Themis: Energy Efficient Management of Workloads in Virtualized Data Centers

Gaurav Dhiman[1], Vasileios Kontorinis[1], Raid Ayoub[1], Liuyi Zhang[1], Chris Sadler[2], Dean Tullsen[1], and Tajana Simunic Rosing[1,*]

[1] UCSD
[2] Google
tajana@ucsd.edu

Abstract. Virtualized data centers facilitate higher resource utilization and energy efficiency through consolidation. However, mixing services-oriented workloads with throughput (batch) jobs is typically avoided due to complex interactions and widely different quality of service (QoS) requirements. We introduce a complete VM resource management framework, called Themis, which manages combined services and batch jobs, maximizing energy-efficient throughput of the latter without sacrificing the service guarantees of the former. Themis' resource management policy outperforms the prior proposed policies by up to 35% on average in work done per Joule when measured on a data center testbed.

1 Introduction

Virtualization has rapidly gained prominence in modern data center deployments, since it provides better fault isolation, improved system manageability, and reduced operational cost through resource consolidation and migration [8]. It is common for a data center to host a mix of interactive service-oriented and throughput-oriented batch jobs. These two types of jobs are usually partitioned into separate sections of the data center [15] because we lack a mechanism for managing their diverse performance requirements. Services typically have strict response time guarantees and the cost of violating those agreements is high [15]. Batch jobs often have long-term performance targets, where immediate response is not vital.

A number of systems for VM management have been proposed in the past. Eucalyptus [23], OpenNebula [25], OpenStack [31] and Usher [18] are open source systems, which include support for managing VM creation and allocation across a cluster, and provide API for migration. However, these solutions generally do not have online VM scheduling policies to dynamically consolidate or redistribute VMs, but instead focus on the initial resource allocation and assignment of VMs to physical machines. In [29], the authors propose a system which dynamically schedules the VMs based on their CPU, memory and network utilization to improve the overall performance. In [6] the authors propose dynamic consolidation and redistribution of VMs for managing QoS requirements of different service VMs in the cluster. Entropy

[*] Corresponding author.

[14] uses constraint programming to determine a globally optimal solution for VM scheduling in contrast to the first fit decreasing heuristic used by [6, 29], which can result in globally sub-optimal placement of VMs. However, these approaches are not QoS aware. They assume that the CPU utilization adds upon VM consolidation, which is not true for heterogeneous VM consolidation.

Management of QoS within operating systems for latency sensitive applications in a heterogeneous workload mix running on standalone systems has been studied in Stanford SMART scheduler [23] and QLinux [28] projects. They ensure timely access to the CPU for the latency sensitive applications while maintaining proportional sharing of CPU for the batch jobs. Similar solutions have been proposed for virtualized environments as well [17, 24]. We show that the software level support for QoS through timely CPU access is not sufficient to guarantee QoS in presence of interference effects in modern multi-core based systems.

The interference effects due to shared resource usage by co-scheduled workloads on modern multi-core based platforms has been studied before at both the OS and hypervisor levels. The work in [5, 9] explores the problem exclusively for batch jobs. The research in [9, 20] takes the same problem of shared resource usage to the cluster level using virtualization, again just for batch workloads. In [21, 26], the authors use CPU capping to ensure QoS requirements are met. However, their focus is either on batch workloads [21] or services [26], but not a mix of the two.

State of the art techniques that focus on consolidating homogeneous workloads do not scale well for resource management with heterogeneous workloads. To solve this problem we designed Themis, a system for VM resource management in virtualized clusters. Themis includes a monitoring infrastructure which allows it to actively measure both performance and QoS metrics of different types of workloads within the data center. It is built as extension to Xen [3], and as such could be integrated into any open source system which can leverage Xen. We evaluate Themis on a testbed built of state-of-the-art servers with workloads representative of both services and batch jobs. Our evaluation shows that Themis can improve energy efficiency by up to 35% over the best proposed policies for resource management in consolidated environments while meeting both service and batch job performance targets.

2 Themis Design

In this section we provide details of the design and implementation of our system, Themis, for managing diverse workloads in the data center. We classify the workloads into two categories [15]: **(1) Services.** The primary goal for these workloads is to serve the user request within a given time bound to maintain a QoS level. In this paper we use RUBiS [1] as representative of services. Rubis is a multi-tier online service that implements the core functions of an auction site including selling, browsing, and bidding. It contains a front-end Apache PHP web server and a back-end MySQL database. RUBiS provides workloads of different mixes for the client sessions that are emulated on a separate machine. We use the ‘browsing mix’, which emulates a web-intensive user browsing experience. **(2) Batch jobs.** These workloads refer to resource intensive jobs that are representative of the analytics, number crunching, and scientific computing class of workloads. The primary goal of these jobs is to maximize the overall instruction throughput, with no specific response time requirements. Since the goal of this paper is to study the effects of CPU and memory interference on the consolidation of different types of data center workloads, we use representative workloads from SPEC2000 and PARSEC benchmark suites, as our batch workloads. These workloads can be used to approximate memory and CPU intensive phases of typical data center batch jobs such as MapReduce [30].

Themis' objectives are to: (1) satisfy the QoS requirement of the service jobs; (2) maximize the batch job throughput; (3) minimize the power consumption. We define a new metric to capture these goals. It measures the batch job throughput/Watt, multiplied by a factor q that reflects how closely services QoS requirements are met. If QoS requirements are *not* met, then q is set to zero. Maximizing qMIPS/Watt implies that it is acceptable to sacrifice a bit of performance of service jobs as long as we still meet the strict service level agreements, increase batch throughput, and increase the overall energy-efficiency of the system.

$$qMIPS/Watt = q * (batchjobMIPS)/SystemWatts \qquad (1)$$

2.1 Motivation for Themis

Many of the modern hypervisor schedulers are based on proportional sharing of CPU resources. Xen uses a credit scheduler, where each VM's proportional share is specified through 'weight'. Based on weight, the physical CPU resources (or credits) are distributed to the virtual CPUs (vCPUs) in proportion of their weight, with vCPU priority recalculated based on the credits the VM has. There are three priority levels: (1) 'Over': the lowest priority that a VM is set to when it exhausts its credits; (2) 'Under': medium priority; and (3) 'Boost': the highest priority used for low-latency tasks such as those that just received an I/O interrupt.

Such a model works well if VM workloads use the CPU resources in a homogeneous fashion. However, CPU residency times of services workloads are very low [10], but at the same time it is important for them to get the CPU as soon as they are ready to run, as otherwise QoS requirements may not be met. The mechanisms provided by the proportional schedulers fail to achieve that, since proportional sharing only promises a higher proportion of CPU without any timing guarantees.

This lack of QoS support in proportional schedulers has been identified as a problem in conventional OS schedulers and is addressed to some extent in [22] and [28], as well as in hypervisor schedulers like Xen [17]. As a part of Themis, we also implement real time scheduling for services similar to QLinux [28], while retaining default proportional scheduling for batch VMs. A new priority state is introduced in the scheduler, referred to as the 'QoS' state, which is between the Boost and Under states. The QoS state can be configured through Xen API for any VM which has a QoS requirement. However, as we show below, providing such real time priority is not sufficient by itself to guarantee QoS for the services in consolidated virtualized environments.

In Figure 1 we show results of an experiment where we run RUBiS web server in a higher priority QoS VM state on one CPU sockett, and a batch VM running *swim*, a compute intensive weather prediction benchmark from SPEC'00, on another socket of an Intel Xeon quad core machine. Figure shows on x-axis time in seconds, and on dual y-axis CPU utilization of the servers and QoS ratio of RUBiS. The target QoS ratio should be less then 1. We also ran RUBiS by itself and found has QoS ratio close to 0.3. The interference effects due to the batch VM slow down the web server, even though they do not share a physical core. This consequently increases the CPU utilization of the web server VM, creates a bottleneck thus worsening the QoS ratio of the application to unacceptable levels that are 3x higher than the specified limit.

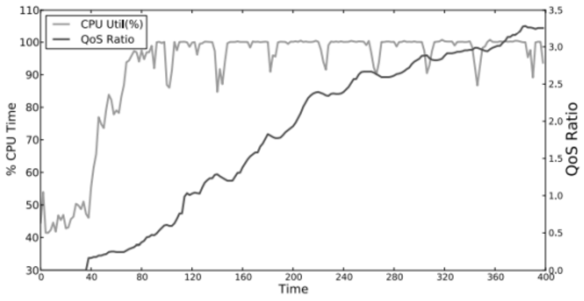

Fig. 1. RUBiS-web with a batch VM running '*swim*'

This example shows that just guaranteeing real time priority is not sufficient to ensure QoS for service VMs in consolidated environments. The interference effects due to shared resource usage can dramatically impact the QoS level even when CPU resources are not shared, and must be explicitly accounted for. These interference effects are a function of how the workloads interact with each other which are difficult to model. Consequently, our approach, as described in the next section, is to infer the interference effects through resource utilization and QoS ratio feedback from the service VMs, and to adaptively provision the resource allocation to alleviate these issues.

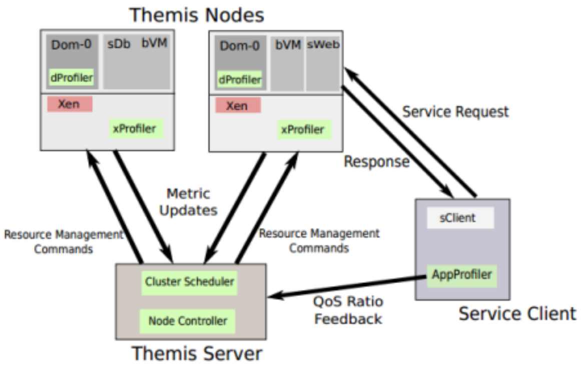

Fig. 2. Themis Design

2.2 Themis Components

The overall objective of Themis is to manage diverse workloads in the data center with the goal of maximizing qMIPS/Watt. It implements a monitoring framework for dynamic VM profiling and policies for dynamic resource management of VMs. Themis has three entities as shown in Figure 2: (1) Themis Nodes: These are the physical machines in the data center that run the actual workloads, which can be a batchVM, serviceVM or both. (2) Services Clients: These are the machine(s) that are running applications requesting service from a serviceVM (which can be a single or multi-tier service) running on the Themis clients. (3) Themis Server: This is the cluster manager, and is responsible for implementing policies for node level resource management and VM scheduling. We now present these entities in greater detail.

1) *Themis Nodes* (tNodes) are physical machines that run workloads. They are equipped with Themis specific profilers to capture live metrics that estimate per-VM resource utilization. There are two such profilers:

xProfiler: It captures throughput (MIPS) and memory access information (MPC) for VMs running on tNodes with CPU performance counters, and communicates that to dProfiler. This data is not used by Themis, but is required for comparison with state of the art scheduling policies [11, 25], and to estimate qMIPS/Watt for all policies.

dProfiler: It compiles per-VM performance and resource utilization information, and communicates it to Themis server. The dProfiler runs inside the Dom-0.

2)Services Clients (sClients) are the applications serviced by the serviceVMs running on tNodes. The sClients use appProfiler to dynamically communicate QoS ratio to the Themis server. We assume that it is feasible to implement such appProfilers for all the services whose QoS needs to be monitored. The QoS ratio is dynamically communicated by the appProfiler to the Themis server.

3)Themis Server (tServer) does resource management across the cluster with the objective of maximizing the qMIPS/Watt. It registers all tNodes and sClients through dProfiler and appProfilers, periodically collects the metric updates and QoS ratios from them, and sends this to the management policies running on the system. The policies convey their management decisions to the dProfiler, which physically implements them on the intended tNode as illustrated in Figure 2. The tServer uses a cluster scheduler similar to the existing state of the art implementations [14, 29], that consolidates batch and serviceVMs based on the CPU utilization metrics of the individual VMs running across the cluster provided by the dProfiler. However, when we co-locate batchVM and serviceVM on a tNode, the interference effects can impact the QoS ratio of the services. While batchVMs need CPU resources to maximize their throughput, the serviceVMs only need the CPU resources for long enough to service the client requests within the required time frame. This observation motivates the design of a resource management policy, referred to as the 'Node Controller' to dynamically control the CPU resource allocation to the serviceVMs.

4) Node Controller (tController) exists for every serviceVM which operates independently based on the metric and QoS inputs. At every time interval tController predicts what is the needed number of CPUs (which we refer to as *nref*) for the next interval based on the CPU utilization and QoS ratio. Its objective is to converge to the number of virtual CPUs sufficient to meet QoS ratio for the serviceVMs, while giving as much CPU headroom to batchVMs as possible. Algorithm 1, used to estimate n_{ref} for the upcoming interval, takes as input the current QoS ratio (QoS_{cur}), CPU utilization ($util_{cur}$) and the vCPU allocation (n_{cur}) for the serviceVM the tController is managing. It further uses two important threshold paramenters: (1) QoS_{th}: This threshold is used by the algorithm to determine whether the QoS of the service being monitored is being safely met. If the QoS_{cur} is less than QoS_{th}, then current resource allocation is more than sufficient to meet QoS metrics. (2) $util_{th}$: CPU utilization threshold that is used to determine if the service needs additional CPU resources. The experiments with servce workload and batch jobs showed that service CPU utilization scales linearly with the number of CPUs regardless of the type of batch jobs.

Algorithm 1. Performance Model

```
Input: $QoS_{cur}$, $util_{cur}$ and $n_{cur}$
1: if $QoS_{cur} < QoS_{th}$ then
2:     $n_{next} \leftarrow n_{cur} - 1$
3:     $util_{next} \leftarrow (util_{cur} \times n_{cur})/n_{next}$
4:     if $util_{next} > util_{th}$ then
5:         return $n_{cur}$
6:     end if
7: else
8:     $n_{next} \leftarrow n_{cur} + 1$
9: end if
10: return $n_{next}$
```

We model the error in selecting the number of vCPUs for each serviceVM, $\delta n(k)$, at each time step k, as a state which is related to the number of currently assigned vCPUs, $n(k)$, and the target number of vCPUs estimated using Algorithm 1, $nref(k)$:

$$\delta n(k + 1) = \delta n(k) + n(k) - nref(k) \tag{2}$$

To maximize the qMIPS/Watt, the error for the next step, $\delta n(k + 1)$, has to converge to zero, as this ensures that we give minimum resources the serviceVM needs to meet its QoS, hence maximizing the achievable MIPS for the batchVM. We use closed loop feedback to achieve this as shown below:

$$n_i(k+1) = -G_i\, \delta n(k) + nref_i\,(k) \tag{3}$$

where G_i is the state feedback gain for the i^{th} serviceVM and $n_i(k+1)$ is the number of vCPUs that serviceVM needs to meet its desired QoS ratio. We pick value of G_i that is between zero and one as this guarantees convergence of the controller per results available from control theory. At each control decision point, the controller calculates $nref_i\,(k)$, estimates current cumulative error, $\delta n(k)$, and based on control shown in Equation (3), it estimates the next step's vCPU allocation, $n_i(k+1)$. In practice we found that this takes only a handful of iterations, with QoS_{th} of 0.6 and T_s of 2s as representative parameters. These parameters need to be choosen appropriately for other deployments. Techniques such as ARMA estimators and maximum likelihood tests can be used to do parameter prediction and selection online.

3 Results

We conduct experiments on a testbed of four dual quad-core Nehalem machines with 24GB memory. Two are tNodes, which run the service and batch VMs, 3rd is sClient which generates workload for services, and the last is tServer, doing scheduling and resource management of the VMs. We compare Themis (labeld as *Controller*) to existing state of the art systems:

(1) *Baseline:* This policy runs service VMs and batch VMs on separate tNodes to avoid any interference effects.

(2) *Consolidation:* This policy consolidates service VMs and batch VMs on the basis of CPU utilization [14, 29] but does not perform resource management with the Node Controller.

(3) *Ideal-tChar:* Systems proposed in [10, 20] identify the memory intensiveness of the batch VMs, and distribute VMs across physical machines to avoid MIPS degradation. We ensure that they only consolidate non-memory intensive batch VMs with the service VMs. If the batch VM is memory intensive, then the VMs run independently on separate tNodes. This helps meet QoS ratio by limiting the degree of consolidation. We guarantee offline that consolidation occurs only if QoS ratio is satisfied, thus labeling the policy Ideal.

(4) *Ideal-tCap*: Systems proposed in [21, 26] manage QoS on consolidated machine by placing a CPU cap on the lower priority VM to ensure that the higher priority VM meets its QoS requirements. A CPU cap limits the amount of time the lower priority VM runs, hence reducing the interference effects. We place a CPU cap on the batch VM based on the QoS ratio feedback of the application being serviced by the service VM to ensure that the QoS requirements are met. This policy is labeled Ideal since we determine the minimum cap for batch VMs so that the QoS ratio is satisfied offline.

In all our experiments, initially one tNode runs the service VMs and the other tNode runs the batch VM. We then monitor the MIPS of the batch VM, QoS ratio for the service VMs and the power consumption of the active tNodes every two seconds (Ts) for the whole duration of the run of the sClient. The power consumption is recorded from the power sensors on the machines, which are accessible through an Integrated Power Management Interface (IPMI) [16]. Using these values, we estimate the qMIPS/Watt. The initial configuration of the batch and service VMs is identical for all policies. All the VMs run Linux as the guest OS, and are configured with 2 vCPUs and 4GB of memory. The service VMs are assigned the 'QoS priority state' for all the policies to ensure timely access to the CPU. RUBiS is configured so that service VMs comfortably meet QoS ratio when running alone. For the batch VMs, we always have as many threads of the benchmark as the number of vCPUs to represent a fully utilized VM. The mix of our batch jobs has an equal number of CPU and memory intensive benchmarks. In Figures 4 and 5 we list results for CPU bound benchmarks on the left, and memory bound on the right, with average over all on the far right.

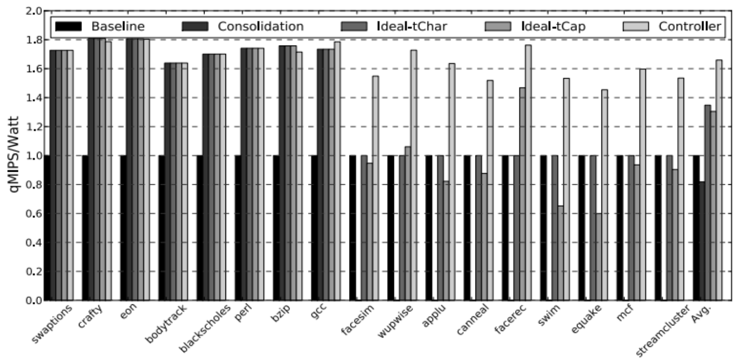

Fig. 3. Energy efficiency comparison

Figure 4 illustrates the overall results in qMIPS/Watt for all the policies, normalized against the baseline policy, with workloads listed on y-axis running inside the batch VM. Higher values of qMIPS/Watt correspond to better system energy efficiency. The more memory intensive the batch job, the more it impacts the execution of service jobs. Baseline policy gives the best MIPS for the batch jobs but is inherently energy inefficient, since it keeps two machines active, resulting in the highest active power consumption. Consolidation policy saves as much as 40% of power through VM consolidation. However, it results in poor qMIPS/Watt because the consolidation with memory intensive batch VMs results in bottlenecks for the service VMs, and consequently violations of their QoS (failure to meet the QoS requirements corresponds to zero qMIPS/WATT). The Ideal-tChar policy combines the best parts of two previous policies. This policy, based on oracle knowledge, consolidates the VMs when their QoS is maintained and keeps them separate when not. Ideal-tChar gets a 40% increase in qMIPS/WATT over the baseline for RUBiS. The policy however, misses out on the opportunity to save energy through consolidation for more memory intensive batch VM workloads. The Ideal-tCap policy, on the other hand, accomplishes consolidation under all circumstances as it places a limit on the CPU utilization of the batch VM so that the service VM just meets it QoS requirement. A cap on CPU allocation results in smaller time spent by batch VM on the CPU, which reduces the interference effects. However, Ideal-tCap not only fails to improve the useful work done per joule but actually results in its slight reduction. This is explained in Figure 5, where we observe that the MIPS of the batch VM because of capping drops considerably. This results in the Ideal-tCap policy performing even worse than the Baseline policy in terms of qMIPS/Watt for some very memory intensive batch VMs. Our Controller policy outperforms all the other policies for both service workloads. It is on average 70% better than the Baseline in qMIPS/WATT and 35% better than the Ideal policies. The large gains in qMIPS/Watt of the Controller policy over the Ideal policies are a consequence of the fact that the controller is better able to exploit the heterogeneity in the way resources are used.

Figure 5 demonstrates that the raw MIPS achieved by the Controller is on an average within 7% of the maximum possible, i.e. the Baseline, and in the worst case within 17%. At the same time it is able to reduce the system power consumption by 50% relative to the Baseline policy. In contrast, Ideal-tCap policy is on an average 30% below and up to 70% worse than the Baseline.

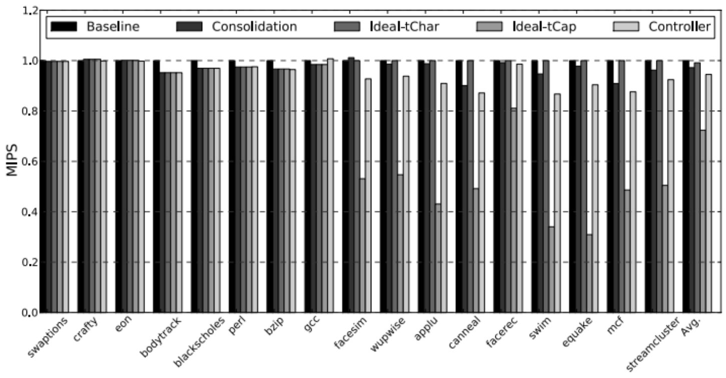

Fig. 4. Batch VM MIPS running with Rubis

4 Summary

This paper explores the challenges of managing latency sensitive services and throughput oriented batch jobs in data centers. We design a new metric, qMIPS/Watt, to capture the amount of work done per joule while maintaining a prespecified level of QoS. Our Themis controller, which leverages the heterogeneity of the workloads when managing resources, outperforms ideal versions of state-of-the art policies in work done per Joule by 35% on average, and by 70% relative to the baseline in today's data centers. Going forward we plan to include other resources to manage, such as I/O, and we plan to integrate Themis as a part of one of the large scale cloud management systems, such as OpenStack [31], which will enable us to more easily evaluate Themis' benefits at larger scale.

Acknowledgments. This work was funded in part by NSF grants No. EEC-0812072, No. CNS-0821155, No. OCI-0962997, MuSyc center, Microsoft and Google.

References

[1] Amza, C., Cecchet, E., Chanda, A., Cox, A.L., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K., Zwaenepoel, W.: Specification & implementation of dynamic web site benchmarks. In: IEEE WWC (2002)
[2] Apache, http://incubator.apache.org/olio/
[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: SOSP 2003 (2003)
[4] Barroso, L., Holzle, U.: The Case for Energy-Proportional Computing. IEEE Computer 40(12) (December 2007)

[5] Blagodurov, S., Zhuravlev, S.: Contention-Aware Scheduling. ACM Trans. on Computing Systems (2010)

[6] Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. IEEE Integrated Network Management (2007)

[7] Chase, J., Anderson, D., Thaka, P., Vahdat, A., Doyle, R.: Managing Energy and Server Resources in Hosting Centers. In: SOSP 2001 (2001)

[8] Clark, C., Fraser, K., Hand, S., Hansen, J., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: NSDI (2005)

[9] Dhiman, G., Kontorinis, V., Tullsen, D., Rosing, T., Saxe, E., Chew, J.: Dynamic Workload Characterization for Power Efficient Scheduling on CMP Systems. In: ISLPED (2010)

[10] Dhiman, G., Marchetti, G., Rosing, T.: vGreen: A System for Energy Efficient Computing in Virtualized Environments. In: ISLPED (2009)

[11] Dhiman, G., Pusukuri, K., Rosing, T.: Analysis of DVFS for Energy Management. In: USENIX-HotPower (2008)

[12] Fan, X., Weber, W., Barroso, L.: Power Provisioning for a Warehouse-sized Computer. In: ISCA (2007)

[13] Ge, R., Feng, X., Feng, W., Cameron, K.: CPU MISER. In: ICPP (2007)

[14] Hermenier, F., Lorca, X., Menaud, J., Muller, G., Lawall, J.: Entropy: a Consolidation Manager. In: VEE (2009)

[15] Hoelzle, U., Barroso, L.: The Datacenter as a Computer (2010)

[16] IPMI, v2.0 Specification, Intel (2004)

[17] Lee, M., Krishnakumar, A., Krishnan, P., Singh, N., Yajnik, S.: Supporting real-time in the Xen hypervisor. In: VEE 2010 (2010)

[18] Mcnett, M., Gupta, D., Vahdat, A., Voelker, G.: Usher. In: LISA 2007 (2007)

[19] Meisner, D., Gold, B., Wenisch, T.: PowerNap: Eliminating Server Idle Power. In: ASPLOS (2009)

[20] Merkel, A., Stoess, J., Bellosa, F.: Resource-Conscious Scheduling for Energy Efficiency. In: EuroSys 2010 (2010)

[21] Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-Clouds: Managing Interference for QoS-Awareness. In: EuroSys (2010)

[22] Nieh, J., Lam, M.: A Smart Scheduler for Multimedia Applications. ACM Trans. Comput. Syst. 21 (2003)

[23] Nurmi, D., Wolski, R., Grzegorczyk, C., Soman, S., Youseff, L., Zagorodnov, D.: Eucalyptus. In: ISCCG 2009 (2009)

[24] Ongaro, D., Cox, A., Rixner, S.: Scheduling I/O in Virtual Machine Monitors. In: VEE (2008)

[25] OpenNebula, http://www.opennebula.org/

[26] Padala, P., Hou, K., Shin, K., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A.: Automated Control of Multiple Virtualized Resources. In: EuroSys 2009 (2009)

[27] Rajamani, K., Lefurgy, C.: On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In: ISPASS (2003)

[28] Sundaram, V., Chandra, A., Goyal, P., Shenoy, P., Sahni, J., Vin, H.: Application Performance in the QLinux Multimedia Operating System. In: MULTIMEDIA 2000 (2000)

[29] Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration. In: NSDI (2007)

[30] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI (2004)

[31] OpenStack (2012), `http://docs.openstack.org`