Review Article   **Open Access**

Allocation of Resource Dynamically in Cloud Computing Environment Using Virtual Machines

Ligade Sunil Subhash[1]* and Dr. K.P. Thooyamani[2]

[1]Department of Computer Science and Engineering, Bharath Institute of Higher Education and Research, Chennai, India
[2]School of Computer Science, Bharath Institute of Higher Education and Research, Chennai, India

Abstract

Cloud computing allows 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 research paper, we present a system that uses virtualization technology to allocate data centre 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 multidimensional 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: Dynamic allocation; Skewness; Resource multiplexing; Virtual machines; Load balancing; Cloud computing; Resource balance; Green computing

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? 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 centres 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 centres [1-9].

System Overview

The architecture of the system is presented in the Figure 1. Each PM runs the Xen hypervisor (VMM) which supports a privileged domain 0 and one or more domain U. 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. 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
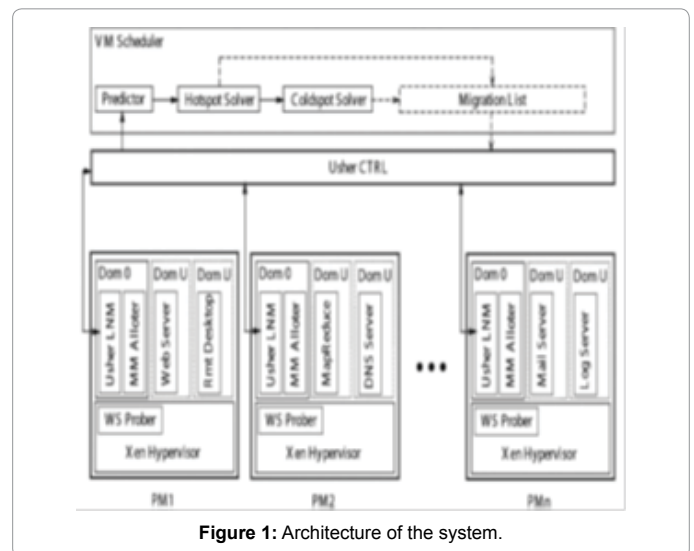

Figure 1: Architecture of the system.

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 Allotter on domain 0 of each

*Corresponding author: Ligade Sunil Subhash, Ph.D. Scholar, Department of Computer Science and Engineering, Bharath Institute of Higher Education and Research, Selaiyur, Chennai 600073, India, Tel: 044 2229 0742; E-mail: sunilligade@gmail.com

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.

Existing System

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. Users with the Amazon EC2 service, 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. The capacity of PMs can also be heterogeneous because multiple generations of hardware coexist in a data centre.

Proposed Work

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:

1. We develop a resource allocation system that can avoid overload in the system effectively while minimizing the number of servers used.

2. 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 multidimensional resource constraints.

3. 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 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:

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

Where, r is the average utilization of all resources for server p. In practice, not all types of resources are performance critical and hence we only need to consider bottleneck resources in the above calculation. By minimizing the *skewness*, we can combine different types of workloads nicely and improve the overall utilization of server resources.

Resource Balance

Recall that the goal of the skewness algorithm is to mix workloads with different resource requirements together so that the overall utilization of server capacity is improved. In this experiment we see how our algorithm handles a mix of CPU, memory, and network intensive workloads. We vary the CPU load as before. We inject the network load by sending the VMs a series of network packets. The memory intensive

applications are created by allocating memory on demand. Again we start with a small scale experiment consisting of two PMs and four VMs so that we can present the results for all servers in following Resource balance for mixed workloads (Figure 2).

The two rows represent the two PMs. The two columns represent the CPU and network dimensions, respectively. The memory consumption is kept low for this experiment. Initially, the two VMs on PM1 are CPU intensive while the two VMs on PM2 are network intensive. We increase the load of their bottleneck resources gradually. Around 500 seconds, VM4 is migrated from PM2 to PM1 due to the network overload in PM2. Then around 600 seconds, VM1 is migrated from PM1 to PM2 due to the CPU overload in PM1. Now the system reaches a stable state with a balanced resource utilization for both PMs – each with a CPU intensive VM and a network intensive VM. Later we decrease the load of all VMs gradually so that both PMs become cold spots. We can see that the two VMs on PM1 are consolidated to PM2 by green computing. 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.

Following Figure 3 for VM distribution over time shows how the algorithm spreads the VMs to other PMs over time. As we can see from the figure, the algorithm balances the two types of VMs appropriately. The figure also shows that the load across the set of PMs becomes well balanced as we increase the load.

Related Work

Resource allocation at the application level

Automatic scaling of Web applications was previously studied in for data centre environments. In MUSE, each server has replicas of all web applications running in the system. The dispatch algorithm in a front end L7-switch makes sure requests are reasonably served while minimizing the number of under-utilized servers. Work uses network flow algorithms to allocate the load of an application among its running instances. For connection oriented Internet services like Windows Live Messenger, work presents an integrated approach for load dispatching and server provisioning. All works above do not use virtual machines and require the applications be structured in a multi-tier architecture with load balancing provided through a front-end dispatcher. In contrast, our work targets Amazon EC2-style environment where it places no restriction on what and how applications are constructed inside the VMs. A VM is treated like a blackbox. Resource management is done only at the granularity of whole VMs. MapReduce is another type of popular Cloud service where data locality is the key to its performance. The "Delay Scheduling" algorithm trades execution time for data locality . Work assign dynamic priorities to jobs and users to facilitate resource allocation.

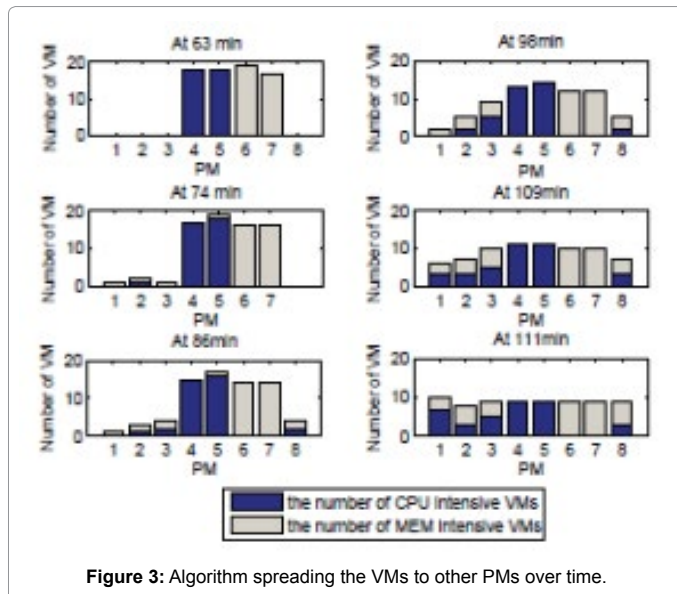**Figure 2:** Resource balance for mixed workloads.
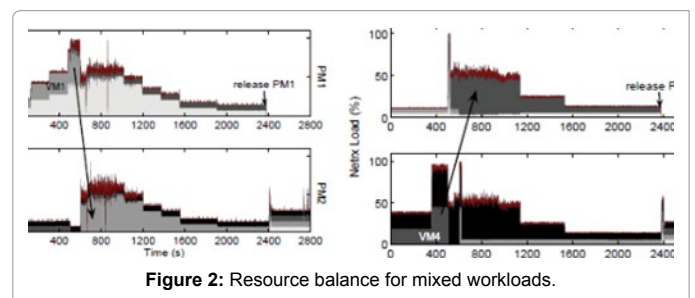
Figure 3: Algorithm spreading the VMs to other PMs over time.

Resource allocation by 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 analysed in Section 5 of the 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 analyse the phenomenon that *VectorDot* behaves differently compared with our work and point out the reason why our algorithm can utilize residual resources better. 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.

Green computing

Many efforts have been made to curtail energy consumption in data centres. Hardware based approaches include novel thermal design for lower cooling power, or adopting power-proportional and low-power hardware. 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 above Section. 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, SleepServer 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. Jettison invents "partial VM migration", a variance of live VM migration, which only migrates away necessary working set while leaving infrequently used data behind.

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. Chen G, Wenbo H, Liu J, Nath S, Rigas L, et al. (2008) Energy-aware server provisioning and load dispatching for connection-intensive internet services. NSDI'08 Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, pp: 337-350.

2. Padala P, Hou KY, Shin KG, Zhu X, Uysal M, et al. (2009) Automated control of multiple virtualized resources. EuroSys '09 Proceedings of the 4th ACM European conference on Computer systems, pp: 13-26.

3. Bobroff N, Kochut A, Beaty K (2007) Dynamic placement of virtual machines for managing SLA violations. Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'07).

4. http://www.tpc.org/tpcw/

5. Chase JS, Anderson DC, Thakar PN, Vahdat AM, Doyle RP (2001) Managing energy and server resources in hosting centers. SOSP '01 Proceedings of the Eighteenth ACM symposium on Operating systems principles 35: 103-116.

6. 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, WWW, pp: 331-340.

7. Zaharia M, Konwinski A, Joseph AD, Katz RH, Stoica I (2008) Improving MapReduce performance in heterogeneous environments. OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation.

8. Agarwal Y, Hodges S, Chandra R, Scott J, Bahl P, et al. (2009) Somniloquy: Augmenting network interfaces to reduce pc energy usage NSDI'09 Proceedings of the 6th USENIX symposium on Networked systems design and implementation, pp: 365-380.

9. Bila N, Lara Ed, Joshi K, Lagar-Cavilla HA, Hiltunen M, et al. (2012) Jettison: Efficient idle desktop consolidation with partial VM migration. EuroSys '12 Proceedings of the 7th ACM European conference on Computer Systems, pp: 211-224