Tide: Achieving Self-Scaling in Virtualized Datacenter Management Middleware

Shicong Meng, Ling Liu
Georgia Institute of Technology
{smeng, lingliu}@cc.gatech.edu

Vijayaraghavan Soundararajan
VMware, Inc.
ravi@vmware.com

ABSTRACT

The increasing popularity of system virtualization in datacenters introduces the need for self-scaling of the management layer to cope with the increasing demands of the management workload. This paper studies the problem of self-scaling in datacenter management middleware, allowing the management capacity to scale with the management workload. We argue that self-scaling must be fast during workload bursts to avoid task completion delays, and self-scaling must minimize resource usage to avoid resource contention with applications. To address these two challenges, we propose the design of Tide, a self-scaling framework for virtualized datacenter management. A salient feature of Tide is its fast capacity-provisioning algorithm that supplies just-enough capacity for the management middleware. We evaluate the effectiveness of Tide with both synthetic and real world workloads. Our results show that the self-scaling capability in Tide can substantially improve the throughput of management tasks during management workload bursts while consuming a reasonable amount of resources.

Categories and Subject Descriptors

C.4 [**Performance Of Systems**]: Performance Attributes; C.5.5 [**Computer System Implementation**]: Servers

General Terms

Management, Performance, Design

1. INTRODUCTION

Datacenter virtualization has attracted attention in recent years due to various benefits it offers. It saves total cost of ownership by consolidating virtual servers[18] and virtual desktops[19]. It also provides high availability to applications that are not designed with this feature by encapsulating them within a highly-available virtual machine (VM)[17]. Virtual machine live migration[16, 7] can nearly eliminate downtime of applications by allowing running virtual machines to be migrated off physical machines that require maintenance, and can also save power[15] during non-peak hours by consolidating virtual machines onto a small number of servers and shutting down the rest. More recently, virtualization has enabled the proliferation of cloud computing platforms such as Amazon's EC2[4], where virtual machines in a datacenter can be provisioned on demand and rented based on usage as if they were physical machines.

In order to provide features like high availability, live migration, and power management, virtualized datacenters rely on a rich management middleware layer. This layer provides unified management of physical hosts, virtual machines, virtualized network switches, and storage devices. One example of a virtualized datacenter management middleware layer is VMware vSphere[20]. VMware vSphere provides three key management functionalities. First, it executes various manual or automated management tasks. For example, if an administrator needs to provision virtual machines to serve as remote desktops for new end-users, the administrator sends these provisioning tasks to the management layer, which creates the new virtual machines and places them automatically on physical hosts. In addition, vSphere may continuously balance the workload of applications running in VMs via automatic live migration[7]. The second functionality provided by vSphere is monitoring the runtime status of each physical server and the virtual machines hosted on them. Finally, vSphere maintains the configuration information of the components in the datacenter like the host hardware configuration, virtual network configuration, and storage device configuration.

The ability of the management layer to support such automated provisioning and load balancing tasks makes it an ideal platform for cloud-based applications. Cloud-based applications require elastic, automated control of resources to meet variable end-user demands. Without efficient resource controls, hot spots can arise in the infrastructure and the performance of end-user applications can suffer. Consider a system without automated load balancing: as more users are added and more VMs must be provisioned, certain hosts may become overutilized, and VM performance suffers. Hence, the performance of the management system can have a direct impact on cloud application performance and cloud user satisfaction[13]. Moreover, the management tasks themselves often must complete within specified time windows to satisfy the periodic maintenance cycles within the datacenter.

1.1 Management Workload

While a typical datacenter operator is often aware of the

resource consumption of end-user applications in the datacenter, it is important to consider the contribution of management tasks as well, especially when the management layer is used in cloud-like environments. For example, consider the cost of powering on a VM. If the management layer is performing automated load balancing, it must select the appropriate host to run the VM[15]. This selection may require a complex series of computations involving load inspection on hosts and compatibility checking between the capabilities of a host and the required capabilities of the VM. For example, the management layer must not only choose a host that has sufficient resources, but must also pick a host that can support the VM's demands (e.g., hardware-based virtualization or connection to a specific iSCSI device). This type of computation can be resource-intensive, mostly CPU and memory intensive, for the management layer, and the resource usage can grow with the number of hosts and VMs, and can also grow if multiple VMs are powered on concurrently.

The management workload consumes considerable resources not just because individual management operations may be expensive, but also because the workload itself is bursty[13]. Figure 1 shows the CDF of management task rates within a 10-minute time window for a production datacenter. While the average workload is fairly small, the peak workload can be nearly 200 times higher than the average workload. Examples of such bursty workloads are prevalent[13]. Companies using virtual machines as remote desktops may perform large numbers of virtual machine cloning and reconfiguration tasks during a small time window to accommodate new users. Moreover, in remote desktop environments, a so-called 'boot storm' of power operations may occur when employees arrive at work. Other examples of potentially-bursty management operations include large-scale virtual machine provisioning for software-as-a-service, live-migration-based datacenter-scale load balancing, and massive security patching. In general, the ability of virtualization to provide automated management can also lead to bursts in management workload in order to accomplish such tasks in specified maintenance windows.

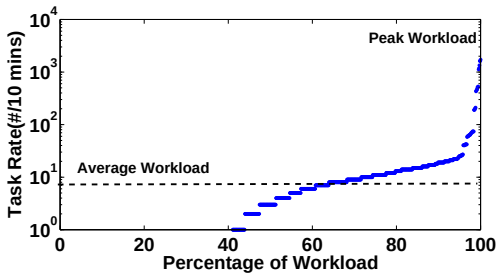


Figure 1: Burstiness of Management Workload

This workload burstiness may further grow in the near future because of two trends. First, increased CPU cores per host may lead to more virtual machines per host, leading to a larger number of entities to be managed within a virtualized datacenter and a larger number of management tasks to be performed on a daily basis. Second, as more datacenters become multi-tenant and support public or private cloud computing platforms, the number of different yet overlapping management workflows that must be accommodated will increase. For example, if a datacenter holds servers that are shared between different companies with different backup strategies (backup quarterly vs. backup bi-weekly), the load on the management layer can be bursty in unpredictable ways, and this burstiness increases as more customers share a given datacenter.

1.2 Challenges in Handling Workload Bursts

As the preceding discussion indicates, the management layer must deal with a diversity of tasks at various intervals and with various resource demands. The capacity of the management layer determines its ability to respond to a management workload burst. For example, if a single server is responsible for servicing all management tasks, it can become a bottleneck, especially if the management tasks are compute-intensive. Another common scenario is spawning a large number of web server VMs within a short period of time to accommodate a flash crowd. The longer it takes for the management layer to create such VMs, the worse the performance for the end user, and the higher potential for an SLA violation. Moreover, if the management layer is busy with such a burst and is unable to perform automated load balancing in a timely manner, end-user performance may also suffer. Each of these scenarios highlights the need for a flexible, elastic management layer to complement the elasticity of the application infrastructure.

Providing the right capacity for the management layer is challenging: sizing for regular workloads may induce excess management task latency when a burst arrives; sizing for the peak may cause a lot of resources to go unused during normal periods, resources that could have been used for applications themselves. We argue that one possible solution to efficiently handle management workload bursts is to make the management system self-scaling, which allows the management system to automatically boost its capacity during peak workloads and drop its capacity during normal workloads.

1.3 Contribution

In this paper, we introducing Tide, a prototype management system with dynamic capacity that scales with management workload. Tide leverages VMware's vSphere[20] management layer. It consists of a primary management server and a number of virtual management server instances that are powered off during normal workloads. When Tide encounters bursts in the management workload, it dynamically powers on additional virtual management server instances to boost overall throughput.

One fundamental requirement of self-scaling is that provisioning of management instances should be fast. In addition, to most efficiently utilize available resources, the number of provisioned instances should be reasonably small. We devise a novel provisioning algorithm that can *quickly* provision *just enough* server instances to the management system for the current workload. The algorithm considers the management system as a black-box and requires no performance modeling of server instances or management workload profiling.

To the best of our knowledge, Tide is the first work dedicated to the design and implementation of self-scaling in virtualized datacenters management systems. To evaluate the effectiveness of Tide, we perform extensive experiments with both synthetic and real-world management workloads collected from several virtualized datacenters. Our experi-

mental results show that Tide can quickly react to workload bursts and efficiently execute large numbers of concurrent management tasks. It also consumes much fewer resources compared with systems implemented with conventional provisioning techniques.

1.4 Organization

The rest of the paper is organized as follows. Section 2 gives an overview of Tide. We describe the fast-provisioning algorithm of Tide in Section 3 and present performance results in Section 4. We discuss related work in Section 5, and provide concluding remarks and directions for future work in Section 6.

2. SELF-SCALING AND TIDE OVERVIEW

The intuition behind Tide is to treat the management workload similar to how one would treat an application workload: when a large number of management tasks must be performed, Tide allocates new management instances to handle the additional load, thus preventing a single management instance (or a small number of statically-partitioned management instances) from becoming a bottleneck in executing the management workload. When the burst subsides, these management instances can be deallocated and their underlying physical resources can be used for the end-user application workloads, allowing more efficient use of the underlying infrastructure. We refer to this automatic provisioning as self-scaling.

To understand the benefits of such an approach, consider a scenario in which an administrator must live-migrate 500 VMs in a period of 30 minutes to prepare for routine maintenance on a datacenter rack. Live migration involves VM-host compatibility computations, placement computations and continuous tracking: such a compute-intensive workload may saturate a single management server. With multiple dynamically-allocated management servers, the management layer can parallelize these computations and minimize the overall latency. Compared with static provisioning based on peak workload (e.g., always keeping 30 management servers running instead of 1), self-scaling provisions resources only during peak periods, potentially saving considerable resources. Previous work has shown that such bursts can be frequent[13] and may last for several hours. Given that administrators prefer predictable time windows and want to minimize any possible business disruptions, it is important to reduce management task latency as much as possible. Tide seeks to reduce the latency to perform management tasks and also limit the resources used by the management layer.

Tide is a distributed system consisting of multiple management server instances, each of which is a vSphere management server. Specifically, Tide has one primary instance and multiple virtual instances. The primary instance is a vSphere server installed on a physical host, while virtual instances are virtual machines running vSphere. The primary instance provides the management interface, and all virtual instances are transparent to administrators. During normal workloads, the primary instance manages all hosts and VMs in the datacenter and all virtual instances are powered off. When encountering workload bursts, the primary instance dynamically powers on virtual instances based on perceived workload and dispatches workload to these virtual instances. By parallelizing task execution, Tide tries to maximize throughput to avoid delaying task execution.

One of the main challenges of self-scaling involves provisioning an appropriate number of virtual management instances quickly enough to benefit the management workload and yet consume minimal resources. Provisioning speed and efficiency are both important because they determine management task execution delay and the overall management cost. As a concrete example, consider the previous scenario of live-migrating 500 VMs within a 30 minutes time window. If the provisioning process takes a long time to complete, task completion delay would be inevitable as Tide has to serialize portions of the migration workflow due to insufficient instances. One may suggest to achieve fast provisioning by simply adding a large number of instances during workload bursts. While this would certainly increase parallelism, the utilization of the provisioned instances may be low as the bottleneck may shift to the network subsystem instead. Moreover, it would also cause the management system to compete resources with user applications.

Rather than trying to develop a model for the highly-variable and unpredictable management workload, Tide leverages an on-line measurement scheme that reactively provisions new instances in an iterative manner. To achieve provisioning speed and efficiency, it continuously measures the change in task execution throughput to determine the best number of instances to provision in future iterations. We discuss the detail of this approach in the rest of the paper.

3. FAST AND EFFICIENT PROVISIONING

Directly estimating the appropriate number of management instances for a given management workload is difficult partly because the resource consumption of management tasks is hard to predict. Different tasks consume CPU and IO quite differently. For example, powering-on a VM involves computations for choosing the best host to run a VM, while cloning a VM is disk-intensive. Even tasks of the same type vary heavily in execution cost due to task heterogeneity. Another difficulty with estimating the number of management instances is that the performance of a management instance is different when it runs on different hosts (e.g., hosts with different CPU types). Capturing these uncertainties with performance modeling is difficult, and also makes provisioning system-dependent.

One possible solution is to use iterative schemes which repeatedly add new instances into the system until doing so does not improve throughput. For such schemes, one difficulty is to determine the number of instances to add at each iteration, a.k.a *the step size*, and straightforward schemes often do not work well. For instance, a scheme that adds a constant number k of instances each time faces a *speed-efficiency* dilemma. If k is small, the scheme may take many iterations and a long time to provision sufficient instances. If k is large, it may unnecessarily provision a large amount of instances for small workload bursts, which causes resource contention between management and applications.

In Tide, we devise a novel adaptive approach that can *quickly* provision *just enough* instances for a workload burst. It is system-independent as it does not rely on specific workload or system performance models. The key technique in our approach is monitoring the speedup in management task throughput and appropriately adding more management instances during the next iteration. We next present details of this approach.

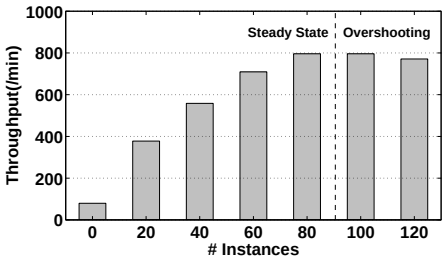


Figure 2: Throughput with Increasing Instances

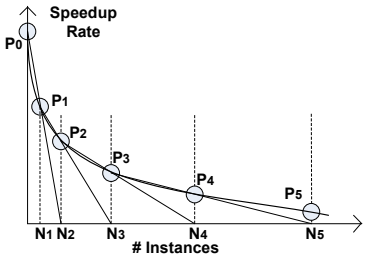


Figure 3: The Speedup Rate Curve

3.1 Speedup-Guided Provisioning

Figure 2 shows the relationship between the number of management instances, denoted by N, and the task execution throughput. Here, we ran a workload trace from a production datacenter on Tide multiple times, with an increasing number of management instances each time. More details on setup can be found in Section 4.

As N increases, the task throughput increases until the throughput of the system matches the workload. After that, the task throughput levels off even with larger N, because the gain in throughput is limited but the cost of coordinating management instances continues to grow. We refer to the throughput increment after adding one or more instances as *throughput speedup*, and the state where the throughput levels off as *steady state*. In addition, we refer to the situation of provisioning after steady state as *overshooting*. Overshooting is clearly undesirable as it wastes resources. Ideally, we should stop provisioning new instances when the system enters the steady state. The question is, how do we reach this state as quickly as possible without causing overshooting?

It is important to note that as the throughput approaches steady state, the change rate of speedup in throughput decreases. Tide uses this change rate information to guide the provisioning process. It iteratively adds a certain number of instances to the management system based on previous speedup change rate. This feature allows us to quickly approximate the steady state.

We use $T(N)$ to represent the throughput given N instances. Since speedup is a function of instance number N, we use $f(N)$ to denote the speedup of N instances. Note that $f(N)$ is the throughput increment of the system from $N-1$ instances to N instances, i.e. $f(N) = T(N)-T(N-1)$. Figure 3 shows the curve of $f(N)$ generated from Figure 2. The system enters the steady state when f reaches the X axis, i.e. $f(N_s) = 0$ where N_s is the number of management instances at steady state. Thus, the estimation of the steady state can be formulated as a root-finding problem.

Our provisioning algorithm is based on Secant method[14], a widely used root-finding technique in numerical analysis. The algorithm combines the root-finding process with the provisioning process. For the first iteration, we measure the initial speedup by adding one instance, i.e. $f(N_0)$ where $N_0 = 1$. Clearly, $f(N_0) = T(1) - T(0)$ where $T(0)$ is the throughput of the primary vSphere server. The value of $f(N_0)$ is shown by the P_0 in Figure 3. Similarly, we then add a fixed small amount of instances to make the total instance number to N_1 and measure the speedup at N_1, i.e. $f(N_1) = T(N_1)-T(N_1-1)$, as shown by the point P_1. Based on P_0 and P_1, we find the number of instances to provision in the next iteration, N_2 as follows. We generate a linear function $S = g(N)$ which passes the point P_0 and P_1. The root of $g(N)$, as shown by N_2 in Figure 3, is the number of instances to provision in the next iteration. Formally, given $N_i, N_{i-1}, f(N_i), f(N_{i-1})$, we generate the number of instances to provision in $(i+1)$-th iteration as follows,

$$N_{i+1} = N_i - \frac{N_i - N_{i-1}}{f(N_i) - f(N_{i-1})} f(N_i) \quad (1)$$

The provisioning process repeats the above iteration by using two previous provisioning points to determine the next one. It terminates when the speedup improvement rate between the two most-recent provisioning is below a predefined threshold γ. Note that users can set γ based on their performance requirements on Tide and the amount of resources they can assign to Tide. In addition, we measure throughput based on multiple samples to avoid incorrect step size estimation due to unstable readings.

3.2 Restrictions For Robustness

To make our algorithm robust, we also apply three restrictions to our algorithm to prevent faulty provisioning.

Restriction 1 $N_{i+1} > N_i$ if $f(N_i) > 0$ ensure N_i is a increasing series (dealing with workload fluctuation)

Restriction 2 When $f(N_i) = 0$, gradually decrease N_i (overshoot prevention)

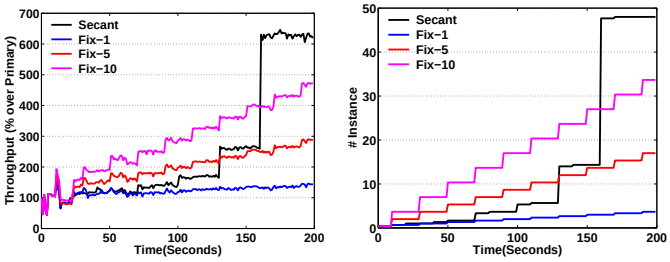
Restriction 3 Ensure $N_{i+1} < N_i+m$, where m is the maximum step size (addressing the divide-by-zero problem)

Note that restriction 2 is also the shrinking process, i.e. scaling-down, in which Tide reduces its management instances when workload bursts disappear. These restrictions ensure the algorithm provisions the right number of instances eventually. Due to space limitations, we refer the reader to our technical report[12] for more details on the provisioning algorithm.

The speedup-guided provisioning algorithm can look at speedup to better approximate the steady state. As an example, if recent provisioning leads to similar speedup, the algorithm would provision much more instances in the next iteration (because the generated linear function $g(N)$ has relatively small tangent and, accordingly, has a relatively large root). Clearly, this feature is desirable, as the system can quickly approach steady state with more instances when the performance is more predictable.

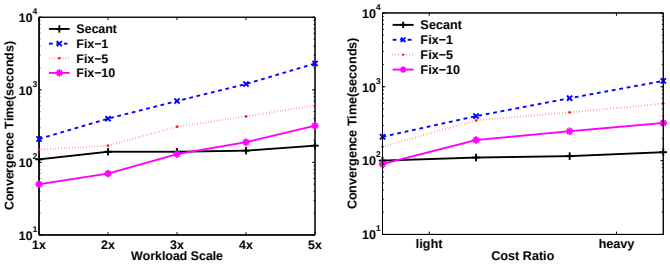
4. EXPERIMENTAL EVALUATION

Our setup includes a primary vSphere server and a total of 50 virtual instances of vSphere. The virtual instances are



(a) Throughput Trace (b) Instance Number Trace

Figure 4: Performance Trace During Self-Scaling



(a) Increasing Workload Scale (b) Increasing Weight

Figure 5: Convergence Time under Different Workload

VMs with vSphere 4.0 installed, and the VMs are placed in a suspended state so they consume no runtime resources unless they are used. The datacenter that we study contains 300 physical hosts with a total of 3000 virtual machines running on them. The primary vSphere server runs on a Dell PowerEdge 1850 with four 2.8GHz Xeon CPUs and 3.5GB of RAM. We installed vSphere 4.0 on all management instances.

We conduct experiments based on both real-world and synthetic workloads. The real-world workload is generated from traces we collected from several customer datacenters. The trace data includes management tasks performed over thousands of virtual machines in a three-year period. From this trace data, we generate two types of real world workloads. The first models short-term workload bursts. We use this workload to evaluate the effectiveness of Tide's self-scaling feature. The second models long-term management workload, which contains both regular and bursty workloads. We use this set of workloads to assess the long-term performance of Tide and its resource efficiency. By creating a composite trace composed of traces from several customers, we simulate the management workload of a cloud provider hosting virtualized datacenters of multiple enterprise users. Because of the variable resource demands of individual customers, this workload is heavier than that of an individual enterprise datacenter. We also use a synthetic workload to allow us to perform sensitivity analysis of Tide with respect to different workload types.

Figure 4 shows the throughput and the number of management instances activated by Tide during a self-scaling process when using different provisioning schemes. Here *Secant* refers to Tide's speedup-guided provisioning scheme and *Fix-N* is the simple provisioning scheme that adds N instances to the system if it observes throughput improvement in the previous iteration. The workload input for this figure is a management workload burst that lasts 200 seconds, and we normalize all throughput by the throughput of the primary instance. The speedup-guided scheme is faster at provisioning the proper number of instances as compared with all fixed-step schemes. It adds a small number of instances at first and gradually adds more instances in each iteration as it predicts the speedup change rate better. Although Fix-10 performs reasonably well, it causes significant resource consumption in the long run due to small and frequent workload bursts[12].

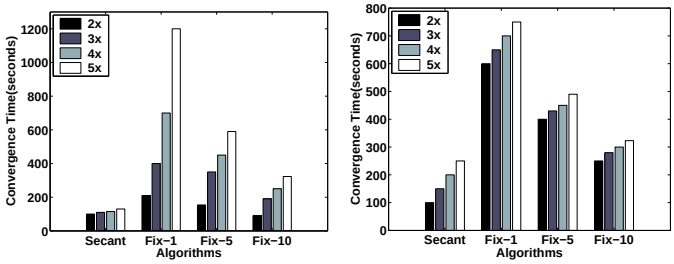
Figure 5 illustrates the convergence time of different provisioning schemes under different workload characteristics. The convergence time measures the time a scheme takes to provision the desirable number of instances (i.e., the time at which adding more instances does not provide sufficient throughput speedup). We use a synthetic workload to control the workload characteristics. In Figure 5(a), we push the incoming rate of tasks from its base level of 1x to 5x (5 times higher). It is clear that the speedup-guided scheme consistently uses less time to converge and its convergence time is barely affected by workload changes. The convergence time of fixed-step schemes such as *FIX-10*, while smaller than that of the speedup-guided scheme under small workloads, degrades with increasing workloads. Because our speedup-guided scheme is robust to various workload types, it is particularly appealing in multi-tenant environments, where the management workload is highly variable. In Figure 5(b), we evaluate different schemes by increasing the workload intensity (i.e., resource demands). We rank different types of tasks by their CPU consumption at the management instance. The heavier a workload, the more CPU-intensive tasks it has. As before, the speedup-guided scheme outperforms fix-step schemes and is insensitive to workload changes.

In Figure 6, we study the convergence time of different provisioning schemes under different types of workload bursts. Again, we use a synthetic workload as it allows us to create workload bursts of different types. Figure 6(a) shows the convergence time of different schemes under workload bursts whose task incoming rate increases from the base level (1x) to higher levels (2x-5x). We can see that the speedup-guided approach consistently achieves much shorter convergence time compared with other approaches. We can observe similar results in Figure 6(b) where the workload bursts drop from a higher level (2x-5x) to the base level (1x). Our approach has higher convergence time in declining bursts as it relies on fixed-step instance reduction to handle overshooting (Restriction 2). However, reducing instances from the steady state does not cause task execution latency.

5. RELATED WORK

There are a number of management systems for virtualized environments such as Usher[11], Virtual Workspaces[9], Cluster-on-demand[6], System Center[2], oVirt[3] and Enomalism[1]. Despite the large number of existing systems, their performance has not been studied in detail.

The concept of auto-scaling is not new, but we extend it beyond application-level scaling and apply it to the management workload. Application-level auto-scaling[5] dynamically adjusts the number of server instances running an application according to application usage. Heinis and *et*



(a) Growing Bursts (b) Declining Bursts

Figure 6: Convergence Time under Different Bursts

al[8] discussed similar ideas in a distributed workflow engine with self-tuning and self-configuration capabilities. In fact, application-level auto-scaling further increases the intensity and burstiness of the management workload, e.g. creating many web server VMs to handle a flash crowd of HTTP requests requires fast VM cloning and power-on. We speculate that Tide may improve application auto-scaling.

The execution of management tasks in a virtualized datacenter involves both the management system and hosts running virtual machines. SnowFlock[10] studies rapid group-instantiation of virtual machines across a group of physical hosts. This work is complementary to ours as we focus on efficient task execution in the management system.

6. CONCLUSIONS AND FUTURE WORK

The prevalence of on-demand computing requires elastic mechanisms for dynamic resource allocation for end-user applications, and virtualization is increasingly being used to deliver such mechanisms. As more and more datacenters move to virtualization and provide multi-tenancy, the management tasks comprise a workload that tends to be bursty and unpredictable, and statically partitioning of management resources may result in both wasted resources and poor compliance with application SLAs. Automatically scaling the management infrastructure is one way to avoid these issues, although it is crucial that auto-scale must provide best performance at minimal resource cost.

In this work, we study the problem of self-scaling of the management infrastructure. We propose Tide, a prototype auto-scaling management middleware. To meet the unique requirements in provisioning speed and efficiency, Tide employs a novel speedup-guided provisioning algorithm. Experimental results show that Tide significantly reduces management task execution delay with efficient resource usage.

To the best of our knowledge, Tide is the first work that employs self-scaling to address management workload bursts in virtualized datacenters. We believe the intensive and bursty nature of management workload is an important problem to virtualized datacenters, especially those that serve as the underlying infrastructure for cloud computing platforms. While we have made an initial attempt to address this problem with Tide, it certainly deserves further study. As part of our ongoing work, we are extending Tide to support multiple management systems with shared resource pools.

7. ACKNOWLEDGMENTS

The first author did the initial work for this project during his internship at VMware. The authors would like to thank Jennifer Anderson at VMware for her strong support to this work. The first two authors are also partially supported by grants from NSF ISE NetSE program, CyberTrust program, an IBM faculty award, IBM SUR grant and a grant from Intel Research Council.

8. REFERENCES

[1] Enomaly homepage. http://www.enomaly.com/.
[2] Microsoft System Center. http://www.microsoft.com/systemcenter.
[3] oVirt home page. http://ovirt.org/.
[4] Amazon. Amazon web service(aws). http://aws.amazon.com.
[5] Amazon. Auto scaling. http://aws.amazon.com/autoscaling/.
[6] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC*, pages 90–103, 2003.
[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.
[8] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. In *ICAC*, pages 27–38, 2005.
[9] K. Keahey, I. T. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific Programming*, 05.
[10] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: rapid virtual machine cloning for cloud computing. In *EuroSys*, 09.
[11] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An extensible framework for managing clusters of virtual machines. In *LISA*, pages 167–181, 2007.
[12] S. Meng, L. Liu, and V. Soundararajan. A self-scaling management system for virtualized datacenters. http://cc.gatech.edu/∼smeng/papers/tide-report.pdf.
[13] V. Soundararajan and J. M. Anderson. The impact of management operations on the virtualized datacenter. In *ISCA*, 2010.
[14] E. Süli and D. F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, '03.
[15] VMware. Distributed resource scheduling and distributed power management. http://www.vmware.com/products/drs/.
[16] VMware. vMotion. http://www.vmware.com/products/vmotion.
[17] VMware. VMware HA. http://www.vmware.com/products/high-availability/.
[18] VMware. VMware Server Consolidation. http://www.vmware.com/solutions/consolidation/.
[19] VMware. VMware View. http://www.vmware.com/products/view/.
[20] VMware. vSphere. http://www.vmware.com/products/vsphere/.