DYNAMIC RESOURCE ALLOCATION USING VIRTUAL MACHINES FOR CLOUD COMPUTING

ATHULYA RAJ, L. MUBARAALI, M.E

PG scholar,Maharaja Engineering College, Avinashi, India
Email:athulyaraj88@yahoo.com
Associate Professor, Maharaja Engineering College, Avinashi, India
Email:muby.star@gmail.com

Abstract-Cloud computing is the long dreamed vision of computing as a utility, where users can remotely store their data into the cloud so as to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources. A virtual machine (VM) is a software implementation of a computing environment in which an operating system (OS). Virtual machines can provide numerous advantages over the installation of OS's and software directly on physical hardware. 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. Introduce the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server. By minimizing skewness, combine different types of workloads nicely and improve the overall utilization of server resources. Here 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.

I. INTRODUCTION

Cloud computing is the long dreamed vision of computing as a utility, where users can remotely store their data into the cloud so as to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources. By data outsourcing, users can be relieved from the burden of local data storage and maintenance. However, the fact that users no longer have physical possession of the possibly large size of outsourced data makes the data integrity protection in Cloud Computing a very challenging and potentially formidable task, especially for users with constrained computing resources and capabilities.

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a metered service over a network. Cloud computing provides computation, software applications, data access, and storage resources without requiring cloud users to know the location and other details of the computing infrastructure. End users access cloud based applications through a web browser or a light weight desktop or mobile app while the business software and data are stored on servers at a remote location. Cloud application providers strive to give the same or better service and performance as if the software programs were installed locally on end-user computers.

The goals are,

Overload avoidance: The capacity of a PM should besufficient to satisfy the resource needs of all VMsrunning 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.

II. RELATED WORK

A. Resource allocation at the application level

Automatic scaling of Web applications was previously studied in [10] and [11] for data center environments. In MUSE [4], 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 underutilized servers. Work [5] 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 [10] presents an integrated approach for load dispatching and server provisioning.

B. 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. Work [5] uses dynamic voltage and frequency scaling (DVFS) to adjust CPU power according to its load. We do not use DVFS for green computing. PowerNap [7] 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 [4] 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.VM live migration is a widely used technique for dynamic resource allocation in a virtualized environment [8], [9],[10].The work also belongs to this category. Sandpiper combines 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 unfortunately abstracts away critical information needed when making the migration decision. It then considers the PMs and the VMs in the presorted order.

We give a concrete example in Section 1 of the supplementary file, which is available online, 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 in Section 5 of the supplementary file, which is available online, 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.

3. SYSTEM ASSUMPTIONS

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 Web server, remote desktop, DNS, Mail, Map/ Reduce, etc. We assume all PMs share a backend storage.

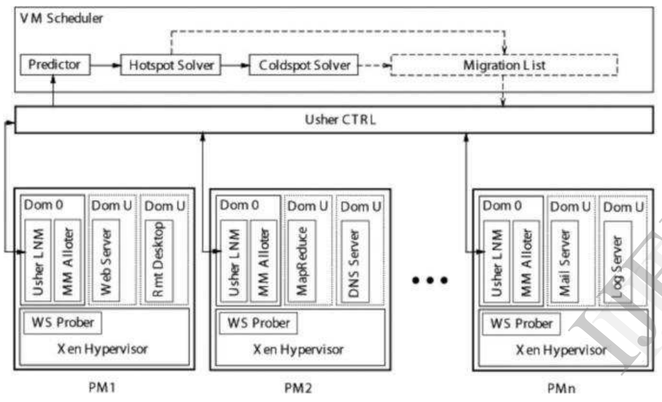


Fig. 1. System architecture.

The multiplexing of VMs to PMs is managed using theUsher framework [7]. The main logic of our system is implemented as a set of plug-ins to Usher. Each node runs an Usher localnode 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 [9].

The VM Scheduler is invoked periodicallyand receives from the LNM the resource demand history ofVMs, the capacity and the load history of PMs, and thecurrent layout of VMs on PMs.The scheduler has several components. The predictorpredicts the future resource demands of VMs and thefuture load of PMs based on past statistics. We compute theload of a PM by aggregating the resource usage of its VMs.The details of the load prediction algorithm will bedescribed in the next section. The LNM at each node firstattempts to satisfy the new demands locally by adjusting the resource allocation of VMs sharing the same VMM.

The hot spot solver in our VM Scheduler detects if theresource utilization of any PM is above the hot threshold(i.e., a hot spot). If so, some VMs running on them will bemigrated away to reduce their load. The cold spot solverchecks if the average utilization of actively used PMs. (APMs) is below the green computing threshold. If so, some ofthose 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.

One solution is to look inside a VMfor application level statistics,e.g., by parsing logs of pending requests. Doing so requiresmodification of the VM which may not always be possibleInstead, 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

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

where E(t) and O(t) are the estimated and the observed load at time t, respectively. _ reflects a tradeoff between stability and responsiveness. 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 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 _ to a negative value. When -1 <=α< 0, the above formula can be transformed into the following:

$$\begin{aligned} E(t) &= -|\alpha| * E(t-1) + (1+|\alpha|) * O(t) \\ &= O(t) + |\alpha| * (O(t) - E(t-1)), \end{aligned}$$

4. THE SKEWNESS ALGORITHM

The concept of skewness to quantify the nevenness 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 ith resource. 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.

Let n be the number of resources we consider and ri be the utilization of the ith resource. We define the resourceskewness of a server p as

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

where r is the average utilization of all resources forserver p. In practice, not all types of resources areperformance critical and hence we only need to considerbottleneck resources in the above calculation. By minimizingthe skewness, we can combine different types ofworkloads nicely and improve the overall utilization of server resources.

4.1 HOT AND COLD SPOT

The algorithm executes periodically to evaluate the resource allocation status based on the predicted futureresource demands of VMs. To define a server as a hot spotif the utilization of any of its resources is above a hotthreshold. This indicates that the server is overloaded andhence some VMs running on it should be migrated away. The temperature of a hot spot p as the square sumof its resource utilization beyond the hot threshold:

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

where R is the set of overloaded resources in server p and rt is the hot threshold for resource r. (Note that only overloaded resources are considered in the calculation.)The temperature of a hot spot reflects its degree of overload.If a server is not a hot spot, its temperature is zero. Different types of resources can have different thresholds.For example, we can define the hot thresholds for CPUand memory resources to be 90 and 80 percent, respectively.Thus a server is a hot spot if either its CPU usage is above90 percent or its memory usage is above 80 percent.

4.2 HOT SPOT MITIGATION

The list of hot spots in the system in descending temperature the goal is to eliminate all hot spots if possible. Otherwise, keep their temperature as low as 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. We aim to migrate away the VM that can reduce the server's temperature the most. In case of ties, we select the VM whose removal can reduce the skewness of the server the most. For each VM in the list, we see if we can find a destination server to accommodate it.

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. If a destination server is found, we record the migration of the VM to that server and update the predicted load of related servers. Otherwise, we move onto the next VM in the list and try to find a destination server for it. As long as we can find a destination server for any of its VMs, we consider this run of the algorithm a success and then move onto the next hot spot. Note that each run of the algorithm migrates away at most one VM from the overloaded server. This does not necessarily eliminate the hot spot, but at least reduces its temperature. If it remains a hot spot in the next decision run, the algorithm will repeat this process. It is possible to design the algorithm so that it can migrate away multiple VMs during each run. But this can add more load on the related servers during a period when they are already overloaded. We decide to use this more conservative approach and leave the system some time to react before initiating additional migrations.

4.3 GREEN COMPUTING

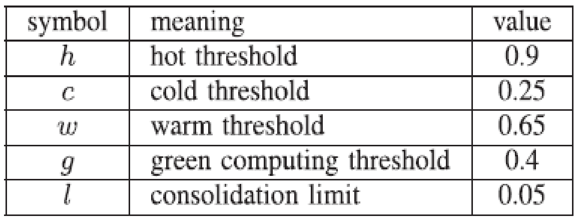
When the resource utilization of active servers is too low some of them can be turned off to save energy. This ishandled in our green computing algorithm. The challengehere is to reduce the number of active servers during lowload without sacrificing performance either now or in thefuture. We need to avoid oscillation in the system. Our green computing algorithm is invoked when theaverage utilizations of all resources on active servers arebelow the green computing threshold. We sort the list ofcold spots in the system based on the ascending order oftheir memory size. Since we need to migrate away all itsVMs before we can shut down an underutilized server, wdefine the memory size of a cold spot as the aggregatememory size of all VMs running on it. Recall that our modelassumes all VMs connect to a shared back-end storage.Hence, the cost of a VM live migration is determined mostlyby its memory footprint.

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 adestination server to accommodate it. The resource utilizationsof the server after accepting the VM must be below thewarm threshold. While we can save energy by consolidatingunderutilized servers, overdoing it may create hot spots inthe future. The warm threshold is designed to prevent that.If multiple servers satisfy the above criterion, we prefer onethat is not a current cold spot. This is because increasingload on a cold spot reduces the likelihood that it can beeliminated. 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.

5. SIMULATION RESULTS

The trace driven simulation is using to evaluate the performance of the algorithm. Note that the simulation uses the same code base for the algorithm as the real implementation in the experiments. This ensures the fidelity of our simulation results. also collected traces from servers and desktop computers in the university including one of mail servers, the central DNS server, and desktops in the department. postprocessed 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.

Parameters in Our Simulation



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

The default parameters we use in the simulation are shown in Table . 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.First evaluate the effect of the various thresholds used inour algorithm. Then simulate a system with 100 PMs and1,000 VMs (selected randomly from the trace).Usrandom VM to PM mapping in the initial layout. Thescheduler is invoked once per minute. To evaluate the scalability of our algorithm by varying the number of VMs in the simulation between 200 and 1,400.The ratio of VM to PM is 10:1.

6. CONCLUSION

A skewness algorithm is proposed to measure the unevenness in the multidimensional resource utilization of a serve, resource management system for cloud computing services. In this system multiplexes virtual to physical resources adaptively based on the changing demand. In 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.

The system multiplexes virtual to physical resources adaptively based on the changing demand. Using the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized. The algorithm achieves both overloadavoidance and green computing for systems with multiresource 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] L. Siegele, “Let It Rise: A Special Report on Corporate IT,” The Economist, vol. 389, pp. 3-16, 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,” Proc. ACM Symp. Operating Systems Principles (SOSP ’03), Oct. 2003.

[4] “Amazon elastic compute cloud (Amazon EC2),” http://aws. amazon.com/ec2/, 2012.

[5] 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.

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

[7] 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.

[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] 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.

[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,” Proc. ACM European conf. Computer Systems (EuroSys ’09), 2009.