International Journal of Computational Intelligence and Informatics, Vol. 3: No. 2, July - September 2013

An Approach to Virtual Machine Placement Problem in a Datacenter Environment Based on Overloaded Resource

T Thiruvenkadam
Department of Computer Science
K S Rangasamy College of Arts and Science
Tiruchengode, India

V Karthikeyani
Department of Computer Science
Thiruvalluvar Government Arts College
Rasipuram, India

***Abstract*-The arrival of the clouds has introduced very strict initiative over the physical resources. A typical resource management system receives queues and finally matches user job requirements with the characteristics of the offered hardware. Server overload is the basis of resource insufficiency and spoils the performance of applications which leads to affect the QoS. Lively consolidation of Virtual Machines is an efficient way to get better use of resources and power efficiency in Cloud data centers. Identifying when it is best to move VMs from a congested host is an aspect of Lively VM consolidation that directly influences the resource consumption and Quality of Service (QoS) delivered by the system. This paper focus on the importance of the detection of server overload and compares the various scheduling algorithms currently used for scheduling virtual machines and also proposes the design methodology of a new algorithm that helps to improve the resource utilization and at the same time energy efficiency.**

Keywords-Virtual Machines, Overloaded resources, Server Consolidation, Hypervisor, Load Balancing

I. INTRODUCTION

One of the major causes of energy inefficiency in data centers is the idle power [1] Data center costs can be reduced by utilizing virtual machines (VMs). Using virtualization, multiple operating system instances can run on the same physical machine, exploiting hardware capabilities more fully, allowing administrators to save money on hardware and energy costs. To maximize the savings, administrators should assign as many VMs as possible to servers given performance requirements. We refer to this problem as the Virtual Machine Assignment Problem. The scheduling of virtual machines in a cloud computing environment has become crucial due to the increase in the number of users.

This is usually done to load balance a system effectively or to achieve a target quality of service [2]. The scheduling algorithm used contributes to the performance of the entire system and the throughput. Server consolidation is the practice of migrating distinct legacy servers from distinct physical machines that possibly use different operating systems to virtual machines on a single more powerful platform. This reduces operational expenses by saving the need to maintain and support all those legacy systems, reducing the floor footprint, and reducing power and cooling requirements. It is especially beneficial when it increases server utilization, e.g. if the legacy servers are not highly utilized, but when consolidated they lead to a reasonably high utilization of the new server.

Another important benefit of consolidation is that it promotes flexible provisioning of resources. With consolidation, the resources provided to each server are not fixed. Rather, the different servers compete for resources, which are provided by the underlying virtualization infrastructure. It is then possible to control the resources provided to each one, and assign them according to need or the relative importance of the different servers. Importantly, this partitioning of the resources can be changed easily to reflect changing conditions. The virtualization infrastructure usually called a hypervisor or virtual machine monitor is therefore found to assume many of the basic responsibilities of an operating system, especially with regard to resource allocation and scheduling.

Indeed, it is natural to consider the use of well-known mechanisms and policies that were originally developed for operating systems and simply port them to the hypervisor or VMM. However, the situation is actually somewhat different. One difference is that hypervisors typically operate with far less information than an operating system. An operating system mediates all interactions with hardware devices for all processes, where the processes

themselves are rather simple in structure. Therefore the operating system can use a pretty simple model of operation, e.g. blocking a process that has requested an I/O operation. But a hypervisor is one level lower down, and supports a virtual machine that in turn runs a full operating system which may support many processes.

When some process in the virtual machine requests an I/O operation, this then does not reflect the activity of the virtual machine as a whole only the activity of that process, which is not even directly known by the hypervisor. Another difference is that the goals are typically different. Operating systems attempt to optimize metrics such as response time of interactive processes, while at the same time providing equitable service to all the processes. With hypervisors, it is more typical to try and control the resource allocation, and ascertain that each virtual machine only gets the resources that it deserves. To complicate matters, this has to be done in multiple dimensions, reflecting the different devices in the system: the CPU, the disks, and the network connectivity.

The question is then what does it mean to provide a certain share of multiple resources, when the processes running on each virtual machine actually require different combinations of resources. The simplest solution is to use the desired allocation as an upper bound. For example, if a certain virtual machine is to be allocated 30% of the resources, it will get no more than 30% of the CPU cycles, 30% of the disk bandwidth, and 30% of the network bandwidth. But this can be very inefficient if a set of virtual machines actually have complimentary requirements, e.g. if one only uses the disk while the other predominantly uses the network.

Our algorithm is designed to deal with this issue. It is based on the combination of two basic ideas: the use precedence based scheduling to control relative resource allocation and the identification of the overload device as the focus where such control should be exercised. By controlling the allocation on the overloaded device, we induce a schedule on other devices as well; this replaces the use of a myopic scheduler on each device that does not take the global system state into account. We claim that this approach provides a meaningful and efficient definition to the concept of predefined allocation of multiple resources.

II. RELATED WORK

The problem of resource allocation has been considered for many years, but generally from diverse angle than the one we use. The requirement for control over the allocation of resources given to different users or groups of users has been addressed in several contexts. It is usually called fair-share scheduling in the literature, where "fair" should be understood as according to each user's due rather than as equitable. Early implementations were based on accounting, and simply gave precedence to users who had not yet received their due share at the expense of those that had exceeded their share [3, 4].

In Unix systems it has also been suggested to manipulate each process's nice value to achieve the desired effect [5], [6]. Simpler and more direct approaches include lottery scheduling [7] or using an economic model [8], where each process's precedence (and hence relative share of the resource) is expressed by its share of lottery tickets or capital. Another approach that has been used in several implementations is based on virtual time [9, 10]. The idea is that time is simply counted at a different rate for different processes, based on their relative allocations.

We use a version called RSVT, for "resource sharing virtual time", which bases scheduling decisions on the difference between the resources a process has actually received and what it would have received if the ideal resource sharing discipline (similar to the ideal processor sharing concept) had been used [11]. A similar approach is used in [12] focusing on virtual machine monitors, Xen uses Borrowed Virtual Time (BVT).

VMware ESX server uses weighted fair queueing or lottery scheduling. The Virtuoso system uses a scheduler called VSched that treats virtual machines as realtime tasks that require a certain slice of CPU time per each period of real time [13, 14]. Controlling the slices and periods allows for adequate performance even when mixing interactive and batch jobs. The main drawback of all the above approaches is that they focus on one resource the CPU. The effect of CPU scheduling on I/O is discussed in [15]. It has also been suggested to temporarily prioritize VMs that do I/O so as not to cause delays and latency problems [16].

However, the interaction of such prioritization with allocations was not considered. Similarly, there has been interesting work on scheduling overloaded devices other than the CPU, but this is then done to optimize performance of the said device and not to enforce a desired allocation [17]. Adapting to multiple resource constraints was addressed by [18]. However, their approach was to control the applications so that they adjust their usage, rather than enforcing an allocation from the outside. The combination between scheduling and multiple resources was discussed in [19].

The context is completely different as they consider targets for migration in the interest of load balancing. Interestingly, the end result is similar to our approach, as they try to avoid machines where any one of the resources will end up being highly utilized and in danger of running out. Perhaps the closest related work is [20]. This paper also considers the interactions and degradations in service created by allocations on multiple distinct devices. However, the context is interactive applications and the need to maintain responsiveness.

The approach is based on specifying the required CPU service as in Virtuoso, and subjecting the other devices to this specification. For example, virtual memory belonging to an application with any such specification will be excluded from paging considerations for a default of 30 minutes.

III. Assessment Of Existing Algorithms

A. Greedy Algorithm

The Greedy algorithm is the default algorithm used for scheduling of Virtual Machines in Eucalyptus. The Greedy algorithm is very simple and straight forward. As a matter of fact, it was the only scheduling policy which was in use for a long time. Only after the cloud started evolving, more complex scheduling policies came into effect [21]. The greedy algorithm uses the first node that it finds with suitable resources for running the VM that is to be allocated. The first node that is identified is allocated the VM.

This means that the greedy algorithm exhausts a node before it goes on to the next node. As an example, if there are 3 nodes and the first node's usage is 40% while the other two are under loaded and if there are two VMs to be allocated, then both are allocated to the first node which might result in the increase of its usage to 90% while the other two nodes will still remain under loaded. As obviously seen, the main advantage of the Greedy algorithm is its simplicity. It is both simple to implement and also the allocation of VMs do not require any complex processing. The major drawback would be the low utilization of the available resources. As illustrated in the example above, even if there are under loaded nodes, an overloading of a node might result.

B. Round Robin Algorithm

The Round Robin algorithm mainly focuses on distributing the load equally to all the nodes [22, 23]. Using this algorithm, the scheduler allocates one VM to a node in a cyclic manner. The round robin scheduling in the cloud is very similar to the round robin scheduling used in the process scheduling. The scheduler starts with a node and moves on to the next node, after a VM is assigned to that node. This is repeated until all the nodes have been allocated at least one VM and then the scheduler returns to the first node again. Hence, in this case, the scheduler does not wait for the exhaustion of the resources of a node before moving on to the next.

As an example, if there are three nodes and three VMs are to be scheduled, each node would be allocated one VM, provided all the nodes have enough available resources to run the VMs. The main advantage of this algorithm is that it utilizes all the resources in a balanced order. An equal number of VMs are allocated to all the nodes which ensure fairness. However, the major drawback of using this algorithm is that the power consumption will be high as many nodes will be kept turned on for a long time. If three resources can be run on a single node, all the three nodes will be turned on when Round Robin is used which will consume a significant amount of power.

C. Power save Algorithm

The Power Save algorithm optimizes the power consumption by turning off the nodes which are not currently used [24, 25]. Instead of keeping all the nodes turned on, resulting in a lot of power consumption, this algorithm aims at turning the unused nodes off which will reduce the power consumed to a reasonable extent. The scheduler allocates a VM to the node and then traverses through the list of nodes to check if the node is unused and if found to be so, turns it off. If the resource of a node which has been turned off is required for the allocation of a VM, the scheduler turns it on again and then allocates the VM to that node.

As an example, consider the scenario in which there are three nodes, two of which are unused. When a new VM is to be scheduled, the scheduler may allocate it to the node which is already being used and would turn off the two nodes which are unused. The Power Save algorithm results in the reduction of power consumption but this is at the expense of lower utilization of resources. This algorithm is used only at places where there is an extreme need for reducing the consumption of power. A new scheduling algorithm has been proposed in this paper which is based on the assignment of lively precedence to the nodes. It overcomes the stated limitations of the existing scheduling algorithms and works effectively under all circumstances.

IV. Proposed Lively Precedence Based Scheduling Algorithm

The proposed algorithm tries to solve the scheduling problem by taking time, precedence and the overloaded device into consideration when scheduling virtual machine requests from the user. Each virtual machine request will have precedence ranking, low, average, or high, and is queued into its respective queue (low, medium, or high). The high queue will be serviced in a first come first served manner with each serviced request being moved to the end of its queue.

After a determined time, the matching queue from the lower precedence has its precedence elevated and is moved to the higher queue to be serviced. If there is no request in the high queue, medium precedence is treated as high precedence and is processed accordingly. If a request comes in that has high precedence, the average will return to average precedence and processed accordingly. The proposed algorithm is divided into the following four sub-problems.

i. Deciding when a host is considered to be overloaded, so that some VMs should be migrated from it to other hosts to meet the QoS requirements.
ii. Deciding when a host is considered to be under loaded, so that its VMs should be migrated and the host should be switched to a low-power mode.
iii. Selecting VMs to migrate from an overloaded host.
iv. Allocating the VMs selected for migration to other active or re-activated hosts.

Our aim is to sustain the perception of restricted allocation of resources to virtual machines, such that we will be able to say that VM1 gets, say, 50% of the resources, VM2 gets 30%, and VM3 gets 20%. The problem is that a virtual machine may require more of one resource than of another. This raises the question of what exactly is meant by "50% of the machine". We answer this using overload-based scheduling. Specifically, we make the observation that at any given time one device is the system bottleneck this is the device that has the highest utilization. Reducing allocations on the overloaded device will likely reduce consumption levels on other resources.

It may not reduce consumption only when a VM's allocation on a resource is above its actual consumption to begin with. In either case, the consumption of other system's resources will not increase therefore no other device can become saturated. The allocations on the overload device will be made using the RSVT algorithm, which uses the notion of virtual time to allocate predefined portions of the resource. The way fractional resource usage is measured depends on the resource:

- For the CPU, it is simply measured as a fraction of CPU time. Note that this includes both user time and system time.
- For the network(s), it is measured as a fraction of the used bandwidth, including bandwidth used for various headers.
- For the disk(s), it is also measured as a fraction of the used bandwidth. The disk space used is of no concern.

V. Using The Proposed Algorithm With Different States

The client request to schedule a Virtual Machine the algorithm checks if the highest resource node has been identified and its precedence has been assigned. If the node has not been identified, then it is identified first. The node with the highest resource is determined and then it is checked whether the node has a load factor less than 85%. This is done to prevent a particular node from being overloaded. Once the suitable node is identified, then it is assigned the highest precedence and the VM is allocated to it. If the highest precedence has a load factor above 85%, then it checks if the next maximum resource node has been identified. If it has been identified and if its load factor is less than 85%, then the VM is scheduled to that node and the search for the next maximum resource node with the load factor less than 85% takes place.

Before assigning the new node as the highest precedence node, the current maximum resource node is assigned to previous maximum resource node and the previous maximum resource node is assigned the next highest precedence. The purpose of keeping track of the previous maximum resource node is to prevent the fluctuations above and below the load factor in extreme cases, when VMs are assigned to it and removed from it. Consider the scenario in which, the maximum resource node has been identified and it has been assigned the highest precedence. The VMs are allocated to it till the load factor is less than 90%. Assume that the current load factor is 80%. After allocating a VM to it, the load factor becomes 90%.On completion of the work, the VM is shut down and the load factor goes to 80%. The load factor fluctuates above and below 85% frequently and will initiate a new search request for finding the new node. To overcome this problem, the previous maximum resource node is kept track off i.e. the next highest precedence node.

If the highest precedence node has the load factor more than 85%, then it keeps allocating the VMs to the next highest precedence node until its load factor becomes greater than 85%. This gives the highest precedence node some time to finish off the work of VM and shut it down so that it would be well below the load factor. If the load factor is still above 85%, then its precedence is decreased and the other node is assigned higher precedence. This algorithm shows remarkable speed in terms of the time taken to schedule a Virtual Machine to a node, once these nodes have been identified. The scheduling would be done to these nodes directly and the other nodes are not taken into consideration when it comes to power efficiency, every time the request for scheduling is received, it also checks for the idle nodes. The idle nodes are one to which, no VM is allocated to it. These nodes are turned off to save power.

A. Advantages

The scheduling can be done very quickly. If the highest and next highest precedence nodes have been identified. It considers the load factor to prevent a particular node from being overloaded. To enable the power efficiency the idle nodes are turned off. It prevents fluctuations around the load factor of 85% in most cases. Fluctuation occurs only under extreme cases, when all the nodes have load factor which are approximately 85%.

B. Result Analysis

The following figure 1 illustrates the relationship of resource utilization rate and number of VMs using proposed and Round Robin VM allocation policy. The parallel axis is the number of VMs requests and perpendicular axis represents the resource utilization rate. The resource utilization rate goes below 50 % in round robin policy when the number of VM Requests increased where as in proposed algorithm above 60% of resources are utilized in all the cases. In this comparison the proposed algorithm performs better than the existing round robin algorithm.

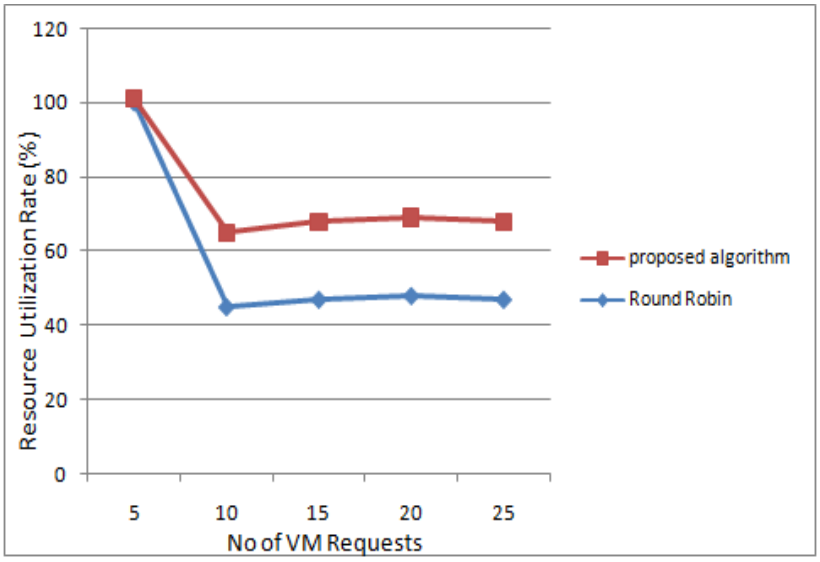

Figure 1. Resource Utilization

The following figure 2 shows the reaction time of the proposed scheduling algorithm based on the number of concurrent VM requests. When the queue length is increased the scheduling time of the VM is also increasing gradually without any spikes.

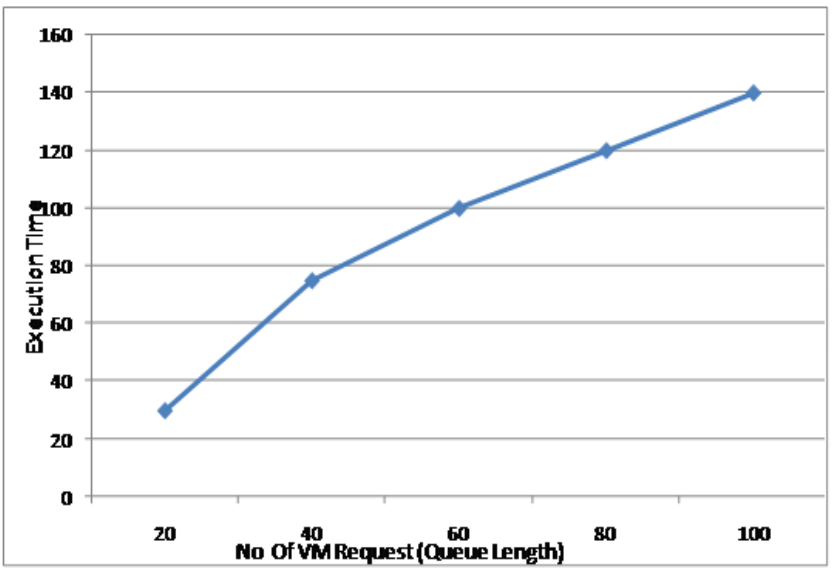

Figure 2. Execution Time

VI. CONCLUSION

We proposed a lively precedence based scheme for the problem of VM allocation in large heterogeneous data centers or server clusters. In this paper it gives the detailed review on existing scheduling algorithms with their advantages and drawbacks. Our framework explores scheduling all the resources so as to achieve a desired allocation while using precedence based VM assignment. We recommend a significant definition of what predefined resource allocations mean in a multi-resource context: we claim that at each instant the system overloaded device should be identified, and that the allocations should be performed on this overload device. These allocations then induce a schedule on the other devices as well, based on the relative precedence on the overload device. The significance of this system lies in its widespread approach to the resource allocation problem. Such an approach is now needed by current installations, especially those used for server consolidation. In future work we

plan to implement the proposed design with all the resources we want to control, namely the CPU, disks, and network. Next, we need a monitoring facility that will identify the overloaded device and choose which module should take precedence over the others. Given a working system we will perform a set of experiments to describe its behavior.

REFERENCES

[1] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. "Energy aware consolidation for cloud computing", In Proceedings of HotPower '08 Workshop on Power Aware Computing and Systems. USENIX, Dec 2008.

[2] Scheduling: http://en.wikipedia.org/wiki/Scheduling (computing)

[3] G. J. Henry, "The fair share scheduler", AT&T Bell Labs Tech. J. 63(8, part 2), pp. 1845–1857, Oct 1984.

[4] J. Kay and P. Lauder, "A fair share scheduler ", Comm. ACM 31(1), pp. 44–55, Jan 1988.

[5] J. L. Hellerstein, "Achieving service rate objectives with decay usage scheduling ", IEEE Trans. Softw. Eng. 19(8), pp. 813–825, Aug 1993.

[6] D. H. J. Epema, "Decay-usage scheduling in multiprocessors", ACM Trans. Comput. Syst. 16(4), pp. 367–415, Nov 1998.

[7] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: flexible proportional-share resource management", In 1st Symp. Operating Systems Design & Implementation, pp. 1–11, USENIX, Nov 1994.

[8] I. Stoica, H. Abdel-Wahab, and A. Pothen, "A microeconomic scheduler for parallel computers", In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), Springer- Verlag, Lect. Notes Comput. Sci. Vol. 949, pp. 200–218, 1995.

[9] J. Nieh, C. Vaill, and H. Zhong, "Virtual-Time Round Robin: an O(1) proportional share scheduler", In USENIX Ann. Technical Conf., pp. 245–259, Jun 2001.

[10] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler", In 17th Symp. Operating Systems Principles, pp. 261–276, Dec 1999.

[11] Y. Etsion, D. Tsafrir, and D. G. Feitelson, "Process prioritization using output production: scheduling for multimedia",ACM Trans. Multimedia Comput., Commun. & App. 2(4), pp. 318–342, Nov 2006.

[12] A. Chandra, M. Adler, P. Goyal, and P. Shenoy, "Surplus fair scheduling: a proportional-share CPU scheduling algorithm for symmetric multiprocessors", In 4th Symp. Operating Systems Design & Implementation, pp. 45–58, Oct 2000.

[13] B. Lin and P. A. Dinda, "VSched: mixing batch and interactive virtual machines using periodic real-time scheduling", In Supercomputing, Nov 2005.

[14] B. Lin and P. A. Dinda, "Towards scheduling virtual machines based on direct user input", In 2nd Intl. Workshop Virtualization Technology in Distributed Comput., 2006.

[15] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors", In 4th Intl. Conf. Virtual Execution Environments, pp. 1–10, Mar 2008

[16] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and co.: communication-aware CPU scheduling for consolidated Xen-based hosting platforms", In 3rd Intl. Conf. Virtual Execution Environments, pp. 126–136, Jun 2007.

[17] B. Schroeder and M. Harchol-Balter, "Web servers under overload: how scheduling can help", ACM Trans. Internet Technology 6(1), Feb 2006.

[18] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server", In Network Operations & Management Symp., pp. 219–234, 2002.

[19] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren, "An opportunity cost approach for job assignment in a scalable computing cluster", IEEE Trans. Parallel & Distributed Syst. 11(7), pp. 760–768, Jul 2000.

[20] T. Yang, T. Liu, E. D. Berger, S. F. Kaplan, and J. E. B. Moss, "Redline: first class support for interactivity in commodity operating systems", In 8th Symp. Operating Systems Design & Implementation, Dec 2008.

[21] Subramanian S, Nitish Krishna G, Kiran Kumar M, Sreesh P and G R Karpagam "An Adaptive Algorithm for Dynamic Precedence Based Virtual Machine Scheduling in Cloud", In IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 6, No 2, November 2012 ISSN (Online): 1694-08.

[22] Tejinder Sharma, Vijay Kumar Banga "Efficient and Enhanced Algorithm in Cloud Computing".In International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Vol.3, Issue-1, March 2013.

[23] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov, "The eucalyptus open-source cloud-computing system", In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, IEEE Computer Society, Washington, DC, USA, pp. 124–131, 2009.

[24] Shingo Takeda and Toshinori Takemura. A rank-based vm consolidation method for power saving in datacenters. Information and Media Technologies, 5(3):994–1002, 2010.

[25] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08, Berkeley, CA, USA, pp. 337–350, 2008.