Bringing Autonomic, Self-Regenerative Technology into Large Data Centers

Ken Birman, Dept. of Computer Science, Cornell University[1]

Abstract

With the introduction of blade servers and the emergence of techniques for spreading applications over clusters to exploit scalability, we're suddenly seeing the emergence of enormous data centers. Google is said to operate more than 100,000 computers in its centers; Amazon has grown from perhaps 25 machines to as many as 2500 over just a few years, and similar stories are now common. These kinds of experiences presage a broader move towards very large cluster-style data centers in a broad range of commercial and military settings. Yet the technology for automated management of big systems is lagging. The most common distributed computing platforms require excessive care and feeding as a deployment grows, and are prone to instability and even melt-downs when large configurations are put under unexpected stress. At Cornell, the QuickSilver project is exploring self-organizing and self-repairing peer-to-peer technologies based on a new kind of epidemic communication protocol. The approach offers promise for such uses as managing, monitoring, and controlling very large-scale data centers. Moreover, same kinds of solutions may be useful in sensor networks.

INTRODUCTION

As corporations and the government move to exploit advances in interoperability technologies such as Web Services, we're seeing explosive growth in the size of data centers and rising demand for what might be called "quality of service" guarantees from such centers. Cornell's QuickSilver group has been working to better understand the nature of these requirements. Figure 1 illustrates what might be called a "canonical architecture" for the systems we've investigated. A tier of front-end platforms offers services to remote clients, over the Internet or a corporate LAN. These platforms could be web systems such as Apache, IBM Web Sphere, BEA Web Logic or Microsoft's Indigo, or they might be Web Services front-ends.

As seen in the figure, a data center is often structured into two tiers. In the case of a client who issues a web-based request, say to an eTailor's website, the request is sent from the client's browser to one of the first-tier web servers using some sort of load-balancing scheme. The front-end basically runs a script, issuing remote procedure calls to the back-end machines, which perform database lookups, maintain product popularity rankings and blogs, track order status, and so forth. As these reply, the front-end constructs a response for the end-user. Similar behavior ensues when the client is a

[1] Other members of the QuickSilver project include Dr. Robbert van Renesse, Dr. Stefan Pleisch, Krzys Ostrowski, Mahesh Balakrishnan, Maya Haridasan and Tudor Marian. Our effort is supported by DARPA's SRS program, the AFRL/Cornell Information Assurance Institute and a MURI grant. We have also benefited from Microsoft Research Foundation support.

computer using a service oriented architecture (SOA), such as Web Services. Here, the first-tier system is a request dispatcher, decoding the incoming invocation, passing it to services layer, encoding the reply, and sending it back.

Between the first tier servers and the second-tier services one typically sees a publish-subscribe infrastructure, as illustrated here, or some other message-queuing middleware product. The approach conceals service availability and configuration issues from the front-end systems, but also places the middleware layer under enormous loads that grow as the system scales up.

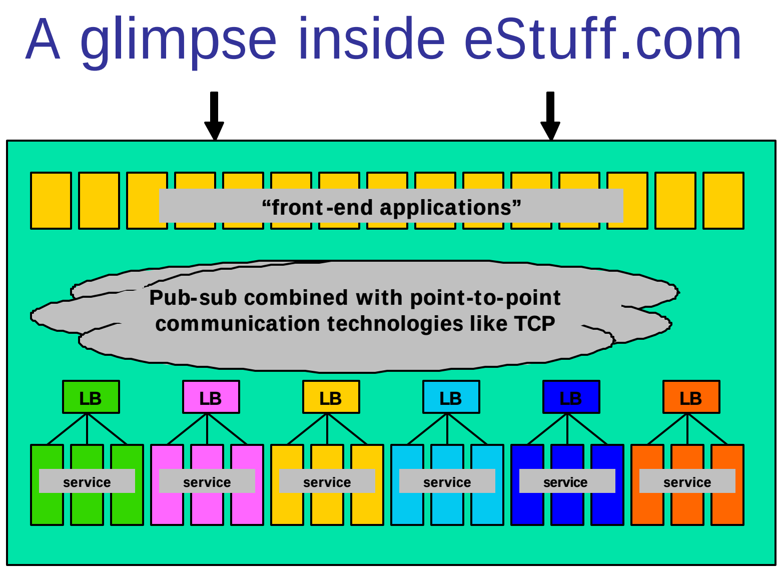
A glimpse inside eStuff.com
"front-end applications"
Pub-sub combined with point-to-point communication technologies like TCP
LB
LB
LB
LB
LB
LB
service
service
service
service
service
service

Figure 1: Internal architecture of a large corporate datacenter

This white paper focuses on quality of service questions that arise in systems having this structure and on technology deficiencies that currently make it hard for developers to achieve their QoS objectives. Our basic finding is new mechanisms are needed to support scalable, secure, robust data replication, at potentially high speed, and often with QoS guarantees. This functionality must be presented in a paradigm that integrates easily with service oriented architectures. The QuickSilver system, under development at Cornell, targets this missing functionality, and can be accessed both through native interfaces and through a publish-subscribe WS_NOTIFICATION interface.

QUALITY OF SERVICE NEEDS IN LARGE DATA CENTERS

At least until recently, data centers have been oriented towards human users and for this reason, they often have surprisingly "weak" reliability requirements. It is not a catastrophe if a user's product inquiry times out and must be reissued. As we move to computer-to-computer interactions mediated through SOA platforms, such outcomes are more problematic, hence SOAs emphasize transactional guarantees and request queuing to ensure that pending requests will not be lost. Yet one sees little attention to end-user guarantees of availability, reliability, fast response, or recovery from disruptive events.

Publish-subscribe products, for example, often offer best effort delivery guarantees. A typical guarantee might take the following form: *a message published by a healthy sender will be delivered to a healthy receiver within 90 seconds. The system will attempt to overcome problems. However, if the system is unable to accomplish do so, or is uncertain about the outcome, an error code is returned to the sender.* (Here we paraphrase the reliability guarantees of one of the most popular off the shelf products).

Users who work with such products find these guarantees are inadequate for the configuration shown in Figure 1. The product just mentioned, for example, can malfunction during load surges or when failures disrupt the stream of data. When this happens, the publish-subscribe protocol generates huge spikes of retransmission requests and attempts, and communication more or less melts down until the 90 second timeout period expires. All requests that arose during the black-out period fail. Such blackouts are never seen in small configurations but become a serious problem as a center scales up to thousands of machines, heavy loads, and more frequent failures.

Data center developers complain that as they scale up the size of a system and the loads upon it, this kind of problem – and here we mean not just problems with the publish-subscribe component of the system, but with *all* technologies used within the center – become common. Even milder QoS requirements such as "rapid event delivery" that can easily be satisfied in small configurations are routinely violated in large configurations. The experience of running a massive data center today is one of unending crises. Perhaps today the publish-subscribe system is suffering 90-second "blackouts". Yesterday a transient inconsistency in a replicated database caused all product popularity queries to fail. Tomorrow, users might be unable to access their shopping carts.

The basic premise behind the QuickSilver effort is that modern distributed systems scale poorly, in many senses of the term:

- They often become fragile and unstable as the size of a deployment grows, and require more and more human care and supervision.
- They are more and more disrupted as participants join, leave and fail. And, of course, such events rise in frequency as we make a system larger.
- As the system size rises, the sustainable loads often drop – for example, a communications technology that was able to send 1000 messages per second in a configuration with 50 participants might be unable to sustain more than 50 per second in a configuration with 1000 participants. Pushed to higher data rates, unexpected and serious outages become apparent.

We believe that trends favoring such standards as Web Services are now making this situation even more extreme. The problem is that Web Services promote a greater degree of communication, and encourage us to build massive systems with components scattered over tremendous numbers of machines [Birman04]. But they lack scalable technologies capable of supporting this model reliably, with high performance, and with adequate security. One way to capture this observation is to say that such systems lack QoS guarantees; another is to suggest that they need to become far more self-managed, stable under stress, and gain the ability to self-diagnose problems and self-repair.

Our decision has been to build a new data replication infrastructure aimed squarely at the developer of this new generation of service-oriented systems. Our goals are broad: we plan to offer both publish-subscribe and native interfaces to the technology, and we see applications outside of data centers. For example, large-scale sensor networks could benefit from the same sorts of technologies. To accomplish these goals QuickSilver reflects the following design principles:

- The design is extremely asynchronous; our components and tools make progress under all circumstances
- They self-stabilize and self-repair when disruptions occur
- They employ hierarchies and functional partitioning to "divide and conquer" where a problem might become larger and larger as a function of scale.
- They make use of aggregation and data fusion techniques, so that applications can see high-value summaries of system state (and exploit those to self-manage) in settings where the "whole" system state is too large to collect at any one spot.
- They employ highly convergent probabilistic protocols to obtain guarantees with overwhelming quality (and strong mathematical techniques to predict behavior and to fine-tune the guarantees to match the needs).

All of this leads to an architecture in which we can actively involve the application in its own problem diagnosis, repair and recovery. Whereas conventional systems operate in the dark, applications integrated with the QuickSilver platform should be able to turn on the lights and "see" system state in a (probabilistically) consistent state, using this information to drive configuration management, adaptation, and control.

Although brevity precludes a discussion of security here, we should mention that our design secures QuickSilver "in depth", and also offers the developer novel security options through QuickSilver's own secured mechanisms.

HOW QUICKSILVER WORKS

QuickSilver is still under development, but we've already completed some component technologies that illustrate the basic ideas underlying the publish-subscribe platform. Briefly, these are:

- *Bimodal Multicast* [Birman99a]: A highly scalable, extremely robust reliable multicast protocol that uses unreliable multicast to disseminate data and then employs a background peer-to-peer gossip protocol to recover missing packets.
- *Astrolabe* [VanRenesse03]: A "virtual" database of system management data and other dynamically collected information that appears to the user as a hierarchically structured relational database, replicated at all nodes in the system,

and usable for self-management, problem diagnosis, data mining and distributed control. Again, the underlying communications technology is peer-to-peer gossip.

- *Kelips* [Gupta03]: A gossip-based distributed hash table supporting very fast O(1)-hop lookup at the cost of O(vN).space overheads.
- *Epidemic, gossip-based, repair mechanisms and membership tracking.* We've developed a family of gossip protocols to track system membership and to compare server states and repair inconsistencies that arise at runtime.
- *Virtually Synchronous Process Groups* [Birman99b]: This is our reliable multicast model for situations requiring strong forms of reliability in smaller groups; the model is strong enough to solve consensus or to implement one-copy serializability. We are also looking at purely gossip-based mechanisms for replication, and at a new scheme called *Chain Replication* [VanRenesse04].
- *Pmap.* A gossip-based membership tracking service that can accommodate a number of kinds of "partitioning functions." We use this to track the mapping of services to sets of machines, and the mapping of requests to a cluster of services within a service. *Pmap* is related to *Kelips* but specialized to the mapping role.

These components come together in support of what Jim Gray has described as a "RAPS of RACS": a *reliable partitioned service* consisting of *reliable clustered servers.* Figure 2 illustrates the basic idea for a single service. A data center would contain many such services, running "side by side" (we saw this in Figure 1; that figure is revisited in terms of RAPS of RACS in Figure 3, below).

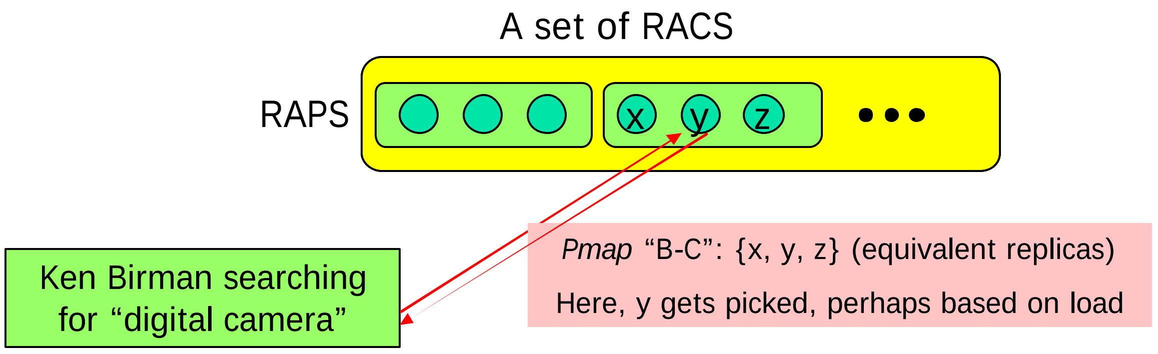
A set of RACS
RAPS
x
y
z
Ken Birman searching for "digital camera"
Pmap "B-C": {x, y, z} (equivalent replicas)
Here, y gets picked, perhaps based on load

Figure 2: In a data center, services are implemented as sets of clusters. Each cluster is a set of service programs that replicate data to share work and achieve high availability.

PUTTING IT ALL TOGETHER

The QuickSilver project, then, integrates these technologies into a single platform and presents that platform through publish-subscribe and group communication interfaces (we're exploring a variety of options) [Birman04]. In tackling this problem, we still face serious challenges:

- Existing publish-subscribe systems are also easily disrupted by overload or failures. We need to ensure that our own infrastructure can repair itself quickly enough to prevent user-visible disruptions in service when such events occur.

- While group communication is used today in many high-profile settings[2] the communication models mentioned above (Bimodal Multicast and Virtual Synchrony) have not been applied in situations when enormous numbers of "groups" overlap. Yet publish-subscribe, used in systems with huge numbers of side-by-side partitioned service and of communication topics, will give rise to tremendous numbers of overlapping groups.
- Many data centers will need new kinds of time-critical event notification services optimized to minimize latency and yet designed to overcome failures.
- Our decision to work within SOA standards requires us to strike a balance between the ambitious goals of these architectures (for example with respect to content filtering) and the limits of what we can actually implement.

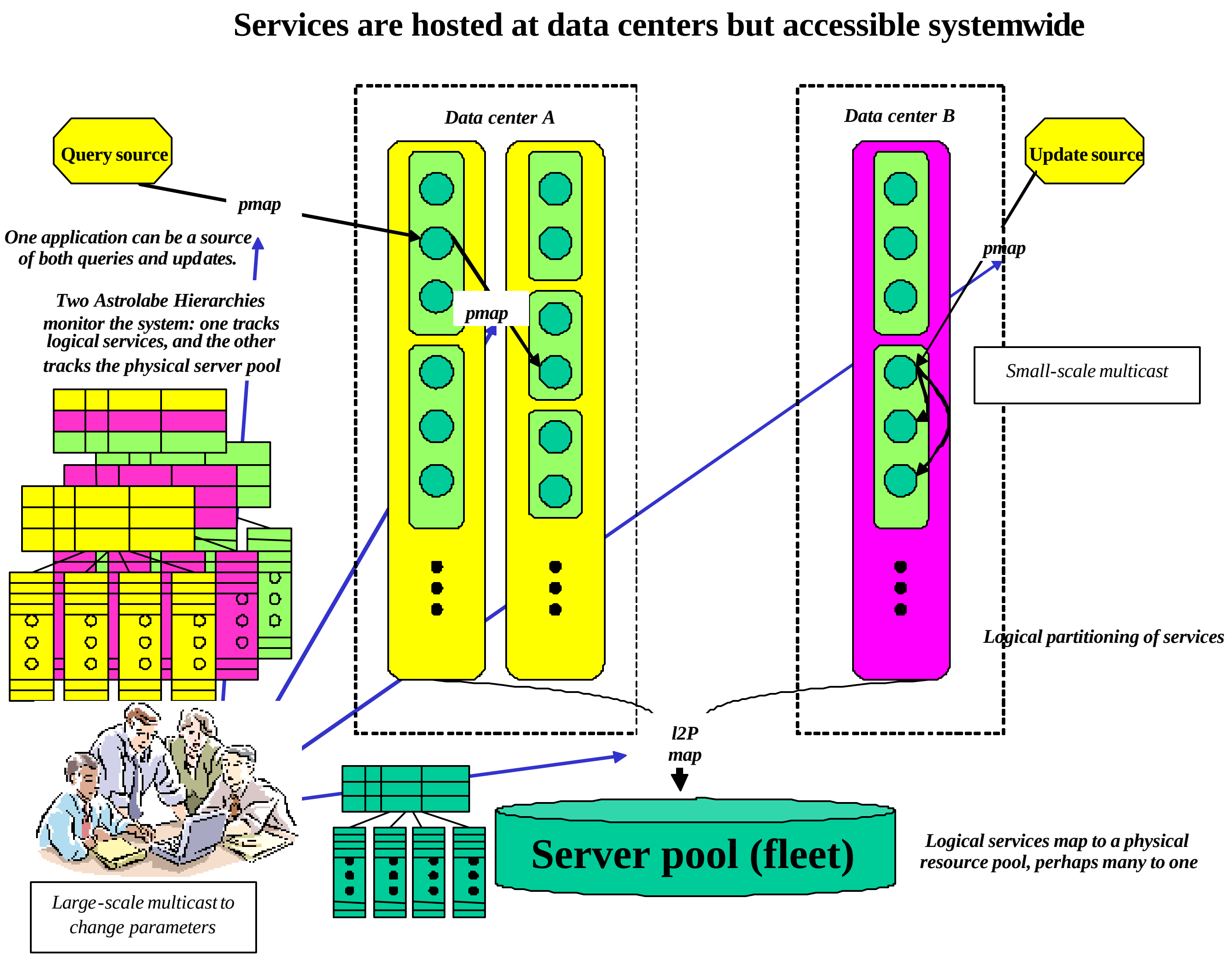
Services are hosted at data centers but accessible systemwide
Query source
pmap
One application can be a source of both queries and updates.
Two Astrolabe Hierarchies monitor the system: one tracks logical services, and the other tracks the physical server pool
Data center A
pmap
Data center B
Update source
pmap
Small-scale multicast
Logical partitioning of services
l2P map
Server pool (fleet)
Logical services map to a physical resource pool, perhaps many to one
Large-scale multicast to change parameters

Figure 3: Our Scalable Services Architecture

[2] Well known examples include the NYSE and Swiss Stock Exchange, French air traffic control system, and AEGIS warship [Birman99b]. Commercial standards and products include the CORBA fault-tolerance standard, IBM WebSphere product, Microsoft Windows Clustering product.

If we are successful, QuickSilver will support the kind of *scalable services architecture* shown in Figure 3. The best way to view this architecture is to see it as using data replication, in various forms, to support the developer of new services for deployment into a large data center. In the figure we see Astrolabe used to monitor, automate and control services and to manage a server pool, and a second Astrolabe instance used to manage a pool of hardware resources. Queries and updates are mapped through the *pmap*, which uses replicated system state data for this purpose. Data replication tools and epidemic repair mechanisms ensure that data is consistent within group members. Thus the idea of replication is pervasive, sometimes through a publish-subscribe paradigm, and sometimes through other interfaces. By making replication tools autonomic and self-regenerative, we can help the designer create applications that share these properties.

CONCLUSIONS

Although our effort is very much a work in progress, we believe that it responds to a clear and growing need for better scalability, robustness, and self-* properties in massive data centers. Although it won't be easy to solve the hard technical problems while also respecting SOA architectural standards, success will enable major advances in the technology options for constructing massive data centers. Moreover, we believe that the same requirements are starting to arise in other settings, such as large sensor networks, and that progress on the problem described here could be transferred to those other domains. More information can be found at our research web site [QS].

REFERENCES

[QS] http://www.cs.cornell.edu/projects/quicksilver/

[Birman99a] Bimodal Multicast. Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu and Yaron Minsky. ACM Transactions on Computer Systems, Vol. 17, No. 2, pp 41-88, May, 1999.

[Birman99b] A Review of Experiences with Reliable Multicast. K. P. Birman Software Practice and Experience Vol. 29, No. 9, pp, 741-774, July 1999.

[Birman04] Adding High Availability and Autonomic Behavior to Web Services. Ken Birman, Robbert van Renesse, Werner Vogels. In the Proceedings of the 26th Annual International Conference on Software Engineering (ICSE 2004). May 23 - 28, 2004. Edinburgh, Scotland.

[Gupta03] Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. Indranil Gupta, Ken Birman, Prakash Linga, Al Demers and Robbert van Renesse. Submitted to: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03); February 20-21, 2003. Claremont Hotel, Berkeley, CA, USA.

[VanRenesse03] Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. Robbert van Renesse, Kenneth Birman and Werner Vogels. ACM Transactions on Computer Systems, May 2003, Vol.21, No. 2, pp 164-206

[VanRenesse04] Chain Replication for Supporting High Throughput and Availability. Robbert van Renesse and Fred B. Schneider, Operating Systems Design and Implementation (OSDI 04); Dec 2004.