin.

It can be proved that based on the above rules, the approximation ratio of this algorithm is bounded by 3/2. The proof is skipped due to lack of space.

However, the original algorithm cannot handle size changing of an item. We extend it by adding a change operation to handle the situation where item type changes and the above rules are violated. It has been proved that a change operation invokes no more than 7 movements (migrations) and an insert operation no more than 3. This property effectively constrains the number of migrations.

We also extend it to multi-dimensional by breaking down items according to their largest dimensions. The approximation ratio is bounded by 3/2*d, where d is the number of dimension under consideration. The worst case is when each item has a different dominating dimension. For example, when d = 3, the items (1, 0, 0), (0, 1, 0), and (0, 0, 1) can be packed into a single bin under the optimal algorithm, but need three bins in this one. This ratio is rather unimpressive at a first glance. But there are not that many dominating dimensions in practice. Most practical systems consider only one or two types of resources (e.g., CPU and memory) in their allocation decision.

We used several optimizations when applying this algorithm to a real environment. The size of each bin is intentionally shrunk a little compared to the real capacity of the PM. The reserved capacity helps avoid SLA violation in face of transient load fluctuation. Though we assume that PMs are homogeneous, a practical data center generally contains different types of servers. This problem can be solved by grouping identical servers together and running a VM scheduler for each group. Sometimes the whole system under the management of the VM scheduler may be overloaded. In contrast to the assumption of infinite bins, the reality is, bins are used up. To solve this problem, the capacity of bins can be magnified by a certain percentage until a solution is found. That technic makes sure that all PMs are evenly overloaded to avoid any application is unfairly treated.

4.2 The Skewness Algorithm

Bin-packing based algorithms are aggressive in packing VMs. Therefore load change can easily incur migrations. In other words, it trades stability and performance for using a fewer number of active PMs. In practice, however, the loss of QoS is much severer than the waste of electricity to a Cloud provider. The pragmatic algorithms are more conservative. They prefer performance to green computing. Many of them even do not perform green computing (Wood et al., 2007) (Singh, Korupolu, and Mohapatra, 2008).

In favor of performance and stability, we designed a pragmatic algorithm, skewness (Xiao, Song, and Chen, 2011), into the aforementioned Cloud system. It is inspired by the fact that if a PM runs too many memory-intensive VMs with light CPU load, much CPU resources will be wasted because it does not have enough memory for an extra VM. We introduce the concept of skewness to qualify the unevenness in the utilization of multiple resources on a server. Let n be the number of resources and ri be the utilization of the i-th resource. The resource skewness of a server p is defined as

$$skewness(p) = \sqrt{\sum_{i=1}^{n} (\frac{r_i}{\bar{r}} - 1)^2},$$

where $\bar{r}$ is the average utilization of all resources for server p. In practice, not all types of resources are performance critical and hence only bottleneck resources are considered in the above calculation.

We use several adjustable thresholds that control tradeoff between performance and green computing. The "hot threshold" defines the acceptable upper limit of resource utilization. We define a server as a hot spot if the utilization of any of its resources is above the hot threshold. We define the temperature of a hot spot p as the square sum of its resource utilization beyond the hot threshold:

$$temperature(p) = \sum_{r \in R} \left(r - r_t\right)^2,$$

where R is the set of overloaded resources in server p and rt is the hot threshold for resource r. (Note that only overloaded resources are considered in the calculation.) The temperature of a hot spot reflects its degree of overload. If a server is not a hot spot, its temperature is zero. The "cold threshold" denotes the acceptable lower limit of resource utilization. A server whose utilization of all resources is under the cold threshold is defined as a cold spot. The "green computing" threshold defines the utilization level of all active PMs, under which the system is considered power-inefficient therefore green computing operations get involved. Finally, the "warm threshold" defines the ideal level of resource utilization that is sufficiently high to justify having the server running but not so high as to risk becoming a hot spot in the face of temporary fluctuation of application resource demands.

For each scheduling round, the skewness takes two steps, hot spot mitigation and green computing, to calculate a migration list. In hot spot mitigation, we try to solve all hot spots in descending order of temperature. For each hot spot, we try to migrate away the VM that can reduce the server's temperature the most. In those servers that can accommodate the VM without becoming a hot spot, we choose a server with most skewness reduction by accepting this VM as the migration destination. This does not necessarily eliminate the hot spot, but at least reduces its temperature. Hot spot mitigation step is finished after all hot spot are processed. If the overall resource utilization of active servers is lower than the green computing threshold, a green computing step is invoked. In the green computing step, we try to solve cold spots in ascending order of the memory utilization, which representing the efforts taken to solve a cold spot. To resolve a cold spot, all of its VMs need to be migrated away. The destination of a VM is decided in a way similar to that in the hot spot mitigation, but its resource utilization should be below the warm threshold after accepting the VM. We also restrict the number of cold spots that can be eliminated in each run of the algorithm to be no more than a certain percentage, for example 5%, of active servers in the system. Those arrangements are to avoid over consolidation that may incur hot spots later. The movements generated in both steps above are then consolidated so that each VM is moved at most once to its final destination. For example, hot spot mitigation may dictate a VM to move from PM A to PM B, while

green computing dictates it to move from PM B to PM C. In the actual execution, the VM is moved from A to C directly.

With lower hot spot threshold, the skewness algorithm reacts earlier to resource shortage by provisioning more resources. Lower cold threshold excludes more servers with relatively low load from being recycled. Lower green computing threshold postpone the green computing operation until the overall load decreases more. Both effects make the skewness algorithm more conservative when performing green computing. It is up to the Cloud provider who decides its tradeoff between performance and green computing. Generally speaking, we recommend low thresholds if applications with unstable workload dominate in the system, because more thrashing workload calls for more conservative resource reservation to absorb transient fluctuation hence SLA is assured.

4.3 Performance of the Two Algorithms

We evaluated both algorithms by simulation to understand their performance. We collected load traces from a wide range of real applications, including Web InfoMall, one of the largest web archive in China, RealCourse, a large scale online learning system that spread over 13 major cities, and Amazing Store, a large P2P storage system. We also collected traces from a DNS server and a mail server for Peking University. Traces are segmented in a per-day granularity. We use random sampling and linear combination to generate workloads at required scales.

Both algorithms are evaluated in four aspects: effect of load balancing, effect of green computing, stability, and decision time. We use the number of hot spots to quantify the effect of load balancing. Small number of hot spots represents good effect of load balancing. We model the effect of green computing as the number of active physical machines (APM) throughout the evaluation. Less active physical machines mean more efficient usage of power. The stability of an algorithm is represented by the number of live migration. We define decision time as time required for an algorithm to calculate a scheduling plan in each scheduling round. In practice, decision time needs to be short enough that the system load distribution doesn't change significantly when a scheduling plan is calculated.

The numbers in Table 1 are average numbers of hot spots in each round just before scheduling. With bin-packing algorithm, the number of hot spots is almost five times as many as that with Skewness algorithm. The bin-packing algorithm tends to maximize the utilization of resource, therefore it generally pack the APMs tighter than Skewness does. Consequently, with bin-packing algorithm, a hot spot is easy to be triggered due to load fluctuation. Unlike bin-packing, the Skewness maintains each active physical machine reasonably loaded so that transient load fluctuation could be absorbed without cause hot spots.The Skewness algorithm actually trade power consumption for performance. As show in Figure 4, we can see that, Skewness uses 10 - 20% more physical servers than bin-packing algorithm.

Table 2 shows the average numbers of migration in each round issued by both algorithms for the same workload. The migration is much more frequent with bin-packing than with Skewness because bin-packing is more sensitive to load

Table 1. Average number of hot spots

Scale in number of VMs	200	400	600	800	1000	1200	1400
Bin-packing	0.60	1.44	2.39	3.41	4.17	4.91	6.37
Skewness	0.15	0.35	0.48	0.57	0.76	0.98	1.10

variance. In addition, the bin-packing algorithm carefully rules the layout of VMs over PMs; therefore it triggers more movement for adjustment than the Skewness algorithm.

Previous analysis (Xiao et al, 2010) (Xiao, Song, and Chen, 2011) reveals that the time complexity of bin-packing and Skewness algorithm are O(log(n)) and O(n2) respectively, where n is the number of VMs. Experimental results shown in Table 3 perfectly conform to the analysis. The decision time of Skewness is more than one second at a scale of 1400 VMs. In a system with 10,000 VMs it is expected to grow to one minute. The decision time and migration time together would exceed scheduling interval. In practice, however, that is not problematic because the scheduling interval is much longer than one minute for high stability. In addition, Servers in big data centers are generally grouped into smaller resource pool so that the scale is manageable.

5 CLOUD ELASTICITY IN INTERNET APPLICATIONS

Sometimes even if each virtual machine of an application occupied a dedicated physical server, the application load still asks for more resources. In such a situation, resource provisioning with local resource adjustment or migration do not work anymore because those mechanisms are unaware of what applications are running in virtual machines. Many commercial platforms, e.g. Google App Engine, are capable of automatically replicating application instance for surging load. This section focused on the solutions adopted by the PKU Cloud for Web applications.

Figure 5 depicts the common architecture of a web application. The front end switch is typically a Layer 7 switch which parses application level information in Web requests and forwards them to the corresponding applications. The switch sometimes runs in a redundant pair for fault tolerance. In the PKU Cloud, the L7 Switch is running on a dedicated physical server, and application components are encapsulated in virtual machines. It is important that the application components are stateless so that they can be replicated safely. The elasticity of storage system belongs to another research domain out of the scope of this chapter. This section focused on the resource provisioning problem for the application tier.

Generally, a resource scheduler is responsible for the resource allocation. It monitors the load of each application as well as the resource utilization statistics of physical servers. Based on the data it collects and layout of the applications over physical servers, it calculates a new resource al-

Figure 4. Number of active PMs

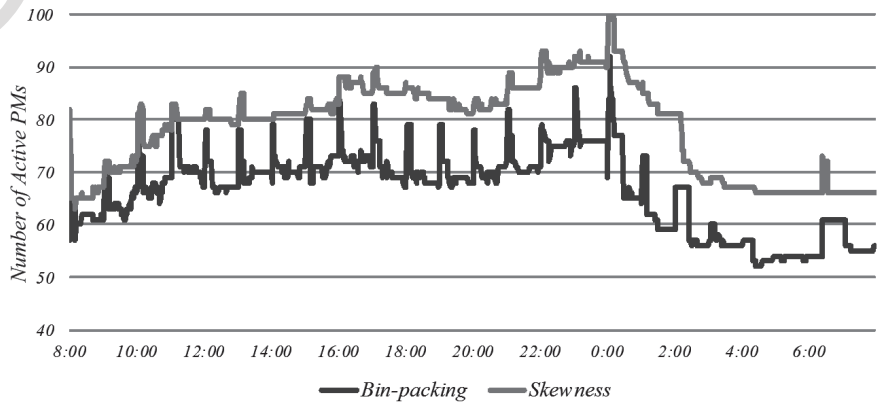
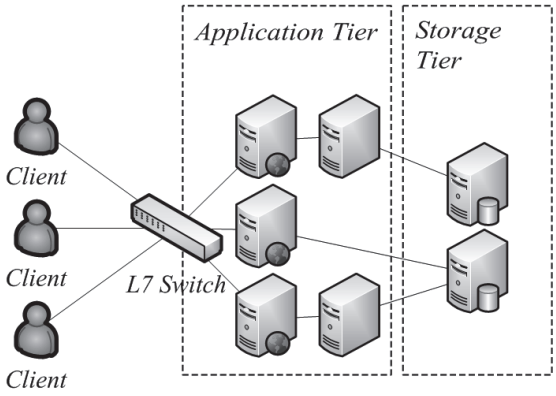


Figure 5. Architecture of a Web application



location strategy that is better than the current one. The criteria for a "good" strategy depend on the particular system. The strategy, however, are commonly composed of two parts: the layout of applications' replica on physical servers and the request rates each replica accepts. A minor adjustment involves improving the dispatch policy of L7 switch. A major adjustment asks for starting/stopping a replica or even starting/stopping a physical server.

5.1 Resource Allocation in MUSE

In MUSE (Chase et al., 2001), all applications are replicated on each server. Therefore all servers in the data center form a unified resource pool. Allocating resource to an application means increasing the number of requests processed, while reclaiming resource from an application means reducing the number of requests processed. The convenience of such settings is that once an adjustment is enforced, it takes effect instantly.

The MUSE system allows each application to bid for its requests, for example, one cent for each request below 1,000 per minute and half a cent for each request above 1,000 per minute. Generally, the application would bid lower when its throughput is higher. It models the throughput as a linear function of CPU utilization, whose parameters are calculated from application performance history. Hence the energy cost of processing a request can be estimated on CPU utilization. By subtract the energy cost from the price an application can afford, the resource scheduler can get the profit by processing each request. The resource scheduler is invoked periodically or by some predefined states such as the occurrence of hot spots. In each round the resources allocations are adjusted in four steps to maximize the total profit. First, the resource with negative return is reclaimed. Then the idle resources, as long as available, are allocated to profitable applications. For each overloaded server, resource allocated to the least profitable applications is reclaimed to bring it back to a normal state. Finally, if there exists any application x whose current bid is higher than application y, then the resource occupied by application y should be reallocated to application x until equilibrium is reached.

This system was designed more than ten years ago when the applications are relatively simple. They do not need much memory so that a server can run a replica of each application. Today, the applications have become much more sophisticated. An application can easily occupy several Gigabytes of memory. Moreover, starting and stopping an application takes a long time. Therefore it is not applicable for the present Cloud environment.

5.2 Starting and Stopping Web Application

Some research works (Karve et al., 2006) (Tang et al., 2007) address resource allocation for sophisticated web applications. They adopt stopping a web application and then starting it on another server as the approach to change the placement of an application. To avoid too much overhead, they manage to minimize the usage of placement. Particularly, the allocation algorithm given by Tang et al. (2007) use network flow programming to maximize the performance of the current placement of web applications. Therefore the placement operation is postponed until the placement cannot satisfy the load anymore.

Table 2. Average number of migration

Scale in number of VMs	200	400	600	800	1000	1200	1400
Bin-packing	4.91	11.66	17.78	23.92	29.95	36.67	43.21
Skewness	0.19	0.36	0.57	0.73	0.90	1.06	1.25

Table 3. Decision time (milliseconds)

Scale in number of VMs	200	400	600	800	1000	1200	1400
Bin-packing	5.9	12.0	19.8	26.0	35.7	39.8	46.4
Skewness	38.8	115.5	233.5	349.9	529.9	674.6	1065.9

The resource allocation can benefit from the VM stop/resume mechanism. VM stop/resume is generally faster than starting/stopping an application directly because they skipped the time-consuming initialization process for large software. The latest stop/resume technology (Zhu, Jiang and Xiao, 2011) can accelerate such an operation to several seconds.

The capacity of data centers in the real world is finite. The illusion of infinite capacity in the Cloud is provided through statistical multiplexing. When a large number of applications experience their peak demand around the same time, the available resources in the Cloud can become constrained and some of the demand may not be satisfied. The amount of computing capacity available to an application is limited by the placement of its running instances on the servers. The more instances an application has and the more powerful the underlying servers are, the higher the potential capacity for satisfying the application demand. On the other hand, when the demand of the applications is low, it is important to conserve energy by reducing the number of servers used.

We develop a system that provides automatic scaling for Internet applications in the PKU Cloud. We model the problem as Class Constrained Bin Packing (CCBP) where each server is a bin and each class represents an application. In the traditional bin packing problem, a series of items of different sizes need to be packed into a minimum number of bins. The class constrained version of this problem divides the items into classes or colors. Each bin has capacity v and can accommodate items from at most c distinct classes. It is "class constrained" because the class diversity of items packed into the same bin is constrained. The goal is to pack the items into a minimum number of bins. The class constraint reflects the practical limit on the number of applications a server can run simultaneously. For J2EE applications, for example, memory is typically the bottleneck resource. The capacity of a bin represents the amount of resources available at a server for all its applications. We develop an innovative auto scaling algorithm that achieves good demand satisfaction ratio and supports green computing.

6 LOAD PREDICTION

Load prediction has significant impacts on resource allocation. With an over-estimated load, a scheduler may allocate more resources than necessary. Therefore some of the resources are wasted. On the contrary, with an under-estimated load, the resource allocation may be insufficient. Consequently, VOD user may complain the video is not fluent and online game players may get angry because they cannot control an avatar.

We found that two categories of load prediction algorithm are widely adopted. One category

composed of variations of the Exponentially Weighted Moving Average (EWMA) algorithm. It is designed based on the assumption that the future value of a random variable has strong relation to its recent history. It has been used in TCP for Round Trip Time (RTT) estimation for decades. Algorithms of the other category adopt the auto-regressive (AR) model. It requires more computation than EMWA based algorithms. But it can incorporate periodicity, which is hard to be utilized in EWMA alternatives, for better precision.

6.1 EWMA Variations

With the original EWMA, load at time t is calculated by $E(t) = \alpha * O(t) + (1 - \alpha) * E(t - 1)$, $0 \leq \alpha \leq 1$, where $E(t)$ and $O(t)$ are the estimated and the observed load at time t, respectively. The parameter alpha reflects a tradeoff between stability and responsiveness. The larger the alpha is, the more agile the estimated load will be (low gain). On the contrary, the smaller the alpha is, the more stable the estimated load will be (high gain).

The load prediction algorithm adopted in MUSE (Chase et al., 2001) is a variation of EWMA. It uses a high gain EWMA and a low gain EWMA. If the latest observed load does not deviate much from recent observations, the low gain EWMA is used. Otherwise the high gain EWMA is used. This eliminates occasionally noisy observations. The output is further processed by a hysteresis filter for stabilization. The working set size estimator (Waldspurger, 2002) in ESX server also incorporates a similar technique. It uses three EWMAs with high, medium and low gain. The highest EWMA is selected as output to avoid under estimation as much as possible.

We designed a "Fast Up and Slow Down" (FUSD) predicting algorithm for the load predictor in the VM Scheduler of the PKU Cloud. It is worth noticing that EWMA does not capture the rising trends of resource usage. For example, when we see a sequence of O(t) = 10; 20; 30; and 40, it is reasonable to predict the next value to be 50. Unfortunately, when alpha is between 0 and 1, the predicted value is always between the historical value and the observed one. This phenomenon easily cause under provisioning when load is rising. To reflect the "acceleration", we take an innovative approach by setting alpha to a negative value. On the other hand, when the observed resource usage is going down, we want to be conservative in reducing the estimation by using a normal alpha. That's why it is called "Fast Up and Slow Down". It dramatically reduces the number of hot spots and live migration for Skewness and bin-packing VM schedulers.

6.2 The AR Model

In some works, future load is modeled as a linear function of several other factors such as the load history, time, or resource allocation. The parameters can be calculated by training with data in the past. Then the model can predict the future load. This methodology is called Auto-Regression (AR), represented as AR(p), where p is the number of factors considered in this model. AR model works well for periodical load.

The Sandpiper VM scheduler (Wood et al., 2007) adopts AR(1). It models the load at time t as a linear function of the average of n latest observations. It cannot utilize periodicity because it is unaware if the application is periodical. In the research on provisioning servers for connection-intensive services (Chen et al., 2008), AR(n) is used to predict the number and login rate of MSN clients. The load is modeled as a linear function of six independent variables, two of the most recent observations and four of the observations at the same time in last four weeks. The results shows perfect fit between the predicted and the observed load. This is because the load of MSN clients presents perfect periodicity in its weekly pattern. We speculate that most popular Internet applications present such characteristics.

7. FUTURE RESEARCH DIRECTIONS

Cloud ecology involves more than one Cloud vendor. The concept of Cloud federation is proposed to architect software over multiple cloud services (Celesti et al., 2010). Besides vendor lock-in avoidance, applications built on Cloud federation enjoy more options for on-demand resource provisioning. Multiple Cloud services may back up each other for fault-tolerance. Or, with carefully arrangement, it may achieve better performance/price ratio than single-vendor approaches do. In other words, Cloud federation brings new possibilities to Cloud elasticity. However, there are challenges to overcome. It is hard to implementing uniform platform layer incorporating Cloud services with distinct service models and user interfaces. The difference among underlying technologies is obstacle to interoperability. Researchers just begin to tackle those problems. Yang (Yang X. et al., 2012) presented a new Cloud federation model for real time applications capable of on-demand resource provisioning across multiple Cloud vendors.

Live migration of VM plays an important role in Cloud elasticity. Current live migration technology, however, is not fully satisfactory. Remote Direct Memory Access (RDMA) infrastructure was facilitated to speed up live migration (Huang et al., 2007), but it is not always seen in Cloud infrastructure other than those dedicated to scientific computing. MECOM (Jin et al., 2009) adopts compression algorithm to reduce the data transferred during live migration and consequently shorten its total time span. MDD (Zhang et al., 2010) takes data de-duplication to achieve the similar effect. They actually trade CPU cycles and memory space for performance of live migration. Such optimizations are not adequate for migrating VMs away to offload a busy physical server. Post-copy (Hines et al., 2009) approach is capable of migrating CPU load away as soon as possible, but exception of either side of migration could cause crash of the migrating VM. Moreover, applications in the migrating VM may experience worse performance degradation than that in pre-copy approach. Shrinker (Riteau, Morin, Priol, 2010) has suggested a real-time fingerprint system for memory pages and DHT-based content sharing system to enable live migration over Wide-Area Network. But they didn't solve the hash collision problem. Since there may be no one live migration technology fit for all purpose, we suggest a hybrid solution. Various optimizations for live migration may be combined in a toolkit. It's up to the resource scheduler which optimization(s) to use.

Latest development of virtualization technology arms Cloud infrastructure with new weapons for Cloud elasticity. Snow Flock (Lagar-Cavilla et al., 2009) enables "fork" operation for virtual machines. Fast VM start-up (Zhu, Jiang, and Xiao, 2011) can start up a VM in milliseconds. Both of them can be used to support fast deployment for flash crowd. Partial migration technology (Bila et al, 2012) extends the post-copy approach to temporarily migrate away the active states of an idle virtual machine, so that the physical server has more chances to sleep and therefore save power. New I/O devices are virtualized at hardware level so that each virtual machine could enjoy high performance with pass-through devices. Live migration with pass-through device is not a trifle because of the difficulty of migrating hardware-specific device states. CompSC (Pan et al, 2012), however, has already added support for pass-through NIC to live migration. Incorporating those technologies into existing Cloud is still open to researchers.

CONCLUSION

Cloud elasticity is an appealing characteristic of Cloud infrastructure. It involves scaling up and down resource allocation according to the real time requirements of applications. In face of resource shortage, an elastic system is able to fairly allocate

resources. Elasticity is implemented in different levels of the Cloud architecture.

Hypervisor is responsible for allocating local resources to the VMs. CPU scheduling algorithm is similar to the process scheduling in the OS. Proportional application performance can be enforced by adjusting scheduling weights of VMs. Ballooning technology realizes memory allocation by inflating or deflating the balloon process residing in each guest OS, while TMEM maintains a public memory pool for page cache and swap. Scheduling I/O resources is a hot research field, where many problems remain to be solved.

As a global resource scheduling mechanism, live migration has its pros and cons. Its advantage is application neutral, but the overhead of migration should be considered carefully. By modeling the scheduling problem with the bin packing problem, we can exploit the abundant existing algorithms in that well studied field. We introduce a practical, online bin-packing scheduling algorithm. Then we introduced the skewness algorithm that avoids uneven utilization of different kind of resources of a physical server. Both of them are incorporated in a real Cloud system.

With internal information of applications, on-demand resource provision technologies at application level can perform a more precise provision. Taking web application as an example, we introduced several elasticity technologies.

Load prediction has significant impacts on resource allocation. Prediction error may invoke under-provisioning or over-provisioning with unpleasant implications. We introduced two categories of prediction algorithms that widely adopted. EWMA based algorithms are simpler, while AR models are more precise.

We finally pointed out future directions of technologies in this field.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China Project 61170056.

REFERENCES

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. Proceedings of the ACM SIGCOMM conference on Data communication, USA, (pp. 63-74). doi: 10.1145/1402958.1402967

Armbrust, M., Fox, A., & Griffith, R. (2009). *Above the clouds: A Berkeley view of cloud computing*. Berkeley: EECS Department, University of California.

AWS Case Study: Washington Post. (n.d.). Retrieved from http://aws.amazon.com/solutions/case-studies/washington-post/

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., & Ho, A. … Warfield, A. (2003). Xen and the art of virtualization. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, USA, (pp. 164-177). doi: 10.1145/945445.945462

Bila, N., Lara, E., Josi, K., Lagar-Cavilla, H., Hiltunen, M., & Satyanarayanan, M. (2012). Jettison: Efficient idle desktop consolidation with partial VM migration. Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12), (pp. 1-14). Retrieved from http://lagarcavilla.com/publications/BilaEurosys12.pdf

Bobroff, N., Kochut, A., & Beaty, K. (2007). Dynamic placement of virtual machines for managing SLA violations. International Symposium on Integrated Network Management, Munich, (pp. 119-128). doi: 10.1109/INM.2007.374776

Celesti, A., Tusa, F., Villari, M., & Puliafito, A. (2010). How to enhance cloud architectures to enable cross-federation. Proceedings of the IEEE third International Conference on Cloud Computing (CLOUD), (pp. 337-345).

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., & Burrows, M. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, *26*(2), 1–26. doi:10.1145/1365815.1365816

Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., & Doyle, R. P. (2001). Managing energy and server resources in hosting centers. Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, (pp. 103-116). doi: 10.1145/502034.502045

Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., & Zhao, F. (2008). Energy-aware server provisioning and load dispatching for connection-intensive internet services. Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, (pp. 337-350).

Clark, C., Fraser, K., Hand, S., Hansen, J. G., & Jul, E. Limpach, C., … Warfield., A. (2005). Live migration of virtual machines. Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, Vol. 2, (pp. 273-286).

Credit Based Scheduler. (2007). Retrieved from http://wiki.xensource.com/xenwiki/CreditScheduler

Gambosi, G., Postiglione, A., & Talamo, M. (2000). Algorithms for the relaxed online bin-packing model. *SIAM Journal on Computing*, *5*(30), 1532–1551. doi:10.1137/S0097539799180408

Garey, M. R., & Johnson, D. S. (1985). A 71/60 theorem for bin packing. *Journal of Complexity*, *1*(1), 65–106. doi:10.1016/0885-064X(85)90022-6

Ghemawat, S., Gobioff, H., & Leung, S. (2003). The Google file system. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, (pp. 29-43). doi: 10.1145/945445.945450

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., & Lahiri, P. … Sengupta, S. (2009). VL2: A scalable and flexible data center network. Proceedings of the ACM SIGCOMM Conference on Data Communication, (pp. 51-62). doi: 10.1145/1592568.1592576

Gulati, A., Merchant, A., Uysal, M., Padala, P., & Varman, P. (2009). Efficient and adaptive proportional share I/O scheduling. *ACM SIGMETRICS Performance Evaluation Review*, *37*(2), 79–80. doi:10.1145/1639562.1639595

Hines, M., & Gopalan, K. (2009). Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09), (pp. 51-60). doi: 10.1145/1508293.1508301

Huang, W., Gao, Q., Liu, J., & Panda, D. (2009). High performance of virtual machine migration with RDMA over modern interconnects. Proceedings of the 2007 IEEE International Conference on Cluster Computing, (pp. 11-20). doi: 10.1109/CLUSTR.2007.4629212

Jin, H., Deng, L., Wu, S., Shi, X., & Pan, X. (2009). Live virtual machine migration with adaptive, memory compression. Proceedings of the 2009 IEEE International Conference on Cluster Computing, (pp. 11-20). doi: 10.1109/CLUSTR.2009.5289170

Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., & Tantawi, A. (2006). Dynamic placement for clustered web applications. Proceedings of the 15th International Conference on World Wide Web, (pp. 595-604). doi: 10.1145/1135777.1135865

Lagar-Cavilla, H., Whitney, J., Scannell, A., Patchin, P., Rumble, S., Lara, E., et al. (2009). SnowFlock: Rapid virtual machine cloning for cloud computing. Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys '09), (pp. 1-12). doi: 10.1145/1519065.1519067

Love, R. (2004). Kernel korner: I/O schedulers. Linux Journal. Retrieved from http://www.linuxjournal.com/article/6931

Magenheimer, D. (2008, April). Add self-ballooning to balloon driver [Electronic mailing list message]. Retrieved from http://old-list-archives.xen.org/archives/html/xen-devel/2008-04/msg00567.html

Magenheimer, D. (2009). Transcendent memory and Linux. Retrieved from http://oss.oracle.com/projects/tmem/dist/documentation/papers/tmemLS09.pdf

McNett, M., Gupta, D., Vahdat, A., & Voelker, G. M. (2007). Usher: an extensible framework for managing clusters of virtual machines. Proceedings of the 21st Conference on Large Installation System Administration Conference, (pp. 1-15).

Meisner, D., Gold, B. T., & Wenisch, T. F. (2009). PowerNap: Eliminating server idle power. Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, (pp. 205-216). doi: 10.1145/1508244.1508269

Miller, R. (2008). Microsoft: PUE of 1.22 for data center containers. Retrieved from http://www.datacenterknowledge.com/archives/2008/10/20/microsoft-pue-of-122-for-data-center-containers/

Padala, P., Hou, K., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., et al. (2009). Automated control of multiple virtualized resources. Proceedings of the 4th ACM European Conference on Computer Systems, (pp. 13-26). doi: 10.1145/1519065.1519068

Padala, P., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., & Singhal, S. … Salem, K. (2007). Adaptive control of virtualized resources in utility computing environments. Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, (pp. 289-302). doi: 10.1145/1272996.1273026

Pan, Z., Dong, Y., Chen, Y., Zhang, L., & Zhang, Z. (2012).CompSC: Live migration with pass-through devices. Proceedings of the 2012 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, (pp. 1-12).

Riteau, P., Morin, C., & Priol, T. (2010). Shrinker: Efficient wide-area live virtual machine migration using distributed content-based addressing. Research Report:RR-7198, INRIA. Retrieved from http://hal.inria.fr/docs/00/45/47/27/PDF/RR-7198.pdf

Seelam, S. R., & Teller, P. J. (2007). Virtual I/O scheduler: A scheduler of schedulers for performance virtualization. Proceedings of the 3rd International Conference on Virtual Execution Environments, (pp. 105-115). doi: 10.1145/1254810.1254826

Shieh, A., Kandula, S., Greenberg, A., Kim, C., & Saha, B. (2011). Sharing the data center network. Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, (pp. 1-14). Retrieved from http://www.usenix.org/events/nsdi11/tech/full_papers/Shieh.pdf

Singh, A., Korupolu, M., & Mohapatra, D. (2008). Server-storage virtualization: integration and load balancing in data centers. Proceedings of the 2008 ACM/IEEE conference on Supercomputing, USA, 1-12.

Tang, C., Steinder, M., Spreitzer, M., & Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. Proceedings of the 16th International Conference on World Wide Web, (pp. 331-340). doi: 10.1145/1242572.1242618

Waldspurger, C. A. (2002). Memory resource management in VMware ESX server. Proceedings of the 5th Symposium on Operating Systems Design and implementation, (pp. 181-194). doi: 10.1145/844128.844146

Wood, T., Shenoy, P., Venkataramani, A., & Yousif, M. (2007). Black-box and gray-box strategies for virtual machine migration. Proceedings of 4th USENIX Symposium on Networked Systems Design and Implementation, (pp. 229-242). Retrieved from http://www.usenix.org/events/nsdi07/tech/full_papers/wood/wood_html/

Xiao, Z., Song, W., & Chen, Q. (2011). Dynamic resource allocation using virtual machines for cloud computing environment (unpublished paper). Peking University.

Xiao, Z., Song, W., Chen, Q., & Luo, H. (2010). Gone with the cloud: Adaptive resource virtualization for Amazon EC2-like environment (unpublished paper). Peking University.

Xu, H., & Li, B. (2011). Egalitarian stable matching for VM migration in cloud computing. IEEE Conference on Computer Communications Workshops, Shanghai, (pp. 631-636).

Yang, X. (2011). QoS-oriented service computing: Bring SOA into cloud environment. In Liu, X. (Ed.), *Advanced design approaches to emerging software systems: Principles, methodology and tools*. Hershey, PA: IGI Global. doi:10.4018/978-1-60960-735-7.ch013

Yang, X., Nasser, B., Surrige, M., & Middleton, S. (2012). A business-oriented cloud federation model for real time applications. Future Generation Computer Systems, Elsevier. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167739X12000386

Zhang, X., Huo, Z., Ma, J., & Meng, D. (2010). Exploiting data deduplication to accelerate live virtual machine migration. Proceedings of the 2010 IEEE International Conference on Cluster Computing, (pp. 11-20). doi: 10.1109/CLUSTER.2010.17

Zhu, J., Jiang, Z., & Xiao, Z. (2011). Twinkle: A fast resource provisioning mechanism for internet services. 2011 Proceedings INFOCOM, Shanghai, (pp. 802-810).

ADDITIONAL READING

Armbrust, M., Fox, A., & Griffith, R. (2009). *Above the clouds: A Berkeley view of cloud computing*. Berkeley: EECS Department, University of California.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., & Ho, A. … Warfield, A. (2003). Xen and the art of virtualization. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, (pp. 164-177). doi: 10.1145/945445.945462

Bobroff, N., Kochut, A., & Beaty, K. (2007). Dynamic placement of virtual machines for managing SLA violations. International Symposium on Integrated Network Management, Munich, (pp. 119-128). doi: 10.1109/INM.2007.374776

Buyya, R., Ranjan, R., & Calheiros, R. N. (2009). Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. International Conference on High Performance Computing & Simulation, Leipzig, (pp. 1-11). doi: 10.1109/HPCSIM.2009.5192685

Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., & Doyle, R. P. (2001). Managing energy and server resources in hosting centers. Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, USA, (pp. 103-116). doi: 10.1145/502034.502045

Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., & Zhao, F. (2008). Energy-aware server provisioning and load dispatching for connection-intensive internet services. Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, (pp. 337-350).

Chisnall, D. (2007). The definitive guide to the Xen hypervisor. Upper Saddle River, NJ: Prentice Hall PTR: Prentice Hall.

Gulati, A., Merchant, A., Uysal, M., Padala, P., & Varman, P. (2009). Efficient and adaptive proportional share I/O scheduling. *ACM SIGMETRICS Performance Evaluation Review*, *37*(2), 79–80. doi:10.1145/1639562.1639595

Hermenier, F., Lorca, X., Menaud, J., Muller, G., & Lawall, J. (2009). Entropy: A consolidation manager for clusters. Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, (pp. 41-50). doi: 10.1145/1508293.1508300

Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., & Tantawi, A. (2006). Dynamic placement for clustered web applications. Proceedings of the 15th international conference on World Wide Web, (pp. 595-604). doi: 10.1145/1135777.1135865

Magenheimer, D. (2009). Transcendent memory and Linux. Retrieved from http://oss.oracle.com/projects/tmem/dist/documentation/papers/tmemLS09.pdf

McNett, M., Gupta, D., Vahdat, A., & Voelker, G. M. (2007). Usher: an extensible framework for managing clusters of virtual machines. Proceedings of the 21st Conference on Large Installation System Administration Conference, (pp. 1-15).

Padala, P., Hou, K., Shin, K. G., Zhu, X., Uysal, M., & Wang, Z. … Merchant, A. (2009). Automated control of multiple virtualized resources. Proceedings of the 4th ACM European Conference on Computer systems, USA, (pp. 13-26). doi: 10.1145/1519065.1519068

Padala, P., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., & Singhal, S. … Salem, K. (2007). Adaptive control of virtualized resources in utility computing environments. Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, USA, (pp. 289-302). doi: 10.1145/1272996.1273026

Seelam, S. R., & Teller, P. J. (2007). Virtual I/O scheduler: A scheduler of schedulers for performance virtualization. Proceedings of the 3rd International Conference on Virtual Execution Environments, (pp. 105-115). doi: 10.1145/1254810.1254826

Singh, A., Korupolu, M., & Mohapatra, D. (2008). Server-storage virtualization: Integration and load balancing in data centers. Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, (pp. 1-12).

Tang, C., Steinder, M., Spreitzer, M., & Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. Proceedings of the 16th International Conference on World Wide Web, (pp. 331-340). doi: 10.1145/1242572.1242618

Turner, A., Sangpetch, A., & Kim, H. S. (2010). Empirical virtual machine models for performance guarantees. Proceedings of the Conference on Large Installation System Administration Conference, (pp. 1-11). Retrieved from https://db.usenix.org//events/lisa10/tech/full_papers/Turner.pdf

Wang, Y., Wang, X., Chen, M., & Zhu, X. (2008). Power-efficient response time guarantees for virtualized enterprise servers. Real-Time Systems Symposium, Barcelona, (pp. 303-312). doi: 10.1109/RTSS.2008.20

Wei, G., Vasilakos, A. V., Zheng, Y., & Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, *54*(2), 252–269. doi:10.1007/s11227-009-0318-1

Wood, T., Cherkasova, L., Ozonat, K., & Shenoy, P. (2008). Profiling and modeling resource usage of virtualized applications. Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, (pp. 366-387).

Wood, T., Shenoy, P., Venkataramani, A., & Yousif, M. (2007). Black-box and gray-box strategies for virtual machine migration. Proceedings of 4th USENIX Symposium on Networked Systems Design and Implementation, (pp. 229-242). Retrieved from http://www.usenix.org/events/nsdi07/tech/full_papers/wood/wood_html/

Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E., & Corner, M. D. (2009). Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers. Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, (pp. 31-40). doi: 10.1145/1508293.1508299

Yang, X., Bruin, R., & Dove, M. (2010). Developing an end-to-end scientific workflow: A case study of using a reliable, lightweight, and comprehensive workflow platform in e-science. Retrieved from http://doi.ieeecomputersociety.org/10.1109/MCSE.2009.211

Yang, X., Wang, L., & von Laszewski, G. (2009). Recent research advances in e-science. Cluster Computing Special Issue, 12(4), 353-356. Retrieved from http://springerlink.com/content/f058408qr771348q/

Zhu, J., Jiang, Z., & Xiao, Z. (2011). Twinkle: A fast resource provisioning mechanism for internet services. 2011 IEEE Proceedings of INFOCOM, Shanghai, (pp. 802-810).

KEY TERMS AND DEFINITIONS

Cloud User: A cloud user refers to the person who use the service provided by a cloud system. According to the type of cloud service, it may be an application user, a software developer or a system architect.

Live Migration: A running virtual machine can be migrated from one physical machine to another without its application being interrupted. That technology is called live migration.

Proportional Resource Allocation: With a proportional resource allocation strategy, when contention encountered, each entity should get a share of resource proportional to its presetting weights, no matter how greedy the other entities are.

Resource Provisioning: Resource provisioning refers to the process of assembling computing resources like CPU, memory, disk and network I/O to serve application computation.

Resource Scheduler: A resource scheduler refers to the entity that performs resource scheduling task.

Scheduling Algorithm: Scheduling algorithm refers to the detailed process of the policy for resource scheduling.

Service Level Agreement (SLA): SLA is the quantized specification of the service a cloud provider promises.