Active Connection Management in Internet Services

Mike Y. Chen and Eric Brewer

Computer Science Division
University of California, Berkeley
{mikechen, brewer}@cs.berkeley.edu

Abstract

We propose a new connection management architecture for clustered Internet services called Active Connection Management (ACM) to improve the availability, quality of service, and manageability of Internet services. ACM extends the API of load-balancing switches to include application-level primitives. These primitives enable Internet services to manage load-balancing switches dynamically to control how client connections are mapped to physical resources as they enter the cluster. ACM provides several properties ideal for Internet services over static configuration. First, ACM increases availability through automated configuration, rolling reboots, and reduced server failure detection and recovery time. Second, ACM provides better load conditioning by enabling dynamic allocation of frontend resources and enabling more control over graceful degradation. Third, it provides better manageability by automating configuration, reducing human errors and improving scalability.

We present a design, implementation, and evaluation of an ACM system, called COMICS (COnnection Manager for Internet-Centric Services), that enables services to actively control how connections are mapped to servers. We believe that ACM primitives should be implemented into future switches and be utilized by application servers [24][25] to build more robust Internet service platforms.

1 Introduction

Internet services present several systems challenges. They must scale to support millions of users, both in terms of performance and manageability. They must be highly available and also be able to evolve quickly with minimal down time. Clustered servers are a natural solution to scaling Internet services, and have been studied extensively. Most Internet services architectures use clustered servers and load-balancing switches [17][18][19] to provide scalability and high availability. These switches, sometimes called "IP sprayers", spread connections across multiple physical servers to make them appear as a single virtual server.

Virtualization of physical resources provides transparent connection management to both clients and servers. On the server side, no modifications to servers are necessary. Each server simply receives a fraction of the total requests. On the client side, requests are typically sent to a pool of virtual IP addresses managed by one or more switches. The switches rewrite the destination address on incoming packets to forward the requests to physical servers. They also rewrite the source address in the reverse direction to return the results to clients. All requests appear to be processed by the virtual server, hiding the complexity of the physical server farm. To provide high availability, switches perform connection failover by using health checks to detect server failures and forwarding client connections to working servers only. The availability provided is such that if a request fails, a retry is independent of the previous request and has a high probability to be redirected to a working server and succeed.

We argue that more control over connection management should be exposed to applications, since applications have more information about resource needs and failures. A number of systems share our philosophy in exposing more control to applications. Scheduler activations [1], application-specific handlers [2], SEDA [4], and operating systems such as SPIN [3], Exokernel [11], and Nemesis [5] augment OS interfaces by giving applications the ability to specialize the policy decisions made by the kernel. We also argue that connection management should be automated to enable systems to be more dynamic and autonomous, rather than relying on manual configuration changes, which are slow and error-prone.

There are several benefits to exposing control over connection management. First, ACM enables flexible control over quality of service. Internet services using ACM can dynamically allocate frontend resources to

quickly respond to changes in load, continuously optimizing the system. Frontend servers can be added and removed from a cluster automatically without dropped connections and without human intervention. This is especially important for systems that host several services on shared resources. Since load for each service may change independently, we want to allocate more resources for those that are under high load. Although several middleware platforms support dynamic allocation of backend nodes, allocation of frontend nodes is only possible with ACM. ACM also enables services to degrade gracefully under overload by allowing services to configure switches to perform admission control at the point of entry.

Second, ACM improves availability through software rejuvenation without downtime, automated switch configuration, and application-assisted failure detection and recovery. Rebooting is a common form of software rejuvenation used to improve system reliability by clearing software bugs that accumulate over time, such as file descriptor and memo ry leaks. It can also be used to perform software upgrades that are version compatible. Rather than rebooting the whole system, rebooting is often performed in a rolling fashion to maintain availability. ACM can be used to automate the rolling reboot process without dropping client connections and without human errors. ACM increases availability through applic ation assisted failure detection and recovery. Even though most switches already perform Layer 2, 3, and 4 health checks and Layer 7 HTTP health checks, applications often have more knowledge about partial and transient failures and can remove faulty resources before the failures are visible to users. Applications can also detect Layer 7 failures other than HTTP. In addition, clustered middleware platforms already perform their own health checks, and can use ACM to complement the health checks performed by the switches.

Third, ACM improves manageability by automating switch configuration. Automated configuration is much faster than manual configuration, enabling configuration changes in less than a second rather than minutes or hours. Also, it eliminates human errors, which is one of the highest ranked causes of system failures [20][21][22][23]. In addition, ACM makes it easier to run multiple service instances on a single box, which is useful on a SMP if a service cannot run as multiple processes. For example, the Squid web server [15] is written as a single-threaded program, so each instance can only utilize one CPU, even if it's running on a SMP. With ACM, each instance can bind to a different server port and is automatically registered with the switch to receive client requests.

We present a design of an Active Connection Management system in section 2. An implementation is described in section 3 and evaluated in section 4. Section 5 presents three sample applications. Section 6 differentiates our work from other systems, and section 7 discusses improvements and experiences with COMICS. Section 8 summarizes our contribution.

2 Design

The primary goals for ACM are the following:

- Enable dynamic resource allocation to respond to changes in load.
- Automate switch configuration to eliminate human errors.
- Provide more flexible control over quality of service.
- Support for starting and stopping services with 100% availability to support online evolution and rolling reboots.

To enable services to manage connections effectively, we have defined the minimal primitives that an active web switch should support. We also present the design of a Connection Manager that enables existing web switches to support these primitives.

2.1 Connection Management Primitives

The core connection management primitives are Layer 3 and Layer 4 operations, and control how TCP connections and UDP streams get mapped to servers. They provide support for basic active connection management that work across all services that use TCP and UDP.

The core primitives are:

- **Add**: binds a physical IP/port to a virtual IP/port. The switch should perform Layer 2, Layer 3, and Layer 4 health checks to ensure such a physical service exists, before forwarding connections to it.
- **Remove**: unbinds a physical IP/port from a virtual IP/port. The switch should stop forwarding new connections to the physical server, but allow existing connections to be processed.
- **Force Remove**: unbinds a physical IP/port from a virtual IP/port immediately, dropping any existing connections.

- **List**: lists all bindings known to the switch. Used to synchronize bindings between the switch and the connection manager.
- **Drop**: drops a specified fraction of new connections to perform admission control. Since TCP performs SYN retries, it effectively queues client connections until SYN retry times out.

Performance improves if services can peek inside requests to distribute them to servers more intelligently, rather than randomly. For example, HACC [8] and LARD [9] partition the working set across physical servers to enhance locality to improve cache performance.

For web services, adding the following application-specific (Layer 7) primitives can improve service performance and robustness:

- **URL hashing**: provides locality-aware request distribution. It ensures that the same request always goes to the same server.
- **URL matching**: such as substring and suffix matching, provides content-aware request distribution. For example, requests that end with ".jpg" can be forwarded to servers that are optimized to serve image files.
- **Cookie hashing/matching**: similar to URL hashing and matching except that cookies are used rather than URLs.

2.2 Connection Manager (CM)

No web switches on the market today are configurable by applications. To enable existing "legacy" switches to support ACM, an external connection manager processes configuration calls from applications and configure switches accordingly.

Since connection managers are external to the switches, they need to be as fault-tolerant as they are. Our first design principle is that the CM should only have soft state to simplify recovery after a CM failure. The second principle is that the CM should be centralized so that only a single node, rather than every node, has to pay the overhead of processing all the requests. The centralized design also matches well with switches that only allow a single root login at any time. Both SNS [12] and Ninja [10] share the same design principles.

CM can be started on any server and is monitored by every server. The goal is to maintain a single CM instance in the system in steady state. CM multicasts heartbeats to announce its presence, and is assumed to be dead if it misses several heartbeats in a row. Servers start new CM instances that are assigned unique IDs when they detect a CM death. An election protocol, where the lowest ID wins, can be used to eliminate redundant instances.

Servers periodically send their port bindings to the CM, which adds them to the switch. Instead of forwarding all requests to the switch, the CM maintains a binding cache and only forwards new bindings. This optimization reduces configuration requests sent to the switches to enable them to support more servers. It also enables servers to beacon more frequently for faster recovery, without increasing the number of actual configuration changes.

In addition, the CM periodically synchronizes its binding cache with the switch. When the CM detects an unknown binding on the switch, it sends a *remove* request to delete that binding from the switch. This mismatch occurs when a server fails during CM election and the switches have stale state. When the CM has bindings that the switch is not aware of, it adds them to the switch. This occurs when a switch resets after failure and clears its bindings.

Redundant switches are often used to tolerate (single) switch failures. Also, redundant ISPs are used to tolerate network failures between the switches and the Internet. The CM needs to be able to support multiple switches that may have different configurations.

3 Implementation

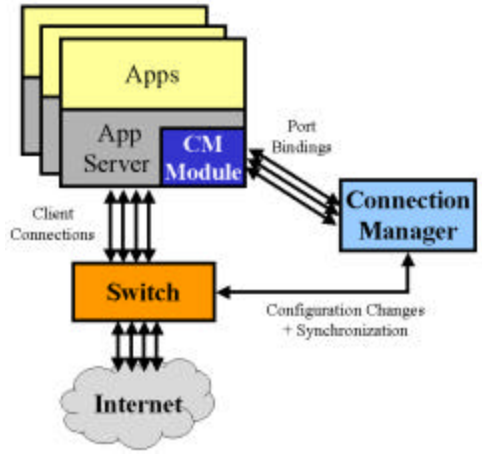

Figure 1: COMICS architecture

COMICS (COnnection Manager for Internet-Centric Services) is a connection manager that supports ACM. As shown in Figure 1, the Connection Manager is the

switch controller module that configures Foundry ServerIron XL switches. The ServerIron switches as shipped can be configured through a command line interface or a web browser. We chose to use the command line interface since it required less parsing than the web interface. We used JExpect [16], a Java telnet package, to implement the configuration library. The configuration library supports *add*, *remove*, and *list* primitives. The Foundry switches we have do not properly support *force remove*, and do not support *drop* at all.

CM client modules are installed on each server that wishes to use ACM. The CM modules are implemented in Java, and supports *add* and *remove* primitives. When a service starts, it initializes the CM module and sends *add* to add itself to the switch. Configuration requests are sent via multicast to the Connection Manager. In addition, each CM module maintains a list of local bindings, and creates a thread to periodically multicast the list to the CM. The beacon interval defaults to 10 seconds, which can scale to 10K servers if it takes 1ms to process each beacon. CM multicasts heartbeats every 3 seconds and is assumed dead if it misses 3 in a row. The combination of soft state and multicast helps the recovery of CM failures and switch failures.

To minimize delay in adding and removing servers, COMICS executes *add* and *remove* requests immediately. It maintains a cache of known bindings so that redundant requests are filtered. This binding cache helps minimize the actual configuration changes made on the server, and helps the system scale to support more servers. COMICS synchronizes its binding cache with the switches using the *list* primitive to garbage collect stale bindings on the switch after a CM failure.

COMICS supports multiple switches through switch profiles. Each switch profile specifies a set of virtual IP addresses and authentication tokens, such as username and password.

COMICS has been integrated with Ninja vSpace's Shogun to leverage Shogun's load-balancing information and capability to start and shutdown servers. Ninja services are event-driven and single-threaded, and needs multiple instances to fully utilize the CPUs on a SMP machine. COMICS achieve this by using a level of indirection. When a service instance starts and requests a server socket locally, the COMICS server module dynamically allocates one on an anonymous port and returns it to the service instance. COMICS then register the server socket with the switches to start forwarding connections to it. Only two lines of code changes are necessary to covert a service to use COMICS: a one-line change in the server socket creation code and another one-line change in the configuration file to specify which client-visible port to bind the server socket to. For example, vSpace may create a random server socket locally for a web server, say port 1234, then binds that port to port 80 on the switches.

4 Evaluation

In this section we present an evaluation of a web server built on top of Ninja vSpace, which uses COMICS to manage client connections. The server cluster consists of 3 two-way 500MHz Pentium III machines running Linux 2.2.16 and IBM JDK 1.3. The closed-loop benchmarking clients run on 10 machines with similar setup. All machines have Layer-3 connectivity to a Foundry ServerIron XL switch, interconnected using 100Mbps Ethernet and Gigabit Ethernet.

4.1 Dynamic Resource Allocation

Load on Internet services vary greatly over time. Load over a 24 hr period and over 7 days can often vary by a factor of 5 or more. Having the ability to dynamically add and remove servers enable a system to respond to changes in load to run more efficiently, and also provides flexibility in managing quality of service. For example, when a service is lightly loaded, it can remove frontend servers and power them off to conserve energy, or release the resources to be used by other services. Conversely, a service can bring in resources to respond to increase in load.

Without ACM, adding and removing a server require an administrator to configure the switches accordingly. Having a human in the loop not only introduces delay, but also introduces errors. A study of failures in fault-tolerant Tandem systems revealed that 42% were due to system administration errors [20]. Without ACM, dynamic resource allocation may be too slow to be effective, or the risk of compromising reliability may be too high for it to be deployed. An alternative is to rely on the health checks performed by switches to add servers. Switches can detect a known server coming online and start forwarding client requests to it. The limitation is that the server has to be known by the switches ahead of time. Removing a server, however, without first removing it from the switches causes all established connections to that server to be dropped, reducing availability.

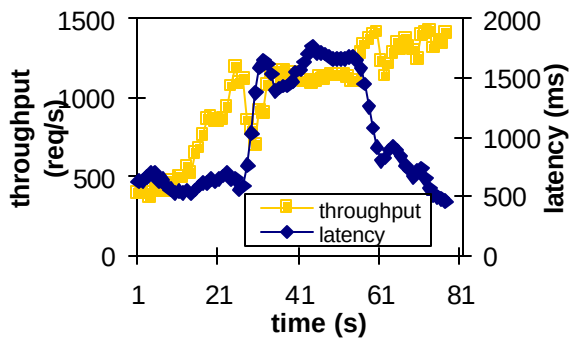

Figure 2: Dynamic resource allocation

Figure 2 shows a system with 1 active web server and 1 standby web server. As load increases, the working set becomes larger than its working memory at t=30. Throughput of the system plateaus and average latency increases significantly. At t=50, the standby server becomes active and begins to service request at t=55. Since the working set now fits into memory, latency drops below 1 second, and the throughput increases.

Sometimes it is useful to preventively shutdown and replace resources before they lead to visible faults. For example, a server may notice its disks are about to fail [26], and removes itself before the faults are visible to users, trading system capacity for availability. **Figure 3** shows how a 3-server system, including 2 active servers and 1 standby server, can maintain throughput while replacing resources without dropping connections. As server s2 is shutting down, COMICS automatically brings in a standby server, s3, to maintain the system throughput. Throughput drops slightly during the transition, but the transition window is quite small, between 5-10 seconds.

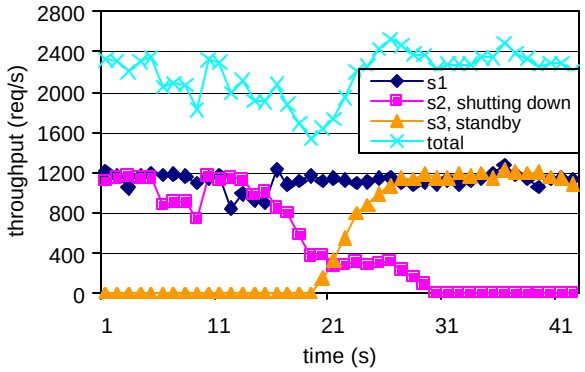

Figure 3: Maintaining fixed resource allocation

4.2 Application-assisted Failure Detection and Recovery

Failures can be categorized into fail-stop failures and partial failures. Fail-stop failures include software and hardware failures that cause the application to stop responding. Web switches can detect fail-stop failures typically in 5 to 10 seconds, depending on how frequent the health checks are performed. Partial failures include application-level failures such as incorrect responses, and corrupted data caused by faulty hardware. Applications with partial failures often respond properly to Layer 3 and Layer 4 health checks performed by the switches. Also, even though some switches are capable of performing Layer 7 health checks, they are limited to HTTP only and cannot handle other protocols such as email and instant messages.

Applications can complement the health checking performed by switches to increase availability in two ways. First, since applications already need to handle exceptions and failures, they can detect partial failures that switches can't. When they detect that a server is returning faulty responses, they can take that server offline before the failure is visible to users by configuring the switches to stop forwarding connections to it. In order to resurrect the server, applications can either notify the system administrators or even try rebooting the server and see if the problems go away. Second, applications can often detect fail-stop failures within a second or two through dropped connections and failed requests, much faster than switches can. Most RPC layers, such as RMI, rely on TCP to provide reliable transport. TCP sockets receive reset packets immediately when a remote application crashes and terminates existing connections. The resets can be used to detect when a remote application has crashed, and should be removed from the switches. In addition, some middleware platforms already perform health checks of their own. Integrating application health checks and switch health checks provides faster recovery and reduces faults visible to users.

When integrated with Ninja, COMICS makes use of Ninja's existing heath checks and can detect and remove a failed server in less than 2 seconds.

4.3 Rolling Reboots and Online Evolution

Since software is unlikely to be bug-free, we should design systems that can tolerate software bugs to improve availability. Bugs such as memory leaks in applications, drivers, and OS can build up over time and degrade performance or even lead to failure. Rolling reboots are useful to reduce software failure by clearing software bugs that accumulate over time.

Inktomi, which provides a search engine service, routinely reboots their servers.

Rolling reboots with 100% availability is supported by COMICS, and is implemented as follows:

1. Remove a server from the switch, allowing it to finish processing existing connections.
2. Reboot the server.
3. Add the server back to the switch.
4. Repeat with the next server.

Without COMICS, administrators have to manually perform the above steps for each server, which is both time consuming and error prone, especially if there are a large number of servers. COMICS makes rolling reboots simpler and faster.

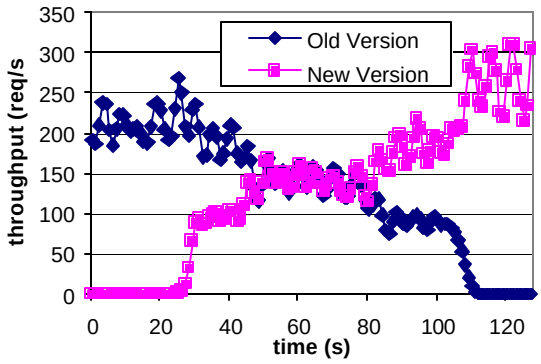

Figure 4: Online evolution using rolling reboots

Support for online software upgrades is also important since Internet services evolve quickly. Bugs fixes and new features are frequently released and need to be deployed without service downtime. Upgrades that are compatible with the current version of services can utilize rolling reboots to simplify the upgrade process. Figure 4 shows a 3-server system where each server reboots and loads a new version of software. Requests are gradually being serviced by the new version until all servers are upgraded. No connections are dropped during the rolling reboot process.

4.4 Graceful Degradation

The goal of graceful degradation is to provide the best quality of service when a system is overloaded. COMICS helps provide several mechanisms to manage quality of service under overload conditions. As a service nears overload, COMICS dynamically allocates all available resources for it until no more resources are available. Once it enters overload mode, COMICS can either take away resources from other services hosted on the same system and give them to the overloaded service, or start degrading QoS selectively for the current service.

COMICS enables two types of selective degradation that can not be achieved through static configuration. The first is through the use of partitions and allocating different amount of resources to each partition, so that quality of service degrades independently. Figure 5 shows a 3-server system with 2 servers allocated to the high-priority partition, and 1 server assigned to the low-priority partition. As load increase in the system, throughput flattens out and average latency increases linearly as shown in Figure 6. The high-priority partition maintains twice the throughput, and its average latency is lower with less variance.

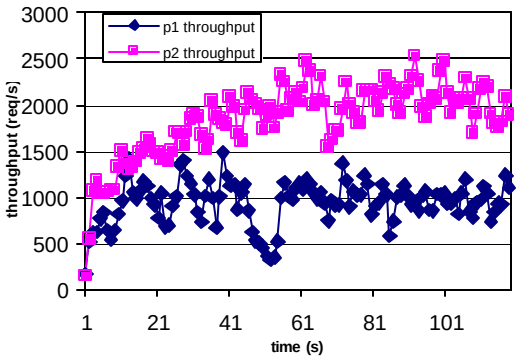

Figure 5: Graceful degradation through partitions, showing throughput

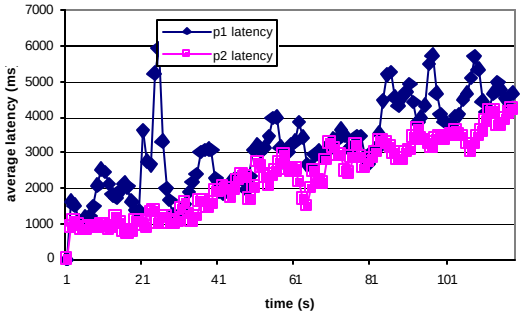

Figure 6: Graceful degradation through partitions, showing average latency

The second mechanism is admission control. Admission control should ideally drop connections as early as possible to minimize resources taken up by connections we don't plan to service. The best place to drop the connections is at the switches, through policies such as content-aware drops. Since no switches available today have such a feature, COMICS could not implement the *drop* primitive that drops connections at

the switches. Instead, we use a drop service to drop connections as they are forwarded to the frontends. Without COMICS, each service would have to implement their own drop mechanism and policy.

When a service is overloaded, COMICS adds a drop service to "service" low-priority requests, in place of an existing server that handles them. It effectively drops 1/N of all low-priority requests if there are N servers servicing them. Figure 7 shows a drop service being added at t=50, dropping throughput of the low-priority requests to half at 500 req/s. At t=80, another drop service replaces a regular server to drop the throughput even further to 0. Throughput for the high-priority requests increases since more resources are available.

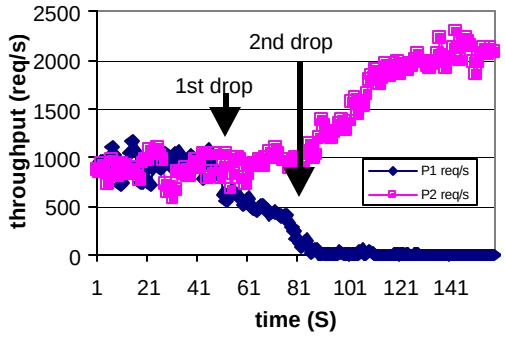

Figure 7: Graceful degradation through admission control

4.5 Switch Configuration Throughput

Switches generally have a forwarding plane with specialized ASIC to forward packets at wire speed, and have a separate general-purpose processor for configuration. Since the processor is not on the critical data path, its performance does not affect data throughput. Its performance does matter in ACM, however, since it limits how dynamic services can be and the number of service instances a switch can support.

We measured the configuration performance of the Foundry ServerIron XL switches, and found that *add* took 26ms and *remove* took 20ms, or about 40 ops/second. If we assume that each service instance has a configuration change every 10 minutes, the Foundry switches can support 24,000 server instances.

5 Example Applications

5.1 ActiveIM Proxy

ActiveIM is an instant messaging proxy that can interoperate with multiple instant messaging systems. It allows users to send and receive messages from AIM, ICQ, MSN, and Yahoo using unmodified MSN clients or a Java applet based client. The proxy logs into all four IM systems on the users' behalf, to utilize the users' own accounts to send and receive messages and buddy status updates. User profiles are stored on the proxy, which allows users to roam and login from different devices.

Instant messaging systems have long-lived connections that are often inactive. Each user typically has one connection for control messages and notification, and one connection for each chat session. The high connection count per user translates into a large number of frontends, and is amplified by ActiveIM proxy where each user connects to multip le IM systems. Scaling the number of frontends has been easy as the same configuration file is used for all frontends. COMICS handles the server port allocation and registration, and provides high availability. Also, since IM protocols are messy and often have hard-to-find bugs, the frontends are frequently rebooted in a rolling fashion. Without COMICS, the frontends would have to be removed from the switch configuration, rebooted, and then added back to the switches, all of which must be performed by administrators.

5.2 NinjaMail

The goal of NinjaMail is to build a scalable email infrastructure other services and application can use to provide more specific functionality.

NinjaMail supports SMTP, POP3, and IMAP4. The SMTP and POP3 components are single-threaded and require multiple instances on a single SMP to utilize all the CPUs. Using ACM, each SMTP instance can requests an anonymous server socket and registers it with the switches to receive requests, rather than having a system administrator statically configuring the server ports. It has enabled us to launch multiple frontends with zero configuration changes.

6 Related Work

Many prior works have used clustering to build web servers that have high performance, scalability, and

availability [8][9][28][29] [30][32][33][34]. They have focused on improving performance using better caching and load-balancing policies, and improving availability through replication. For example, HACC [8] and LARD [9] improve cache locality by reducing the working set on individual servers using content-aware request distribution. Simulation and measurements show substantial improvement over other Layer 3 and Layer 4 request distribution such as round-robin. HACC's Smart Router monitors server load and balances working set size to spread load evenly across a static set of servers. All these systems have assumed that administrators configure the load-balancing switches. These systems are restricted to distribute load, how ever intelligently, within a fixed set of resources, since they cannot dynamically add or remove servers without human intervention. They also do not address manageability and techniques to improve availability such as preventive shutdowns rolling reboots. ACM complements these performance improvements made in clustered servers and offers higher reliability, availability, and manageability.

SNS [12] shares our goal in providing a self-managing system that provides high availability and scalability. SNS uses a fault-tolerant load manager to decide how requests should be distributed across physical servers. SNS focuses on how requests are processed once they are received by fronends, whereas COMICS focuses on enabling applications to control how to map requests onto frontends as they arrive.

IBM Director [27] is a suit of system management tools that includes a software rejuvenation agent (SRA). SRA is capable of predicting impending outages and reboot servers, as well as time-based reboots. SRA relies on the underlying system to perform failover of the rebooted server. In the case of clustered web servers, rebooting a server drops all established connections to that server. COMICS supports rebooting without downtime, thus is ideal to be integrated with SRA to providing higher availability.

Several other systems use programmable network routers and switches, but have different goals. Active Networks [13] have routers execute packets that contain program code. The goal is to have intelligence in the network itself to adapt to network conditions. Popeye uses programmable Ethernet switches to provide network access control for authenticated users. It only routes packets from users that have authenticated themselves through a web-based interface. SPINACH [6] and its successor [7] were similar network access control systems, but used Linux-based PC routers to enforce the packet filtering.

7 Discussion and Future Work

Support for online evolution of incompatible versions requires more support from an ACM system than COMICS provides. One approach is to first partition the servers into two halves and upgrade one half of the servers. The switch is then configured to start sending some connections to the upgraded half and stop sending connections to the old half. Support for check-pointing and rollback can save a snapshot of a clean state that is known to work. Such support makes it possible to back out of failed upgrades, reducing down time.

When a server is being rebooted, it is first removed from switches to stop receiving new connections, and is allowed to finish processing existing connections. Since requests, such as downloading huge files over a modem link or an instant messaging user that never disconnects, can take a long time to finish, we must have a timeout mechanism to close the connections. The timeout value is dependent on the target availability guarantee. Once the target availability is achieved, the remaining connections can be dropped and the server can be rebooted.

COMICS makes configuring multiple instances of applications on the same SMP trivial. Each instance run as a separate process and can leverage the protection provided by the OS. However, it makes debugging trickier since servers now run on non-standard ports. We have seen a few faulty servers that required us to manually go through the list of servers running on random ports, and then do a linear search to pinpoint the faulty server. Debugging tools that can help automate the process would significantly shorten debugging time.

COMICS can be extended to support other existing switches by implementing the switch controller module for each type of switches. Most switches on the market today support both a command line interface (CLI) and a web-based interface. Currently, COMICS is external to the switches in order to support existing switches. Future switches may wish to implement the ACM primitives directly to provide a highly available Connection Manager, and provide client modules for application to configure the switches.

8 Summary

Internet systems need to be more self-managing and autonomous to improve on availability, quality of

service, and manageability. Active Connection Management enables clustered Internet systems to control how connections are mapped to servers and provide several properties ideal for Internet services over static configuration. ACM increases reliability through automated configuration and rolling reboots. It provides better load conditioning through dynamic resource allocation and graceful degradation. It improves availability by shortening server failure detection and recovery time. Last, it provides better management scalability by automating configuration.

We have evaluated and shown that our ACM implementation, COMICS, provides the above properties. We have also shown that integration of ACM and application server makes it trivial to develop and deploy services that have these properties. We would like to see ACM primitives to be implemented and supported by future switches.

Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency (grants DABT63-98-C-0038 and N66001-99-2-8913). We would like to thank Rob von Behren, Andy Huang, and Larence Melloul for their valuable input on this paper. Eric Fraser, Matt Massie, Albert Goto, and Philip Buonadonna provided support for the Berkeley Millennium cluster used to obtain performance measurements.

References

[1] T. E. Anderson, B. N. Bershad, E. D. Lazowska, and H. M. Levy. Scheduler activations: Effective kernel support for the user-level management of parallelism. In Proceedings of the 13th ACM Symposium on Operating System Principles, pages 95--109, October 1991.

[2] D.A. Wallach, D.R. Engler, and M.F. Kaashoek. ASHs: Applicationspecific handlers for high-performance messaging. In ACM Communication Architectures, Protocols, and Applications (SIGCOMM '96), pages 40--52, August 1996.

[3] B. Bershad, S. Savage, P. Pardyak, E. G. Sirer, D. Becker, M. Fiuczynski, C. Chambers, and S. Eggers. Extensibility, safety and performance in the spin operating system. In Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP15) , pages 267--284.

[4] M. Welsh, D. Culler, and E. Brewer. "SEDA: An Architecture for Scalable, Well-Conditioned Internet Services." To appear in Proceedings of the 18th Symposium on Operating Systems Principles (SOSP-18), Lake Louise, Canada, October 21-24, 2001.

[5] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. IEEE Journal on Selected Areas in Communications, 14(7):1280--1297, September 1996. Yatin Chawathe and Eric A. Brewer. “System support for scalable and fault tolerant Internet service.” In IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98), Lake District, UK, Sep 1998

[6] Elliot Poger and Mary Baker. Secure public internet access handler (spinach). In Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1997.

[7] Guido Appenzeller, Mema Roussopoulos, and Mary Baker, "User-Friendly Access Control for Public Network Ports". Proceedings of IEEE INFOCOM '99, March 1999.

[8] Zhang, M. Barrientos, J. B. Chen, and M. Seltzer. HACC: An Architecture for Cluster-Based Web Servers. In Proceedings of the 3rd USENIX Windows NT Symposium, Seattle, WA, July 1999.

[9] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, Oct. 1998.

[10] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Josheph, R. H. Katz, Z. M. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. Special Issue of Computer Networks on Pervasive Computing, 2000.

[11] D. R. Engler, M. F. Kaashoek, and J. O'Toole Jr. Exokernel: an operating system architecture for application-specif ic resource management. In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, December 1995.

[12] Yatin Chawathe and Eric A. Brewer. System support for scalable and fault tolerant internet service. In IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98), Lake District, UK, Sep 1998.

[13] D. Tennenhouse and D. Wetherall. Active Networks. In 15th Symposium on Operating Systems Principles, December 1995.

[14] SPECWeb99 http://www.spec.org/osg/web99/

[15] Duane Wessels, Squid Internet Object Cache, http://squid.nlanr.net/Squid, May 1996

[16] Java expect library, Dave Jarvis, http://www.joot.com/software/jexpect/

[17] Alteon ACEdirector http://www.alteonwebsystems.com/products/acedirector/

[18] Foudry ServerIron web switches http://www.foundrynet.com/products/webswitches/serveriron/

[19] Cisco Content Services Switches http://www.cisco.com/warp/public/cc/pd/si/11000/

[20] J. Gray. Why Do Computers Stop and What Can Be Done About It? Symposium on Reliability in Distributed Software and Database Systems, 3-12, 1986.

[21] B. Murphy and T. Gent. Measuring System and Software Reliability using an Automated Data Collection Process. Quality and Reliability Engineering International, 11:341-353, 1995.

[22] D. R. Kuhn. Sources of Failure in the Public Switched Telephone Network. IEEE Computer 30(4), April 1997.

[23] J. Menn. Prevention of Online Crashes is No Easy Fix. Los Angeles Times, 2 December 1999, C-1.

[24] BEA WebLogic E-business platform http://www.bea.com/

[25] Microsoft .NET Enterprise Servers http://www.microsoft.com/servers/evaluation/overview/net.asp

[26] Aaron Brown, David Oppenheimer, Kimberly Keeton, Randi Thomas, John Kubiatowicz, and David A. Patterson. ISTORE: Introspective storage for data intensive network services. In Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII), March 1999.

[27] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert. Proactive management of software aging. IBM Journal of R & D, Volume 45, No. 2, 2001

[28] M. Baentsch, L. Baum, G. Molter, S. Rothkugel,P. Sturm, "Enhancing the Web's Infrastructure: From Caching to Replication", in IEEE Internet Computing, pp. 18-27, 1997 R.B. Bunt, D.L.

[29] Eager, G.M. Oster, C.L. Williamson, “Achieving Load Balance and Effective Caching in Clustered Web Servers”, Proc. 4th International Web Caching Workshop, San Diego, CA, March 1999.

[30] M. Colajanni, P. S. Yu, D. M. Dias, “Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems”, IEEE Trans. on Parallel and Distributed Systems, Vol. 9, N 6, pp. 585-600, June 1998.

[31] E.D. Katz, M. Butler, R. McGrath, “A Scalable HTTP Server: The NCSA Prototype”, Comp. Net. and ISDN Sys., 27 (2), pp.155-164, November 1994.

[32] J. Watts, S. Taylor, “A Practical Approach to Dynamic Load Balancing”, IEEE Trans. on Parallel and Distributed Systems, Vol. 9, NO. 3, March 1998, pp. 235-248.

[33] P. Damani, P.E. Chung, Y.Huang, C.Kintala, Y. M. Wang, “ONE-IP: Techniques for Hosting a Service on a Cluster of Machines”, Comp. Net. and ISDN Sys., 29, 1997, pp. 1019-1027. IEEE Network, “Special Issues – Web performance”, Vol 14, No. 3, May/June 2000.

[34] A. Iyengar, J. Challenger, D. Dias, P. Dantzig, “High-Performance Web Site Design Techniques”, IEEE Internet Computing, Vol. 4., N. 2, March/April 2000, pp. 17-26.