OnCall: Defeating Spikes with a Free-Market Application Cluster

James Norris, Keith Coleman, Armando Fox, George Candea
Department of Computer Science, Stanford University
{jcn, keith, fox, candea}@cs.stanford.edu

Abstract

Even with reasonable overprovisioning, today's Internet application clusters are unable to handle major traffic spikes and flash crowds. As an alternative to fixed-size, dedicated clusters, we propose a dynamically-shared application cluster model based on virtual machines. The system is dubbed "OnCall" for the extra computing capacity that is always on call in case of traffic spikes. OnCall's approach to spike management relies on the use of an economically-efficient marketplace of cluster resources. OnCall works autonomically by allowing applications to trade computing capacity on a free market through the use of automated market policies; the appropriate applications are then automatically activated on the traded nodes. As demonstrated in our prototype implementation, OnCall allows applications to handle spikes while still maintaining inter-application performance isolation and providing useful resource guarantees to all applications on the cluster.

1. Introduction

Today's Internet application clusters face severe limitations in responding to major traffic spikes and flash crowds. A typical cluster runs a single application on a fixed set of machines, each of which may require hours of configuration time before becoming operational. Were they sufficiently overprovisioned to handle all spikes, clusters would become exceedingly expensive and would waste resources while idling during more typical traffic workloads.

As a more efficient alternative, we developed a shared, dynamic application cluster design based on virtual machines (VMs). The system is dubbed "OnCall" for the extra computing capacity that is always on-call in case of traffic spikes. OnCall is a cluster management system designed to multiplex several (possibly competing) e-commerce-type applications onto a single server cluster in order to provide enough aggregate capacity to handle temporary workload surges for a particular application while guaranteeing some capacity to each application in steady state.

Our development efforts focus on a marketplace-based resource manager that enables applications to handle traffic spikes by acquiring additional computing capacity from other applications on the cluster. OnCall accomplishes this while still maintaining inter-application performance isolation and providing useful performance guarantees to all applications on the cluster. Allocation decisions (and server switches) are fully automated and run without human intervention.

OnCall is targeted for applications that serve dynamic content; much work has already been done to address the simpler problem of handling spikes in the realm of static content. OnCall does not address surges resulting from denial-of-service (DoS) attacks. Rather, many of the other research or industry solutions to detect and filter DoS-driven requests can be run in front of an OnCall cluster to reduce the influence and effects of a DoS attack.

1.1. Spike handling is the critical problem

Provisioning for load in steady state is the "easy" case, while spike handling is the real challenge. Much useful work on allocation policies attempts to optimize resource allocation for applications in steady state. However, even with the temporal traffic swings that do occur in steady state, resources aren't constrained, so even the simplest hosting solutions like over-provisioned, fixed-size clusters will fulfill performance need; but during a spike, resources are highly constrained such that maintaining desired performance becomes difficult or impossible.

The primary challenge for hosting services, then, is to provide adequate performance during unexpected spike conditions. Beyond that, if hosting platforms can handle steady state "well enough"—that is, with at least the same efficiency and performance guarantees of a static cluster—then applications will be well off. We intend to demonstrate that OnCall not only handles the difficult case of spikes but also provides a mechanism for economically-efficient resource allocation during steady state.

1.2. Contributions and organization

The primary contribution of this work is a spike management system for shared hosting clusters that relies on an economically-efficient marketplace for computing resources to allocate capacity between potentially competing applications. An implementation also demonstrates the use of VMs as a platform for shared clusters that serve dynamic content.

The remainder of the paper is organized as follows: Section 2 describes the OnCall Marketplace, the mechanism through which resource allocation decisions are made. Section 3 discusses the details of the OnCall Platform, including the VM-based implementation. Section 4 discusses OnCall's strengths. Section 5 provides experimental validation of the system. Section 6 presents areas for future work. Section 7 compares OnCall with related work. Section 8 concludes.

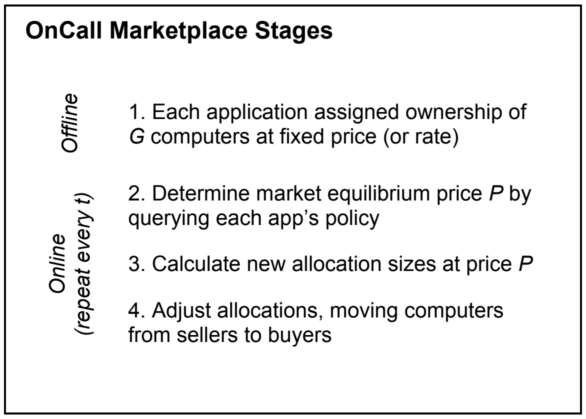
2. OnCall Marketplace

The OnCall marketplace facilitates resource allocation between various, possibly competing, applications on the cluster. Each application has the ability to incrementally grow its capacity by adding additional nodes to its allocation, and decrease its capacity by releasing nodes currently allocated to it.

2.1. Marketplace operation

As shown in Figure 1, each application is initially statically assigned ownership of a fixed subset of the cluster, such that every cluster node is statically owned by exactly one application. Applications negotiate their static allocations offline, based on their own estimates of expected traffic and desired steady-state utilization. In other words: there are N applications; each application i pays a fixed price (or rate) at configuration time for ownership of G_i cluster nodes such that $\sum G_i$ is the total number of nodes on the cluster. The fixed price for the G_i nodes could either be a one-time fixed price or regular fixed rate, such as a monthly fee that might be agreed upon in a month-to-month hosting contract.

Normal operation is divided into time quanta of length t (typically on the order of minutes). At the beginning of each time quantum, any application can offer to rent spare capacity on its statically-owned nodes to other applications or borrow extra capacity from other applications' static allocations. If an



OnCall Marketplace Stages

Offline

1. Each application assigned ownership of *G* computers at fixed price (or rate)

Online (repeat every t)

2. Determine market equilibrium price *P* by querying each app's policy

3. Calculate new allocation sizes at price *P*

4. Adjust allocations, moving computers from sellers to buyers

Figure 1. OnCall marketplace has an offline stage that occurs once, and a series of online stages that are repeated every time quantum *t*.

application rents nodes (beyond its own G_i nodes) from another application, it pays that application the agreed upon price.

An application defines the number of nodes it wishes to use by providing a deterministic policy function that maps from *(price to rent 1 node during this quantum)* to *(number of nodes it would be willing to rent at that price)*. The policy function is also provided with usage and performance statistics to help it make allocation decisions.

The marketplace determines an equilibrium rental price, P, by conducting a binary search on the price space, querying each application's policy until the sum total number of nodes desired by all applications is equal to the number of nodes on the cluster. If exact equilibrium is impossible, the marketplace selects the highest price at which the total number of nodes desired is greater than the number of nodes on the cluster. Since this means there are more nodes to be allocated than are actually available, strict allocation priority goes to those applications that would have been willing to pay 1 unit more. Effectively, the Marketplace should behave much like a Dutch Auction where the buyers and sellers are the same parties. Once the equilibrium solution is determined, the resources are allocated and money changes hands. At the end of the time quantum everything starts over.

Some additional notes and assumptions:

- The cluster machines are assumed to be homogeneous and all nodes are rented at the same price. This assumption is made purely for simplicity—there is nothing fundamental about OnCall that imposes this limitation.

- There is a time delay cost when starting up and shutting down applications on nodes that change hands (see Section 4 for details). However, if a

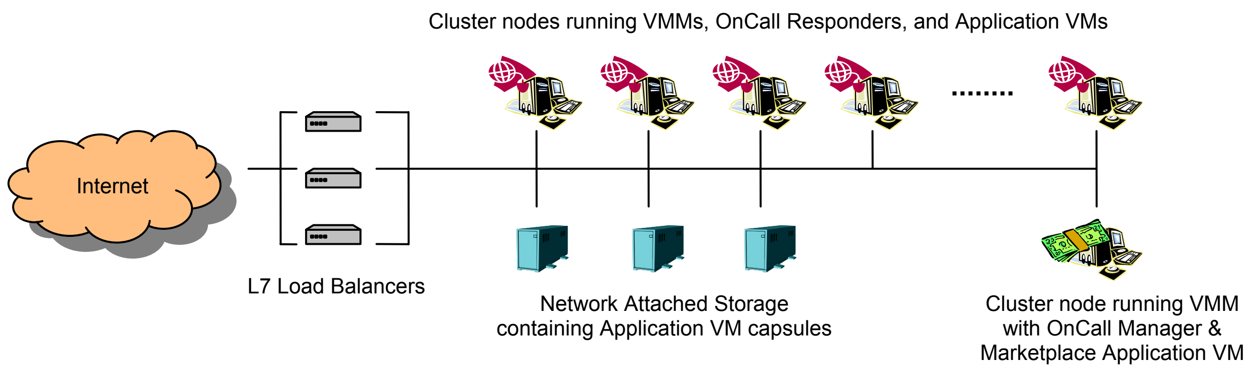

Cluster nodes running VMMs, OnCall Responders, and Application VMs
....
Internet
L7 Load Balancers
Network Attached Storage containing Application VM capsules
Cluster node running VMM with OnCall Manager & Marketplace Application VM

Figure 2. An overview of the OnCall platform.

rental contract is renewed in the next time quantum, the given application will continue to run on the same node.

- Applications only pay the market price, *P*, for those nodes above and beyond their static allocations. For example, if an application has 30 nodes in its static allocation but is using 40 nodes during a given time quantum, it will only pay the current market price for 10 nodes. Use of the first 30 nodes is pre-paid with the static allocation.

2.2. Resource guarantees

The OnCall marketplace provides useful resource guarantees (and the implied performance guarantees that follow) to its member applications. Namely, an application will be guaranteed use of the G_i computers it buys in the offline portion of the marketplace. If an application's market policy requests G_i or fewer nodes during any market round, it is guaranteed to receive the requested nodes and will not pay for their use (beyond the fixed price it is already paying). The only case where the application will have fewer than G_i nodes is when it decides to rent out use of some nodes to another application at the equilibrium price *P*. It can of course recapture those nodes in any future market round without paying additional fees.

The offline agreement of G_i is thus what provides the resource guarantee. A risk-averse application owner may thus choose G_i to be the number of nodes it would install in a static, fixed-size cluster, while a more risk-seeking owner (i.e. one who is willing to assume more risk for the chance to save or make money) may choose a lower G_i with the hope that extra capacity will be affordable when needed.

2.3. Market policies

Each application may develop its own custom market policy to accurately reflect its performance-to-dollar value relationship. OnCall provides every custom policy engine with performance statistics (in the form of CPU and disk usage) for every node on which its application is running.

This research is not focused on developing practical or effective policies, but rather on providing the mechanism to enable efficient allocation of resources through such policies. In fact, economically efficient policies can only truly be developed by application owners who understand the economics of their own application as well as the traffic patterns that their application typically experiences.

A simple market policy might look something like the following: the application owner knows or can calculate (a) expected number of nodes needed to handle current traffic, (b) dollar value of one additional user being served, (c) number of users each node can serve, and (d) price each additional node is worth, *p*.

The policy decision process is then two separate steps: (1) determine the number of nodes desired, n_i, and (2) if n is less than G_i, sell the excess nodes (excess = $G_i - n_i$) for any price; if n is greater than G_i, use all G_i nodes and if $p <= P$, buy to $n_i - G_i$ extra nodes at price *P*.

Additional policy features can be used to avoid thrashing, such as (a) use the max of short and long-term historic usage averages as the expected number of nodes needed, or (b) employ an idle resource tax or similar technique to control overprovisioning.

3. OnCall Platform

OnCall operates on a cluster of computers linked together by a high-speed LAN and connected to the public Internet through a wide-band link (Figure 2). Each physical machine in the cluster runs a virtual machine monitor (VMM) such as VMware GSX server. Individual web applications are packaged together with their associated, configured operating systems into VM capsules that reside on network attached storage. It is of course assumed that all

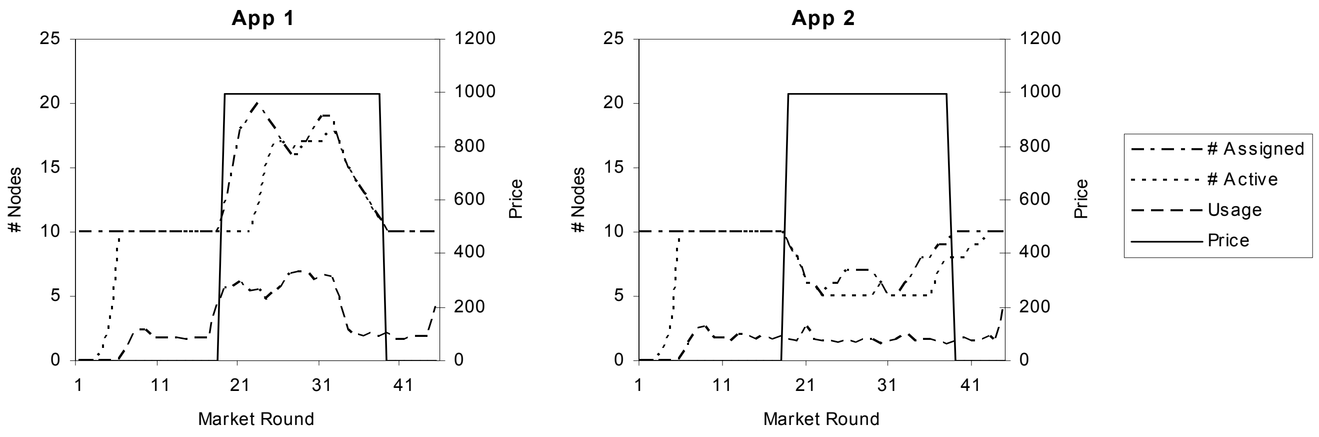


Figure 3. In the first simulation, App 1 experiences a spike and rents extra capacity from App 2.

application components managed by OnCall be can be replicated simply by adding nodes.

Between the OnCall cluster and the Internet are load balancers that redirect requests to the appropriate physical machines, depending on which machine is running which application at a given time.

One cluster node runs the special OnCall Manager application. The Manager is the central decision maker on the cluster, containing the marketplace as well as each application's market policy engine.

The remaining cluster nodes run OnCall Responders, which talk to the Manager, stopping, starting and booting new application VM capsules on their respective machines, as necessary. The VM-based architecture provides a number of benefits, including fast node activation, interapplication security, and application generality, as application components can run on any physical node with their own custom-configured operating systems.

3.1. Runtime operation

Standard runtime operation runs in cycle with each marketplace round. A full cycle runs as follows:

(1) Manager employs Marketplace to calculate an equilibrium market price used to determine the resource allocation for each application.

(2) Manager decides which application each physical node will run, minimizing the number of shutdowns and startups.

(3) Manager signals each node's Responder to shut down the current app and subsequently start up the new one, should a switch be necessary.

(4) At the end of the round, Manager gathers updated usage and performance statistics from each node and reports them to the respective application's policy engine.

3.2. Multi-tiered applications

The platform provides an interface that allows a multi-tiered application to replicate its various components in different numbers. For instance, an application could specify that of its N nodes, m should be front end servers while p should be middle-tier application servers. Databases must currently be fixed onto specific nodes, but in the future we hope to enable replication of data stores that provide such capabilities.

4. Simulation results

Our simulation test platform consists of a 40 node cluster. Each node is configured with dual 1.0 GHz Intel Pentium III CPUs, 1.5 GB RAM, and dual 36 GB hard disks; nodes are connected via gigabit Ethernet. We employ VMware's GSX Server as the VMM, running on top of Linux 2.4. OnCall's Marketplace was set to use a time quantum of 30 seconds. Load was generated using the Apache JMeter testing tool.

Our simulations demonstrate four important traits of the OnCall system: (1) its ability to handle traffic spikes under unconstrained resources, (2) its ability to handle traffic spikes under constrained resources, (3) its ability to provide resource guarantees in the face of spikes and constrained resources, and (4) fast server-switching and boot time performance.

4.1. Spike handling and profit earning

Our first simulation (Figure 3) demonstrates how an OnCall application can handle a traffic spike by renting nodes from other applications.

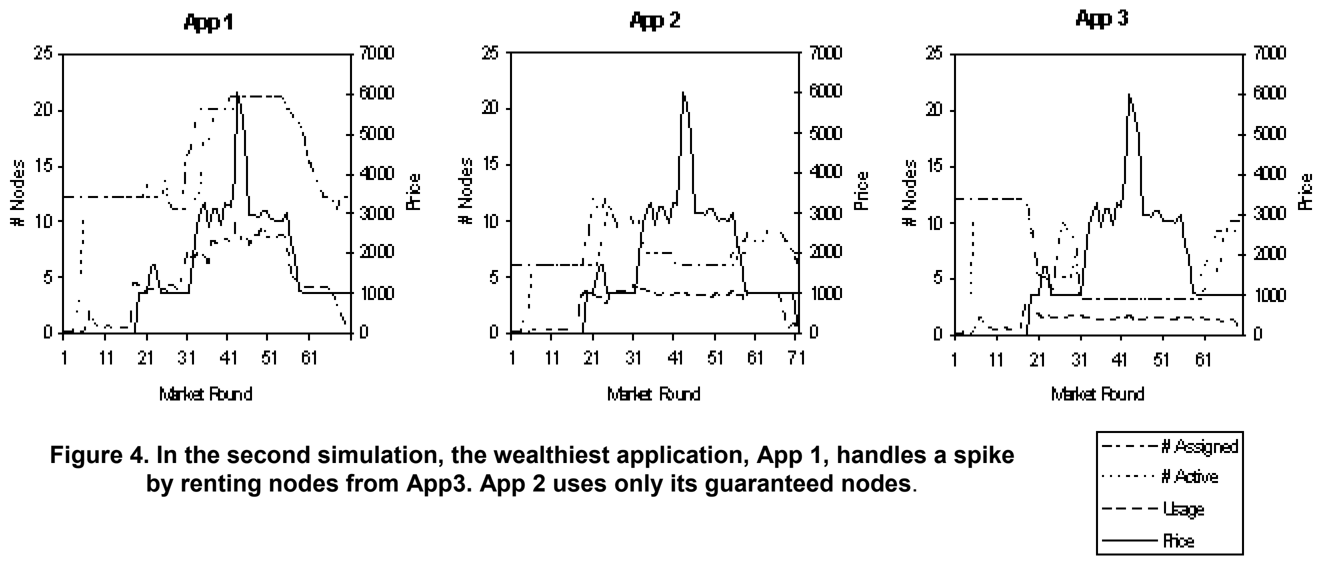


Figure 4. In the second simulation, the wealthiest application, App 1, handles a spike by renting nodes from App3. App 2 uses only its guaranteed nodes.

This test was run on a cluster with 30 nodes. Three applications are running on the cluster, each with a G_i of 10. Between Market Rounds 19 to 34, App 1 experienced a traffic spike that required more than its own 10 nodes to handle. Thus, throughout this period, it rented extra nodes from Apps 2 and 3 at the market equilibirum price of 999 (in abstract currency). It continued to rent these nodes until Round 40 at which point its market policy determined that the spike was over and it was time to release the extra nodes.

The graphs also demonstrate the lag time between a node being assigned to an application and the application software becoming active on that node. The activation time delay varies on a number of factors and is discussed in greater detail in Section 4.3.

4.2. Resource guarantees in the face of spikes

Our second simulation (Figure 4) demonstrates OnCall's ability to provide resource guarantees in the face of spikes, despite the fact that cluster resources are constrained (e.g. the applications desire, but obviously cannot afford, more nodes than are on the cluster).

The cluster was configured with 30 nodes, 12 owned by Apps 1 and 3 each and 6 owned by App 2. In this case, App 1 experienced a gradually rising load from Rounds 18 through 55 that eventually required more than its given 12 nodes to handle. At the same time, App 2 also experienced additional traffic that caused it to look to the marketplace for additional nodes. With both Apps 1 and 2 looking to buy nodes and App 3 experiencing a low usage rate, App 3 was willing to rent out many of its unused nodes for a high price and keep only what it considered necessary to serve its requests. Because App 1 was configured with a higher budget than the other two applications, it drove up the market price between Rounds 43 and 55 to a point where App 2 could no longer afford to rent more than its original 6 guaranteed nodes. Thus, App 2 operated only on its guaranteed nodes during this period and subsequently began renting additional nodes at Round 56 when App 1's spike subsided and the price became affordable.

4.3. Fast server activation

Our third test demonstrates the speed with which OnCall can activate new application nodes:

Table 1. OnCall activates nodes at least two times faster than a "standard" dynamic cluster platform that does not use VMs.

Platform	OnCall Optimal	OnCall Limited	Standard with OS	Standard w/out OS
Time until Active (s)	5-10	50-120	270-330	710-750

To obtain the first two data columns we timed application activations over a number of runs. "OnCall Optimal" is the case where OnCall is able to load VMs from a suspended state and resume them on new cluster nodes. Unfortunately, because of limitations in VMware's MAC address controls, OnCall is generally required to boot a VM from a shut down state in order to ensure that each node has a unique MAC address. The "Optimal" run times were acquired in cases where OnCall activated only a single instance of an application. The full boot process (the "Limited" case) adds roughly one to two minutes to the activation time, depending on the number of simultaneous nodes accessing the same virtual disk on a central NFS file store. A distributed file store or caching system that bypassed the centralized bottleneck could consistently hold this time to under a minute.

Nonetheless, even in worst case "Limited" mode, OnCall nodes activate over two times faster than the best case for nodes on a more typical dynamic cluster platform such as Oceano. The data in the "Standard" columns are approximate activation times as measured on the Oceano system [1]. In the best case, when the appropriate OS is already installed on a given node, the Oceano system takes 4.5 to 5.5 minutes to activate a node; in the worst case, when an OS must first be installed, the activation time is 11.5 to 12.5 minutes. Both the OnCall and Oceano measurements were taken over relatively small samplings, rather than from rigorous performance analyses [1].

These speed improvements are significant, as OnCall's short activation times reduce the need to have accurate future traffic predictors. Since the system can activate new nodes quickly, predictors must only predict a short time (1-2 minutes) in advance.

5. Discussion

5.1. Marketplace optimality and fairness

The marketplace simulates a competitive market that, subject to certain conditions, is Pareto efficient and achieves the desired result that, within resource constraints, those applications with the most utility to derive from the use of additional nodes are given those nodes. Some of the conditions mentioned above fall under the category of preventing anti-competitive behavior, which is the subject of the following section. Other conditions involve assumptions about the shape of applications' utility curves.

An application's utility curve specifies the dollar value an application derives from possessing a certain number of nodes for a specific time quantum. We can say trivially that utility curves will always be monotonically non-decreasing—that is, it is never worse to own more nodes at a given total cost. The assumption we've made for optimality to hold in the current OnCall marketplace model is that utility functions are also smooth—that is, the marginal price an application is willing to pay for an additional node is a monotonically non-increasing function. In the case where this condition doesn't hold—for example, when an application requires nodes to be activated in pairs—then the application risks overpaying for some nodes if an exact equilibrium cannot be found. Since applications with these types of utility curves restrict the marketplace's flexibility, we believe that it is not inappropriate for the marketplace to restrict their flexibility. Nonetheless, we hope to examine additional market options, such as a multi-round marketplace, that could improve efficiency under these conditions.

5.2. Preventing market tampering

The efficiency of the OnCall marketplace rests on the existence of a free and fair market, so it is essential to prevent tampering of the marketplace. It is easy to envision scenarios in which tampering could occur: an app that owns a majority of cluster nodes artificially inflates the equilibrium price; an evil app waits until other apps are spiking, at which point it buys the vast majority of machines on the cluster at an extremely high price but for only a single time quantum, thus incurring little aggregate expense but forcing spiking apps to temporarily shut down.

In OnCall, just as in most real world markets, such tampering is prevented through regulation. Most of these measures are not technical in nature, but rather take place in the real world. Some examples might be:

- Ensure that a cluster with competing apps has enough distinct applications and sufficiently diverse "fixed" ownership allocations such that no monopoly or oligopoly exists.
- Always select the lowest of the possible equilibrium prices, making price inflation less likely in less competitive marketplaces.
- Fine or ban any application that engages in overtly anti-competitive behavior.

Though they are likely to be effective, the obvious downside of these solutions is that they rely on deterrence and retribution rather than direct prevention.

5.3. Competitive or cooperative

OnCall works in both competitive or cooperative environments. The system could be used internally within a single corporation to manage a cluster that runs all of the business's various services. Additionally, because of the security and performance isolation provided by the VM framework, OnCall could be used by a third-party hosting service to manage a cluster containing applications from varied and potentially competing owners.

5.4. Profit through efficiency

OnCall allows both application owners and hosting providers to financially benefit from efficient allocations of computing resources. Applications can gain by selling any capacity that they own but are not using. The hosting provider who owns the cluster can generate extra profits through two methods:

(1) The provider can shut down unused computers to save on utilities and maintenance cost.
(2) The provider can offer additional capacity on the cluster (above the sum of the applications' fixed size allotments) and sell that capacity when profitable.

Both goals can be accomplished through a host-owned "Shut Down" application that is willing to buy capacity at any price less than the base operational expense. The Shut Down application turns off nodes it is able to buy, then restarts those nodes when it can sell them at a price greater than the operational expense.

6. Future work

There are a number of platform features that were not critical for the prototype presented here, but would benefit a production system. We are currently developing and exploring these features: VM caching—cache VMs to local disk either speculatively or as they are read from network attached storage. Fault tolerance—add master-backup fault tolerance to the OnCall Manager. Performance statistics—provide market policies with additional statistics (e.g. end-to-end response time). Advanced policies—examine the use of feedback control loop theory in the creation of market policies. Scalable data layer—add support for new scalable persistent stores [2, 3] that would allow replication on the data tier. Multiplexing—study the trade-offs of running several applications on one node.

7. Related work

Much work has been done in the areas of cluster design and resource allocation. The work can largely be divided into mechanism- and policy-related work.

7.1. Mechanism-related work

Mechanism related work focuses on platforms for supporting either spike handling or dynamic clusters. Several projects are related to OnCall with regard to our use of VMs but lack a marketplace-type allocation policy mechanism. Jiang and Xu [4] make a case for the use of VMs on shared clusters for purposes of security. Figueiredo et al [5] make a case for the use of VMs in Grid computing, highlighting VMs' surprising efficiency. The Collective [6] provides a mechanism for managing VM capsule-based server updates and replication—some of the inspiration for OnCall came from this work. The vMatrix project [7] promoted a VM-based cluster that was much like a simplified OnCall cluster. Denali [8] makes the case for the use of a new, lightweight isolation kernel in a hosting cluster context instead of virtualizing at the hardware layer.

Other systems for load/spike handling differ from OnCall in their goals and approaches: IBM's Oceano [1] system is a dynamic cluster that can shift unvirtualized resources between various applications. A number of P2P systems [9, 10] [Padmanabhan] have been developed to handle flash crowds hitting privately hosted static content, whereas OnCall concentrates on dynamic content in shared hosting environments.

7.2. Policy-related work

Many contributions have been made in the realm of resource allocation policies on shared cluster systems, and the concepts developed therein could influence the design of OnCall market policies and increase their efficiency. Much of the work (Clockwork [11], Resource Overbooking [12]) focuses on optimizing allocation policies during steady state instead of rapidly changing demands. [13] focuses on static data, and its models don't directly apply to dynamic content. The "dynamic surge protection" approach of [14] uses regression analysis techniques to determine resource requirements for a given application, but does not provide a method for allocating resources between competing applications. A study by Chandra et al [15] demonstrated the effectiveness of resource multiplexing on shared application clusters and highlighted several potential optimization variables. Other work [16-18] is largely tangential but is worth mentioning for those interested in this area.

8. Conclusions

We proposed a spike management system for shared hosting clusters that serve dynamic content. The system, OnCall, relies on an economically-efficient marketplace for computing resources to reallocate capacity to spiking applications, as needed. Our prototype implementation demonstrates (a) OnCall's spike handling ability, (b) the economic gains and savings that accrue as a result of the resource marketplace, (c) the resource guarantees provided to each application, and (d) the speed benefits gained through the use of VMs.

In a broader sense this work demonstrates that market models are an effective and efficient way to allocate resources on shared clusters (particularly under spike conditions), and that even greater benefits would be gained from the expansion of market models to accommodate more complex application structures and market policies.

OnCall is able to automatically reallocate resources at a rate that current manually controlled systems cannot. OnCall makes decisions on the fly based on application-specified policies, increasing economic efficiency and reducing management overhead.

9. Acknowledgments

We would like to thank UC Berkeley's Mike Howard for his tremendous help in configuring our testbed cluster and Yoav Shoham for his insights into the economics of resource markets.

10. References

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar, "Oceano: SLA Based Management of a Computing Utility," *IFIP/IEEE Intl. Symposium on Integrated Network Management*, May 2001.

[2] A. C. Huang and A. Fox, "Decoupled storage: State with stateless-like properties," *Submitted to the 2004 USENIX Annual Technical Conference*, 2003.

[3] B. Ling, E. Kiciman, and A. Fox, "Session State: Beyond Soft State," *In Proc. First USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, March 2004.

[4] X. Jiang and D. Xu, "Protection of Application Service Hosting Platforms: an Operating System Perspective," *CS Technical Report TR-03-010, Purdue University*, February 2003.

[5] R. J. Figueirido, P. A. Dinda, and J. A. B. Fortes, "A Case for Grid Computing on Virtual Machines," *Proceedings of IEEE ICDCS 2003*, May 2003.

[6] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual Appliances for Deploying and Maintaining Software," *17th Large Installation Systems Administration Conference (LISA 2003)*, October 2003.

[7] A. Awadallah and M. Rosenblum, "The vMatrix: A Network of Virtual Machine Monitors for Dynamic Content Distribution," *7th Int'l Workshop on Web Content Caching and Distribution*, 2002.

[8] A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and Performance in the Denali Isolation Kernel," *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

[9] T. Stading, P. Maniatis, and M. Baker, "Peer-to-Peer Caching Schemes to Address Flash Crowds," *First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002.

[10] A. Stavrou, D. Rubenstein, and S. Sahu, "A Lightweight, Robust P2P System to Handle Flash Crowds," *10th IEEE International Conference on Network Protocols*, November 2002.

[11] L. W. Russell, S. P. Morgan, and E. G. Chron, "Clockwork: A New Movement in Autonomic Systems," *IBM Systems Journal*, vol. 42, no. 1, 2003.

[12] B. Urgaonaker, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

[13] P. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vadhat, "Model-Based Resource Provisioning in a Web Service Utility," *USENIX Symposium on Internet Technologies and Systems (USITS '03)*, March 2003.

[14] E. Lassettre, D. W. Coleman, Y. Diao, S. Froehlich, J. L. Hellerstein, L. Hsiung, T. Mummert, M. Raghavachari, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, and P. Ye, "Dynamic Surge Protection: An Approach to Handling Unexpected Workload Surges With Resource Actions That Have Dead Times," *IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, October 2003.

[15] A. Chandra, P. Goyal, and P. Shenoy, "Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers," *First Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.

[16] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated Resource Management for Cluster-based Internet Services," *USENIX Symposium on Operating Systems Design and Implementation (OSDI '02)*, December 2002.

[17] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "QoS-Driven Server Migration for Internet Data Centers," *Tenth International Workshop on Quality of Service (IWQoS '02)*, May 2002.

[18] T. Kelly, "Utility-Directed Allocation," *First Workshop on Algorithms and Architectures for Self-Managing Systems*, June 11, 2003.