International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE) ISSN: 0976-1353 Volume 9 Issue 1 –JULY 2014.

Fair Resource Allocation in Cloud Using Virtual Machines

A.Nandishwar
Mtech-CSE, Teegala Krishna Reddy Engineering College, Medbowli, RN.Reddy nager, Meerpet, Andhra Pradesh, India.

Abstract--

Cloud computing is an emerging technology in the present generation. Cloud provides various resources and services to its users most of them are for free of cost. Cloud computing allows business customers to develop their business by providing resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. In this paper, we present a system that uses virtualization technology for fair allocation of data center resources based on application demands and support green computing by optimizing the number of servers in use. In this paper we introduce the concept of "skewness" to measure the unevenness in the multi-dimensional resource utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources. We develop a set of heuristic solutions that prevent overload in the system effectively while saving energy used. Our experimental simulation and results demonstrate that our algorithm achieves good and offers better performance.

***Keywords*—Virtualization, Skewness, Green Computing, Resource Multiplexing.**

1 INTRODUCTION

The services and the resources offering by the cloud are attracting manybusinesses. There is a lot of discussion on the benefits and costs of the cloud model and on how to move legacy applications onto the cloud platform. Here we study a different problem: how can a cloud service provider best multiplex its virtual resources ontothe physical hardware? This is important because much of the touted gains in the cloud model come from such multiplexing.Studies have found that servers in many existing data centers are often severely under-utilized due to over-provisioning for the peak demand. The cloud model is expected to make such practice unnecessary by offering automatic scale up and down in response to load variation. Besides reducing the hardware cost, it also saves on electricity which contributes to a significant portion of the operational expenses in large data centers. Virtual machine monitors (VMMs) like Xen provide a mechanism for mapping virtual machines (VMs) to physical resources. This mapping is largely hidden from the cloud users. It is up to the cloud provider to make sure the underlying physical machines (PMs) have sufficient resources to meet their needs. VM live migration technology makes it possible to change the mapping between VMs and PMs While applications are running. However, a policy issue remains as how to decide the mapping adaptively so that the resource demands of VMs are met while the number of PMs used is minimized. This is challengingwhen the resource needs of VMs are heterogeneous due to the diverse set of applications they run and vary with time as the workloads grow and shrink. The capacity of PMs canalso be heterogenous because multiple generations of hardware Co-exist in a data center.We aim to achieve two goals in our algorithm:

_ Overload avoidance: the capacity of a PM should be sufficient to satisfy the resource needs of all VMs running on it. Otherwise, the PM is overloaded and can lead to degraded performance of its VMs.

_ Green computing: the number of PMs used should be minimized as long as they can still satisfy the needs of all VMs. Idle PMs can be turned off to save energy. There is an inherent trade-off between the two goals in the face of changing resource needs of VMs. For overload avoidance, we should keep the utilization of PMs Low to reduce the possibility of overload in case the resource needs of VMs increase later. For green computing, we should keep the utilization of PMs reasonably high to make efficient use of their energy. In this paper, we present the design and implementation of an automated resource management system that achieves agood balance between the two goals.

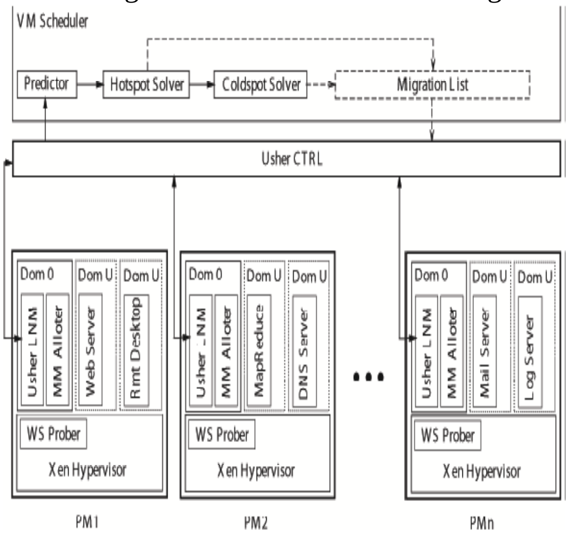

Fig. 1. System Architecture

2 OVERVIEW

The architecture of the system is presented in Figure 1. Each PM runs the Xen hypervisor (VMM) which supports a privileged domain 0 and one or more domain U [3]. Each VM in domain U encapsulates one or more applications such as Web server, remote desktop, DNS, Mail, Map/Reduce, etc. We assume all PMs Share backend storage. The multiplexing of VMs to PMs is managed using the Usher framework. The main logic of our system is implemented as a set of plug-ins to Usher. Each node runs an Usher local node manager (LNM) on domain 0 which collects the usage statistics of resources for each VM on that node. The CPU and network usage can be calculated by monitoring the scheduling events in Xen. The memory usage within a VM, however, is not visible to the hypervisor. One approach is to infer memory shortage of a VM by observing its swap activities. Unfortunately, the guest OS is required to install a separate swap partition. Furthermore, it may be too late to adjust the memory allocation by the time swapping occurs. Instead we implemented a working set prober (WS Prober) on each hypervisor to estimate the working set sizes of VMs running on it. We use the random page sampling technique as in the VMware ESX Server. The statistics collected at each PM are forwarded to the Usher central controller (Usher CTRL) where our VM scheduler runs. The VM Scheduler is invoked periodically and receives from the LNM the resource demand history of VMs, the capacity and the load history of PMs, and the current layout of VMs on PMs. The scheduler has several components. The predictor predicts the future resource demands of VMs and the future load of PMs based on past statistics. We compute the load of a PM by aggregating the resource usage of its VMs. The details of the load prediction algorithm will be described in the next section. The LNM at each node first attempts to satisfy the new demands locally by adjusting the resource allocation of VMs sharing the same VMM. Xen can change the CPU allocation among the VMs by adjusting their weights in its CPU scheduler. The MM Alloter on domain 0 of each node is responsible for adjusting the local memory allocation. The hot spot solver in our VM Scheduler detects if the resource utilization of any PM is above the *hot threshold* (i.e., a hot spot). If so, some VMs running on them will be migrated away to reduce their load. The cold spot solver checks if the average utilization of actively used PMs (APMs) is below the *green computing threshold*. If so, some of those PMs could potentially be turned off to save energy. It identifies the set of PMs whose utilization is below the *cold threshold* (i.e., cold spots) and then attempts to migrate away all their VMs. It then compiles a migration list of VMs and passes it to the Usher
CTRL for execution.

3 PREDICTING RESOURCES

We need to predict the future resource needs of VMs. As said earlier, our focus is on Internet applications. One solution is to look inside a VM for application level statistics, e.g., by parsing logs of pending requests. Doing so requires modification of the VM which may not always be possible. Instead, we make our prediction based on the past external behaviours of VMs. Our first attempt was to calculate an exponentially weighted moving average (EWMA) using a TCP-like scheme:

$E(t) = \alpha * E(t-1) + (1-\alpha) * O(t),$

$0 \le \alpha \le 1$

Where$E$ (t) and O (t) are the estimated and the observed load at time t, respectively. _ reflects a trade-off between stability and responsiveness. We use the EWMA formula to predict the CPU load on the DNS server in our university. We measure the load every minute and predict the load in the next minute. Figure 2 (a) shows the results for _ = 0:7. Each dot in the figure is an observed value and the curve represents the predicted values. Visually, the curve cuts through the middle of the dots which indicates a fairly accurate prediction. This is also verified by the statistics in Table 1. The parameters in the parenthesis are the _ values. *W* is the length of the measurement window (explained later). The "median" error is calculated as a percentage of the observed value: $:|E(t) - O(t)|/O(t)$. The "higher" and "lower" error percentages are the percentages of predicted values that are higher or lower than the observed values, respectively. As we can see, the prediction is fairly accurate with roughly equal percentage of higher and lower values.

TABLE 1
Load prediction algorithms

	ewma(0.7) *W* = 1	fusd(-0.2, 0.7) *W* = 1	fusd(-0.2, 0.7) *W* = 8
median error	5.6%	9.4%	3.3%
high error	56%	77%	58%
low error	44%	23%	41%

Although seemingly satisfactory, this formula 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 α is between 0 and 1, the predicted value is always between the historic value and the observed one. To reflect the "acceleration", we take an innovative approach by setting α to a negative value. When $-1 \le \alpha < 0$, the above formula can be transformed into the following:

$$E(t) = -|\alpha| * E(t-1) + (1+|\alpha|) * O(t)$$

$$= O(t) + |\alpha| * (O(t) - E(t-1))$$

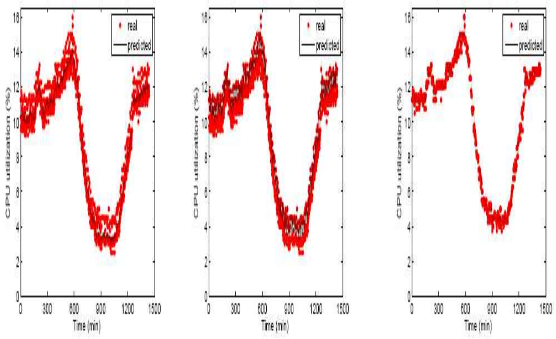

Fig. 2. CPU load prediction for the DNS server at our university. W is the measurement window.

4 SKEWNESSALGORITHMS

We introduce the concept of *skewness* to quantify the unevenness in the utilization of multiple resources on a server. Let *n* be the number of resources we consider and *ri*be the utilization of the *i*-th resource. By minimizing the*skewness*, we can combine different types of workloads nicely and improve the overall utilization of server resources. In the following, we describe the details of our algorithm.

4.1 Hot spots and cold spots

Our algorithm executes periodically to evaluate the resource allocation status based on the predicted future resource demands of VMs. We define a server as a *hot spot* if the utilization of any of its resources is above a *hot threshold*. This indicates that the server is overloaded and hence some
VMs running on it should be migrated away. We define the*temperature* of a hot spot p as the square sum of its resource utilization beyond the hot threshold.We define a server as a *cold spot* if the utilizations of allits resources are below a *cold threshold*. This indicates thatthe server is mostly idle and a potential candidate to turn offto save energy. However, we do so only when the averageresource utilization of all actively used servers (i.e., APMs)in the system is below a *green computing threshold*. A serveris actively used if it has at least one VM running. Otherwise,it is inactive. Finally, we define the *warm threshold* to be alevel of resource utilization that is sufficiently high to justifyhaving the server running but not so high as to risk becominga hot spot in the face of temporary fluctuation of applicationresource demands.

4.3 Green computing

Green computing is the practice of using computing resources efficiently. The goals are to reducethe use of hazardous materials, maximize energy efficiency during the product's lifetime, andpromote recyclability orbiodegradability of defunct products and factory waste. Such practicesinclude the implementation of energy-efficient central processing units (CPUs), servers andperipherals as well as reduced resourceconsumption and proper disposal of electronic waste (ewaste). In 1992, the U.S. Environmental Protec tion Agency launched Energy Star, a voluntarylabelin g program which is designed to promote and recognize energy-efficiency in monitors, climate control equipment, and other technologies. This resulted in thewidespread adoption of sleep mode among consumer electronics. The term "green computing" wasprobably coinedshortly after the Energy Star program began; there are several USENET posts dating back to 1992 which use the term in this manner.

Our green computing algorithm is invoked when the averageutilizations of all resources on active servers are below the green computing threshold. We sort the list of cold spots in the system based on the ascending order of their memory size. Since we need to migrate away all its VMs before we can shut down an under-utilized server, we define the memory size of a cold spot as the aggregate memory size of all VMs running on it. Recall that our model assumes all VMs connect to a shared back-endstorage. Hence, the cost of a VM live migration is determined mostly by its memory footprint.We try to eliminate the cold spot with the lowest cost first. For a cold spot *p*, we check if we can migrate all its VMs somewhere else. For each VM on *p*, we try to find a destination server to accommodate it. The resource utilizations of the server after accepting the VM must be below the *Warmthreshold*. While we can save energy by consolidating under-utilized servers, overdoing it may create hot spots in the future. The warm threshold is designed to prevent that. If multiple servers satisfy the above criterion, we prefer one that is not a current cold spot. This is because increasing load on a cold spot reduces the likelihood that it can be eliminated. However, we will accept a cold spot as the destination server if necessary. All things being equal, we select a destination server whose skewness can be reduced the most by accepting this VM. If we can find destination servers for all VMs on a cold spot, we record the sequence of migrations and update the predicted load of related servers. Otherwise, we do not migrate any of its VMs. The list of cold spots is also updated because some of them may no longer be cold due to the proposed VM migrations in the above process. The above consolidation adds extra load onto the related servers. This is not as serious a problem as in the hot spot mitigation case because green computing is initiated only when the load in the system is low. Nevertheless, we want to bound the extra load due to server consolidation. We restrict the number of cold spots that can be eliminated in each run of the algorithm to be no more than a certain percentage of active servers in the system. This is called the *consolidationlimit*.

5 EXPERIMENTALSIMULATIONS

We evaluate the performance of our algorithm using trace driven simulation. Note that our simulation uses the same code base for the algorithm as the real implementation in the experiments. This ensures the fidelity of our simulation results. Traces are per-minute server resource utilization, such as CPU rate, memory usage, and network traffic statistics, collected using tools like “perfmon” (Windows), the “/proc” file system (Linux), “pmstat/vmstat/netstat” commands (Solaris), etc.. Theraw traces are pre-processed into “Usher” format so that the simulator can read them. We collected the traces from a variety of sources like: Web InfoMall: the largest online Web archive in China (i.e., the counterpart of Internet Archive in the US) with more than three billion archived Web pages.RealCourse: the largest online distance learning systemin China with servers distributed across 13 major cities.AmazingStore: the largest P2P storage system in China. We also collected traces from servers and desktop computers in our university including one of our mail servers, the central DNS server, and desktops in our department. We post-processed the traces based on days collected and use random sampling and linear combination of the data sets to generate the workloads needed. All simulation in this section uses the real trace workload unless otherwise specified.

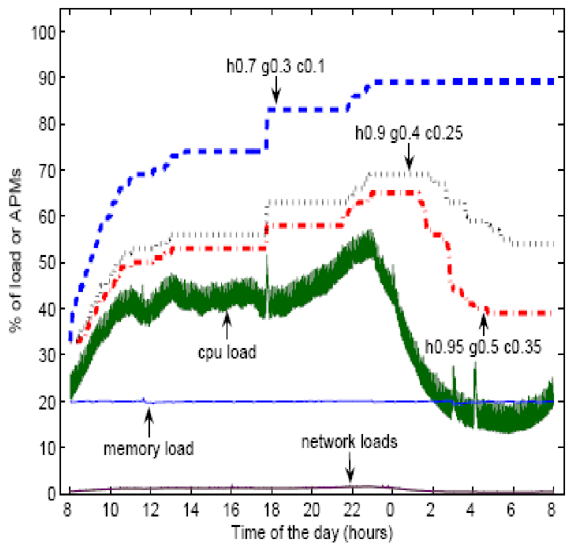

Fig.3. Impact of thresholds on the number of APMs

5.2 Scalability of the algorithm

We evaluate the scalability of our algorithm by varying the number of VMs in the simulation between 200 and 1400. The ratio of VM to PM is 10:1. The results are shown in Figure 4. The left figure shows that the average decision time of our algorithm increases with the system size. The speed of increase is between linear and quadratic. We break down the decision time into two parts: hot spot mitigation (marked as 'hot') and green computing (marked as 'cold'). We find that hot spot mitigation contributes more to the decision time. We also find that the decision time for the synthetic workload is higher than that for the real trace due to the large variation in the synthetic workload. With 140 PMs and 1400 VMs, the decision time is about 1.3 seconds for the synthetic workload and 0.2 second for the real trace. The middle figure shows the average number of migrations in the whole system during each decision. The number of migrations is small and increases roughly linearly with the system size. We find that hot spot contributes more to the number of migrations. We also find that the number of migrations in the synthetic workload is higher than that in the real trace. With 140 PMs and 1400 VMs, on average each run of our algorithm incurs about three migrations in the whole system for the synthetic workload and only 1.3 migrations for the real trace. This is also verified by the right figure which computes the average number of migrations per VM in each decision. The figure indicates that each VM experiences a tiny, roughly constant number of migrations during a decision run, independent of the system size. This number is about 0.0022 for the synthetic workload and 0.0009 for the real trace. Thistranslates into roughly one migration per 456 or 1174 decision intervals, respectively. The stability of our algorithm is very good. We also conduct simulations by varying the VM to PM ratio. With a higher VM to PM ratio, the load is distributed more evenly among the PMs. The results are presented in Section 4 of the supplementary file

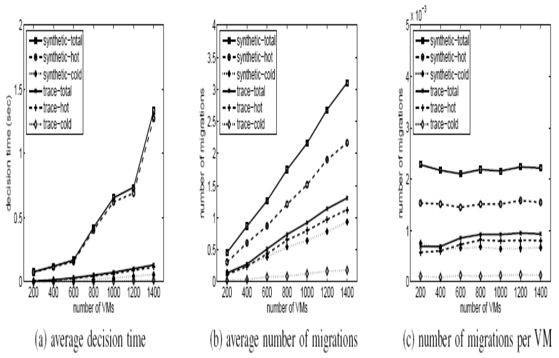

Fig. 4. Scalability of the algorithm with system size

6 EXPERIMENTS

6.1 Impact of live migration

One concern about the use of VM live migration is its impact on application performance. Previous studies have found this impact to be small [5]. We investigate this impact in our own experiment. We extract the data on the 340 live migrations in

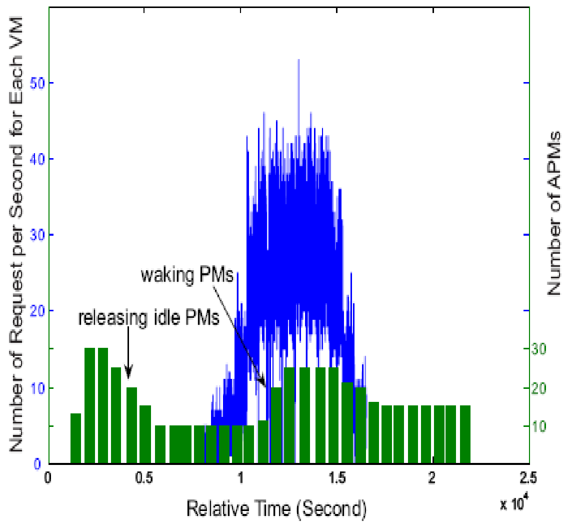

Fig. 5. #APMs varies with TPC-W load

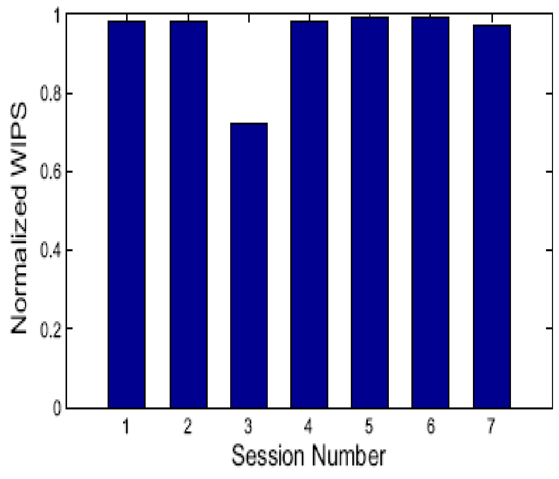

Fig. 6. Impact of live migration on TPC-W performance

Our 30 server experiment above. We find that 139 of them arefor hot spot mitigation. We focus on these migrations because that is when the potential impact on application performance is the most. Among the 139 migrations, we randomly pick 7 corresponding TPC-W sessions undergoing live migration. All these sessions run the “shopping mix” workload with 200 emulated browsers. As a target for comparison, we re-run the session with the same parameters but perform no migration and use the resulting performance as the baseline. Figure 6 shows the normalized WIPS (Web Interactions per Second) for the 7 sessions. WIPS is the performance metric used by TPC-W. The figure shows that most live migration sessions exhibit no noticeable degradation in performance compared to the baseline: the normalized WIPS is close to 1. The only exception is session 3 whose degraded performance is caused by an extremely busy server in the original experiment. Next we take a closer look at one of the sessions in figure 9 and show how its performance varies over time in figure 10. The dots in the figure show the WIPS every second. The two curves show the moving average over a 30 second window as computed by TPC-W. We marked in the figure when live migration starts and finishes. With self-ballooning enabled, the amount of memory transferred during the migration is about 600MB. The figure verifies that live migration causes no noticeable performance degradation. The duration of the migration is under 10 seconds. Recall that our algorithm is invoked every 10 minutes.

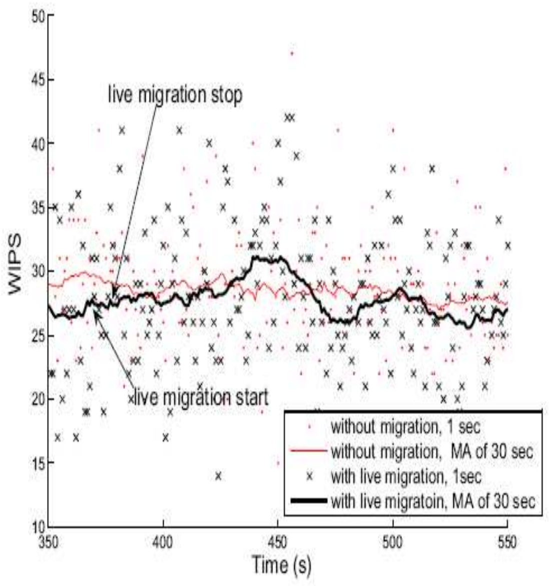

Fig. 7. TPC-W performance with and without live migration

7 RELATED WORK

7.1 Resource allocation by using live VM migration

VM live migration is a widely used technique for dynamic resource allocation in a virtualized environment. Our work also belongs to this category. Sandpiper combines multi-dimensional load information into a single *Volume* metric. It sorts the list of PMs based on their volumes and the VMs in each PM in their volume-to-size ratio (VSR). This unfortunately abstracts away critical information needed when making the migration decision. It then considers the PMs and the VMs in the pre-sorted order. We give a concrete example in Section 1 of the supplementary file where their algorithm selects the wrong VM to migrate away during overload and fails to mitigate the hot spot. We also compare our algorithm and theirs in real experiment. The results are analyzed inthe supplementary file to show how they behave differently. In addition, their work has no support for green computing and differs from ours in many other aspects such as load prediction. The HARMONY system applies virtualization technology across multiple resource layers. It uses VM and data migration to mitigate hot spots not just on the servers, but also on network devices and the storage nodes as well. It introduces the *Extended Vector Product (EVP)* as an indicator of imbalance in resource utilization. Their load balancing algorithm is a variant of the Toyoda method for multi-dimensional knapsack problem. Unlike our system, their system does not support green computing and load prediction is left as future work. In Section 6 of the supplementary file, we analyze the phenomenon that *VectorDot* behaves differently compared with our work and point out the reason why our algorithm can utilize residual resources better. Dynamic placement of virtual servers to minimize SLA violations is studied in. They model it as a bin packing problem and use the well-known first-fit approximation algorithm to calculate the VM to PM layout periodically. That algorithm, however, is designed mostly for off-line use. It is likely to incur a large number of migrations when applied in on-line environment where the resource needs of VMs change dynamically.

7.2 Green Computing

Many efforts have been made to curtail energy consumption in data centers. Hardware based approaches include novel thermal design for lower cooling power, or adopting power-proportional and low-power hardware. Our work uses Dynamic Voltage and Frequency Scaling (DVFS) to adjust CPU power according to its load. We do not use DVFS for green computing, as explained in the complementary file. PowerNap resorts to new hardware technologies such as Solid State Disk(SSD) and Self-Refresh DRAM to implement rapid transition(less than 1ms) between full operation and low power state, so that it can "take a nap" in short idle intervals. When a server goes to sleep, Somniloquy notifies an embedded system residing on a special designed NIC to delegate the main operating system. It gives the illusion that the server is always active. Our work belongs to the category of pure-software low-cost solutions. Similar to Somniloquy, Sleep Server initiates virtual machines on a dedicated server as delegate, instead of depending on a special NIC. LiteGreen does not use a delegate. Instead it migrates the desktop OS away so that the desktop can sleep. It requires that the desktop is virtualized with shared storage. Jettisoninvents "partial VM migration", a variance of live VM migration, which only migrates away necessary working set while leaving infrequently used data behind.

8 CONCLUSION

We have presented the design, implementation, and evaluation of a resource management system for cloud computing services. Our system multiplexes virtual to physical resources adaptively based on the changing demand. We use the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized. Our algorithm achieves both overload avoidance and green computing for systems with multi-resource constraints.

REFERENCES

[1] M. Armbrust *et al.*, "Above the clouds: A Berkeley view of cloud Computing," University of California, Berkeley, Tech. Rep., Feb 2009.

[2] L. Siegele, "Let it rise: A special report on corporate IT," in *The Economist*, Oct. 2008.

[3] 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," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP'03)*, Oct. 2003.

[4] "Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/."

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc.of the Symposium on Networked Systems Design and Implementation(NSDI'05)*, May 2005.

[6] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proc. of the USENIX Annual Technical Conference*, 2005.

[7] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: An extensible framework for managing clusters of virtual machines," in *Proc. of the Large Installation System Administration Conference(LISA'07)*, Nov. 2007.

[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. Ofthe Symposium on Networked Systems*

Design and Implementation(NSDI'07), Apr. 2007.

[9] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. of the symposium on Operating systems design andimplementation (OSDI'02)*, Aug. 2002.

[10] G. Chen, H. Wenbo, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. of the USENIXSymposium on Networked Systems Design and Implementation(NSDI'08)*, Apr. 2008.

[11] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proc. of the ACM European conference on Computersystems (EuroSys'09)*, 2009.

[12] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing slaviolations," in *Proc. of the IFIP/IEEEInternational Symposium on Integrated Network Management (IM'07)*, 2007.