Challenges to Building Scalable Services

A Survey of Microsoft's Internet Services

MSR-TR-2015-29

Comments from the Authors:

This paper was originally circulated as a Microsoft Confidential memo in fall 1999. Its purpose was to document the findings of the co-authors as we attempted to understand the state-of-the-art of large internet services. Our original intent was to gather the data documented in this paper purely for our own to understand. However, as we discussed early findings with our colleagues, we quickly realized the value of circulating them to a wider audience. The original memo was circulated to Microsoft's entire executive staff and quickly passed around. From file server data, we believe over 1,000 MS employees read the original memo in the first three months after internal publication.

This release of the memo has been modified slightly from the original to remove non-technical information, such as business plans.

Due to an unfortunate oversight on my part, the original memo did not name each of the people we interviewed. Those pioneers deserved recognition at the time and their groundbreaking work deserves now to be remembered by history.

In the 15 years since this paper's circulation, much has changed at Microsoft and in the industry. Experience gathered in writing this paper directly lead to our discovery of the core principles of what is now widely known as cloud computing*. In 1999, Microsoft's largest internet service had just over 2,000 computers. Today, many cloud services use over 100,000 servers. Many of the services and technologies described in this paper no longer exist. Those that remain have morphed beyond recognition under the intense pressure of growing by orders of magnitude.*

So, why publish such an out-of-date document? To document the world that was; to provide those who want to understand the past with authoritative data on how Internet Services worked at the end of the last millennium.

Galen Hunt
April 2015
Redmond, WA

Challenges to Building Scalable Services

A Survey of Microsoft's Internet Services

Version 1.0

September 24, 1999

Steven Levi and Galen Hunt
Microsoft Research

Acknowledgments

We have been continually impressed at the caliber of the people building and operating Microsoft's megaservices. Don't underestimate either their intelligence or their commitment to deliver. We have borrowed their time and their wisdom. Without exception, these people have graciously shared both with us. Any value derived from this report comes from their contribution. Any inaccuracies are solely ours.

Revision History

Version 0.6	August 24, 1999	Rough draft distributed to executives.
Version 0.7	September 3, 1999	Text complete and revisions from sites.
Version 0.8	September 15, 1999	Added LinkExchange.
Version 0.9	September 17, 1999	Revisions from LinkExchange.

CHALLENGES TO BUILDING SCALABLE SERVICES 1

ACKNOWLEDGMENTS 1
REVISION HISTORY 1

1. INTRODUCTION 3

2. WHAT IS A SERVICE? 3

2.1. SERVICE CHARACTERIZATION 4
2.2. LOAD BALANCING 6
2.3. EXAMPLE 8

3. OBSERVATIONS 9

3.1. MAINTAINABILITY 10
3.2. SCALABILITY 11
3.3. AVAILABILITY 13

4. SERVICES 14

4.1. HOTMAIL 14
4.2. HOME.MICROSOFT.COM (HMC) 17
4.3. SIDEWALK 18
4.4. MSNBC 21
4.5. INSTANT MESSAGING 23
4.6. EXPEDIA 25
4.7. MONEYCENTRAL 27
4.8. WINDOWS UPDATE 29
4.9. CARPOINT 31
4.10. CALENDAR 33
4.11. CHAT 34
4.12. COMMUNITIES 35
4.13. WEBTV 36
4.14. LINKEXCHANGE 41
4.15. LINKEXCHANGE LISTBOT 45
4.16. HYDROGEN 46
4.17. PASSPORT/WALLET 47
4.18. ADSTECH 50
4.19. MSN OPERATIONS 52

5. CONCLUSIONS 53

1. Introduction

Desktop applications and megaservices have fundamentally different per-user scalability models. The developers of desktop applications like Office, Flight Simulator and Visual Studio scale to more users by shipping more CD-ROMs. The developers of megaservices like HotMail, Expedia, and home.microsoft.com must physically scale their servers and software to support millions of simultaneous users.

While programming and execution in the two domains can be quite different, deployment and operations are radically different. Software developers bridging the two domains, whether MS product teams or ISVs, have little incentive to use MS technology when moving to the Internet because frankly, MS at present has little, if any, technology to aid with the task of making their application scalable, deployable or operable with millions of users. Moreover, the skill set of the desktop developer is radically different than the skill set needed to build and deploy large, scalable services.

Three months ago, we initiated a study of Microsoft megaservices. Recognizing our complete ignorance, we endeavored to visit each megaservice within the company. Our goal was to understand their architectural, programming, and deployment challenges. Our hope was to identify a set of commonalties upon which we could then propose a new Microsoft application platform for megaservices and create an explosion of new applications reminiscent of the Win3 desktop application explosion.

This report represents the first major milestone of our study. We visited eighteen megaservices, including virtually all of the MSN properties, HotMail, LinkExchange, and WebTV. We believe we have a strong basic understanding of the challenges of building, deploying, and operating today's megaservices and want to share our understanding with a broader audience.

The outline of the remainder of this paper is as follows:

Section 2 develops a notion of what a service is by introducing an extremely simple definition and then layering on top of that additional constraints and complexities that one must consider as part of any viable (internet) service. To round out a common base of understanding a discussion of load balancing mechanisms is included. Finally, this notion is applied to a real-world example where some of the more subtle design trade-offs are explored.

Section 3 details our key observations. The reader is urged not to skip Section 2 before reading Section 3. Without the base of understanding developed in Section 2, the findings in this section will not be as clear.

Section 4 describes each of the Microsoft services.

Section 5 summarizes our most important findings.

2. What is a Service?

We will not propose in this report an application platform for megaservices. Building scalable systems is hard. Building reliable megaservices is even harder.

2.1. Service Characterization

Two-tier client/server computing has evolved into multi-tier service-based computing. In this newer approach, servers provide logical services to their clients by partitioning the application workload among themselves, and by depending upon a common communication model to share their distributed computation.

In practice, application-level megaservices are often recursively composed of multiple services that cooperatively span many machines. As simple as this statement is, it has profound implications. The application author must become responsible for matching his or her code to the hardware and communications capabilities of what is typically a large and difficult-to-understand cluster of computers. Thinking about multiple types of machines, and multiple instances of each type, forces the developer to think about network connectivity. One must consider communications patterns between machines and the mechanisms needed/desired to balance both machine load and internal/external network capacity.

Virtually all of the Microsoft services we reviewed minimally decomposed their service into what are typically called front-end (FE) machines and back-end (BE) machines. The FE machines usually service incoming user requests, acquire, generate or read some data, render this data into HTML, and then issue a response to the user. For services that support the HTTP 1.0 protocol, a TCP connection is created and torn down for each individual user request.

Front-end (FE) machines are generally stateless. That is to say, after a connection is torn down there is no need to remember anything about that request to process future requests. Because of their statelessness, most FE machines are considered interchangeable. Back-end (BE) machines are typically stateful and are used to retrieve and persist data. Examples of BE components include the Exchange Store, a SQL or Oracle database, and the HotMail USTORE machines. FE machines quite often make requests to BE machines to get or store data. End users rarely communicate directly with BE machines.

As services scale, data partitioning becomes an issue. In the case of a small service where there is only one BE machine, the way in which the data is partitioned does not matter much. As the system scales and the needed BE output rate exceeds the resources of a single box (CPU cycles, network bandwidth, number of disk spindles, etc.), more often than not there will be a need to find a natural partitioning of the data. Fortunately, many of the current services being developed are "embarrassingly parallel," which is to say that data is easily partitioned, usually on a per-user basis. The developer now concerns himself not only with how the data can and must be partitioned, but also how it will be accessed and what the topology of the connectivity between the FE machines servicing the requests and the set of BE machines. If not careful, the developer will find himself with a full mesh connectivity from every FE to every BE. A mesh may or may not be a bad thing depending on how it is used. If, however, the service accesses SQL via ADO in ASP it will open and tear down connections between the FE and the BE machines for each HTTP request. The mesh in this case is disastrous.

Of course, our poor developer is not out of the woods yet. Not only does he have to figure out how to partition the data to accommodate the current BE expansion but he must provision for all future expansions and new versions as well! Now the developer probably

needs to come up with some virtual resource manager (VRM) that provides an indirection between the logical data and its physical partition. As new data BE machines come on line he needs to adjust the VRM to reflect the new partitioning. Most likely this new partitioning will necessitate the migration of data stored on one or more of the BE devices to other BE devices. The developer has to either handle this directly in the service or provide support tools for migration.

As a service grows in scale and importance, more and more attention is paid to smaller and smaller details of performance. With a single machine fulfilling only one functional aspect of the service, the developer's attention turns to specific details of what is going on each box. Where is performance going? Are there things on this box that get in the way? Are there services or processes running on this box that are not need? Recall that a subcomponent of the system may run on a very large number of machines. HotMail has over 1300 Front Door machines all doing exactly the same thing. The cost of additional fine tuning is amortized over the entire set of machines on which it runs.

To understand the system better, the developer must add all manner of runtime instrumentation. Sources of runtime instrumentation data serve two distinct purposes: to aid in monitoring the health of a current system; and to enable understanding of how the system behaves during real on-line operations. Deeper understanding of the current system increases the chance of improving its implementation. The most successful services have built extensive monitoring and logging facilities into their architectures.

As the number of components in the service increases, so does the likelihood of component failure. Interestingly, as a consequence of their distributed nature, even if a function (or a function servicing a subset of the user community) is not operational most likely some portion of the service is still operating. It behooves the developer to exploit partial failure. Because of partial failure, the developer must treat failures and/or the lack of sub-services (or infrastructure services) with style and grace when writing his components. He needs to handle failures from all external components as gracefully as possible and certainly not cause errors to ripple back to the user in some unintelligible form (or gods forbid, crash!).

Up to this point, a strong system developer or architect with some distributed systems knowledge could grapple with most of the concepts and issues presented. The challenge is to expand their mindset to incorporate an intrinsic quality of services: *megaservices must always be up!* This is a massive mental shift. For services to become a cornerstone of electronic business, they must always be available, supporting at least a minimal quality of service. Think telephone: you can (almost always) depend upon it.

No number of developers and architects versed in distributed computing are sufficient to ensure high availability – this guarantee can only be offered by a top-notch operations staff. The requirements placed on a service by operational constraints are as important (and perhaps even more important) than end-user requirements. Especially in the resource-rich environment of dedicated personal computer hardware, developers have often paid less attention to operational ease-of-use than to end-user visible application features. This lack of engineering detail if left unchecked can have disastrous effects upon the scalability and availability of a service, however, and must be avoided. Administrative simplicity, ease of configuration, ongoing health monitoring, and failure detection are as high priorities as any application feature; because of this, the developer must fully understand the context in

which a service is deployed and run. Conversely, the operations staff must also understand all partitioning schemes, administrative tools, and communications patterns that characterize the service and its runtime presence on the net.

2.2. Load Balancing

In the presence of diverse and plentiful machines, the developer is forced to think about partitioning the application workload, as well as the network connectivity needed to support each partitioning scheme. One very important aspect of connectivity exists between external users and the service itself. Except for the smallest of services, some form of load balancing mechanism is needed to distribute the load of incoming connections over the FE boxes. Many megaservices also have sub-services, which are themselves load balanced.

Three typical load balancing mechanisms exist: multiple IP addresses (DNS round robin), hardware support for virtual-to-real IP address mapping, and software support for virtual-to-real IP address mapping.

2.2.1. Multiple IP Addresses (DNS Round Robin)

DNS name resolution is the process of translating a domain name to an IP address. In "round robin" DNS, a random IP address will be returned with each DNS resolution request (if multiple entries exist in the DNS entry.)

The purpose of round robin is to allow use of multiple HTTP servers (with identical contents) in order to distribute the connection loads. Round robin is not random, though it gives a random effect. It operates in a round-robin fashion (as the name implies), in that it rotates the return record sequence by one for each response – one address is handed out, put at the end of the list, and then the next address is handed out for the next translation request yielding something like a translation list.

2.2.2. Hardware Solutions (Cisco, Alteon, F5 and others)

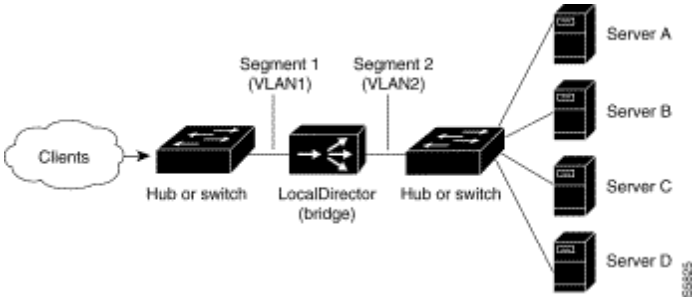
The Cisco LocalDirector is an example of a hardware-based load-balancing solution. Several companies make similar devices. Most of these devices manifest themselves as switches or bridges with additional software for managing specialized routing. The switch learns the IP addresses of all the servers connected to it. Based on machine availability and a balancing algorithm the switch takes the incoming packets, all with the same destination IP address (the LocalDirector's IP address), and re-writes them to contain the appropriate chosen server's IP address. The high-end LocalDirector can re-write packets for a 230Mbps data stream with up to 32 destination servers. Moreover, we believe, the director can support up to 16 independent data segments (sub-nets such that traffic on the sub-net is completely isolated from traffic on the other sub-nets)

The following is a blurb from http://www.cisco.com/warp/public/cc/cisco/mkt/scale-/locald/index.shtml

> *"Cisco System's Local Director is a high-availability, Internet scalability solution that intelligently load balances TCP/IP traffic across multiple servers. Servers can be automatically and transparently placed in or out of service, and LocalDirector itself is equipped with a hot standby failover mechanism, eliminating all points of failure for the server farm. LocalDirector is a high performance Internet appliance with proven performance in the highest traffic Internet sites."*

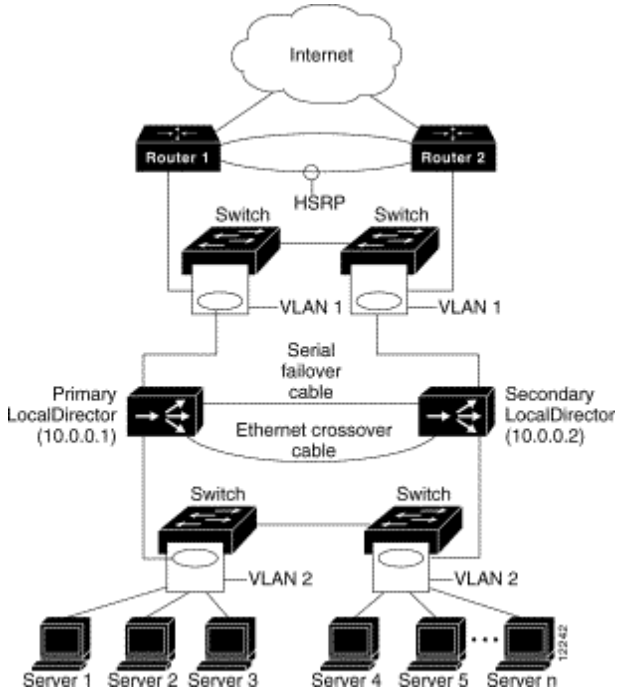
Below is an example of a LocalDirector configured in one of its simplest forms. A set of server machines is connected via a hub on one common Ethernet segment. The segment is connected to the LocalDirector (bridge).

LocalDirector with Hubs or Switches (simplest configuration):



The example below is considerably more complex. This configuration can survive any single point of failure (up to but not including, the servers) without the users being adversely affected. This example is included to demonstrate the complexity and sophistication that can occur with basic network building blocks.

LocalDirector in Highly Fault-Tolerant Configuration:



2.2.3. Software Solution (WLBS)

Microsoft's Windows NT Load Balancing System (WLBS), internally known as Convoy, is a software-only load balancing solution. WLBS creates a shared virtual IP address (VIP) among a cluster of Windows NT servers. WLBS load balances incoming TCP connections to the VIP across the members of the cluster.

WLBS is an NDIS packet filter driver. WLBS sits above the network adapter's NDIS driver and below the TCP/IP stack. Each machine in the cluster receives every packet for the VIP. WLBS decides on a packet-by-packet basis which packets should be processed on each machine. If the packet should be processed by another machine in the cluster, WLBS throws the packet away. If the packet should be processed locally, it is passed up to the TCP/IP stack.

The WLBS driver on each machine sees all incoming packets because all members of the cluster sit on a shared Ethernet segment. In essence, each packet is broadcast to every machine and each machine individually decides which packets to process.

WLBS uses a distributed hash function to determine which packets should be accepted on a given machine. WLBS hosts exchange heartbeat messages with each other once a second. The heartbeat message contains a host's view of the cluster. Based on heartbeats, each host knows about the state of other cluster members. Each host independently makes the decision to accept or reject the packet based on its host ID, the number of hosts in the cluster, and the information in IP header of the packet.

WLBS effectively partitions (through hashing) the IP client space among available cluster hosts and lets each handle its share of the workload. WLBS does not modify any information in the packet.

Using a management console, the system administrator can remove or add any host to the cluster at any time.

2.3. Example

Other architectural components of a megaservice include file storage, database storage (like SQL), database-access mechanisms (like ODBC), network bandwidth limitations, and server hardware (number of CPUs, hardware class, etc.). While each of these issues is important, we've found that people at Microsoft tend to have a much better understanding of these fundamental problems and scalability trade-offs than they do of load balancing.

Naturally, not every service needs to worry about all of these architectural problems. On the other hand, there is more than one service today that deals with virtually all the above-described issues. Future services (or services that have yet to scale sufficiently) will likely have to deal with all of these issues.

To reinforce the difficulty of constructing a megaservice, consider HotMail. It is a service that easily scales by user. In condensed form, HotMail consists of four classes of machines:

- Web server Front Doors (FDs): Stateless web servers that present the HotMail user interface via HTML.
- User data stores (USTOREs): Stateful servers persisting the email folders for up to 2M users.
- Member index servers (MSERVs): Stateless servers containing the global mapping from user ID to USTORE.
- Incoming SMTP servers: These servers accept incoming email messages and save them to the appropriate USTORE.

Hotmail is arguably the largest service Microsoft owns. HotMail has 47M users and handles many millions of email messages a day. On one occasion when HotMail suspended incoming SMTP connection for 2 hours, AOL's outgoing SMTP queue grew to about 2M messages. HotMail can independently scale each of the four classes of machines. They currently have over 1,300 Front Doors and 54 USTOREs. Cisco LocalDirectors allow the Front Doors to share a common IP address and automatically balance incoming HTTP requests.

HotMail's most costly scaling unit is the USTORE. Each new USTORE currently cost over $750,000. The number of I/O requests a USTORE can fulfill per second bounds the per user scalability of the HotMail architecture. The USTOREs are continuously pounded by multiple I/O requests per user page view. Furthermore, the USTORE is a single point of failure. If a USTORE goes down, up to 2M users lose access to their email (although they can still send outgoing email messages).

One of the proposed solutions to fix HotMail's problems is the following: since a user's email is limited to 2MB (and in fact it is often closer to 400KB), transfer the entire email folder to the Front Door and back in a single pair of I/O operations per user session. Furthermore, the email folder could be RAID striped across a cluster of USTOREs. The Front Door reads the email folder, twiddles on the bits over the lifetime of the HTTP connection, then flushes the email folder back to the USTOREs.

This "easy" solution overlooks the realities of the web. First, the law of large numbers: the access patterns of 47M users are indistinguishable from random noise. Second, the inherent parallelism of web activity: while one Front Door is rendering an email message to HTML, another email message may arrive for the same user. Third, the law of large numbers again: thousands of email messages arrive at a given SMTP server in any given minute, and they just keep coming. Megaservices don't get coffee breaks.

The "easy" solution becomes complex very quickly. Moving the email folder for every connection can be very expensive. HTTP and SMTP activity for the same folder can be concurrent forcing either expensive queuing or locking. The Front Doors and SMTP servers become stateful drastically complicating load balancing. Finally, HTTP connections can be very short lived: HTTP 1.0 clients reconnect on every page item request.

Building scalable megaservices is not easy, but it can be done. One can make the "easy" solution work and, in fact, a number of low-tech solutions work extremely well in the megaservice space. Our challenge is to gather the collective wisdom of those who have built scalable megaservices and harness it for the benefit of the company and our customers.

3. Observations

In this section, we present several of the most important insights and observations we have gathered from our survey. While these insights are presented here for conciseness, we strongly encourage a complete reading of subsequent sections. The true value of these insights is best appreciated in context. The bulk of this report contains our detailed descriptions of each of the megaservices we visited.

In order to drive this point home the lessons we learned have been classified into three buckets: Maintainability, Scalability and Availability.

3.1. Maintainability

- **Simple solutions are often best.** Many web services are basically, in the words of the parallel computing community, "embarrassingly parallel." For example, HotMail has exploited the inherent simplicity of per-user email partitioning to avoid the extra layers of software and architectural complexity that come with general-purpose extensibility models such as Application Server Providers (ASP) or Enterprise Java Beans. They have created a service that directly reflects the natural partitioning of the domain being modeled, and that scales and performs exceptionally well because of this. Whether this service will scale well into an era of inter-service integration remains an open question, but the simplicity remains striking – general architectures designed for effectively sharing the resources of a single machine are unlikely to adapt to a world in which the machines themselves constitute the component boundary.

- **Hire the best people for operations.** When we visited HotMail, we spent an afternoon with two development leads and the operations lead. Almost without exception, the operations lead answered all of our software architecture questions. He knew every scalability pitfall in the system. He knew the architecture as intimately as the people who wrote it did. Why does HotMail have fewer operators by almost any metric than any of our sites? Our conjecture: because the operations team knows the software.

  It is critical to recognize that while developers create code, operators create process. In the same sense that a service needs strong developers, it also needs strong operators to create the appropriate processes. Code may be the backbone of our products, but process is the backbone of our services.

- **Operations teams should be integrated into product development.** Development management at several of Microsoft's largest megaservices insisted that one of Microsoft's current failings is that we separate operations and development into separate teams, often in separate divisions. At both HotMail and WebTV, the operators are intimately involved in product development. The WebTV operators are pushing features one, and even two, software releases in advance. We found at least one example at HotMail where operations correctly predicted the lack of scalability of a particular feature long before the developers came to that same conclusion. Operators feel their users' pain, successful developers will feel their operators' pain.

- **Configurable off-the-shelf solutions are preferred to custom code.** When possible, it is better to adapt the design of the cluster to accommodate optimized hardware or software than to write custom code. LocalDirector switches from Cisco are widely deployed for this reason; they are reliable, well understood, predictable, and can easily be dropped into a network without adverse impact. This flies in the face of Microsoft's traditional extensibility solution, calling user-supplied code. Although code is a very general solution, it also demands much

more operational coordination than LocalDirector's simple configuration-driven solution. Likewise, many services expressed a desire for off-the-shelf state management for both user profiles and content management.

- **Low-tech rules.** Low-tech systems are often much easier to operate because they are much easier to understand. For example, with few exceptions the MSN properties use ROBOCOPY instead of CRS for content replication because it is restartable, reliable, and easy to understand. When it comes to managing a site with hundreds or thousands of machines, command-line scriptable tools always beat a fancy GUI.

  The value of low-tech is particularly evident in the megaservices messaging infrastructures. Only one of the sites we visited uses cross-machine DCOM. Even sites using very RPC-like communication patterns and proprietary software, such as the next version of MSN Chat, use low-tech message passing.

- **Less is more.** Internet users have shown a surprising tolerance for systems that are operationally reliable and responsive, but that aren't feature complete. This is probably symptomatic of both the immaturity of the market and the volatility of the Internet. This suggests that, within reason, manageability and operations should be given a higher priority than feature creep. Again, this is a shift from our traditional product focus.

- **Side-by-side component versioning.** Launching a new version of a service or one of its subcomponents involves risk. Each new deployment should include a backward path in the case of failure, as well as an incremental rollout strategy that can be adjusted in real time. Most sites employ physical side-by-side operations to accomplish this: the new version of service is deployed on new hardware parallel to the existing service. Either a DNS or a router switch is thrown to enable the new service. WebTV upgrades on the same hardware, in parallel directory trees; they switch between versions of the system by changing a soft link and restarting.

- **A service is never finished.** The Internet changes, competitors change, and the load on a service changes. Constant change demands that service be malleable and maintainable. A web service is only truly finished when the developers give up and quit. Once again, megaservices don't get coffee breaks.

3.2. Scalability

- **The network is an integral part of the system.** Network topology including placement of routers, switches, LocalDirectors, and subnets is crucial to service scalability. Service providers manipulate and exploit the network; it isn't just part of the environment. Imagine building HotMail without control over its physical network topology. Although they have not yet moved to hardware assisted routing of data, services such as Instant Messaging that depend upon multicast topologies or distributed event routing are interested in the deployment possibilities represented by programmable switches.

- **Careful partitioning.** As mentioned above, one common feature of all of the services we examined is that at some fundamental level they are embarrassingly

parallel. Whether separating HotMail data by user, or chat rooms by topic, the data decomposition is parallel and, for the most part, obvious. It is likely that as the Web evolves, the granularity of partitioning will become larger (moving from per-user to per-community, etc.), but ample opportunities for data partitioning and operation parallelism will persist.

- **Load balancing is a core component.** Load balancing is the invocation model that, when coupled with partitioning, enables scalability. At the level of IP packets and TCP connections, load-balancing solutions (like DNS round robin, WLBS, and LocalDirector) are readily available.

- **Stateless front ends.** Most of the megaservices we visited employ stateless FE machines (although they often depend upon state passed by the client within their logic). The front ends render HTML and embody the control logic used to issue requests to stateful back ends. In one sense, the stateless front end is just the bottom half of the UI layer in a three-tier system. In another sense the FE machines act as a high-level, application specific switch since they often switch back-end connections based on information in the HTTP request rather than data in IP or TCP headers.

  The primary disadvantage of stateless front-end architectures is that state must be pushed either to the client or to the back-end servers. Pushing state to the server implies that the site must support a notion of a unique ID (or login) and must provide a state database. Pushing state to the client-side implies that state must either be embedded in the URL or transported in a browser cookie. Cookies are problematic because they do not support user roaming and they are often considered an invasion of privacy.

- **Understand your connectivity.** Services must understand the nature of their internal dataflow in order to scale. This is one reason that the FE/BE distinction is so useful – the FE can offload the processing bottleneck associated with slow client connections and enable greater BE concurrency. In general, impedance matching by service components based on the connectivity patterns and bandwidths is an important part of megaservice design. Concern with the impedance relationships between database processing, rendering, and personalization is a topic that recurred with several of the groups interviewed.

- **Cost and performance matter.** Scalability of a service is affected by the performance of individual components. HotMail uses 9GB drives on their front doors, rather than cheaper 2GB drives, simply to get faster RPM drives; they don't use the space. Performance of ASP was a common complaint of the service developers we met with in our study. By rendering its top four pages from pure ISAPI (instead of ASP), MoneyCentral reduced its average response time from 32ms to 8ms per page. Optimization for the common case leads to a willingness to factor components to a very fine grain when necessary – any attempt at automatic or tool-generated factoring is better when it takes empirically gathered statistics into account.

3.3. Availability

- **A component should never fail due to an external component failure.** Individual components in WebTV have defined behaviors for all possible external failures. For example, if the login server is unavailable, WebTV will service the user with "unauthenticated" permissions. WebTV can function even when a remarkable number of its servers have failed. The WebTV home page service will render a start page even if the mail, ad, and stock ticker services are all down. It is considerably better to have broad and robust error coverage than to have an unprotected component that implements more features.
- **Components should fast-fail on inconsistent state.** The quicker an errant component fails, the sooner the rest of the system can work around the failure. As a negative example, consider the interaction between LocalDirector and IIS. Cisco's LocalDirector routes incoming packets to the server with fastest turn-around time. When IIS runs out of internal resources, a fast path immediately replies to all HTTP requests with a "come back later" reply. LocalDirector currently interprets this behavior as a very fast server and begins to route all incoming packets to the overloaded server. Similarly, WLBS currently only considers a server failed if it no longer issues a heartbeat, regardless of the health of the resident IIS server.
- **Monitoring is absolutely essential.** Many of the megaservices use extensive logging and counter-based monitoring, along with as much remote administration as possible, to both ensure continuous availability and to provide data with which to improve their infrastructure. Filtering, alerting, and visualization tools are an absolute necessity for sites with hundreds of machines in order to filter out important events from background "noise".
- **Systems should be designed (and tested) with component failure as a rule not an exception.** The standard for designing communicating components for the Internet is considerably different from Microsoft's traditional LAN-centric models such as ODBC and DCOM. For the most part simple socket-based protocols are used by megaservice components when they cannot piggyback upon an existing web protocol. Although not all of these protocols are designed to fail in a recoverable way, they all are designed to come back up as quickly as possible in the face of failure, and the services interviewed were always aware of the failure characteristics of their infrastructures.
- **The system should work partially even when components fail.** Running an Internet service is a double-edged sword. Users expect service. Sometimes they expect full service; sometimes they'll tolerate less. Even if a USERVE is offline, the affected users still get a HotMail web page and can still send email. On the other hand, Expedia has found that when it loses connection to its BE machines, it is better to deny users entry (with a stylized retry-later message) than to let them get halfway through a ticket purchase before failing.
- **Test suites should be delivered to operations as part of the platform**. At most of the sites, the test suites used by development are also used continually to sanity check the health and the performance of the system. Many of the sites (see full

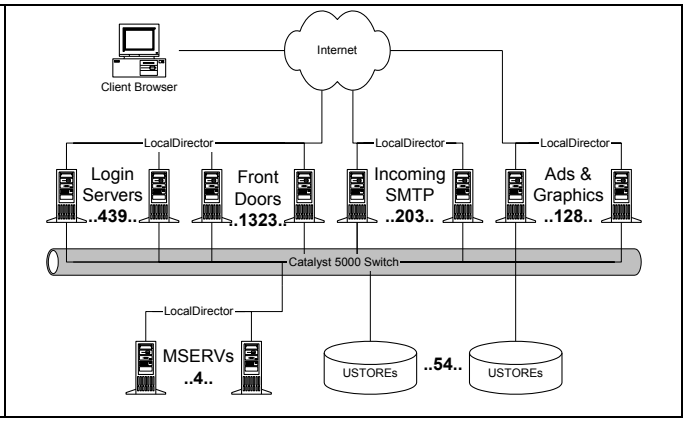
descriptions) have a “hidden” web page that exercises important site features. Viewing the hidden page alerts operations staff of any problems in functionality or user-perceived performance.

4. Services

This section describes the eighteen Microsoft Internet services we visited and c with some observations from the MSN Operations team. Each subsection describes a service’s basic function, physical architecture, operations issues, and scalability issues.

4.1. Hotmail

Configuration	
Front Ends:	2198
Back Ends:	58
Other Boxes:	54
Load Balancing:	LocalDirector
Interview Date:	July 22, 1999



HotMail is an Internet mail service founded in 1996. HotMail was acquired by 1997 and is still in the planning stage of its transition to Microsoft technologies. HotMail currently has 47 million users, and supports up to 77K simultaneous online connections. HotMail serves approximately 90M ad impressions per day.

Users connect to www.hotmail.com with a standard Internet browser. HotMail.com is really a set of Front-End (FE) machines behind a local director. All of the FE machines run FreeBSD and Apache. The FE machines communicate with one of the member index servers (MSERVs), a set of replicated machines that contain the index of username, <machine name, data segment> mappings, to determine which USTORE machine to connect to for data. The USTOREs are large SUN Solaris boxes that contain the user’s mail, password and customizations (aliases etc.).

The FE acts as an agent for the end user; it reads and writes files on the USTORE through the XFS protocol (an atomic mail storage protocol with some similarities to NFS) and generates the appropriate HTML. Ads and images are stored on separate servers to keep the load down on the front ends.

Instant Messaging, which is housed at HotMail, is detailed in a separate section.

4.1.1. Architecture

For Scaling, HotMail has defined a Hotmail Capitalization Unit (HCU). An HCU is the incremental unit of scale for adding new users to the system. Given today’s’ hardware a HCU covers approximately 2M users. An HCU is added approximately every month.

The prototypical HotMail HCU (HotMail Capitalization Unit) consists of the following:

- **User data server (USTORE) (1 machine).** Typically a very large Solaris box (latest machines are E4500s with 8 processors). The USTORE holds all of the email for a large group of user (up to 2M users). Backup is done to tape units attached directly to the USTORE. Other machines (FE boxes) access the USTORE files through XFS. Contrary to popular belief, XFS is not a file system; it is really an atomic mail storage and retrieval protocol. USTOREs are bound by two constraints: the amount of time needed for backup and the number of I/O operations per second. It takes 18 hours to back up a USTORE. USTOREs typically have a CPU load of about 5%, but a disk utilization of 100%.
- **Front Door servers (16 machines).** Front doors are stateless front-end servers; their primary responsibilities are accessing storage and HTML rendering. In addition to HTML, FDs also run spell checking, thesaurus, and dispatch outgoing email from HotMail users. Current FDs are 400MHz P2 boxes running FreeBSD and Apache. FDs are CPU and network bound. Each incoming connection requires at least two FreeBSD processes. The FDs within a cluster share a common IP address. Incoming requests are distributed with a Cisco LocalDirector.
- **Login servers (15 machines).** Login servers are stateless web servers that redirect users at login to the appropriate cluster. Physically they have the same configuration as the Front Doors.
- **Member index server (MSERVs (1 machine).** A global directory mapping users to USTOREs. All MSERVs share a common IP address distributed with a LocalDirector. Each MSERV contains the entire user directory.
- **Graphics servers (4 machines).** Simple Web servers for static graphics content. The graphics servers load all images into cache on boot up to reduce request latency.
- **Incoming mail servers (4 machines).** These run SMTP to accept incoming mail and dispatch it to the appropriate USTORE. Mail servers sit behind a single IP address and LocalDirector.

Multiple HCUs are combined behind common Cisco LocalDirectors to form a cluster. HotMail currently runs seven mail clusters; six at the Lawson facility and one at Wyatt. HotMail runs the LocalDirectors beyond the maximum recommended speed with the expected instabilities.

The HCU is an idealized model. As HotMail has scaled from 9M users at time of acquisition to 45M users, the ratios have morphed. For example, a common set of four MSERVs is shared across all of the HotMail clusters. Ad service has now moved to MSN Ads.

Given gains in hardware, offloading of ad tracking, and generally better performance from their code, the team expects that within 12 months one HCU (one USTORE), with the addition of the appropriate number of disk spindles, will be capable of supporting 4M users.

To support multiple clusters, users enter the site through a set of login servers (via DNS round robin). The login servers then redirect the user to the appropriate cluster. At login

time, a file containing the user's last access time is updated. A user's account is deleted after 120 days of inactivity.

At time of acquisition, the typical user had 240KB of stored email. Storage has grown to 400-600KB per user primarily because of email attachments. Cost has grown to $1.62/user/year with an addition $.60/user/year for Instant Messaging. Approximately 70% of the hardware costs are in the USTOREs.

Load during off-peak is 2/3s of load during peak time; many of their customers are foreign based.

4.1.2. Operations

The operations team is intimately familiar with all of the quirks and details of the system (including the actual code). In fact, HotMail's director of operations provided most of the technical development details during our interview.

HotMail's machines are housed at Exodus. Exodus supplies cages, power, and Internet connectivity. Due to Exodus' pricing model, HotMail has added redundant network connectivity to many of the major ISPs.

Operations are monitored remotely; all management is through remote shells and scripting. Updates are propagated to all of the live machines at once using RDIST. Feature upgrades are applied incrementally. HotMail has never had a "major" release.

Operations is deeply involved in all phases of development. They are a major feature driver for the system. Operations is involved at least one version ahead of deployment.

Server up time can be as much as 300 days (USTOREs), with 94.5% uptime for the service as a whole. However, on average across the whole system, they experience one USTORE kernel panic every two days. USTORE hardware failures are due, in order, to tape drives gone bad, RAID controllers fried, or dead CPUs. Front doors experience as many as eleven crashes per day. Luckily, front-door crashes are largely masked by the LocalDirectors.

Nightly backup has proven useful. HotMail has restored one tape backup in the last month. The login process had not been correctly updating the last access time for 120 days. After the garbage collector deleted a large faction of a USTORE, the operators noticed, the bug was fixed, and the previous tape was restored. This was the third restore in the history of the site. The other restores were due to serious RAID failures.

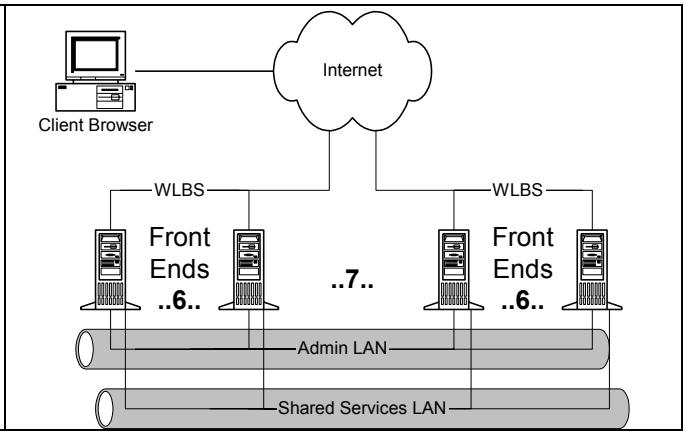
It is worth noting that users are routinely migrated between USTOREs. For example, in the last week, approximately 1M accounts were moved in order to retire two old USTOREs. The average churn rate between USTOREs is probably closer to 250K accounts per week. The migration of users from one USTORE to another is a good example of dynamic partitioning in action with the smallest unit of granularity being a single mailbox. It is interesting to note that migration is under administrative control.

The system has to be brought down briefly when adding a new HCU (all MSERV index structures have to be updated). Migrating users between stores does not necessitate downtime.

HotMail still is vulnerable to single points of failures – When a USTORE goes down mail is not available to 2M accounts. HotMail is also vulnerable to catastrophic events as all of their machines are hosted in one site (excluding ad servers).

4.2. Home.Microsoft.Com (HMC)

Configuration	
Front Ends:	42
Back Ends:	?
Other Boxes:	?
Load Balancing:	DNS/WLBS
Interview Date:	August 3, 1999



HMC supplies the main web page for the Microsoft portal. Much of the complexity of the home page arises due to personalization. Personalization information is kept in cookies on the user's machine and read before the page is rendered. How many things are on the page, their order, and localized content are all determined at runtime from the cookie.

4.2.1. Architecture

HMC consists of seven clusters of six machines each. Each box is connected to three networks: the Internet, an administrative LAN, and the shared services LAN. Ads, profiles and stock quotes are all accessed through the shared services LAN. Each machine is a quad-processor Xeon. Ideally, each machine would run with just four threads; one thread per processor for the Internet, the shared services LAN, IIS, and NT. In reality, they use approximately 40 threads due to the "lemmings" problem: large groups of users with the same source IP address (like AOL) hitting a single server.

Load is balanced between the clusters with DNS round robin; load is balanced within each cluster through WLBS.

Browser clients are bucketized into four classes:

1) IE4+ (support for DHTML).
2) ECMA Script.
3) HTML 2.0 - requires more server round trips.
4) IE2 - pop up a forced upgrade dialog.

The ASP code detects the 70 or so flavors of browser in use and reduces them to one of these four cases. The rest of the rending code is conditionalized to output for these four buckets.

4.2.2. Development/Testing methodology

HMC tests performance with ad hoc packet drops. They also rely on MSNSRVT (the MSN Server Test team) a user experience tool, SOC watch, and the WebCatThreads from the IIS test team. Most of their in-house testing is focused on UI.

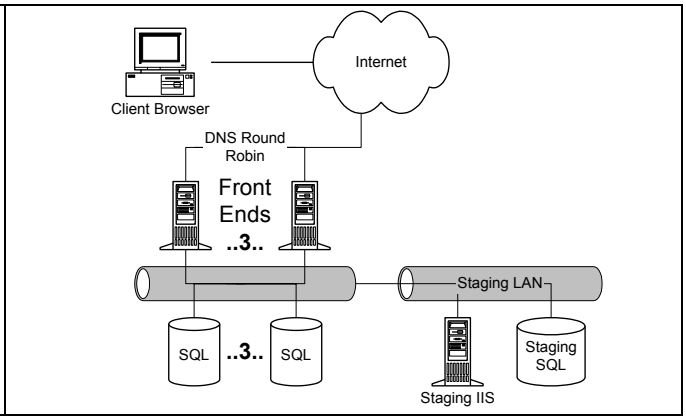
Aside from scale, HMC's major problem is content management. For example, HMC acquires stories from the Wall Street Journal (WSJ). Data arrives from the WSJ in a CDF file encoded in XML. The file is then split into headlines, HTML payloads and redirections for WSJ pages. The file is retrieved from a WSJ-provided URL every 5-60 minutes (depending on the specific file). The fetcher applies transformations to the data, publishes into HMC's CORAL SQL server and schedules it for display. The SQL data is then copied to a stager, which pushes data out to the web servers.

Another part of content management is the application of business logic rules. For example, sports scores come in unprocessed. HMC sorts the scores by team and league using business logic that defines the league for each team and adds URLs such as the team's home page. The logic of the team/league hierarchy must be encoded into the business logic; it isn't described in metadata delivered by the WSJ. Another example of business logic is the conversion of times from GMT to local time. As a final example of the complexity of business logic, HMC renders local news in the US based on the user's zip code if known. Internationally, local news is determined by region, but there exists no standard taxonomy for defining local regions on an international basis.

Content management in the human world is governed by defined processes. However, the state of web development is that processes must be captured and expressed in code.

4.3. Sidewalk

Configuration	
Front Ends:	3
Back Ends:	3
Other Boxes:	2
Load Balancing:	DNS
Interview Date:	August 2, 1999



Sidewalk is an online guide to entertainment with targeted content for 74 cities, primarily in the US. Sidewalk's primary challenges are content management (with customized data for 74 cities and 3-4 updates per day) and HTML rendering. In July, MS sold the arts and entertainment sections of its first 10 cities to TicketMaster's City Search. Essentially, TicketMaster bought version 2.0 of the Sidewalk rendering engine. The other cities (and the rest of the site) use version 3.0. The yellow pages section (along with local yellow page advertising) is the most lucrative side of the Sidewalk business.

4.3.1. Architecture

The sidewalk architecture consists of the following:

- **Front End (FE) IIS servers (3 machines).** Running a custom ISAPI rendering engine, each machine is DNS registered with 74 names (one for each Sidewalk city). An ISAPI filter maps friendly URLs to internal URLs. FE servers are connected to both the Internet and a local 100BaseT publishing LAN. Load is balanced between the FE servers using DNS round robin.
- **Mid-Tier (MT) SQL Servers (3 machines).** There is a 1:1 relationship between FE IIS servers and MT SQL servers. All page content is stored on the SQL servers. Custom OLEDB data service objects provide optimized support for connection management and query. A fourth MT SQL server stores non-content data.
- **IIS staging server (SS) and SQL staging server (1 machine each).** Identical to the FE IIS and MT SQL servers, these servers run on CorpNet. The IIS server is actually registered under 592 names (through a custom HOSTS.TXT file propagated to Sidewalk editors' and developers' machines). The 592 names cover the cross product of 74 cites, 2 modes (preview or live) and 4 content bases (current, next, and 2 others). New data is replicated from the staging servers to FE and MT servers every 5 minutes. When a new edition of Sidewalk is ready to publish (3-4 times per day), an update notification is sent to each of the IIS servers to swap the URL filter for "current" and "next" content.

The Sidewalk rendering platform (now the MSN rendering platform) creates a page composed of five panes. Traditionally these panes are the Header, Browse, Content, Ads, and Footer panes. Each pane is a COM object, called a response object, with an extensible set of properties and an output string. Coincidentally Sidewalk renders HTML into the output strings, but the engine does not assume HTML at any point. In addition to panes, the render platform manages any number of Render Support Objects (RSOs). RSOs contain state useful across the panes.

When a request enters IIS, the rendering platform creates a response object with three strings, called the Header, Head, and Body. As rendering progresses, text is copied into these strings. Just before returning, the rendering platform concatenates and flushes the strings to IIS. Request processing occurs in the following steps:

1) Request arrives in IIS
2) The friendly filter maps the request URL into internal canonical form (based on content switches).
3) Rendering runtime receives request.
    a) Platform asks AppObject for page layout. Note that the AppObject is just a blessed RSO. A ContextObject, ServerObject and RequestObject are passed to the AppObject and all RSOs.
    b) AppObject hands back the page layout in the form of a GUID for each pane object (inside glue for layout).
    c) The platform initializes all RSOs.

d) The platform initializes all panes (allocated from a shared pool using the Rental Apartment threading model). Typically, at this point, each pane sets properties on the ContextObject it wishes to propagate to the other panes.
e) The platform notifies all RSOs (and the AppObject) that initialization has finished.
f) The platform invokes the Render method on all panes.
g) The platform notifies all RSOs that rendering has completed.
h) The platform exits all panes and zeros the thread heap.

4) The page is flushed to IIS.

4.3.2. Development Methodology/Issues

Sidewalk use DNS round robin over WLBS because DNS responds well to stalled systems (i.e. the user presses refresh). Users balance themselves on a bad experience. WLBS would meet their needs better if it could adjust its load balancing based on machine load read from an NT performance counter.

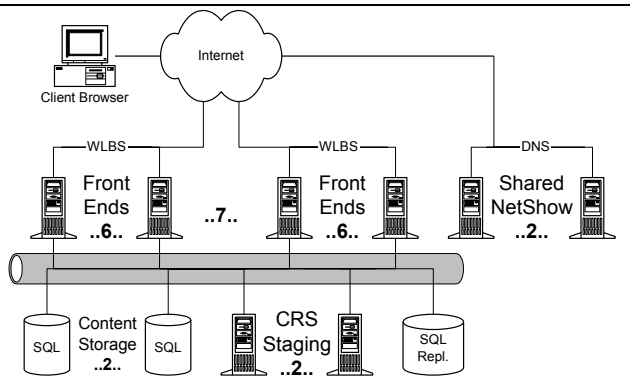
The IIS data is replicated using ROBOCOPY; SQL data is replicated with SQL replication. Sidewalk uses ROBOCOPY instead of CRS for file replication because it is restartable, reliable, and easier for operators to understand.

Memory management is a huge issue when rendering many pages. No matter what else changes, the maximum number of page faults the machine can service per second remains constant. The key to improving performance is to maximize the work done before the next page fault. This isn't just a matter of putting more RAM in a box. Soft page faults are still very expensive (due to limited number of cached TLB entries). Cache pollution is also a problem.

Sidewalk experienced major problems with heap fragmentation. To code around fragmentation they use "heap pairs", two heaps per thread. The thread makes the first 100K allocations on the first heap, and then switches to the second heap for the next 100K allocations. By the time the thread switches back to the first heap, it is most likely empty (or at least very un-fragmented). Two heaps actually use less real memory than one because they are less fragmented. Other memory tricks they use are expanding string buffers, global arrays, recycled memory, and judicious use of VirtualAlloc. It literally takes 1000s of string concatenations (via strcat) to build a single page. Only the outermost string is on the heap, the rest are on the stack. One could reduce the number of strcats by "pre-render" HTML, but the HTML header needs a content length for keep-alive.

4.4. MSNBC

Configuration	
Front Ends:	42
Back Ends:	2
Other Boxes:	5
Load Balancing:	DNS/WLBS
Interview Date:	August 3, 1999



MSNBC.com is the Microsoft-NBC joint venture whose charter is to lead the web with coverage of breaking news in general news, sports, business, and other categories, such as health, entertainment, local, weather, and so on.

In the last two years, MSNBC has grown from 100K unique users/day to 2.5M users/day (10M/month). The average user makes 3.4 visits per month vs. seven for the average web page and 3.5 for CNN. The average user views 3-5 pages before leaving the site. Scaling the service by orders of magnitude without a parallel scaling of cost is their primary problem

4.4.1. Architecture

Principal components of the MSNBC architecture are:

- **Front-end (FE) web servers (42 machines).** Each FE runs IIS with ASP and XML, WMP, and SQL in 512MB of memory. MSNBC averages 500 simultaneous online connections (SOCs) per box, but peaks at 2000. Every server has every single page. Peak output from the servers is ~50-60Mbps. ASP server-side objects cache frequently used content such current weather, headlines, and other material required to build personalized pagelets. The FE boxes are organized into seven clusters of six servers each. Between clusters they use DNS round robin; within a cluster they use WLBS across a shared IP address.
- **Front-end SQL servers (2 machines).** These machines contain online content indexes, scores, and surveys.
- **CRS staging servers and SQL replication server (2 machines and 1 machine).** The CRS servers update the entire website every hour to each FE box. The entire web site is approximately 1.1GB.
- **Shared NetShow servers (2 machines).** Sponsored by the NetShow team, these shared servers are available to any MS property.

The editors of MSNBC publish somewhere between 100 and 1000 articles per day, or about 25MB/day. Wire services, partners, and the MSNBC editorial staff generate content. Approximately 100 articles are written by the MSNBC editor staff each day; the remaining articles are acquired through largely automated processes. For automated content, data

feeds from the wire services are automatically categorized and beautified into HTML. For original content, editors place the data onto the staging servers.

Original content is generated using the WorkBench environment, which provides editing, workflow and publishing. Workbench is a template-based system. WorkBench has about 40 templates with specialized wizards to create dynamic content like surveys and live maps. Content from WorkBench is published through the "Borg" tool, which consumes XML and XSL (from WorkBench) and spits out content for MSN, HPC, SNAP, and GenStar.

MSNBC publishes up to 50 video-on-demand clips per day. Videos are served from the two, shared ITG/NetShow servers. Approximately 80% of the usage on those servers is from MSNBC. MSNBC anticipates using broadband to exploit always-on for news alerts. The hope is to do so without needing extra or secondary editorial staff. Photos are processed with tools like PhotoShop outside of WorkBench and then attached to Workbench stories for publication to the staging servers. A new system, FastNS, is about to be deployed that will allow creation of WMP files for on-demand viewing about 5 times faster than is currently the case.

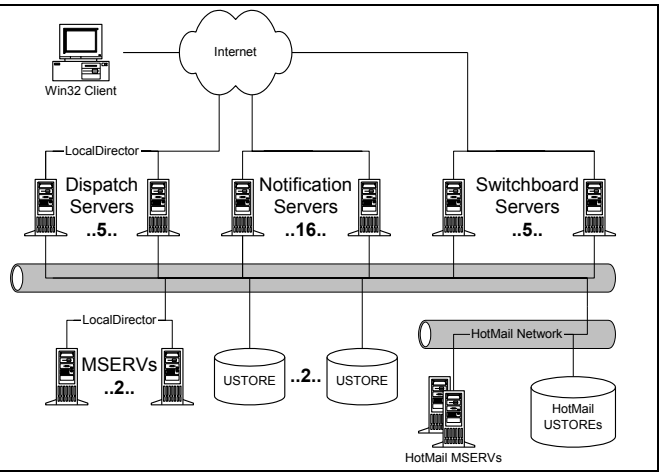
MSNBC recently developed and deployed an SSO replacement for the old MPS personalization system. This SSO presents an MPS-compatible API, so they don't have to change their pages, but decouples them from the old MPS storage system, which was slow and unreliable. In general, MSNBC tries to minimize dependence on data storage outside IIS servers, such as MPS, SQL, or the MoneyCentral quote cache, to avoid bottlenecks. Hence, MSNBC caches large amounts of external data in their global ASP objects, and refresh those objects on a frequent basis (approximately every ten minutes).

Unlike services such as Sidewalk, MSNBC does not use friendly URLs. All URLs are exposed directly. In addition, MSNBC actually changes the web page for a specific URL over the lifetime of a story. For example, they used a single web page for the entire Clinton/Lewinsky story over the period of a year. An implication of page reuse is that MSNBC does not support access to archived news articles.

Users navigate within MSNBC through an ActiveX menu control. A configuration file that is downloaded onto the client box drives the control. To reduce load on the system during peak periods, the configuration file is reduce from ~18K to a few K. Reducing the size of the menu configuration file is tantamount to pruning the space of categories/articles the user can see. Roughly 60% of MSNBC users have browsers with ActiveX support. Long term, MSNBC is very interested in pushing XML to the user. Many of their pages contain per-browser ASP code. Slideshows and some other client-side gadgets are DHTML; they send different versions from the server depending on the client.

4.5. Instant Messaging

Configuration	
Front Ends:	27
Back Ends:	4
Other Boxes:	HotMail Servers
Load Balancing:	LocalDirector
Interview Date:	July 14, 1999



Instance Messaging is Microsoft's entry into the online messaging field. Instant Messaging is really two services in one: a notification service and an instant messaging capability.

The notification service provides mail notification and online buddy-list services. Users connect to Notification Servers (NSs) and see a window listing the online status of their friends. As friends connect or disconnect (or request a status change), the client is notified and status is updated. Notifications occur through a persistent TCP connection between the NS and the client. The Instance Messaging protocol is open, but it is not intended to be "the" universal standard for Internet instant messaging.

The instant messaging service is started when a user initiates an online conversation session or double-clicks on a friend's name to send the friend a message that immediately appears on the friend's screen through a one-time, one-way conversation session. To create a messaging session, the client sends a request to the NS; the NS selects a switchboard server on which the communication will take place; the NS redirects the client to the switchboard server; the NS sends a message to the buddy's client via the buddy's NS; and finally, the buddy connects to the designated switchboard.

Instant Messaging started Spring '97 as an offshoot from NetMeeting. Initially the team planned to build both a client and a server, but stopped building the server in January '98 in favor of the HotMail-developed server. In January '99, Instant Messaging acquired ownership of the server code. While Instant Messaging is now responsible for the software, the service is hosted at HotMail's Lawson facility.

In its first 12 hours of production (July 22, 1999), the Instant Messaging service received 4000 connects, peaked at a network bandwidth of 400Kbps, and hosted 300 simultaneous online connections per server. Within 36 hours of launch, Instant Messaging was hosting 50K simultaneous online connections. The site regularly maintains over 120K connections, but has been tested to 500K simultaneous online connections.

4.5.1. Architecture

The Instant Messaging architecture consists of the following components:

- **Client software.** The user runs proprietary client software. The client connects to Instant Messaging through a persistent TCP connection. In order to encourage migration from AOL (who purchased ICQ and is the largest Internet instant messaging competitor), the client can connect simultaneously to both AOL's instant messaging servers and Microsoft's Instant Messaging.
- **Notification Servers (NS) (16 machines).** Clients maintain a persistent connection to one of the NSs. Clients are assigned to a specific NS using a static hashing function. If the number of servers changes, the entire service must be brought down to adjust the hashing function. Each NS maintains a list of all connected users. Notifications, such as on a client login event, are sent to the appropriate peer NSs based on the client "reverse" buddy list.
- **Dispatch Servers (DS) (5 machines).** Dispatch servers are used to locate the correct NS. Dispatch servers run the same code as notifications, but they have no assigned hash slots.
- **User storage servers (USTOREs) (2 machines).** Each user's buddy list is persisted to a USTORE (running the same XFS protocol and server code as HotMail's USTOREs). Each user has two buddy lists: a forward buddy list, people whose status the user wants to know about, and a reverse buddy list, people who want to know the user's status.
- **Member index server (MSERVs) (2 machines).** The MSERVs act as a global directory mapping users to USTOREs. All MSERVs share a common IP address distributed through a LocalDirector. Each MSERV contains the entire user directory.
- **Switchboard servers (5 machines).** Each online conversation is hosted through a switchboard server. Switchboards broadcast their state to the NSs for load balancing.

Instant Messaging users can control who has access to their online state (online, busy, offline), etc. A user can block buddies from knowing their state in which case the user always appears to be offline to their blocked (un) buddy. Users also control on whose buddy list they appear. When users add someone to their buddy list, a message is first sent to the buddy for authorization. Only consenting buddies are added to the user's list.

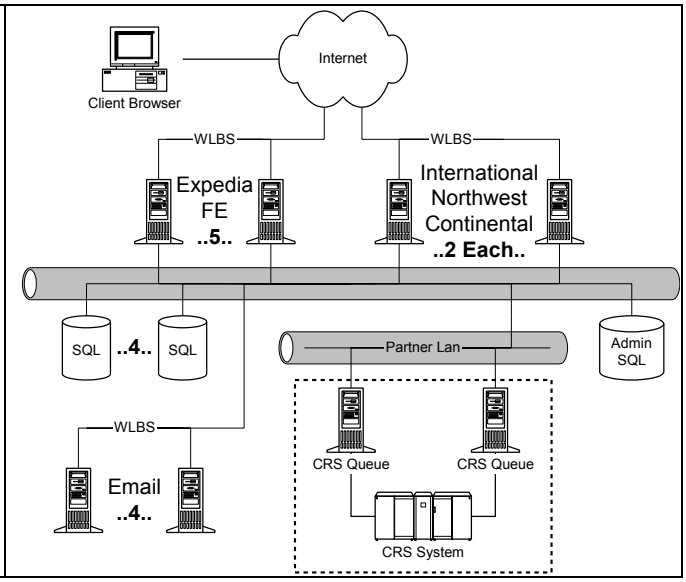
Once a user decides to communicate, they send a request to the notification server. The server chooses a switchboard machine to host the conversation, routes the initiator to the switchboard, validates that the receiver has permissions and is authenticated, and finally redirect the receiver to the same switchboard machine.

In addition to buddy-list notifications, Instant Messaging user's with HotMail accounts are notified of status of their mailbox at login time and as new email arrives. To enable the initial email status notification, the user's NS sends a message to their HotMail USTORE at login. To handle notification of incoming mail, the NS writes a notification file onto the user's USTORE. When a new message arrives, the HotMail postman sends a message to the user's NS if the notification file exists.

The Instance Messaging software sends asynchronous messages between servers, from client to server, and from server to client. They use a simple ASCII protocol instead of RPC or DCOM because RPC and DCOM contain far too much overhead for their purposes.

4.6. Expedia

Configuration	
Front Ends:	11
Back Ends:	12
Other Boxes:	5
Load Balancing:	WLBS
Interview Date:	June 21, 1999



Expedia is Microsoft's travel service. It provides users with the ability to find out flight information, plan trips, and choose the best fares. Expedia hosts about 230K user/day. They currently have 7M unique users with about 6M of those being registered users.

The most expensive request a user can make, in terms of cost to the Expedia service, is a PowerShopper query. A PowerShopper is a request to find the lowest X fares in a specific category. Expedia must pay their external partners for each PowerShopper whether the query translates into a purchase or not.

4.6.1. Architecture

Expedia consists of four components: web server front-ends, travel servers in the middle tier, external back-end travel services, and local databases.

- **Stateless web server front-ends (11 machines).** Connection state (such as user's universal identifier, called a TUID, etc.) is maintained on the browser in a cookie. The server processes the page request (retrieving the user's profile based on their TUID in the process), calls the travel server as needed to field travel requests, and returns the reply. Expedia 4.0 uses 8-10 front-end machines. They are proud of the stability of their service: web servers stay up about 1 week at a time. Load is balanced with WLBS.
- **Back-end, local databases (4 machines).** The local databases contain information like user profiles. Every single page access goes back to the profile database using the TUID as a key. All access to backend databases is through SQL stored procedures.

- **Back-end Travel Servers (8 machines).** Travel servers provide generic travel services to the front ends. Web servers send requests to the travel servers through synchronous DCOM calls, but transfer bulk data through sockets because DCOM is too slow. In-line components on the travel servers connect the travel servers to external back ends. Each component connects to a separate Computer Reservation System (CRS) such as SABRE, American Express, etc. Expedia 4.0 uses four CRS providers.
- **Back-end CRS servers.** Run externally by the various CRS systems, each CRS is fronted at Expedia.com by two CRS-specific queue servers. Most of Expedia.com's transactions flow to WorldTran via a message queue. Incoming messages are queued onto WorldTran's two Tandem servers, and are then fed to their mainframes for processing. In most cases, WorldTran's mainframes process transactions by contacting airline computers.

It can take almost 5 minutes for a ticket purchase to execute through WorldTran. Expedia can do nothing to reduce that transaction time. As such, Expedia has throttle mechanisms in the front end. It is Expedia's goal to accept a user connection only if they will receive good service. The travel servers give load feedback to web servers. In addition, they can manual throw a switch in the front-ends to reject 70% of all users for 1 hour. They would like to have special classes of users and to offer them preferred service. As is, users who have purchased a ticket in the past are not turned away when the service is throttled down.

In the past, all commerce has occurred below Expedia in WorldTran. They are exploring options to move the commerce closer. Part of this effort is a new best fare search (BFS) engine that is currently under development. BFS will serve as a pricing engine. The most important objective of BFS is to reduce the need to use "PowerShoppers" on the CRS systems since Expedia must pay the CRS providers for each PowerShopper.

4.6.2. Development and Testing

Pages are rendered from C++ ISAPI extensions. Expedia developed a custom rendering/scripting library, QScript, which hides authentication details behind roles, like administrator, user, etc. QScript was roughly a three man-year development effort, but they are continually improving it.

Current development model is to upgrade the entire system at once. However, over the last 4 months they have made 98 hot fixes. Hot fixes cover things like new airports, new or deleted airlines, etc. They want to move to a new development model from large fixed releases to slipped-in upgrades. Primary goal is to be more nimble.

Expedia has explored using Yukon, written by MSN search, as a 3-level cache to hold temporary (last 5 minutes) state. Expedia would prefer that the cache be permanent to get around problems with storing state on the browser.

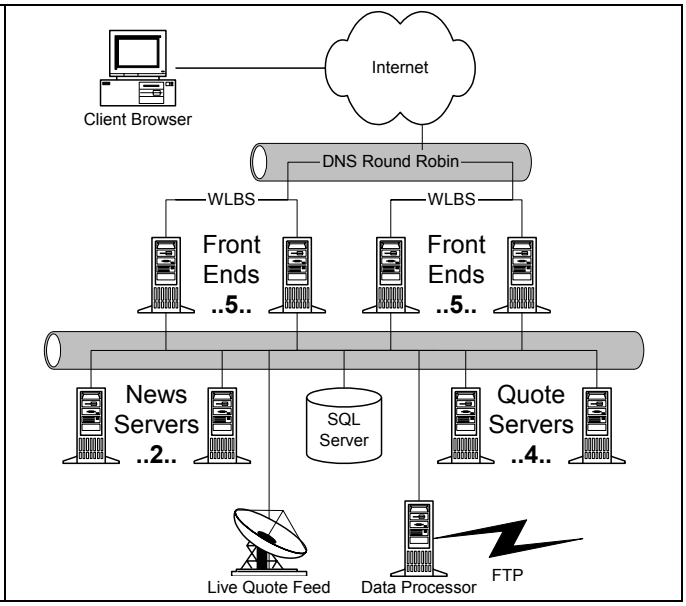
Front-end Test uses Orville, which does cloned multiple-replication testing. They aggressively use NT performance counters. Every piece of code dumps performance counters for operators. They also rely on IIS logging.

On the client side, they had a bad experience with ActiveX controls. ActiveX became a huge debugging nightmare due to all of the variations in clients. Only about 60% of their

clients are IE. They are starting to exploit DHTML. DHTML is structured so that it degrades gracefully on down level clients.

4.7. MoneyCentral

Configuration	
Front Ends:	10
Back Ends:	7
Other Boxes:	2
Load Balancing:	DNS/WLBS
Interview Date:	June 23, 1999



MoneyCentral's vision is to be the best place to research, make decisions, and take action on personal finances.

MoneyCentral serves 6.5M pages per day to 721K users (4.3M users per month). MoneyCentral's generates revenue primarily from ad views and value to the rest of the portal team.

MoneyCentral has 2 operators and 12 developers: three for client, five for web, and four for servers. They have their own operators to handle site-specific issues and rollout.

4.7.1. Architecture

Key components of the architecture:

- **Client's browser** (with ActiveX control for graphing). All user state is cached on the client. However, they support roaming profiles through the subscriber database.
- **FE machines (10 machines).** Quad Xeon 500MHz machines. Each request creates a new socket to back-end servers. Historical data is stored on web FEs and accessed via a 600MB memory mapped file through ISAPI extensions with two DLLs. Web servers are purely monolithic. Requests are load balanced with DNS round robin between two IP clusters, then WLBS within each cluster of five machines.
- **Quote servers (4 machines).** Input comes either from live feeds (satellite) or from the data processor. The quote server was written largely by one D14. It is a highly optimized table database engine. Written using the pipelined server model. The FEs "round robin" requests to the quote servers. Other services receive feeds from the quote engine (HMC, mobile).

- **Data processor (1 machine).** Retrieves quotes via FTP from providers.
- **News servers (2 machines).** Handle news, billing and some email. Bulk outgoing email is outsourced to Communiqué. News articles are retrieved from MSNBC via FTP.
- **SQL server (1 machine).** Contains down level portfolios, subscriber database (billing, etc.), and assorted databases with small hits (banking ranks, etc.). Approximately 80-90% of their SQL code is in stored procedures.

Goal of the architecture is to handle the traffic of a market crash plus 25% or about 2x the average daily usage.

Most of the pages are written in pure ASP. The site contains about 200 core pages plus thousands of articles rendered through those pages. The top four pages (accounting for ~50% of hits) are rendered directly from ISAPI extensions. The system used to average 30ms/page, but changing these four pages to ISAPI reduced average time to 8ms/page including the time to call quote server. There is no feedback from back-end to throttle front-ends because the front-ends are the bottleneck.

One third of quote requests come from external source, primarily HMC. The HMC quote client, written by MoneyCentral, caches the top 1000 stocks.

Ads are served by the MSN AdsTech. The Advisor FYI runs as a data-mining engine on the quote servers. It passes over the data with a set of filter formulas and generates events.

4.7.2. Development and Operations

MoneyCentral's product cycle is a big release about once every 6 to 9 months.

They upgrade hardware as quickly as possible (largely due to the per-machine overhead costs on older hardware at Canyon Park)"

ITG charges roughly $7000/month/server at Canyon Park for network infrastructure, etc:

- $4K overhead
- $1K network
- $400 per ticket - each time an onsite operator has to touch a machine.

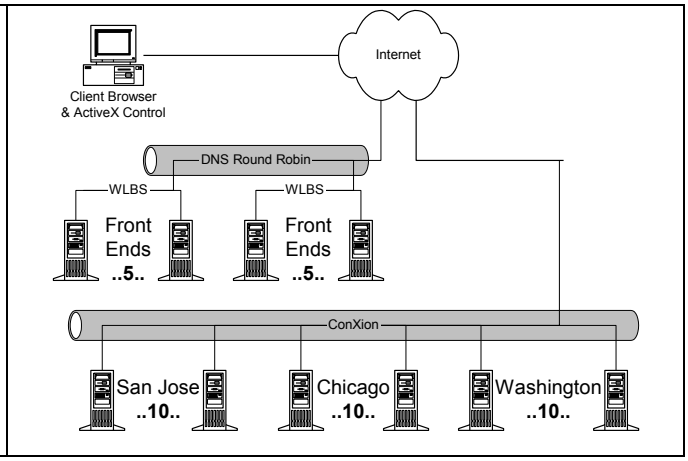
Major scaling issue: ASP performance. They try to use HTML that works on all browsers as opposed to writing slow ASP code to do browser detection. As a result, like many other sites, they have least common denominator support for browsers. They believe their pages are two heavy. Yahoo's pages are under 10KB. People value the quick download.

Profiling is done primarily through NT PerfMon and event logging. They also use WebMon to answer simple questions like: "Is the server up?", "Can I get a page?", "Can I get a quote?", etc. WebMon has a scheduling component with a retry count. Failures are sent to an event dispatcher via email.

For thin clients, they use down-level pages with GIFs for charts. They support mobile clients through separate quote pages authored by the MSN Mobile team.

4.8. Windows Update

Configuration	
Front Ends:	10
Back Ends:	30
Load Balancing:	DNS/WLBS
Interview Date:	Aug. 23, 1999



The Windows Update Service (WUS) is the first step on the road to extending the notion of Windows onto the web. The WUS objective is to ensure Windows is a living constantly evolving entity. Equally important is the objective of simplifying the care and maintenance of PCs for consumers and small businesses and fostering a lasting relationship with those customers. To motivate the team on one of their walls is a quote from Barry Schuler, president of online services for AOL:

> *"The reason Microsoft has failed in the online service business and hasn't been able to get going is that they don't have that relationship with the customer"*

4.8.1. Architecture

The Windows Update architecture for the system shipping today is quite simple. It contrasts in some fundamental ways with most of the other services. In particular, their *raison-d'etre* is the rich client. All of their clients are Windows-based and must be running at least IE4.0. All their clients support ActiveX controls. Consequently, they can use DHTML (perhaps XML in the future) and client side computation to offload the service complexity.

The basic architecture consists of:

- **Client components.** ActiveX controls and DLLs to aide in machine configuration detection.
- **Front-end (FE) web servers (10 machines).** Quad Xeon 450MHz machines running IIS. FE boxes are load-balanced via two groups of five machines. DNS round robin is used to select a group and WLBS is used within the group.
- **ConXion back-end servers (30 machines).** WUS downloads come from ConXion through an existing contract with ITG. There are 30 machines serving Windows Update downloads: 10 in San Jose, 10 in Chicago, and 10 in Washington D.C. Each machine houses all of the downloadable bits.

The basic control-flow is as follows (client side operations):

Presenting the user with the list of items to update:

- Controls are downloaded to the client, if needed,

- The client controls rummage around to determine the machine configuration. The DHTML is actually static with VB scripts that dynamic hide/show contents based on results returned from the client controls. In fact, everything is driven by the DHTML scripts calling into the client control (detection, download and install).
- Download from the server the list of available catalogs
- From the machine configuration determine which catalogs are needed, and download them. These catalogs are un-pruned with minimal info (like version number, location of installation data on server, etc).
- Merge user configuration description with catalog descriptions looking for things that need to be updated.

Performing an update:

- User selects an item to be updated.
- Request is made to FE to download the information needed to perform the update (e.g. location of the all the individual pieces [binary chunks] that are needed to install the package).
- Over the course of minutes, hours, or days, in the background, acquire all the little pieces.
- Assemble the pieces into the installation package.
- Perform the install.
- Send a post to the service as to the success/failure of the process.

Multitudes of small details must be handled on the client. The client control needs to determine when the box is online. It needs to detect when the user connection and machine are idle. It needs to bandwidth-throttle the download so as not to swamp-out user requests. It needs to keep track of which pieces it has and which ones it needs to get. Note that these issues apply to the next release, not the current one.

There are no connections between FE and BE machines. There is no dynamic content of the FEs – pure DHTML. There is no DB access. The client does all the work.

Load balancing for download servers is done via "Hot Route" a patented, proprietary mechanism used to determine from which machine at which ConXion site the client should download via client re-direction at ConXion's site.

4.8.2. Development Methodology and Issues

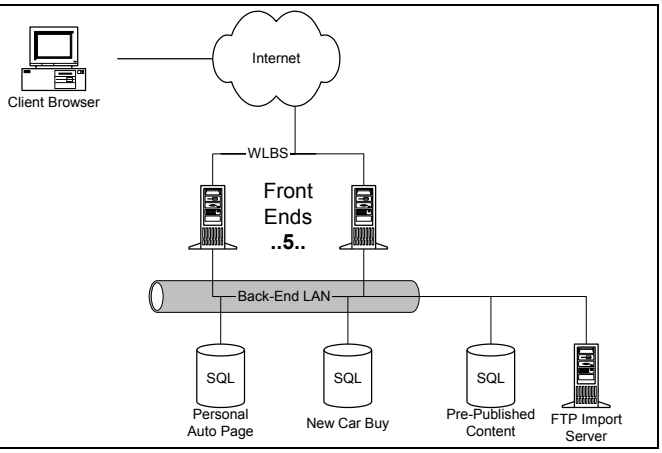
Today, one of the WUS team's challenges is the creation and management of the DHTML catalogs. WUS supports updates from Microsoft internal customers as well as certified drivers from ISVs. Content for updates from partners is created manually and added to a DB. Every two weeks they go through a complex build process, taking into consideration support for 10 platforms and 31 languages, which builds up DTML catalogs and additional data.

When the bits are ready, those destined for the FE boxes are propagated via the PUBWIZ tool (based on NT's multi-threaded file copy program – similar to ROBOCOPY). The

packetized bits destined for the download boxes are sent to one central site in San Jose and ConXion propagates the bits to their boxes.

4.9. CarPoint

Configuration	
Front Ends:	5
Back Ends:	2
Other Boxes:	2
Load Balancing:	WLBS
Interview Date:	Aug. 19, 1999



CarPoint provides end users with access to new car reviews, surround video, reliability review information, and competitive price shopping. When appropriate, users are put in contact with local dealers to expedite the conversion to a real sale. CarPoint is really three services in one: the one the users see, CarPoint; and the two the dealers see: New Car Buying (NCB) and the Used Car Marketplace (UCM).

4.9.1. Architecture

CarPoint's architecture consists of the following components:

- **Front-end (FE) web servers (4 machines).** Quad P6 200 MHz IIS machines. Most data served to customers are preprinted HTML pages that are replicated on all FE machines. All FE machines are running MSN AdsTech's client SSO. FE machines are load balanced using WLBS
- **SQL servers (2 machines).** Contain all the articles, videos, reviews and customer correspondence.
- **SMTP daemon.** Co-resident with one of the IIS servers, the SMTP daemon accepts incoming mail from clients and dealers.

CarPoint has a simple email notification system. As part of their personal page, users can supply information about their car (like make, model and date purchased) with this info CarPoint notifies the user via email when their car needs an oil change or needs to be serviced. The personalized data is stored in a SQL database, which is mined nightly. The CarPoint team rolled its own email notification system. They started using ECHO, and then grew their own. Although they spend little time maintaining their own user profile component, they would consider using a common profile store if it met their needs.

Content management

Offline they have an FTP import server (as well as other forms of data collection) that acquires data and stores it in an offline SQL database. This data is then massaged and

converted to either ASP or HTML. One significant problem they have is that there is no common schema to represent a car's year, make, and model, etc. As a result, they have a full time person checking the data for both format and validity. In theory, they think that BizTalk will help in this area. Pre-publishing data is stored in the SQL database as XML. Once a week, XSL is used to convert the XML data to ASP and HTML, which is then flushed to the IIS Front-End servers.

DealerPoint Architecture

DealerPoint has the same basic hardware architecture as CarPoint. It is built using a three-tier model with data access on the back end, business objects written in Visual Basic on the middle tier, and a specialized presentation (rendering) language and engine on the front end called TWERP. The CarPoint team built TWERP because of perceived problems with XML and XSL. Specifically, their HTML was not completely well formed, necessitating a large HTML to XML migration effort. They also saw problems with handling BLOBs in XML. They have since changed their opinion of XML and will probably migrate to XML in the near future.

4.9.2. Issues

Tradeoffs between rendering speed and flexibility have been an ongoing struggle. In the first implementation, CarPoint used SILK as their rendering language instead of ASP.

Originally, CarPoint kept all of their live data in SQL databases. For every page, they used ADO via ASP to connect to SQL and get the most up-to-date data. They soon realized that access to the database (setup and teardown of connections) was killing them. Around the same time SILK was developed. Its claim to fame was that it optimized connections to the database, cached data, and hid most of the details.

At some point, they realized that SILK was not caching everything and there were still performance problems. They thought about rolling their own FE cache manager but decided that it was too expensive to build.

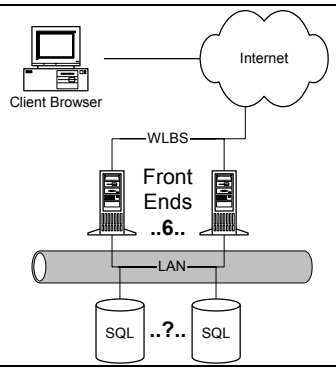
At this point, they stepped back and analyzed their content flow. They concluded that if they updated the bulk of the articles only once per week they could pre-print most content and push it to the FE boxes. This had three significant benefits. First, the cost of database access went down. Second, the throughput of the FE boxes increased as most of the rendering went from dynamic to static content. Finally, because the pages were now static, they knew at design time exactly what the user was going to see.

They are currently exploring XML/XSL for DealerPoint and possibly CarPoint.

CarPoint updates the contact database as mail comes in. They are considering MSMQ for reliability between SMTP and the database.

4.10. Calendar

Configuration	
Front Ends:	6
Back Ends:	TBD
Load Balancing:	WLBS
Interview Date:	July 22, 1999



Calendar is a personal calendaring service in development from the Jump team. Jump Networks was purchased in April 1999. The Jump site (http://jump.com) is operating as a beta, but does not accept new user registrations. Jump has 60K users. Over the last 60 days, there have been ~120K calendar transactions: 20K index requests, 3K add requests, 1.5K modify requests, 63K Put requests from synchronizing software (1 per calendar entry), and 38K batched Get requests from synchronizing software.

The primary usage of Calendar is for external, not self-generated events. Examples of external events include television schedules, CD release schedules, and concert schedules. Aside from banner advertising, the majority of income comes from advertisers who schedule promotional events into user's calendars (subject to user approval).

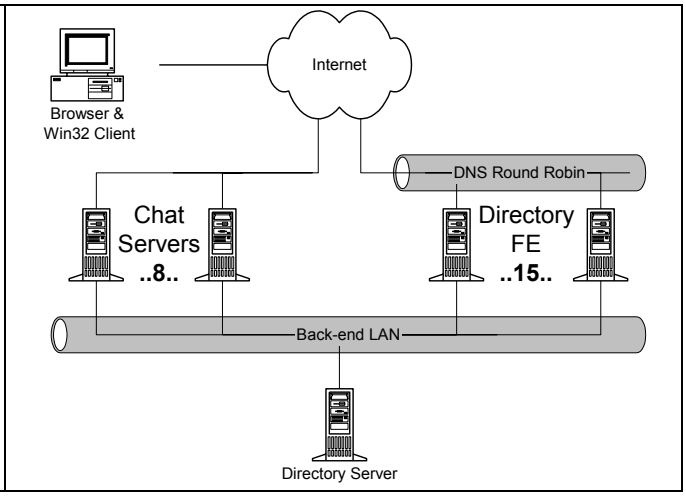
4.10.1. Architecture

The proposed Calendar architecture consists of the following:

- **Client software.** Users can access Calendar either through HTML, XML, or program specific relay protocols.
- **Web Front Ends (6 machines, 4x Xeons).** These run ISAPI extensions in IIS on Win2K. FEs communicate with back-end servers using one OLE DB (ODBC) connection per server. FEs select the back-end server for a user using the virtual resource layer (VRL) developed by MSN Communities. The VRL hashes the user ID into 10K buckets, then redirects to the resource based on the value in the bucket. XML front ends and relay front ends exist to support synchronization with client-based software.
- **SQL Back Ends (number of SQL server is still unknown).** SQL acts as store for calendar data. The original Jump architecture used Oracle with no stored procedures due to poor performance. When we visited the Calendar team, Jim Gray suggested strongly that they use SQL stored procedures. Oracle's support for stored procedures is very weak, but stored procedures work very well under SQL Server.

4.11. Chat

Configuration	
Front Ends:	15
Back Ends:	9
Load Balancing:	DNS
Interview Date:	July 21, 1999



Chat support two types of community rooms: user-created communities (UCCs), chat rooms created by users for the topic of their choice, and social chat communities (SCCs), chat rooms sponsored by MSN. UCCs tend to be short lived and low volume. The typical user will check in and out of a number of chat rooms before settling into a room for the evening. Peak time is in the evenings.

4.11.1. Architecture

The current MSN Chat system uses an Exchange 5.5 backend with a large set of modifications for performance and customization in a star topology. Users connect to an Exchange server on the edge of the star. Chat messages to other users in the same room, but on a different server must pass through a server at the center of the star. Center servers are the primary bottleneck. Effectively users are bound to machines. Each Exchange server will support about 4000 simultaneous online connections. The average room contains 20-30 people.

Chat's new architecture will bind rooms to a machine and direct users to that machine. No room will ever span more than one machine, reducing the cluster from a star to a flat topology. Release date for the new architecture is late September 1999.

The Chat architecture consists of:

- **Chat client software.** The current system supports any chat client using the IRC protocol. About 78% of users come in through custom Win32 clients. In the next release, chat will move to a proprietary client. The proprietary client will support MSN advertising. While the change will reduce the number of current clients, simultaneous they will advertise Chat heavily to HotMail users (over 80% of whom use Win32 machines). Chat will contract out development work for Macintosh and WebTV clients. WebTV users account for about 15K of Chat's simultaneous online connections.
- **Front-end chat servers (FEs) (15 machines).** Users connect to the front ends to find and create chat rooms. Pages are rendered with ASP. FEs retrieve chat room information from the directory server (DS).

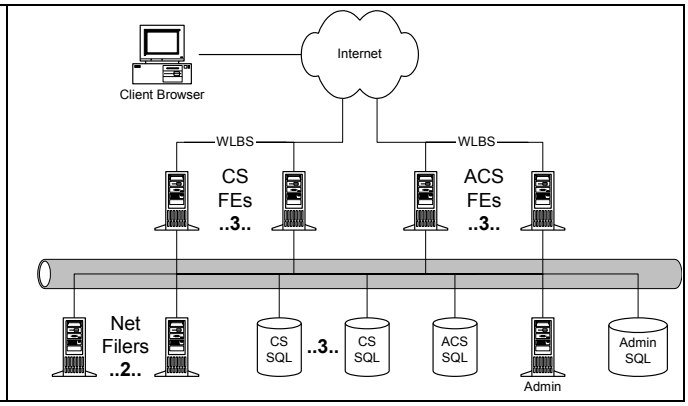
- **Directory Server (DS) (1 machine).** Maintains list of all chat rooms and mappings of users to chat rooms. Gather statistics from Exchange back ends. Design goal is to support 40K simultaneous online connections per DS with one DS in the system. The interface from DS to Exchange is proprietary.
- **Exchange servers (8 machines).** Support chat rooms. Both the old and new architectures use eight Exchange servers, although in the new architecture all eight servers directly host users.

With the new architecture, their single point of failure moves from the hub of the Exchange star to the DS. They are mulling over ways to make the DS more scalable.

Their intention is to push point-to-point communication (like file transfer, audio, and video) to MSN Instant Messaging.

4.12. Communities

Configuration	
Front Ends:	6
Back Ends:	8
Other Boxes:	2
Load Balancing:	WLBS
Interview Date:	July 15, 1999



The MSN Communities service provides a site for users with common interests to share a space. The shared community space includes web pages, picture albums, distribution lists, and a chat room. Their primary competitor, Yahoo, has about 150K clubs. The service is inherently scalable because it can be partitioned by community.

4.12.1. Architecture

Key components of the architecture:

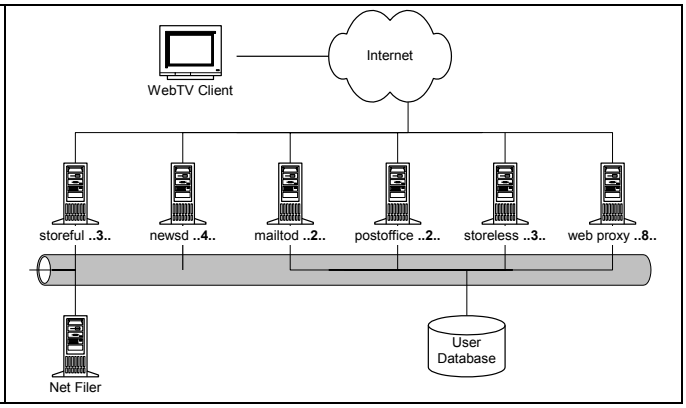
- **Client browser.** Basic HTML and some JavaScript, no ActiveX controls.
- **Front-end web servers (FEs) (6 machines).** MSN Communities uses two classes of front-ends: Content Stores (CSs) and Attributed Community Stores (ACSs). They use WLBS to load balance across one shared IP address for the CS servers and another for the ACS servers. A "vanity" ISAPI filter converts user friendly URLs to site internal URLs. Each CS can process approximately 300 requests per second; each ACS can process approximately 250 requests per second. The rendering engine is based on the Sidewalk engine.
- **Virtual Resource Layer (VRL).** The VRL sits on the FEs and maps requests from web FEs to backend SQL databases and NetFilers. The VRL hashes a key into one of 10K buckets. Each bucket contains a set of resource pointers (about 6 different

data types) to backend servers. The system can be re-balanced by updating the resource pointers in the buckets.

- **Passport authentication.** External machines.
- **Back-end SQL servers (8 machines).** SQL servers are divided into storage for CS and ACS.
- **Back-end NetFilers (2 machines).** NetFiles provide BLOB storage for web pages.
- **Other servers (5 machines).** One administrative machine, one administrative SQL server, and three machines for publishing and notification services. The notification service sends email to users on activity or content change within the communities to which they belong.

4.13. WebTV

Configuration	(Per cluster)
Front Ends:	23
Back Ends:	1 1/3
Other Boxes:	
Load Balancing:	Custom
Interview Date:	August 6, 1999



WebTV is a full service comprised of WebTV boxes (thin clients with no unique persistent data) that reside at the customer's premises and a set of service machines located in WebTV run data centers.

Approximately 75% of boxes are turned on each day – average session per box is ~100 minutes. Users access the WebTV servers through 36 ISPs, both regional and national. Selection of the ISP is transparent to the customer and is driven based on analysis of WebTV's data warehouse. Boxes dial into different ISPs based on time of day and user usage patterns to minimize cost while still giving a good customer experience. WebTV's single largest expense is the cost of user connectivity even though costs have been reduced by a factor of three since the service was launched.

4.13.1. Architecture

WebTV's has a theoretical Service Group, the set of machines with inter-dependencies required to deliver service to a customer and provides a deployment guideline for capitalization costs. However, they acknowledge that the "clean" Service Group turns into a myth as the operators drive to reduce cost. Cost reduction often drives "pooling". Pooling involves hosting more than one service on the same machine and sharing machines between service groups when isolation between service groups is not required. Another interesting aspect of "pooling" is that it affords the opportunity to run a small service on a

collection of machines to achieve N+1 availability. WebTV defines the following clustering rules:

- Persistent data servers (the back-ends) tend to define how you cluster the front-end machines.
- Grouping will be influenced by costs
- Service group size is defined by the exposure you are willing to face when a back-end machine dies.

Front ends run services that customers hit directly. WebTV front ends have no unique data and can be easily replaced. The hardware for front ends is selected for the price-performance "sweet spot," but not necessarily the machines best suited for the task. Front-end machines are expected to die and the system provides mechanisms to route around failed front ends. In the most catastrophic event, a WebTV set-top box will dial a set of 800 numbers to hunt for a WebTV configuration server.

WebTV back ends are stateful. Backend services must be hosted on highly available platforms. Backend hardware is selected for maximum performance to reduce the number of expensive servers and enable large pools of front-end servers. WebTV typically uses Network Appliance Filers running the NFS protocol for persistent data storage. These machines were selected for their high performance characteristics and high reliability. WebTV is not pleased with all of the characteristics of NFS, but chose it as a matter of expediency. WebTV built a "UserStore" abstraction so NFS filers could be replaced at a later date if appropriate.

A WebTV Service Group contains the machines needed to service 75K customers. For provisioning purposes, WebTV assumes a maximum of 12% of their customers will be online at any given point in time. As a result they need to configure for ~9,000 simultaneous online connections per service group.

The unique part of a service group is seven "storeful" machines bundled around a NetApp Filer for persistent storage. Storeful servers support user-oriented services (except cookies) that require persistent storage, including mail and Usenet news.

As noted above, each service group also needs a portion of some pooled resources. Normally these resources are left in one large pool. When WebTV stages roll outs, some machines might be un-pooled and assigned directly to a specific service group to isolate new service software. When a new service group is installed, the following resources are added to WebTV's pooled resources:

- Fair share of a Cookie Clusters (3 FE machines, 1/3 of a NetApp Filer)
- Storeless servers (1.5 machines).
- Proxy servers (12 machines), which are used to access all external web pages and convert them to WebTV device friendly data.

There following services that are shared globally:

- 4 Customer Database: 1 Central Read/Write Database, 2 Load shedding read-only databases (can be fail-over read/write if needed), 1 Billing read-only database.

- 3 Electronic Program Guide Services (Sun E450s with lots of disk)
- 10 Mail Notify Servers (tracks online clients and send UDP packets to boxes to tell them they have new mail)
- 8 Mail Gateways: 2 internal transit, 8 incoming (will drop to 2 when we deploy next generation MTA in the next few weeks), 2 outgoing mail hosts
- 7 Logging Hosts: 1 harvester (aggregated all logs), 2 servers which make the aggregated data available to various tools, 4 machines which run various other monitoring services.
- 4 Administrative Hosts: remote console service, network boot server (golden master machine) and general administrative tools.
- 2 Radius servers used by external ISP/IAPs to authorized our customers access
- 3 DNS name servers
- 2 Ad servers (the only NT boxes in our service)
- 3 FlashROM servers – used to upgrade client boxes
- 2 machines used for running backups
- 2 Scriptless servers to configure new machines.

Communication between set-top boxes and the FEs is handled though persistent TCP connections. Set-top boxes communicate via a custom protocol called WTVP (Web TV Protocol). Essentially, WTVP is an extended HTTP 1.0 (WTVP was created before HTTP 1.1), WTVP includes:

- Persistent connections.
- Encryption and authentication using a shared secret.
- Compression.
- Specialized headers with commands to reboot the WebTV box, flush its cache, and manipulate the backlist on the WebTV browser.
- "Tickets" – an opaque blob similar to today's cookies. The ticket arrives in the HTTP header encrypted with a service key and is passed on every connection.

Load balancing is achieved with two technologies. First, the login service for WebTV, called headwaiter, provides a service routing function. It hands a round-robin list of multiple servers for each service a client requires. If one server fails for a given service, the client retries the next server in the service list. Additionally, WebTV uses Alteon switches in front of a set of hosts. The first "server" in a client's service list is typically a "virtual" host, which the Alteon hands to an appropriate host.

4.13.2. Development methodology

WebTV's Solaris-based services use a single master configuration file. The configuration file lists the identity of all machines in the Service Group, their desired configuration, and optional settings. All system software is distributed to all machines in the Service Group.

A particular machine knows which services to enable based on its entry in the configuration file. The configuration files uses prototype style inheritance, so configuration values can be set for the entire service while permitting over-rides based on class of machines, service groups, or individual services on individual machines.

The WebTV service is highly instrumented thanks to an event logging system that integrates data from the clients as well as all the services. As a result, it is nearly possible to track user clicks through the entire WebTV service. WebTV's logging service aims to offer high performance and low overhead. WebTV logs over 140GB/day to a central data center repository. Logs are continually spooled to the repository to avoid the need to make "offline" moves during off-peak periods. WebTV uses a flexible logging API with support for strings, integers, doubles, a GMT timestamp, and arbitrary BLOBs. Logging uses a publisher/subscriber model. Data can be streamed to an arbitrary hierarchy of servers. Subscribers can filter just the events they want. GMT timestamps facility correlation of logged events.

Each service has a corresponding test harness used by QA for validate correctness of a service. These tests are tied together into a test system that is deployed in WebTV's production environment. This permits end-to-end functional testing of the running system including verifying provisioning for new users.

WebTV's design expects applications to fail. The design philosophy is that, "if applications must die, do it quickly; come back quickly; don't stay in a half dead state." WebTV employs a ServiceLauncher that watches for dying application processes. ServiceLauncher records the failure in the event log, opens an entry in WebTV's bug tracking system, saves the process core dump, and performs a rate limited restart to throttle any machine spinning on a bad restart. Furthermore, WebTV attempts to provide the best possible degraded service in the face of failure. For example, if a sub-service can't write to a database, it provides whatever functionality is available in read only mode. Operations exploited read-only operation to enable "online" backup of the user database. The WebTV home page renders whatever information it has so if, for example, news headlines are unavailable the server doesn't include any headline related HTML. Finally, most services are self-healing. For example, if the mail service detects that the table of contents (TOC) does not match the individual messages, it automatically rebuilds the TOC.

One weakness in the current release of WebTV is a centralized user database. When the WebTV service was created, it was recognized that there was not enough time to build a highly available distributed database. So, the WebTV service launched with a single machine that ran an Oracle 7 database. Most of WebTV's major outages have been caused by database failures. Protecting the database (with aggressive caching and carefully crafted queries) has been a high priority in the software development. In the last four years, the service has evolved so that the service can run in a degraded mode against a read-only database while an automated process replicates the customer database to backup machines. Related to WebTV's database is a proprietary billing system which was embedded in the OLTP database used for customer information. The next release of the WebTV service (their 8th major release) finally splits the billing database from the OLTP database.

4.13.3. Operations

WebTV servers are housed in their data center in Palo Alto and in collocation facilities owned by GNAC (Redwood City), and Compaq (PAIX). Additional machines will be hosted in the new SVC once it is constructed and possibly in the Exodus facility. WebTV also experimented with placing servers in the network fabric of one of its major ISPs to minimize backhauling traffic, though they concluded this was not cost-effective.

All administration and monitoring of WebTV hosts are done "remotely" (e.g. no touch). Once a machine is racked and wired, it is possible for the operations staff to perform a diskless boot, load a current operating system, load applications software, and add the new machine to the WebTV service. All WebTV needs to run a machine remotely is a pair of hands which can plug cables and replace failed components.

Applications are distributed using multicast. Each service release is in its own directory tree with no shared bits. It is possible to push new releases while the service is live. Switching between service releases requires changing a single symbolic link and restarting the services. The restart is not a machine reboot so machines can be rolled forward (or back) in a matter of a few seconds. Most OS reboots have been caused by hardware failures. All of the reboots traced to software were due to Oracle or the Veritas file system. WebTV has many machines with more than 300-day uptimes.

WebTV operations motto #1: "If you can't measure it, you can't see it, and you're dead!"

Without appropriate measurement tools, "you are stuck in voodoo administration." Be able to log every event, but be prepared to change what you store as your needs change. Data isn't enough though. Once you pass a hundred machines there is too much noise. Visualization tools are essential to really understand the health of the service.

WebTV operations motto #2: "Leverage Commands"

WebTV has a tool called NETEXEC, which executes a command on a set of machines that match a specification in the master configuration file. For example, it is possible to execute a set of commands on all machines running a proxy server, or all machines that are part of a particular service group. All operations have command lines, so it is easy to script a common procedure and then execute that procedure on appropriate machines.

WebTV operations motto #3: "Process Matters"

The larger an organization gets, the more important it is to have well documented (and tool assisted) processes and procedures. Communication and coordination is extremely important in a production environment. It is quite possible that human error (often communication related) is the first or second most likely cause of our customer visible service outages (the other is Oracle bugs).

WebTV operation motto #4: "We are one."

WebTV employs an integrated approach to management. With the exception of the underlying server OS, they control all aspects of their service. (They have no desire to control the underlying OS.) Both clients and servers are carefully configured, as are the network and the application.

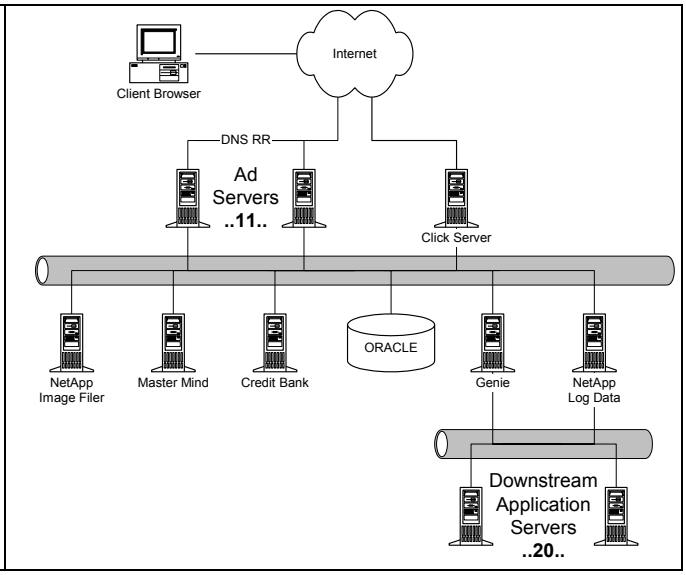
Integration has three key benefits. First, it offers multiple levels of defense for both security and fault tolerance. Second, it provides opportunities for global optimizations – problems can be solved were it is easiest. If one type of solution fails to yield the desired result, they can often fix the problem in a different layer of the system. Finally, integration reduces finger pointing and turf wars because the entire team works together. Problems can be resolved readily without waiting for a third party.

Examples of WebTV's integrated approach to development and operations:

- "We couldn't make an application protect itself, so we add protection (packet filtering) at the network layer. Eventually the host can protect itself, but we need the network protection in place, just in case a mistake is made on the host."
- "Headwaiter gives basic service routing. Our network team will mostly add Alteon switches into the network fabric to complement existing service routing."

4.14. LinkExchange

Configuration	
Front Ends:	12
Back Ends:	6
Other Boxes:	20
Load Balancing:	DNS
Interview Date:	Sept. 10, 1999



LinkExchange provides Internet services for small businesses. Microsoft acquired LinkExchange in November 1998. They are early in their planning to transition to Microsoft technologies.

LinkExchange's flagship service is BannerNetwork, an advertising network through which small web sites exchange advertising space. For every two ads participants display on their web site, they get credit to have their ad displayed once on another web site. LinkExchange sales the surplus ad space through their Ad Store. BannerNetwork currently serves between 45 and 50 million ad impression per day.

When a new site joins BannerNetwork, the owner submits a subscription request including self-categorization any number of BannerNetwork categories. BannerNetwork has over 2,000 ad categories, such as soccer sites or Spanish-speaking sites.

4.14.1. Architecture

BannerNetwork consists of the following components:

- **Oracle database server (1 machine).** Running Oracle 7.6 on a Sun E3500, the database contains all ad and customer information. The server has 2GB of RAM.
- **Front-end Ad Servers (11 machines).** The Ad Servers process all ad opportunities. Ad Servers run FreeBSD and Apache. While LinkExchange has not modified the FreeBSD sources, they have tuned FreeBSD to support larger shared memory regions and more processes. In the near future, the front-ends will be replaced with dual processor boxes.
- **Click Server (1 machine).** The click server processes user click-through requests on ads. One click server is more than adequate as typical ads have a click-through rate of less than 1%.
- **Mastermind (1 machine).** The Mastermind is responsible for all ad scheduling.
- **NetApp image server (1 machine).** Web servers retrieve ad images from the NetApp using NFS.
- **Genie server (1 machine).** The Genie server aggregates logs from the front-end web and click servers. It then dispatches the logs to downstream applications. The file-based logs are collected and dispatched via NFS. A fortuitous side effect of using file-based logs is that if any downstream machine goes down, its incoming log file just grows in its absence until it comes back online and consumes the log.
- **Credit Bank (1 machine).** Acting as a downstream application, the Credit Bank accumulates ad credits for BannerNetwork participants. Credits are then fed back to the Mastermind through the Oracle database. Credits are accumulated in a large memory-mapped database. The Credit database has transactional, but not relational, properties.
- **Downstream application servers (20 machines).** Including the Credit Bank, BannerNetwork has twenty-three downstream applications include tools for counting inventory (for sales), site profiling, and log archival. Some of the downstream applications are small enough that they share a single server. One of the downstream apps is large enough to have a dedicated Oracle database machine. The site profiler maintains a running profile for all account and user profiles for targeted marketing.

All of the BannerNetwork machines are located at the Frontier Global Center in Sunnyvale, a one-hour drive from LinkExchange's San Francisco offices. As such, remote consoles are an absolute necessity. In general, LinkExchange personnel only travel to Sunnyvale to replace hardware.

Once accepted into the BannerNetwork, the participating web master adds the following HTML to the site:

```
<!-- BEGIN LINKEXCHANGE CODE -->
<IFRAME SRC=http://leader.linkexchange.com/5/X1132997/showiframe?
  WIDTH=468 HEIGHT=60 MARGINWIDTH=0 MARGINHEIGHT=0 HSPACE=0 VSPACE=0
  FRAMEBORDER=0 SCROLLING=NO>
  <A HREF="http://leader.linkexchange.com/5/X1132997/clickle" TARGET="_top">
  <IMG WIDTH=468 HEIGHT=60 BORDER=0 ISMAP ALT=""
    SRC="http://leader.linkexchange.com/5/X1132997/showle?">
  </A>
</IFRAME>
<!-- END LINKEXCHANGE CODE -->
```

When a browser views the participating web page, the LinkExchange code issues an ad request to the Ad Servers. The Ad Server writes the request to the Mastermind, records information about the request to the Click Server, and logs an event. The Ad Server then returns the selected banner URL to the browser based on response from the Mastermind. Image URLs are actually a redirect to the image servers. This redirect defeats caching for ad counting, but enables caching of previously seen banner images. The architecture supports multiple Masterminds, but only a single Credit Bank as the Credit Bank owns synchronization of the authoritative ad-credit database.

Ad Servers cache account and ad information from the database. The receiving Ad Server asks the Mastermind which banner should be shown through Mastermind relay daemon running on the Ad Server.

In general, processes in the LinkExchange system communicate through three mechanisms:

- Relay daemons with local shared memory. To improve tolerance of network latencies and simplify timeout-related code, LinkExchange components seldom cross-communicate across machines. Instead, the client component communicates with a local server relay through a shared memory segment. The server relay forwards the request to the server machine. If communication with the server machine times out, the server relay provides the client with a reasonable default.
- Logs. The producing process writes events into the common log. Logs are aggregated by the Genie and dispatched to downstream applications. The downstream applications filter and consume log events. The front-end web servers generate between six and eight GB of log data every day.
- Shared database tables. The producing process writes into the database table and the consuming processes reads from the table. Shared database tables are used primary for latency-tolerant feedback, such as to adjust ad schedules based on credit for ads shown.

Multiple Apache processes on a single Ad Server communicate with one local Mastermind relay. The Mastermind relay can aggregate requests to the Mastermind machine as appropriate. The Mastermind caches account information from the account database and receives updated ad counts from the Credit Bank every 10 minutes. The Credit Bank filters log events to determine which participating web sites should receive ad credits.

Ad inventory management is a problem. BannerNetwork has approximately 1 million participating web sites (acting as 1 million ad sponsors) with 400K active sites in any given month. Ads must be shown based on credit information and site characteristics like “soccer

sites". A given site can have multiple overlapping characteristics such as "soccer" and "Spanish". In addition to participating web sites, ad space can also be purchased through the Ad Store. It took one year to develop a working prototype of the inventory management system.

Variables affecting ad inventory include site category, site rating, banner type, banner size, site exclusions, IP domain (like .edu or .uk) for either site or user, user's browser type, user's operating system, or stored user information like their geographic region from ListBot. 33K sites account for 80% of BannerNetwork's ad inventory.

4.14.2. Development

All of the LinkExchange front-end code is written in ModPerl and runs inside the Apache process. ModPerl is a sophisticated, in-process Perl5 interpreter. About one year ago, LinkExchange did a whiteboard survey of a number of rendering options ranging from Perl, Java, and C++ to Scheme, Python, ColdFusion, and ASP. Perl was chosen primarily for its large body of readily available libraries; the Comprehensive Perl Archive Network (CPAN) has over 200 modules. Perl also has a strong developer community. ModPerl was chosen for performance; it operates in process and does not start a new process on each web request.

LinkExchange management feels that the choice of scripting language is of lesser importance compared to the choice of database connectivity. Connectivity is a major performance problem. Rather than use solutions such as ADO, RDO, or even ODBC, LinkExchange uses a custom Oracle OCI data connector. While their OCI data connector has been great for performance, it has impeded their movement from Oracle 7 to Oracle 8. Called DA Server, LinkExchange's data connector has explicit notions of database caching and database modeling.

LinkExchange's developers are intimately involved with operations; they carry pagers.

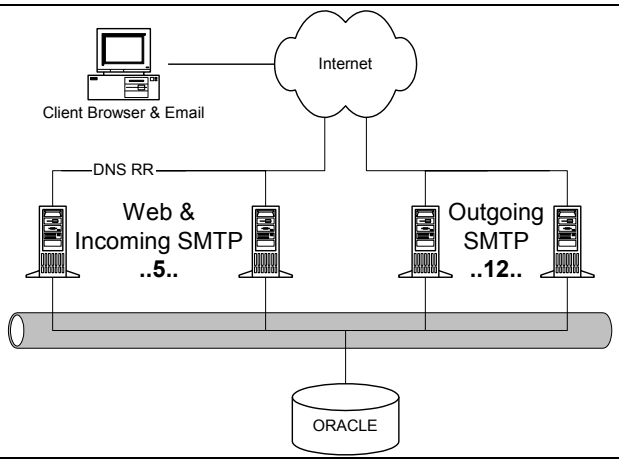
LinkExchange uses a 13-week planning cycle feeding into a 6-month development cycle. In general, no single program takes more than 13 weeks to develop and deploy. Infrastructure development is separated from product planning. Development resources are divided with 40% to infrastructure, 40% to new features, and 20% to reactionary needs.

They are 6 months into the rollout of their next infrastructure with an 18-month planned lifecycle. The goal of the new infrastructure is to scale to 1 billion page impressions per day. The target was set in January and the first pieces of the architecture were rolled into day-to-day production in April.

In the past, LinkExchange didn't have separate staging servers and deployed servers. They once lost their entire database due to a programming error. Through heroic efforts, they recovered the database in one day, but they have since opted to stage deployment.

4.15. LinkExchange ListBot

Configuration	
Front Ends:	17
Back Ends:	1
Other Boxes:	
Load Balancing:	DNS
Interview Date:	Sept. 10, 1999



ListBot is LinkExchange's free service for email distribution lists. Just six months ago, a prior version of ListBot, written by a single developer, ran on a single machine. As demand on the service increased, the system has been re-architected to run across a cluster of at least 18 machines. ListBot has 16.7M subscribers, supports 560K distribution lists, and processes approximately 5.7M email messages per day. Thirty percent of ListBot's email recipients are international.

4.15.1. Architecture

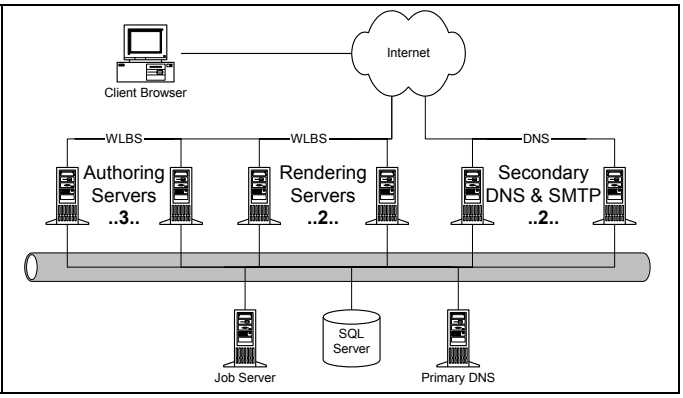
ListBot consists of four components:

- **Oracle backend database (1 machine).** The database runs on a SPARC Ultra E450. The size of the machine was chosen to pick a "sweet spot" in Oracle's pricing. The server has 150GB of online storage of which 90GB is currently used for live data. Most of the live data is devoted to an online message archive for the distribution lists. Other components of the architecture connect to the database through LinkExchange's DA Server OCI connection provider.
- **Front-end servers (5 machines).** The front-end servers act as web servers and incoming SMTP servers. ListBot front-end servers are dual processor Pentium IIIs running Solaris. Users access the web servers to manage their current distribution list subscriptions. Users can also access the distribution list archive through the web servers. The current web servers run out-of-the-box Apache with pure C++ CGI extensions. Load is balanced across the web servers through DNS round robin. The incoming SMTP daemons receive all email messages bound for distribution lists. The SMTP daemons filter email messages to remove spam. Only authenticated members of a distribution list are permitted to submit messages. After filtering, the email message is copied to the Oracle archive and placed on an outgoing Oracle queue
- **Outgoing SMTP servers (12 machines).** The outgoing SMTP servers poll for messages on the Oracle queue. A child process forks with the email message, attaches a list of recipients, then launches a copy of QMAIL to distribute the message. QMAIL forks off one child process per recipient. The turnaround time,

from receipt at the incoming SMTP server to full distribution is normally less than one minute. Outgoing SMTP servers are dual-processor Pentium IIIs with eight 10K RPM disks and 512MB of RAM each.

4.16. Hydrogen

Configuration	
Front Ends:	7
Back Ends:	3
Other Boxes:	1
Load Balancing:	DNS/WLBS
Interview Date:	Aug. 17, 1999



Hydrogen is the code name for Microsoft's entry into shared-hosting web site solutions for small business. Hydrogen also provides the capability to commerce-enable these sites to allow small businesses to sell products over the web.

In its first release, Hydrogen plans to appeal to small business end-users, by:

- Allowing them to build web sites that are useful, attractive, and tailored to their business.
- Enabling them to easily create and manage sites by themselves.
- Making it easy for them to upgrade an existing web site to a commerce-enabled web site.

Users create web pages with a template-based web wizard. After creation, pages can be edited through the wizard or through Microsoft Front Page. Users can combine templates with Hydrogen-provided services such shopping carts.

Templates separate the look and feel (in HTML) of a page from its data (in XML). Updates to a template are automatically visible in both pages created with the template and the web wizard. Templates are authored in Front Page and can be edited by the 6000+ Certified Front Page Professionals.

Hydrogen enters beta testing for version 1 in October 1999, followed by a full roll out in November 1999. Service goal is to support 5000 sites by late December. Hydrogen can support approximate 500 sites per server. Yahoo, Hydrogen's primary competitor hosts 7000 users after 1 year in operation.

4.16.1. Architecture

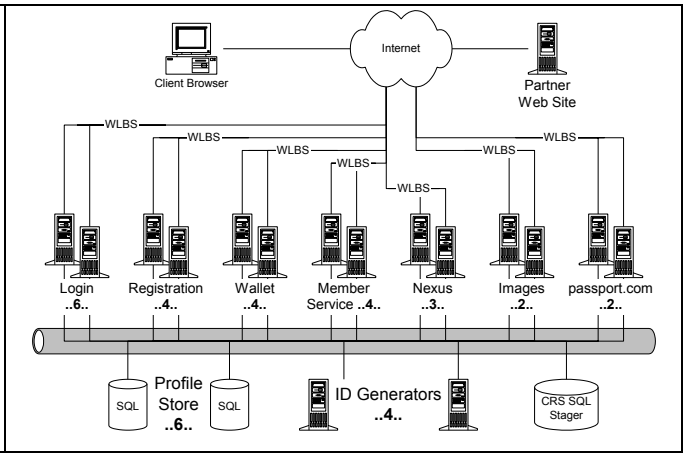
Hydrogen's major components are:

- **Authoring web servers (3 machines).** Small business clients connect to the authoring servers to generate their web pages. Business web pages are stored on the backend SQL database.

- **Rendering web servers (2 machines).** These servers render web pages when end users (clients of the small businesses) connect to the small business web sites.
- **DNS servers (3 machines).** One primary and two secondary DNS servers host the top-level DNS names of the small businesses.
- **SQL Server (1 machine).** Holds XML for all small business web sites and Hydrogen support data.

4.17. Passport/Wallet

Configuration	
Front Ends:	25
Back Ends:	10
Other Boxes:	1
Load Balancing:	WLBS
Interview Date:	Aug. 11, 1999



Passport will provide a unified site logon for all MSN sites. In addition, Passport will provide a universal login across potentially thousands of partner sites. Passport hopes to lower the barrier for e-commerce by eliminating forms and increasing users' sense of trust and security. As a secondary feature, Passport will share profile information with partners.

Passport consists of two major, end-user visible components: Passport, which gives the user one login ID associated with an email account and granted by a Domain Authority, and Wallet, which provides secure access to credit card information enabling one-click buying to become pervasive.

Passport has two sets of customers: end users, who register with passport to receive an ID, and partners, sites like MoneyCentral, barnesandnoble.com, etc., who want to use passports single user ID and wallet information to facilitate e-commerce. Partners will install the Passport manager to broker Passport logins, manage cookies, and transfer wallet data.

Passport implements "Kerberos" style authentication with cookies, all created on the Passport login server. Passport supports three types of cookies: ticket-granting cookies (SSL only), Passport domain cookies (encrypted using Passport key), and partner site cookies (encrypted using site key found in Nexus data).

The ticket-granting cookie (ID + timestamps) is created at login time and marked as SSL only. The process of acquiring a ticket-granting cookie involves authentication. The goal is to have this occur only one time per session. Typically, a Passport domain login cookie is also created at the same time.

If a partner does not have a cookie for a user, the user is redirected to a Domain Authority. Since all Domain Authorities are children of the Passport domain, the (parent's) cookies

are sent as part of the (redirected) request as well. If the info in the Passport cookie is still valid, it is re-encrypted using the site's key and sent back to the user with a redirect back to the partner site. The site then uses this info to write a local cookie for the user (to shortcut this the next time). The implication here is that every time a new site is hit the some (Domain Authority) login server will be accessed. If the passport ticket is still valid, a simple .ASP or CGI script can be used to transform the data. This will therefore not involve either user intervention or membership identification lookup – this can be done on a login server (frond end box) of any Domain Authority.

In the advent of ticket expiration, access to the "correct" Domain Authority is needed. The initial steps are similar to above with the following exception: when the Domain Authority recognizes that the ticket is invalid, it inspects domain name of the user. It then additionally responds to the user another re-direct to the "right" Domain Authority over HTTPS (SSL) (the domain is derived from membership ID in passport cookie). This time, the ticket-granting ticket is passed as well and is updated (this may involve a membership database lookup to validate the password).

4.17.1. Architecture

The MSN Passport cluster consists of 25 front-end IIS servers and 11 back-end SQL servers:

- **Front-end Nexus servers (3 machines).** Central servers that maintain domain map including Domain Authority partners, encryption keys for all partners, and miscellaneous info like URL for co-branding logins. Nexus info is pushed/pulled to all Passport domain components as well as all Passport partners at regular intervals. The Nexus is stateless; all information is gathered dynamically when a new Nexus comes up.
- **Front-end passport domain servers (e.g. www.passport.com) (2 machines).** A namespace containing one or more Domain Authorities. Domain Authorities issue user IDs (and supply email). They are responsible for Passport registration, login (revalidation), and update according to spec. Domain Authorities are free to authenticate users however they like (HotMail will use their current scheme, MSN is growing their own, MSNIA will use Concorde). Domain servers are stateless.
- **Front-end wallet servers (4 machines).** Contain user credit card(s) information. Credit card information is sent via SSL as a post from centralized servers (MSN) to the partner. Uses ECML (Electronic Commerce Model Language) to describe data (nothing more than well-defined named parameters for HTTP post). The Passport wallet feature is intended to simplify "buy-now" and "one-click buying" by eliminating (repeatedly) filling out forms. This feature really has nothing to do with passport authentication. Wallet servers are stateless.
- **Front-end login servers (6 machines).**
- **Front-end registration servers (4 machines).**
- **Front-end update/member services servers (4 machines).**
- **Front-end static image servers (2 machines).**

- **Back-end SQL profile stores (6 machines).**
- **Back-end ID generators (4 machines).**
- **Back-end CRS/SQL stager (1 machine).**

4.17.2. Deployment Plans

The initial deployment of Passport will occur at HotMail. HotMail front-end boxes will continue to run FreeBSD. Login will be done using a CGI script.

HotMail now has three login server clusters: the old cluster, an SSL cluster, and a non-SSL cluster. The old will now act as a partner site (hotmail.com). The non-SSL version of the Passport login server (*.passport.com) exists because HotMail is worried about SSL performance even though they use external boxes to generate SSL keys.

Abstractly, the HotMail MSERVs and USTOREs will act as the "SQL backend". MSERVs are the indexes. USTOREs provide the data, including password for authentication. All data to user is served via the front-end engine. Front ends will continue to communicate with USTORES via XFS.

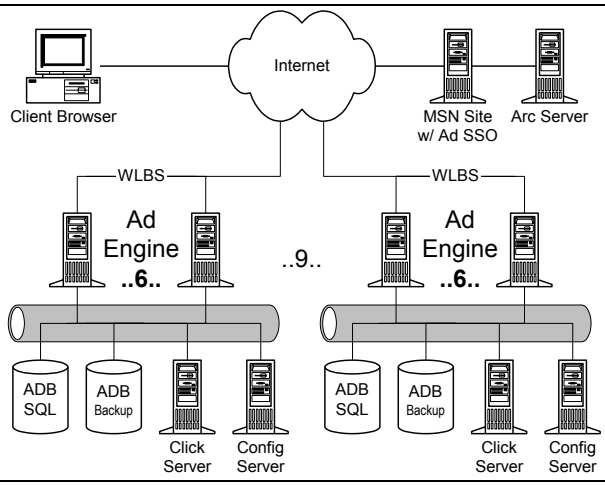
In the current plan, HotMail will be the only Domain Authority at Passport launch. Passport.com will be hosted at MSN for third parties (and possibly MSN Passport accounts). Third party mail will be hosted at HotMail. Passport will ask merchants to support Wallet initiative for Christmas this year, but not participate in Passport authentication.

In the next release, Passport will define more data to put into user profile by augmenting the static schema with a flexible schema. It will allow users to define what is discretionary and what is not, although choosing an appropriate user interface for this is very difficult. They hope that the flexible schema will be an incentive for merchant adoption.

There are a number of interesting and unresolved issues relating to international Passport Domain Authorities. They may want to have a Domain Authority in a country X so that access for users and merchants in country X is optimized. However, users in country X may roam the entire planet; for these cases, access could be much worse if, for example, users from country X spend most of their money at US web sites.

4.18. AdsTech

Configuration	
Front Ends:	54
Back Ends:	36
Other Boxes:	
Load Balancing:	WLBS
Interview Date:	August 4, 1999



MSN Advertising Technology Group (AdsTech) provides advertising technology to all MSN sites. AdsTech serves close to 150M ad impressions per day. Like Passport, AdsTech provides an infrastructure service rather than an end-user service. The "customers" of AdsTech are FE boxes at the various properties that need to insert an ad onto a page before shipping it to the final end user.

AdsTech runs nine advertising clusters: seven for sites in the US and two for international sites. Large services, such as MSNBC, use an entire cluster. AdsTech recently began to host HotMail's advertisements.

4.18.1. Architecture

The AdsTech cluster architecture consists of the following:

- **Ad Client SSO.** Supplied by AdsTech and hosted on FE machines of properties utilizing the AdsTech service.
- **Ad Engines (AEs) (6 machines).** Ad Engines receive the GetAd requests from the AdClient SSO, select an ad, and return HTML. The Ad Engine will be described in more detail to follow.
- **Ad Database (ADB) (2 machine).** The Ad Database contains the list of all ads to be served, their schedules, and their impression counts. The ADB is also the integration point with line-of-business (LOB) tools.
- **Image Servers (ISs).** IIS servers that serve up static GIF, JPG, HTML, and dynamic ASP. The HTML generated by the Ad engine includes URLS to these machines. The current Quad 500MHz Xeons can server 150 images per sec (average ad size is 10KB).
- **Click Server (CS) (1 machine).** Needed to support down-level clients that do not support floating frames (IFRAME)

Flow of control through the servers is as follows:

1) Client browser requests content from MSN content server. Content server makes ad request via client SSO. Ad Server retrieves correct ad from schedule. Ad Server sends context of request to Click Server. Content is returned.
2) As content renders it encounters HTML tags (SRC=) asynchronously directing request to Core Ad Server, which returns path to correct ad.
3) Browser retrieves ad from Central Ad server.
4) User with current browser clicks on ad banner. Ad Engine retrieves the redirect URL and counts the click through. User with downscale browser clicks on ad banner, Click Server binds context then retrieves the redirect URL and counts click through.

4.18.2. Software Description

On quad-processor Xeon 500MHz machines, the AE runs 16 threads, and fulfills 800 ad requests per second. Performance is affected by three main factors: number of schedules, number of targets per schedule, and hardware (faster hardware really helps). The MSNBC Engine (4x500 Xeon) has 2000 schedules, and averages 1.7 targets per schedule. For MSNBC, AdsTech processes 300 ads/second at 20% CPU utilization. AdsTech goal is to increase to 2000 ads/second.

On every ad opportunity, the FE invokes the SSO that, in the general case, makes one round-trip "GetAd" call to the ad server. The GetAd call includes a set of properties. Two properties are mandatory: 1) the ad size, an abstract value that includes not only the size of the ad, but also the placement of the ad (such as on the top banner, etc.). The GetAd call also includes a MS GUID. The FE SSO has fail-over strategies to do things like display a standard ad when it can't reach an ad server. The return value from the GetAd call is stream of HTML.

GetAd uses a custom ASCII protocol. To improve performance, high volume page groups are cached entirely within the SSO. High volume pages account for about 1% of the site pages, but drastically reduce the number of GetAd calls.

Hotmail (and other remote sites such as WebMD and FairMarket) access the GetAd call via HTML that calls a cluster of FE servers that are run by AdsTech that have AdClient running as an ISAPI extension (these servers are called Arc servers). The HTML makes use of the IE <IFRAME> tag to serve the ad. Here is an example of the HTML:

```
<IFRAME SCROLLING=NO HEIGHT=60 WIDTH=468 FRAMEBORDER=0
 src="http://arc5.msn.com/ADSAdClient31.dll?GetAD?PG=HOTROS?SC=LG">
  <A HREF="http://ads.msn.com/Clicker/ADSClicker31.dll?Redirect?PG=HOTROS">
   <IMG SRC="http://arc5.msn.com/ADSAdClient31.dll?GetImage?PG=HOTROS?SC=LG>
  </A>
</IFRAME>
```

AdClient can also serve from a local “valve” cache (like High Volume page groups) based on an ad per second threshold. The current HotMail Arc cluster (15 servers) allows 800 ads per second to pass through to the AE and the rest are served from the valve cache.

With newer browsers, ads are embedded in an HTML IFRAME. When a user clicks through an ad, an ad ID embedded in the referenced IFRAME HTML tells the AE which

ad the user was shown. The AE then retrieves the destination URL (all click throughs return to MSN AEs for accounting). Due to architecture constraints, if the user's browser does not support IFRAMEs, the ad HTML cannot contain the ad ID. Instead, the ad ID is stored on the CS by the IS redirector, then retrieved from the CS at click through.

Ads are displayed based on an ad schedule. The schedule includes the ad's content (called its creative), the impression goal (number of views), and the period to which the schedule applies. MSNBC runs approximately 2000 ad schedules at a time.

Every six hours, the latest ad contract data are converted to ad schedules and placed in the ADB. Ad contracts come from network promotions (about 10% of all ads on MSN), line-of-business (sales), and local site-originated ads. Every 5 minutes the AEs flush their private impression counts to the ADB and then retrieve the latest schedules (including the global impression counts).

Within the AE, ads are served from a list sorted by priority of display. The list only includes those ads that are below quota based on their ad schedule. When a new ad request arrives, the AE linearly searches through the list for the first ad that matches the GetAd's properties. Each ad schedule is augmented with a frequency control counter (valued from 0 to 15). An ad is only displayed if its frequency control counter is zero. If the ad matches the properties, but has a non-zero count, the count is decremented and the search continues down the list. If the search reaches the end of the list, it restarts at the beginning. The AE guarantees that there is always at least one ad in the list. In the worst case, the AE must loop through the list 15 times (to decrement the single matching ad's counter from 15 to 0. The AE's active schedule list is updated every 30 seconds. The update algorithm creates a new list then swaps the new list with the old list to minimize synchronization costs.

4.19. MSN Operations

Configuration N/A **Interview Date:** June 22, 1999	**N/A**

The MSN Operations Team is charged with day-to-day operations of most of the MSN properties. Functionally, they operate between MSN sites and onsite ITG Canyon Park operations staff.

The typical MSN property uses WLBS for balancing across disjoint clusters (HMC has 42 web servers, 7 groups of 6). The typical MSN property consists of a front-end cluster, an ad cluster in AdsTech, an image cluster for static content, redirection (through WLBS), and a layer three switch (Alteon or Cisco 6500).

Features MSN Operations would like to see in the software platform and services:

- I/O Filters.
- Dump tracing tools.
- Serviceability: applications should work in "read-only" mode if they system can't write to store, etc.

- Configuration management and propagation (how to change registry key on all 42 machines).
- Mechanisms to create an aggregate view of the system for marketing, application owners, etc. MSN servers generate on the order of 3-4GB of IIS logs per day. The backend LAN is 100Mbps. They have to schedule data-movement jobs carefully to avoid using all of the bandwidth.
- Job engines, batching, etc.
- Tighter integration with line-of-business systems, for example to transfer output from e-commerce to SAP, etc.

5. Conclusions

If the megaservices we have visited are any indication, Microsoft's markets are changing from product-oriented to service-oriented. In shifting to service markets, we must change our mentality to focus firstly on maintainability, scalability, and availability. Service is about more than features; it is about providing features when the user wants and expects them.

Part of providing a service is having a firm understanding of customer needs and usage patterns. Almost without exception, both the developers and operators at every site we visited wanted a better understanding of how their customers use their system. They want to know which features customers are using and for how long.

Customer knowledge is critical to operating a responsive service. One of the hardest lessons our developers have learned is that in the service business, data is more valuable than user interface or algorithms.

Most importantly, a service must stay up. It must give the user service no matter how serious the internal failure. Services must assume that failures will occur and must plan for them. Because operators are the first line of defense to maintain the service, they are the developer's most important customers. Developers must "delight" operators!

Operators are not just customers; they must also be the developers' closest partners. Operations personnel must be involved in the development and deployment planning of the service. As the developers create the service's code, the operators must create the processes that allow the service to function from day to day. The developer's most important objective should be to make the operator's life simple.

The service architecture should be understood by the operators at a very deep level. Specifically at HotMail and WebTV, the operators have repeatedly saved the service by exploiting knowledge about the system architecture, sometimes by exploiting features in ways unintended by the developers. The operator as an adversary won't do that, but the operator as a friend and partner will.

In Section 3, we presented our key observations from visits to eighteen of Microsoft's Internet sites. We iterate them here by the three crucial abilities of any Internet service: maintainability, scalability, and availability.

Maintainability:

- Operations teams should be integrated into product development.
- Simple, understandable solutions are best.
    - Configurable off-the-shelf solutions are preferred to custom code.
    - Low-tech rules.
    - Less is more: Users and operators choose service over features.
- A service is never finished.
    - Side-by-side component versioning for rollout is crucial.
    - Process isolation and restart increase reliability.

Scalability:

- The network is an integral part of the system.
    - Understand your connectivity.
    - Partition data careful.
    - Load balancing is a core component.
- Cost and performance matter.

Availability:

- Systems should be designed with component failure as a rule not an exception.
    - The system should work partially even when components fail.
    - A component should never fail due to an external component failure.
    - Components should fast-fail on inconsistent state.
- Monitoring is absolutely essential.
    - Test suites should be delivered to operations as part of the platform.