INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)

Allocation of Dynamic Resources for Cloud Computing Environment using Virtual Machines

M. Sujitha
Department of MCA, Santhiram Engineering College
Nandyal, Affiliated to JNTU, Ananthapur, India
E. Sreenivasulu
Assistant Professor, Department of MCA, Santhiram Engineering College
Nandyal, Affiliated to JNTU, Ananthapur, India

Abstract:
Cloud computing enables business customers to scale up and down their 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 to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use. 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 heuristics that prevent overload in the system effectively while saving energy used. Trace driven simulation and experiment results demonstrate that our algorithm achieves good performance.

Keywords: *Cloud Computing, Resource Management, Virtualization, Green Computing*

1. Introduction

The elasticity and the lack of upfront capital investment offered by cloud computing is appealing to many businesses. 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 onto the physical hardware? Studies have found that servers in many existing data centers are often severely under-utilized due to over-provisioning for the peak demand [1] [2]. 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 [3]. This mapping is largely hidden from the cloud users. Users with the Amazon EC2 service [4], for example, do not know where their VM instances run. 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 [5], [6]. 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. The capacity of PMs can also 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 a good balance between the two goals. We make the following contributions:

- We develop a resource allocation system that can avoid overload in the system effectively while minimizing the number of servers used.
- We introduce the concept of "skewness" to measure the uneven utilization of a server. By minimizing skewness, we can improve the overall utilization of servers in the face of multi-dimensional resource constraints.
- We design a load prediction algorithm that can capture the future resource usages of applications accurately without looking inside the VMs. The algorithm can capture the rising trend of resource usage patterns and help reduce the placement churn significantly.
- The rest of the paper is organized as follows. Section 2 provides an overview of our system and Section 3 describes our algorithm to predict resource usage. The details of our algorithm are represented in Section 4.Section 5 and 6 present simulation and experiment results, respectively.

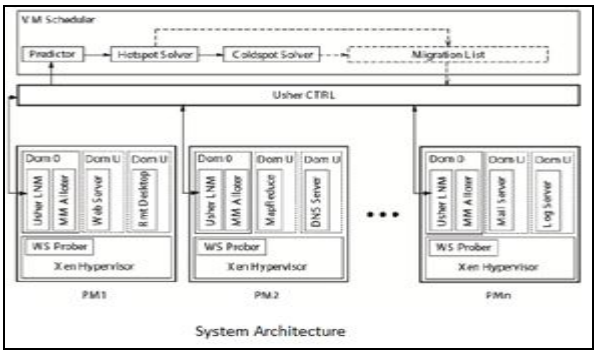


Figure 1: System Architecture

2. System 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 a backend storage.

The multiplexing of VMs to PMs is managed using the Usher framework [7]. 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.

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 MM allotter 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 then compiles a migration list of VMs and passes it to the Usher CTRL for execution.

3. Predicting Future Resource Needs

We need to predict the future resource needs of VMs. 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 behaviors of VMs. Our first attempt was to calculate an exponentially weighted moving average (EWMA) using a TCP-like scheme:

$E(t) = \alpha * E(t-1) + (1-\alpha) * O(t)$, $0 <= \alpha <= 1$

Where E (t) and O (t) are the estimated and the observed load at time t, respectively. α reflects a tradeoff 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.

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

Table 1: Load Prediction Algorithms

Although seemingly satisfactory, this formula does not capture the rising trends of resource usage. For example, when we see a sequence of O (t) = 1 0 2 0 3 0 and 4 0 , 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 historic value and the observed one. To reflect the "acceleration", we take an innovative approach by setting to a negative value. When –1 <=α<0, the above formula can be transformed into the following:
E (t) = -|α|*E (t – 1) + (1+| α|)*0(t)
=O (t) +| α|*(O (t) – E (t – 1))

4 The Skewness Algorithm
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. We define the resource skewness of a server p as
Skewness (p) =

$$\sqrt{\sum^{n}_{i=1}\left(\frac{ri}{r}-1\right)2}$$

Where r is the average utilization of all resources for server p. In practice, not all types of resources are performance critical and hence e only need to consider bottleneck resource in the above calculation. By minimizing the skewness, we can combine difference 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 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:
Temperature (p) = $\sum_{r \pounds R} (r -- r_t)^2$
Where R is the set of overloaded resources in server p and rt is the hot threshold for resource r. The temperature of a hot spot reflects its degree of overload. If a server is not a hot spot, its temperature is zero.
Different types of resources can have different thresholds. For example, we can define the hot thresholds for CPU and memory resources to be 90% and 80%, respectively. Thus a server is a hot spot if either its CPU usage is above 90% or its memory usage is above 80%.

4.2. Hot Spot Mitigation
We sort the list of hot spots in the system in descending temperature (i.e., we handle the hottest one first). Our goal is to eliminate all hot spots if possible. For each server p, we first decide which of its VMs should be migrated away. We sort its list of VMs based on the resulting temperature of the server if that VM is migrated away. The server must not become a hot spot after accepting this VM. Among all such servers, we select one whose skewness can be reduced the most by accepting this VM. Note that this reduction can be negative which means we select the server whose skewness increases the least.

4.3. Green Computing
When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm. The challenge here is to reduce the number of active servers during low load without sacrificing performance either now or in the future. We need to avoid oscillation in the system.
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 warm threshold.

4.4. Consolidated Movements

The movements generated in each step above are not executed until all steps have finished. The list of movements is 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.

5. Simulations

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.. The raw traces are pre-processed into "Usher" format so that the simulator can read them. We collected the traces from a variety of sources:

We collected the traces from a variety of sources:

- Web Info Mall: 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.
- Real Course: the largest online distance learning system in China with servers distributed across 13 major cities
- Amazing Store: the largest P2P storage system in China.

The default parameters we use in the simulation are shown in Table 2. We used the FUSD load prediction algorithm with = 0 2 , = 0 7 , and W = 8. In a dynamic system, those parameters represent good knobs to tune the performance of the system adaptively. We choose the default parameter values based on empirical experience working with many Internet applications. In the future, we plan to explore using AI or control theoretic approach to find near optimal values automatically.

Symbol	Meaning	Value
h	hot threshold	0.9
c	cold threshold	0.25
w	warm threshold	0.65
g	green computing threshold	0.4
l	consolidation limit	0.05

Table 2: Parameters in Our Simulation

5.1. Effect of thresholds on APMs

We first evaluate the effect of the various thresholds used in our algorithm. We simulate a system with 100 PMs and 1000 VMs (selected randomly from the trace). We use random VM to PM mapping in the initial layout. The scheduler is invoked once per minute. The bottom part of Figure 4 shows the daily load variation in the system. The x-axis is the time of the day starting at 8am. The y-axis is overloaded with two meanings: the percentage of the load or the percentage of APMs (i.e., Active PMs) in the system. Recall that a PM is active (i.e., an APM) if it has at least one VM running. As can be seen from the figure, the CPU load demonstrates diurnal patterns which decrease substantially after midnight. The memory consumption is fairly stable over the time. The network utilization stays very low.

To examine the performance of our algorithm in more extreme situations, we also create a synthetic workload which mimics the shape of a sine function (only the positive part) and ranges from 15% to 95% with a 20% random fluctuation.

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 5. 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.

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.

5.3. Effect of Load Prediction

We compare the execution of our algorithm with and without load prediction in Figure 6. When load prediction is disabled, the algorithm simply uses the last observed load in its decision making. Figure 6 (a) shows that load prediction significantly reduces the average number of hot spots in the system during a decision run. Notably, prediction prevents over 46% hot spots in the simulation with 1400 VMs. This demonstrates its high effectiveness in preventing server overload proactively. Without prediction, the algorithm tries to consolidate a PM as soon as its load drops below the threshold. With prediction, the algorithm correctly foresees that the load of the PM will increase above the threshold shortly and hence takes no action. This leaves the PM in the "cold spot" state for a while. However, it also reduces placement churns by avoiding unnecessary migrations due to temporary load fluctuation.

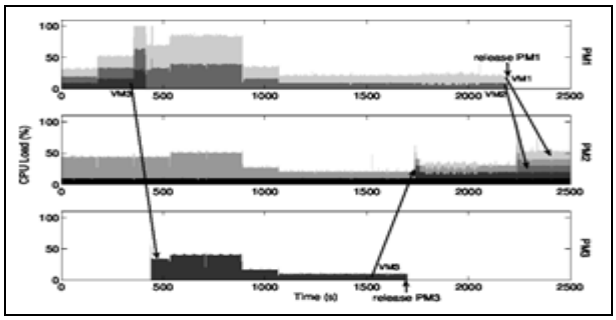


Fig.ure 7: Algorithm effectiveness

Consequently, the number of migrations in the system with load prediction is smaller than that without prediction as shown in Figure 6 (c). We can adjust the conservativeness of load prediction by tuning its parameters, but the current configuration largely serves our purpose (i.e., error on the side of caution). The only downside of having more cold spots in the system is that it may increase the number of APMs. This is investigated in Figure 6 (b) which shows that the average numbers of APMs remain essentially the same with or without load prediction (the difference is less than 1%). This is appealing because significant overload protection can be achieved without sacrificing resources efficiency. Figure 6 (c) compares the average number of migrations per VM in each decision with and without load prediction. It shows that each VM experiences 17% fewer migrations with load prediction.

6. Experiments

Our experiments are conducted using a group of 30 Dell PowerEdge blade servers with Intel E5620 CPU and 24GB of RAM. The servers run Xen-3.3 and Linux 2.6.18. We periodically read load statistics using the `xenstat` library (same as what `xent op` does). The servers are connected over a Gigabit ethernet to a group of four NFS storage servers where our VM Scheduler runs. We use the same default parameters as in the simulation.

6.1. Algorithm Effectiveness

We evaluate the effectiveness of our algorithm in overload mitigation and green computing. We start with a small scale experiment consisting of three PMs and five VMs so that we can present the results for all servers in figure 7. Different shades are used for each VM. All VMs are configured with 128 MB of RAM. An Apache server runs on each VM. We use httperf to invoke CPU intensive PHP scripts on the Apache server. This allows us to subject the VMs to different degrees of CPU load by adjusting the client request rates. The utilization of other resources are kept low.

We first increase the CPU load of the three VMs on PM1 to create an overload. Our algorithm resolves the overload by migrating VM3 to PM3. It reaches a stable state under high load around 420 seconds. Next we extend the scale of the experiment to 30 servers. We use the TPC-W benchmark for this experiment. TPC-W is an industry standard benchmark for e-commerce applications which simulates the browsing and buying behaviors of customers [13]. We deploy 8 VMs on each server at the beginning. Each VM is configured with one virtual CPU and two gigabyte memory.

6.2. Impact of Live Migration

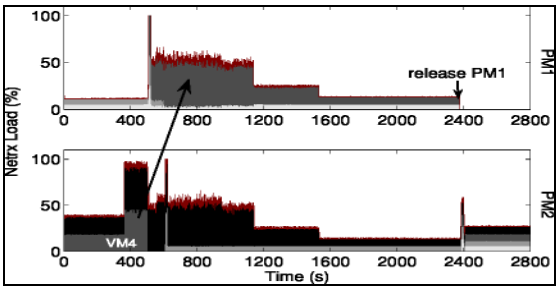
One concern about the use of VM live migration is its impact on application performance. Previous studies have found this impact to be small [5]. We investigate this impact in our own experiment. We extract the data on the 340 live migrations in our 30 server experiment above. Next we extend the scale of the experiment to a group of 72 VMs running over 8 PMs. Half of the VMs are CPU intensive, while the other half are memory intensive. Initially, we keep the load of all VMs low and deploy all CPU intensive VMs on PM4 and PM5 while all memory intensive VMs on PM6 and PM7. Then we increase the load on all VMs gradually to make the underlying PMs hot spots.

7. Related Work

7.1. Resource Allocation at the Application Level

Automatic scaling of Web applications was previously studied in [14] [15] for data center environments. In MUSE [14], each server has replicas of all web applications running in the system. The dispatch algorithm in a frontend L7-switch makes sure requests are reasonably served while minimizing the number of under-utilized servers. MapReduce [16] is another type of popular Cloud service where data locality is the key to its performance.

7.2. Resource Allocation by Live VM Migration



VM live migration is a widely used technique for dynamic resource allocation in a virtualized environment [8] [12] [20]. Our work also belongs to this category. Sandpiper combines multi-dimensional load information into a single Volume metric [8].

The HARMONY system applies virtualization technology across multiple resource layers [20]. 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.

7.3. 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.

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.

9. Acknowledgements

The authors would like to thank the anonymous reviewers for their invaluable feedback. This work was supported by the National Natural Science Foundation of China (Grant No. 61170056) and National Development and Reform Commission (Information Security 2011, CNGI2008-108).

10. References

1. N. Bila, E. d. Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, “Jettison: Efficient idle desktop consolidation with partial vm migration,” in Proc. of the ACM European conference on Computer systems (EuroSys’12), 2012.
2. L. Siegele, “Let it rise: A special report on corporate IT,” in The Economist, Oct. 2008.
3. “Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/.”
4. M. Nelson, B.-H. Lim, and G. Hutchins, “Fast transparent migration for virtual machines,” in Proc. of the USENIX Annual Technical Conference, 2005.
5. “TPC-W: Transaction processing performance council, http://www.tpc.org/tpcw/.”
6. T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, “Black-box and gray-box strategies for virtual machine migration,” in Proc. of the Symposium on Networked Systems Design and Implementation (NSDI’07), Apr. 2007.
7. C. A. Waldspurger, “Memory resource management in VMware ESX server,” in Proc. of the symposium on Operating systems design and implementation (OSDI’02), Aug. 2002.
8. T. Das, P. Padala, V. N. Padmanabhan, R. Ramjee, and K. G. Shin, “Litegreen: saving energy in networked desktops using virtualization,” in Proc. of the USENIX Annual Technical Conference, 2010.
9. 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 Computer systems (EuroSys’09), 2009.
10. N. Bobroff, A. Kochut, and K. Beaty, “Dynamic placement of virtual machines for managing sla violations,” in Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM’07), 2007