Volume 4, Issue 7, July 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Allocation of Resources Dynamically in Cloud Computing Using Virtual Machines

J. Ramesh*
M- Tech Scholar, Dept of CSE
SCVRIT College
Tadipatri, Anatapur (Dist)
India

M. Bindu Mahitha
Asst. Prof Dept of CSE
SCVRIT College
Tadipatri, Anantapur (Dist)
India

P. Amrutha
Asst. Prof Dept of CSE
SCVRIT College
Tadipatr, Anantapur (Dist)
India

Abstract*— *In cloud computing cloud users or customers use resources based on their needs. Cloud providers multiplex the resources and provide to users through virtualization technology. In this document, we introduce a system that uses virtualization technology to allocate data center resources dynamically based on application demands and best utilization of the number of servers in use thus supporting green computing. We introduce "skewness" the concept used to measure the unevenness in the multidimensional resource utilization of a server. By minimizing skewness, we can unite different types of workloads nicely and advance the overall consumption of server resources. We develop a set of heuristics that avoid overload in the system effectively while saving energy used.

Keywords*— *Cloud computing, green computing, resource management, virtualization

I. INTRODUCTION

One of the main advantage of the cloud computing is the lack of capital investment and scalability, due to this many firms prefer cloud computing. 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 technology. the different problem here we can learn is how can a cloud service provider(csp) best multiplex its virtual resources onto the physical hardware? this difficulty is significant because much of the touted gains in the cloud model come from such multiplexing. surveys have found that servers in several existing data centers are frequently rigorously underused due to over provisioning for the peak demand.

The cloud model is predicted to make this practice unnecessary by offering automatic scale up and down in response to load variation. By this it not only saves on electricity which contributes to a significant portion of the operational expenses in large data centers but also reduce the hardware cost, the popular trend cloud computing attempts to provide low-priced and easy access to computational resources. Compared to previous models, cloud computing focus on treating computational resources as measurable and billable utilities. Cloud computing hides the underlying hardware architecture from the clients' point of view. This abstraction saves them the costs of design, setup and maintenance of a data center to host their Application Environments (AE), whereas for cloud providers, the arrangement yield an opportunity to profit by hosting many AEs. This economy of scale provides benefits to both parties, but leaves the providers in a position where they must have an efficient and cost effective data center.

Infrastructure as a Service (IaaS) cloud computing focuses on providing a computing infrastructure that leverages system virtualization [1] to allow multiple Virtual Machines (VM) to be amalgamated on one Physical Machine (PM) where VMs often represent components of AEs. VMs are loosely coupled with the PM they are running on; because of this resultant, not only can a VM be started on any PM, but also, it can be migrated to other PMs in the data center. Migrations can either be accomplished by temporarily suspending the VM and transmitting it, or by means of a live migration in which the VM is only stopped for a split second [5].

With the current technologies, migrations can be performed on the order of seconds to minutes relying on the size and activity of the VM to be migrated and the network bandwidth between the two.

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. The cloud provider task is to make sure the underlying physical machines (PMs) have sufficient resources to satisfy their demands. VM live migration technology brings it to be possible to change the mapping between VMs and PMs even applications are running [2], [6]. But when the number of PMs used is minimized how the resource demands of VMs are met and how the mapping can be done adaptively. This is challenging when the resource needs of VMs are heterogeneous due to the diverse set of applications they run and vary with time as the workloads arise and reduce. The capacity of PMs can also be heterogeneous because multiple generations of hardware coexist in a data center.

We intend to attain two goals in our algorithm:

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

- Green computing. The needs of all VMs will be satisfied even though the number of PMs should be minimized. Idle PMs can be turned off to save energy.

There is an inbuilt tradeoff between the two goals in the face of changing resource needs of VMs. For overload avoidance, the utilization of PMs should be kept low to reduce the possibility of overload in case the resource needs of VMs increase afterward. On behalf of green computing, the consumption of PMs should be kept reasonably at 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 compose the following assistance:

- 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 consumption of a server. By minimizing skewness, we can get better the overall utilization of servers in the face of multidimensional resource constraints.
- We design a load prediction algorithm that can capture the future resource usages of applications accurately without looking in the interior of the VMs. The algorithm can capture the expanding tendency of resource usage patterns and help reduce the placement churn significantly.

II. SYSTEM OVERVIEW

Figure 1.1 exemplifies a cloud computing system comprising of service layers. The bottom layer is occupied by hardware layer. Above the hardware resources virtualization is located which provides high - level cloud services.

On the top of the virtualization layer IaaS layer is located. The IaaS layer allows an infrastructural environment with a set of standard self - service interfaces for controlling, management and provisioning of virtualized resources, allowing stakeholders for directly communicating by virtualized resource instances.

Above the layer of IaaS, Paas layer is located. These are used for building, designing, and deploying cloud applications with the absence of any complexity. Above the top of PaaS, SaaS layer is located, and delivering specific applications as a service to its clients Security is the main aim at each of the service delivery layers guaranteeing legitimate and authorized cloud services for delivering to the customers. User account management besides is also handled appropriately. The significant feature of the cloud computing model includes accounting, virtualization , programming APIs and managing etc.

In virtualization, several computers are associated to a single physical computer. This is termed as full virtualization since it permits an operating system to be installed in a separated virtual environment several simulating computers are called virtual machines. Scope of Virtualization is not restrained to the simulation of a whole computer machine.

There are numerous diverse virtualization technologies related to other hardware resource types, such as database virtualization, networking virtualization, storage virtualization and memory virtualization, each of which allows an abstraction layer between physical resources and virtualized resources.

A cloud computing system may utilize more than one virtualization technology; each one has custom interfaces specified for virtualized resource management. The cloud infrastructure layer (the IaaS) as specified in the architecture tower objective is at allowing a higher-level abstraction and a set of standard self-service interfaces for control, management and provisioning of virtualized resources hosted by various virtualization technologies.

Server Virtualization has two types:-

1)Full Virtualization

2)Para Virtualization

In full virtualization, the hypervisor function as a separating layer between the virtual server and the server hardware. It mediates accession of peripherals and hardware controllers of server's. Operating systems installed with the full server concept is not having awareness of the virtualized environment.

Full virtualization refers to the capability to run a program, most possibly an operating system, straightly above the virtual machine and without any changes, as however it were run on the raw hardware. To make this potential, virtual machine managers are necessary to provide a full simulation of the complete underlying hardware. The major gain of full virtualization is absolute isolation, which contributes to improved security, simplicity of simulation of dissimilar architectures, and coexistence of diverse systems on the same platform. Whereas it is a preferred objective for numerous virtualization solutions, full virtualization poses essential concerns associated to technical and performance implementation. A key challenge is the prevention of privileged information such as I/O instructions: Since they alter the status of the resources disclosed by the host, they have to be enclosed within the virtual machine manager. A straightforward resolution to attain full virtualization is to grant a virtual environment for all the instructions, thus posing some restrictions on performance.

A well-organized and successful achievement of full virtualization is achieved with a combination of software and hardware, not letting possibly risky instructions to be executed directly on the host. This is what is achieved during hardware-assisted virtualization. VMware and Microsoft are the best example of it.

Para-virtualization allows greater performance over full virtualization. Hence the guest operating system is customized so that it gets aware of the virtual environment. Para-virtualization is only appropriate for making use by open source OS.

This is a non-transparent virtualization solution that provides implementing thin virtual machine managers. Para-virtualization techniques render a software interface to the virtual machine that is modified to some extent from the host

and, as a result, guests need to be altered. The intention of para-virtualization is allowing the capability to demand the execution of performance-critical processes directly on the host, consequently restricting performance losses that would or else be occurred in managed execution. This grants a simpler implementation of virtual machine managers that makes simply transfer the implementation of these operations, which were tough to virtualize, directly to the host. To take benefit of this opportunity, guest operating systems have to be changed and explicitly carried by remapping the performance-critical functions during the virtual machine software interface. This is potential when the source code of the operating system is accessible, and this is the cause that para-virtualization was generally explored in the academic and open source environment. Where this technique was primarily employed in the IBM VM operating system families, the term para-virtualization was bring out in literature in the Denali project at the University of Washington. This procedure has been effectively used by Xen for allowing virtualization solutions for Linux-based operating systems specially carried to run on Xen hypervisors. Operating systems that can't be carried can still obtain benefit of para-virtualization with the use of ad hoc device drivers that remap the execution of serious instructions to the para-virtualization APIs bring out by the hypervisor. Xen allows this solution for implementing Windows-based operating systems on x86 architectures. Other solutions with para-virtualization include parallels, VMWare, and some solutions for embedded and real-time environments such as Wind River, XtratuM and TRANGO. Citrix Xen is the finest instance of it.

The architecture of the system is presented in Fig. 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 DNS, Map/ Reduce, Remote Desktop, Web Server, Mail, etc. We assume all PMs share a backend storage.

The multiplexing of VMs to PMs is managed by means of the Usher framework [10]. The main logic of our system is implemented as a set of plug-ins to Usher. Every node carries an Usher local node manager (LNM) on domain 0 which collects the usage information of resources for each VM on that node. The network and CPU usage can be computed by monitoring the scheduling events in Xen.

The memory usage within a VM, however, is not visible to the hypervisor. One approach is to deduce memory shortage of a VM by observing its swap activities [8].

Regrettably, the guest OS is required to install a separate swap partition. Furthermore, it may be too behind schedule to adjust the memory allocation by the time swapping occurs. As an alternative we implemented a functioning set prober (WS Prober) on each hypervisor to estimate the working set sizes of VMs operating on it. We make use of the random page sampling technique as in the VMware ESX Server [9].

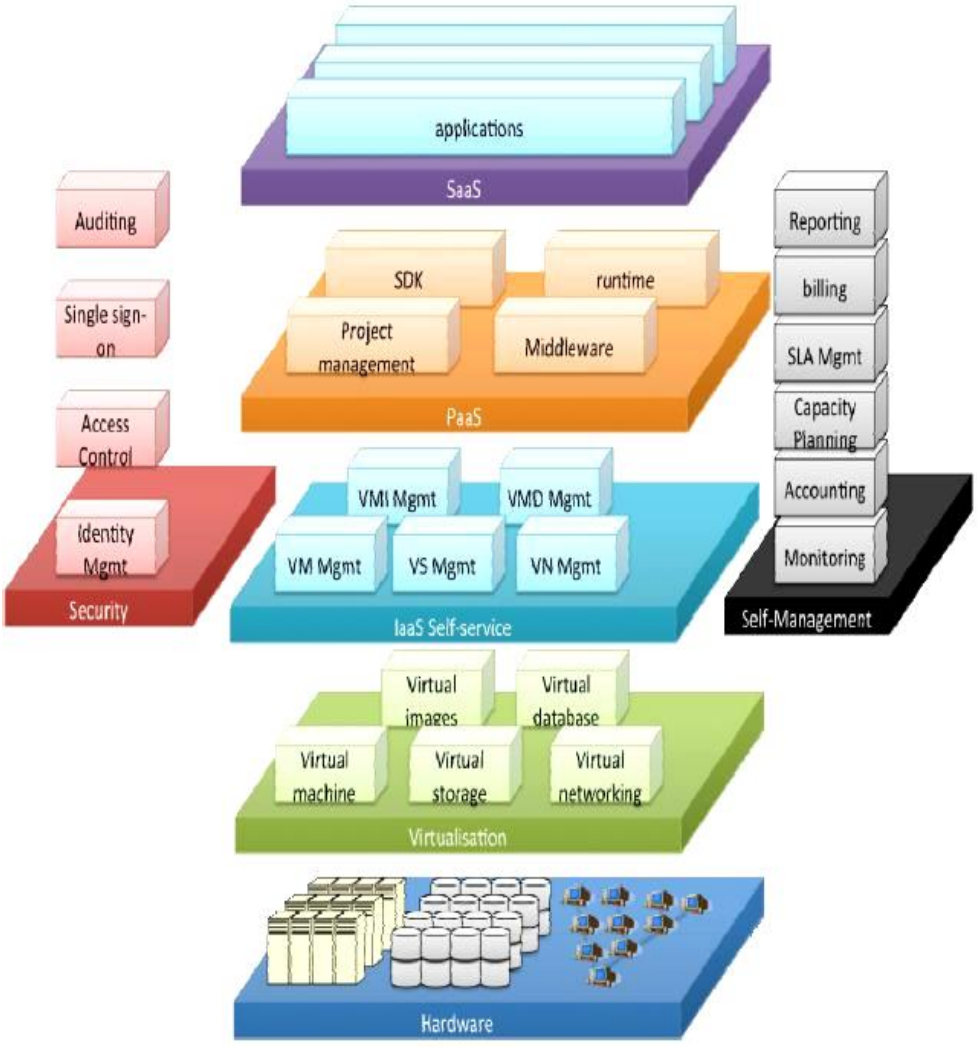

Fig 1.1 Cloud Computing System

The information collected at each PM are forwarded to the Usher central controller (Usher CTRL) where our VM scheduler runs. The VM Scheduler is invoked from time to time and receives from the LNM the load history and the capacity of PMs, the resource demand history of VMs, , and the present layout of VMs lying on PMs.

The scheduler has numerous components. The predictor predicts the future resource demands of VMs and the future load of PMs grounded on past statistics.

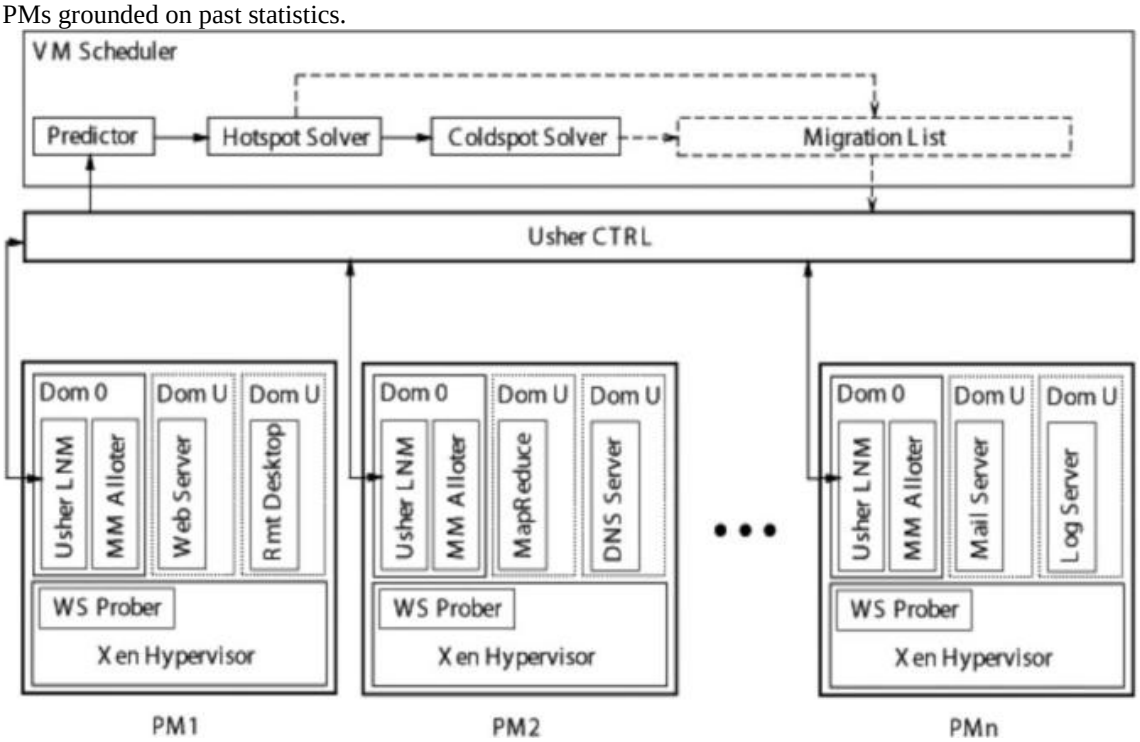

Fig. 1. System architecture.

We calculate the load of a PM by combining the resource utilization of its VMs. The LNM at every node primarily attempts to meet the new demands locally by adjusting the resource allocation of VMs sharing the similar VMM. Xen can modify the CPU allotment amongst the VMs by adjusting their weights in its CPU scheduler. The MM Allotter on domain 0 from every node is responsible for adjusting the local memory allocation.

The hot spot solver in our VM Scheduler observes if the resource utilization of any PM is above the hot threshold (i.e., a hot spot). If this is the case, some VMs running on them will be migrated away to diminish their load. The cold spot solver checks if the average consumption of actively used PMs (APMs) is below the green computing threshold. If this happens, a few of those PMs would possibly be put off to save energy. It identifies the set of PMs whose usage is below the cold threshold (i.e., cold spots) and then attempts to migrate away every one of their VMs. After that it collects a migration list of VMs and passes it to the Usher CTRL for execution.

III. PREDICTING FUTURE RESOURCE NEEDS

We should predict the future resource needs of vms. as said previously, our focus is on internet apps. one key is to come across the interior of vm for application level data, e.g., via analysing logs of awaiting requests. doing so requires alteration of the vm which might not all the time be possible. rather, we compose our calculation founded on the past external behaviours of vms. our preliminary effort was to compute an exponentially weighted moving average (ewma) using a tcp-like scheme

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

Where E(t) and O(t) are the estimated and the observed load at time t, respectively. α reflects a trade-off between stability and responsiveness.

We use the EWMA formula to forecast the CPU load. Detailed investigation on prediction algorithms are left as future work.

IV. THE SKEWNESS ALGORITHM

We bring in the concept of skewness to compute the unevenness in the utilization of numerous resources on a server. Let n be the quantity of resources we regard and r be the utilization of the i^{th} resource. We define the resource skewness of a server p as

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

Where $\bar{r}$ is the average utilization of all resources or server p. In observation, not every type of resources are performance critical and hence we only need to consider bottleneck resources in the above computation By reducing the skewness, we may unite different types of workloads properly and get better the overall utilization of server resources.

A. Hot and Cold Spots

Our algorithm executes from time to time to calculate the resource allocation status based on the predicted future resource needs of VMs. We identify a server as a hot spot when the utilization of any of its resources is exceeding a hot threshold. This indicates that the server is stuffed and hence some VMs running on it ought to be migrated away. We describe the temperature of a hot spot p as the square sum of its resource utilization beyond the hot threshold:

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

Where R is the set of congested resources in server p and r is the hot threshold for resource r. (Notice that simply jammed resources are considered in the computation.) The measure of a hot spot reflects its degree of excess. If the condition of the server is not a hot spot, then its temperature is zero.

We define a server as a cold spot if the utilizations of all its resources are beneath a cold threshold. This shows that the server is mostly idle or inactive and a potential candidate to turn off to keep energy. Nevertheless, we do so only whenever the average resource utilization of all actively used servers (i.e., APMs) in the system is underneath a green computing threshold. A server is actively utilized whenever it has at the least one VM running. Otherwise, it is static. Eventually, we describe the warm threshold to be a level of resource utilization that is sufficiently high to justify having the server running but not as high as to risk becoming a hot spot in the face of temporary fluctuation of application resource demands.

Different types of resources can have diverse thresholds. For instance, we can describe the hot thresholds for CPU and memory resources to be 90 and 80 percent, correspondingly. Thus a server or host is a hot spot if either its CPU usage is higher than 90 percent or its memory usage is more than 80 percent.

B. 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 get rid of all hot spots if possible. Or else, keep their temperature as low as possible. For each server p, we first come to a decision which of its VMs should be migrated away. We sort its list of VMs based on the resultant temperature of the server if that VM is migrated away. We aim to migrate away the VM that can lessen the server's temperature the for the most part. Whenever tie occurs we choose the VM whose exclusion can reduce the skewness of the server the majority. For each VM in the list, we see if we can find a destination server to adopt it. The server mustn't turn into a hot spot after accommodating this VM. Among the entire such servers, we pick one whose skewness can be reduced the most by tolerating this VM. Note that this reduction can be negative which means we select the server whose skewness increases the least. If a destination server is established, we document the migration of the VM to that server and renew the predicted load of related servers. Or else, we move onto the next VM in the list and try to find a destination server intended for it. As long as we can come across a destination server for whatever of its VMs, we conceive this run of the algorithm a success and then move onto the after that hot spot. Note that every run of the algorithm migrates away at most one VM from the overloaded server. This does not inevitably eliminate the hot spot, but at the least trims its temperature. When it continues a hot spot after that conclusion run, the algorithm may replicate this process. It is likely to figure the algorithm so that it can migrate away multiple VMs through each run. But this can put in additional load on the related servers during a period when they are already congested. We come to a decision to make use of this more conservative approach and leave the system some time to react before starting additional migrations.

C. Green Computing

When the resource utilization of active servers is too short, a number of them can be put off to save energy. Such method is addressed in our green computing algorithm. The dispute here is to lessen the number of active servers during low load without giving performance either now or in the future. We need to evade fluctuation in the system.

Our green computing algorithm is invoked when the average utilizations of all resources on active servers are lower than the green computing threshold. We arrange the record of cold spots in the system based on the mounting order of their storage volume. Since we need to migrate away all its VMs before we can shut down an underutilized server, we define the memory size of a cold spot as the aggregate memory size of all VMs running on it. Remember that our model assumes all VMs connect to distributed back-end storage. Consequently, the price 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 someplace else. For each VM on p, we attempt to find a destination server to contain it. The resource usage of the server after accepting the VM must be below the warm threshold. While we can save energy by combining underutilized servers, exaggerating it may create hot spots in the future. The warm threshold is designed to avoid that. If multiple servers satisfy that standard, we choose one that is not a current cold spot. This is since growing load on a cold spot reduces the possibility that it can be wiped out. However, we will accept a cold spot as the destination server if needed. All things being equal, we select a destination server whose skewness can be reduced the most by allowing this VM. When we come across destination servers for all VMs on a cold

spot, we have to record the sequence of migrations and update the predicted load of related servers. Or else, we don't migrate none of its VMs. The record of cold spots is too updated since some of them may no longer be cold due to the proposed VM migrations in the above process.

The above consolidation adds additional load onto the related servers. This is not as serious a trouble as in the hot spot mitigation case since green computing is initiated only when the load in the system is low. However, we want to bound the extra load due to server integration. We limit the numeral 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 termed as the consolidation limit.

Note that we eliminate cold spots in the system only when the average load of all active servers (APMs) is below the green computing threshold. If not else, we leave those cold spots there as potential destination machines for upcoming unload. This is reliable with our viewpoint that green computing should be carried conservatively.

D. *Consolidated Movements*

The movements generated in each step above are not carried out until all steps have done. The list of activities is then consolidated so that each VM is moved at most once to its concluding destination. For example, hot spot mitigation might state a VM to move from A's PM to B's PM, whereas green computing states it to move from B's PM to C's PM. In the genuine execution, the VM is proceeding from A to C directly.

V. ALGORITHM EFFECTIVENESS

We assess the efficacy of our algorithm in overload mitigation and green computing. We initiate with a small level test lie of three PMs and five VMs so that we can present the results for all servers. Different shades are employed for every VM. each and every VMs are assembled with 128 MB of RAM. An Apache server execute on each and every VM.

We employ httperf to invoke CPU concentrated PHP scripts on the Apache server. This lets us to subject the VMs to dissimilar degrees of CPU load by adjusting the client request rates. The consumption of other resources is kept low. We first raise the CPU load of the three VMs on PM to create an excess. Our algorithm resolves the overload by migrating VM_3 to PM_3. It reaches a stable state under high load approximately 420 seconds. About 890 seconds, we reduce the CPU load of all VMs progressively. Since the FUSD prediction algorithm is conservative when the load reduces, it carries out a while earlier than green computing takes effect. About 1,700 seconds, VM_3 is migrated from PM_3 to PM_2 so that PM_3 can be put into the stick by mode. In the region of 2,200 seconds, the two VMs on PM1 are migrated to PM2 so that PM1 can be free as well. As the load becomes changeable, our algorithm will repeat the above process: spread over or combine the VMs as needed.

Next we extend the scale of the experiment to 30 hosts. We make use of the TPC-W benchmark for this test. TPC-W is an industry criterion level for e-commerce to applications which assumes the browsing and buying behaviors of customers [13]. We deploy 8 VMs on each server at the beginning. Every VM is assembled with one virtual CPU and two gigabyte storage of memory. Self-ballooning is enabled to permit the hypervisor to retrieve idle memory. Every VM runs the TPC-W in the server side benchmark equivalent to various types of the workloads: workloads, hybrid, shopping browsing, etc. Our algorithm is invoked every 10 minutes.

To compute the energy saving, we considered the electric power consumption under various TPC-W workloads with the built-in watt-meter in our blade systems. We find that an idle blade server takes about 130 Watts and a fully utilized server takes approximately 205 Watts. In the previous experiment, a server on mean spends 48 percent of the time in standby mode due to green computing. This translates into approximately 62 Watts power-saving for each server or else 1,860 Watts intended for the cluster of 30 servers utilized in the experiment.

VI. RELATED WORK

A. *Resource Allocation at the Application Level*

Automatic scaling of Web applications was previously studied in [13] and [19] for data center environments. In MUSE [13], each server has replicas of all web applications running in the system. The discharge algorithm in a forepart L7-switch makes sure requests are logically served while minimizing the number of underused servers. Work [19] uses network flow algorithms to allocate the load of an application among its operating instances. On behalf of connection oriented Internet services like Windows Live Messenger, work [7] presents an incorporated approach for load dispatching and server provisioning. All works other than this do not employ virtual machines and require the applications be structured in a multitier architecture with load balancing provided through an front-end dispatcher. As a counterpart, our effort aims Amazon EC2-style environment where it places no constraint on what and how applications are constructed within the VMs. A VM is handled as a black box. Resource administration is done only at the granularity of whole VMs.

Map Reduce [15] is another type of admired Cloud service where data locality is the key to its performance. Quincy [16] adopts min-cost flow model in task scheduling to maximize data locality while keeping fairness among different jobs. The "Delay Scheduling" algorithm [17] trades execution time for data locality. Work [18] assigns dynamic priorities to jobs and users to ease resource allocation.

B. Resource Allocation by Live VM Migration

VM live migration is a extensively used technique for dynamic resource allocation in a virtualized environment [8], [11], [16]. Our work also belongs to this class. Sandpiper aggregates multidimensional load information into a single

Volume metric [8]. It sorts the list of PMs based on their volumes and the VMs in each PM in their volume-to-size ratio (VSR). This regrettably abstracts away critical information needed when making the migration judgment. It then considers the PMs and the VMs in the pre-sorted order.

The HARMONY system employs virtualization technology across multiple resource layers [14]. It uses VM and data migration to mitigate hot spots not just upon the servers, but also on storage nodes and the network devices as well. It inaugurates the Extended Vector Product (EVP) as an indicator of unevenness in resource utilization. Their load balancing algorithm is a variation of the Toyoda method [20] for multidimensional knapsack problem. Not like our system, their system doesn't support green computing and load prediction is left as future work.

Dynamic placement of virtual servers to minimize SLA model it as a bin packing problem and use the well-known first-fit approximation algorithm to compute the VM to PM layout from time to time. That algorithm, however, is intended mostly for offline utilization. It is possibly to acquire a great quantity of migrations when applied in online environment where the resource needs of VMs change dynamically.

C. Green Computing

Many efforts have been made to restrain energy utilization in information centers. Hardware related approaches contain novel thermal design for lower cooling power, or assuming power relative and low-power hardware. Work [21] uses dynamic voltage and frequency scaling (DVFS) to adjust CPU power according to its load. PowerNap [22] appeals to new hardware technologies such as solid state disk (SSD) and Self-Refresh DRAM to employ rapid transition(less than 1ms) between full operation and low power state, so that it can "take a nap" in short inactive intervals. As a server befallen to sleep, Somniloquy [23] notifies an embedded system residing on a special designed NIC to delegate the main operating system. It makes the false impression that the server is always active.

Our work belongs to the category of pure-software low-cost solvents [7], [11], [13], [24]. Parallel to Somniloquy [23], Sleep Server initiates virtual machines on a devoted server like delegate, as a replacement for of depending on a special NIC. LiteGreen [24] does not use a delegate. As an alternative it migrate the desktop OS away so that the desktop be able to nap. It necessitates that the desktop is virtualized with shared storage. Jettison invents "partial VM migration," a distinction of live VM migration, which only migrate away essential effective set while leaving rarely used data behind.

VII. CONCLUSIONS

We have presented the design, execution, and assessment of a resource administration system for cloud computing services. Our system aggregates resources of virtual to physical adaptively based on the varying requirement. We utilize the skewness metric to unite VMs with different resource characteristics properly so that the capabilities of servers are well used. Our algorithm attains both overload avoidance and green computing for systems with poly resource constraints.

REFERENCES

[1] M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," technical report, Univ. of California, Berkeley, Feb. 2009.

[2] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," Proc. Symp. Networked Systems Design and Implementation (NSDI '05), May 2005.

[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," Proc. ACM Symp. Operating Systems Principles (SOSP '03), Oct. 2003.

[4] "Amazon elastic compute cloud (Amazon EC2)," http://awsamazon.com/ec2/, 2012.

[5] L. Siegele, "Let It Rise: A Special Report on Corporate IT," The Economist, vol. 389, pp. 3-16, Oct. 2008.

[6] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," Proc. USENIX Ann. Technical Conf., 2005.

[7] 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," Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI '08), Apr. 2008.

[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-Box and Gray-Box Strategies for Virtual Machine Migration," Proc. Symp. Networked Systems Design and Implementation (NSDI '07), Apr. 2007.

[9] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," Proc. Symp. Operating Systems Design and Implementation (OSDI '02), Aug. 2002.

[10] M. McNett, D. Gupta, A. Vahdat, and G.M. Voelker, "Usher: An Extensible Framework for Managing Clusters of Virtual Machines," Proc. Large Installation System Administration Conf. (LISA '07), Nov. 2007.

[11] "TPC-W: Transaction Processing Performance Council," http:// www.tpc.org/tpcw/, 2012.

[12] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers," Proc. ACM Symp. Operating System Principles (SOSP '01), Oct. 2001.

[13] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A Scalable Application Placement Controller for Enterprise Data Centers," Proc. Int'l World Wide Web Conf. (WWW '07), May 2007.

[14] Y. Toyoda, "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems," Management Science, vol. 21, pp. 1417-1427, Aug. 1975.

[15] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair Scheduling for Distributed Computing Clusters," Proc. ACM Symp. Operating System Principles (SOSP '09), Oct. 2009.

[16] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," Proc. European Conf. Computer Systems (EuroSys '10), 2010.

[17] T. Sandholm and K. Lai, "Mapreduce Optimization Using Regulated Dynamic Prioritization," Proc. Int'l Joint Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '09), 2009.

[18] A. Singh, M. Korupolu, and D. Mohapatra, "Server-Storage Virtualization: Integration and Load Balancing in Data Centers," Proc. ACM/IEEE Conf. Supercomputing, 2008.

[19] M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," Proc. Symp. Operating Systems Design and Implementation (OSDI '08), 2008.

[20] R. Nathuji and K. Schwan, "Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems," Proc. ACM SIGOPS Symp. Operating Systems Principles (SOSP '07), 2007.

[21] D. Meisner, B.T. Gold, and T.F. Wenisch, "Powernap: Eliminating Server Idle Power," Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '09), 2009.

[22] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: Augmenting Network Interfaces to Reduce Pc Energy Usage," Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI '09), 2009.

[23] T. Das, P. Padala, V.N. Padmanabhan, R. Ramjee, and K.G. Shin, "Litegreen: Saving Energy in Networked Desktops Using Virtualization," Proc. USENIX Ann. Technical Conf., 2010.

[24] Y. Agarwal, S. Savage, and R. Gupta, "Sleepserver: A SoftwareOnly Approach for Reducing the Energy Consumption of PCS within Enterprise Environments," Proc. USENIX Ann. Technical Conf., 2010.

[25] N. Bila, E.d. Lara, K. Joshi, H.A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, "Jettison: Efficient Idle Desktop Consolida- tion with Partial VM Migration," Proc. ACM European Conf. Computer Systems (EuroSys '12), 2012.

