Zimbra Collaboration Suite
Architectural Overview

Zimbra, Inc.[1]
www.zimbra.com

Table of Contents

1. Introduction

The purpose of this whitepaper is to provide an architectural overview of the Zimbra™ Collaboration Suite (ZCS) as well as share some of the messaging and web systems expertise we brought to bear in building Zimbra. Since Zimbra is open source, we also intend this whitepaper to be a starter kit for prospective Zimbra community members seeking to delve into the code and detailed documentation.

While developers and systems administrators are our primary target audience, architecture is not, however, just for information technology (IT) gurus: the underlying architecture of a piece of software is the single most significant factor in determining

- How secure it is;
- How reliable it is;
- How fast it is (i.e., what is the response time per user);
- How scalable it is (i.e., how many simultaneous users for a specific hardware configuration);
- How effectively and quickly it can evolve/support new features;
- How easily it can be customized, extended, or integrated with third-party technologies; and last but not least,

[1] Zimbra is a trademark of Zimbra, Inc. All other trademarks belong to their respective owners.

- How easy it is to administer.

So when you choose to place a bet on a new software system, that bet is inherently a vote of confidence in its underlying architecture. Yet one more reason to love open source is that architectural beauty (or the lack thereof) is subject to easy scrutiny.

One particular caveat is in order: herein we occasionally describe functionality that is not yet shipping in order to shed light on how Zimbra works and what the Zimbra Community can expect to see in the future. Any such discussions of Zimbra "futures" will be explicitly marked.

2. Why New Messaging & Collaboration Software?

There are clearly mature business-oriented messaging & collaboration servers available in the market—consider Microsoft Exchange and Lotus Domino. Other vendors (*e.g,* Openwave, Sun iPlanet) have successfully delivered consumer-oriented messaging solutions, generally via service providers—telecoms, broadband providers, Internet Service Providers (ISPs). Last but not least, the open source community already has widely-deployed software in Cyrus IMAP, SendMail, and so on. So why, then, has Zimbra built something new? What are the technical motivations for constructing yet another messaging & collaboration solution?

Impact of the World-wide Web. Email and instant messaging (IM) use has exploded as messaging has emerged as one the two "killer applications" of the Internet (the other, of course, being browsing). Early email users may remember the quaint old days when ten or twenty emails constituted a "busy day". As a result, information technology (IT) workers (*a.k.a.* power users) may spend the majority of their on-line time (3+ hours per day) interfacing with their messaging software. But while messaging usage has changed, the majority of the messaging solutions on the market today were built prior to advent of the World-wide Web. If you were instead to "start from scratch" in 2003, as we did at Zimbra, you might reconsider certain of these fundamental architectural decisions:

1. **Search** – Search engines have redefined the way we manage large datasets: *a prior* organization via folders is far less efficient than automated preindexing and very fast (nearly instantaneous) search across messages, contacts, documents, meetings, and so on. With richer search criteria (such as the domain of the sender, approximate date, kind of attachment, regards customer problems, etc.), saved searches (*a.k.a.* "virtual folders") provide yet another powerful mechanism toward a more "auto-organizing" mailbox better suited to today's volumes;
2. **Native security** – For any mailbox that has been in existence for more than, say, six months, protection from spam, viruses, phishing attacks, and so on is now essential. End-users rightly expect such security processing—anti-spam (AS) and anti-virus (AV)—to be well-integrated so that (a) it can be easily trained, (b) false positives can be easily resolved, and (c) protection can be extended to other collaborative communications;
3. **Web privacy** – Why install Virtual Private Networks (VPNs) on home computers when the web security model (based on SSL/TLS encryption) has been proven out over years of successful E-Commerce? Web security combines strong privacy (never broken outside of laboratory settings?) with single sign-on to business portals, backing applications and systems. (Of course, nothing in Zimbra precludes the

additional protection afforded by a VPN, particularly for non-TLS applications.) Moreover, with Ajax clients like Zimbra no user data is cached on the client, so secure remote access can be afforded even from public kiosks; and
4. **Server-centric deployment** – Messaging and collaboration has become a mission-critical application, and as such, the “store of record” must be a secure, managed server rather than the more-vulnerable PC client. Server-centric (rather than client-centric) architecture also affords easier access across a range of PC and mobile devices.

Richer end-user functionality. At the same time as the Internet has pumped up message volumes, near universal connectivity and software innovation have expanded the scope of what users expect from messaging & collaboration servers:
5. **Shared/group calendaring** – Universal connectivity has extended collaborative scheduling between organizations as well as among friends and family. With www.iCalShare.com, there are now thousands of public calendars that keep you abreast of your favorite bands, sports teams, gaming, programming, and so on;
6. **Address book/contact management** – Software has superseded physical address books and business card files. Moreover users want easy sharing as well as integration with their Customer Relationship Management (CRM) applications;
7. **Mobility** – As the smart phone has emerged as an Internet device, end-user’s justifiably expect their email, address book, calendar, etc. to be automatically synchronized “over the air” from the server, generally without having to install special-purpose software on their device; and
8. **Mix and match client software** – Businesses and even individual users want the freedom to use a range of clients (Windows, Mac, Linux), and existing personal productivity applications (Microsoft Outlook, Apple Desktops (& iPods), Mozilla Thunderbird, Novell Evolution) to which they have become accustomed. The messaging and collaboration server, then, ties together these disparate clients;

Impact of Web 2.0. While there is no doubt a significant amount of hype associated with the buzzword *du jour*—Web 2.0, the term nevertheless encompasses a significant set of new technologies. After all, Web 1.0 started off on a similar wave of confusing, consumer-focused hype. Today, however, there is a consensus that the broad range of consumer and business applications of Web 1.0 all rely on a common technology stack founded upon HTML, HTTP, and SSL/TLS. Web 1.0 had a huge impact on both consumers and on businesses, and the range of applications that were shoehorned into the basic model far exceeded expectations. We can expect the same thing for the Web 2.0 technologies:
9. **Rich browser UI via Ajax** – Ajax is really just a paradigm for leveraging Web 1.0 technologies like JavaScript, Cascading Style Sheets (CSS), Dynamic HTML (DHTML), XML, and XMLHttpRequest object (XHR) programming to build more richly interactive user interfaces that run natively in the browser. Web 1.0 didn't turn out to be much of a dislocator for email user interfaces even though email was one of the "killer" applications of the Internet. The issue, of course, is that HTML is not itself rich enough to provide a great email experience. Ajax is, and moreover allows for some blending of the browsing ("pull") experience and the messaging ("push") experience to reduce all of that cutting and pasting from email to browser and back.
10. **Inter-application communications via XML, SOAP, REST, etc. -** Of course, Ajax leverages XML for the client browser application to server application

communications, but this is just a special case of XML for Service-Oriented Architecture (SOA) in general. SOA has been around for decades, but back then we talked about Enterprise JavaBeans, stored procedures, CORBA, DCOM, and so on. The difference is that with the success of the Internet, XML, SOAP, et al., as well as Software as a Service (SaaS), we have finally achieved critical mass. Moreover, open bindings (via XML, SOAP, REST, etc.) allow other business applications to directly access the messaging and collaboration server (such as to check presence or send a work request) so that it need no longer be a black box that does not play well with others.

11. **Collaboration-oriented Internet protocols** – The Web itself is growing up into a collaboration platform: Really Simple Syndication (RSS) for publication; Wiki for collaborative authoring and versioning; iCalendar for sharing meetings and events on intranets and the Internet; instant messaging (XMPP) and presence; Voice over IP (VoIP) and Session Initiation Protocol (SIP); and so on. The future winning collaboration servers will be the ones that embrace rather than compete with this standardization;
12. **Collaborative authoring** – Web 2.0 is most associated with collaboratively generated content, and it stands to reason that collaboration servers should embrace the success of Wikis (Wikipedia), tagging (Flickr, Del.icio.us), the mash-up of other intranet/Internet applications & systems, and now Ajax Linking and Embedding (ALE). (ALE in particular may prove to be a tipping point in terms of WYSIWYG authoring within the browser, which will allow users to far more easily create rich content.)

Archiving and compliance. The email archiving market is growing explosively with the proliferation of retention and compliance policies (often motivated by the increased regulatory overhead of Sarbanes Oxley (SOX), Health Insurance Portability and Accountability Act of 1996 (HIPAA), and so on). Current archiving and compliance systems can be expensive, often doubling the necessary storage, server, administrative, and security overhead. Archiving and compliance capabilities will instead be built into future messaging and collaboration solutions:

13. **Unified archiving** – If organizations must retain email anyway, why not utilize the same underlying software and storage as for the on-line systems? Write-once, read-many (WORM) storage can guarantee the integrity of the archive without requiring the replication of each and every message on disk (only the meta-data need be stored on read-write storage—messages themselves are immutable).
14. **Unified cross-mailbox search & policy** – Since we have already made rich syntactic search available to individual users (so that they can lay their fingers on the right email as quickly as possible), why not enable the same mechanisms for administrators to be able to choose appropriate compliance policies as well as conduct efficient, comprehensive cross-mailbox searches, such as for litigation/discovery.

Simplifying administration.

15. **Server proliferation.** As administrators have worked to address the proceeding fourteen factors with their existing messaging and collaboration solutions, they have generally had to resort to "bolt on" technologies for security (AS/AV), mobility, search, archiving, cross-mailbox search/discovery, and so on—each such addition requiring its own hardware footprint, as well as administration and security overhead; and

16. **Storage management** – Email administrators are under increasing pressure to cheaply accommodate ever larger mailbox quotas. (Gmail is now approaching free, 3Gb quotas for consumer mailboxes!) The answer lies in leveraging commodity storage (*e.g.*, SATA drives) and archiving solutions (*e.g.*, hierarchical storage management/HSM) without sacrificing user experience or reliability. Also key is single-copy message storage—avoiding redundantly storing copies of the same message as well as any attachments as it is delivered to multiple users. Finally, the messaging and collaboration solution must itself automate routine administration like garbage collection/defragmentation and back-ups, and should allow restoration of individual mailboxes or even messages/files into a live system (rather than requiring an entire "storage group" of users to have to be restored in bulk to extra a single item).

Impact of Open source and Software as a Service (SaaS). While open source and SaaS (or on-demand) are primarily viewed as software "business" dislocators, each also has underlying impacts on system architecture:

17. **Open source infrastructure** – Open source messaging and collaboration solutions are expected to leverage mature open source systems software building blocks to provide better caching and reduced write overhead (I/O being the largest barrier to scalability/performance in messaging systems): Linux file system (message store), Apache (server container), MySQL (meta-data), Apache Lucene (search), Postfix (Mail Transfer Agent), OpenLDAP (config. data), and so on;
18. **Open source quality** – To ensure software quality, there is nothing quite like the added developer accountability that results from the public scrutiny of (and commentary on) his/her code base. Products that pass muster are likely to have more elegant, more secure internal architectures that will make them a better long-term investment (more on this below);
19. **Open source community** – Successful open source technologies draw broad, diverse communities that make substantial contributions in innovation, quality assurance, customization, integration, localization, and technology transfer. Modern software architectures that has been designed with such community contributions in mind will thrive under the open source model; and finally,
20. **SaaS deployments** – Messaging and collaboration, particularly for consumers and smaller organizations, is easily packaged as a service that can be subscribe to across a network, thereby leaving the specialized job of server and storage administration to experts. SaaS, however, places special requirements on the server software: multi-tenancy (sharing across multiple domains), delegated administration (so that end-users can tune behavior without interfering with others), intra-server security (to protect against leaks across end-users), automated provisioning, and so on.

Now that we have shared twenty reasons why the Zimbra Collaboration Suite was built, let's dig into describing how Zimbra works. We will look at the client architecture first, and then the server.

3. Zimbra Client Architecture

Zimbra Ajax client. Our philosophy at Zimbra is to support existing email/messaging client applications, such as Microsoft Outlook, Apple Mail, Novell Evolution, Mozilla Thunderbird, and so on. At the same time, there are advanced features of the Zimbra Server (*e.g.*, mash-ups, search, conversations, tagging, Ajax Linking & Embedding/ALE, *etc.*) that cannot be surfaced through existing user interfaces. Users interested in leveraging these advanced features will need to use the Zimbra Ajax client (at least until existing clients are similarly enhanced).

Ajax stands for Asynchronous JavaScript and XML. While the name Ajax has only recently been coined, the underlying technologies—JavaScript (*a.k.a.* ECMAScript), Cascading Style Sheets (CSS), XML, SOAP, and so on—have been maturing for years.

Ajax has become a "next big thing", because Ajax applications uniquely combine:

- Richly interactive user interface;
- Rich client/server communications;
- The zero-cost administration of a web client;
- The look and feel of the web (minus the back button and bookmarks);
- Seamless integration with other web content (e.g., "mash ups"); and
- The investment protection of a web standard.

As a demonstration of Ajax being web standard, the Zimbra Ajax client runs and behaves uniformly across Mozilla Firefox, Microsoft Internet Explorer (IE), and Apple Safari.

The Zimbra Ajax client is in actually an extensive (more than 100K lines of code) object-oriented user interface programmed in JavaScript. The client follows a Model-View-Controller (MVC) architecture, with state changes both rendered in the UI and sent to/retrieved from the backing server. The basic Zimbra Ajax browser client has no persistence—all persistent state is being maintained on the server-side. Cascading style sheets are used as much as possible to govern the painting of the UI. Optimization techniques essential to the user experience/lower latency include aggregating multiple JavaScript and CSS files, compressing the result, and browser caching (of code, not user data). One third of the ZCS client code is made up of the underlying Kabuki Ajax widget toolkit, authored by Zimbra and open-source licensed separately from the full ZCS client/server solution.(For more on Ajax, please see the Zimbra/Kabuki Ajax Toolkit whitepaper.)

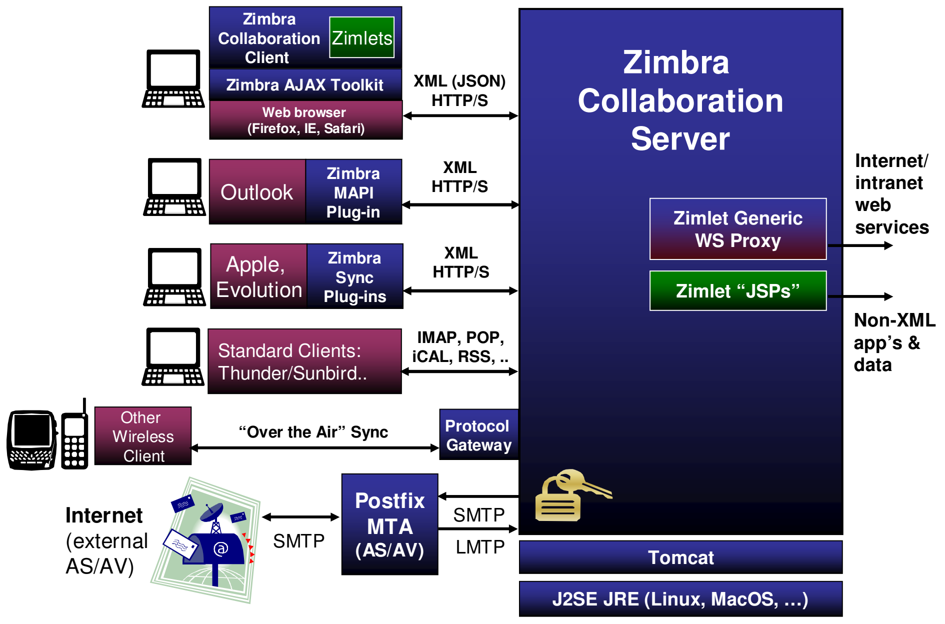

Figure 1: Front-End Architecture

Client/server communications. The XML/SOAP bindings used for Zimbra client/server communications are well-documented endpoints so that programmers are free to access any Zimbra messaging and collaboration services—such as sending an email, scheduling a meeting, or submitting a work request—from any web service-enabled application. At the same time, all that XML crunching can be non-optimal for Ajax client performance, and so we also provide a "fast path" alternative in which the Zimbra server optionally ships a optimized representation—JavaScript Object Notation (JSON)—better suited to efficient processing within the Ajax client. This more closely-coupled alternative transport is automatically selected as appropriate (version compatibility is assured since the Ajax client is downloaded from the server, and there is always the XML default interface behind it).

Existing collaboration clients. As illustrated in Figure 1, the Zimbra Collaboration Suite supports a broad range of existing messaging and calendaring clients—such as Outlook, the Apple desktop; Mozilla Thunderbird, Novell Evolution, Eudora, and so on—via Zimbra-provided client plug-ins and industry standard protocols:

- **Outlook (MAPI)** - Zimbra includes a MAPI provider plug-in (packaged as a universal Windows Installer or MSI file) that directly links with Outlook, and hence must be installed upon the Windows client. This is the Microsoft-sanctioned and supported approach for synchronizing Outlook features like email as well as calendaring and contacts (not supported by IMAP) with messaging servers like ZCS. The Zimbra MAPI provider speaks the identical secure web services protocol as the Zimbra Ajax client (minus the JSON optimization). The Zimbra MAPI provider supports cached mode for efficient disconnected operation and resynchronization, and since it natively supports SSL/TLS, does not require a VPN to be secured over public networks.

- **Apple Desktop (iSync) –** The ZCS client sync plug-in for Apple is similar in its architecture to the ZCS Connector for Outlook, but provides synchronization from Apple Address Book, iCal & iCal Events to the ZCS server. Apple Mail is synchronized as well, albeit via IMAP. All network communications support privacy via SSL/TLS.
- **Novell Evolution –** The ZCS client sync plug-in for Novell Evolution (Ximian) is very similar in architecture, using the Evolution Data Server (EDS) as its client-side integration point and providing synchronization via the same secure web services transport.
- **POP3 –** SSL/TLS or clear text transport. Underlying asynchronous I/O implementation ensures efficient handling of many concurrent clients.
- **IMAP4 –** Zimbra supports most of the IMAP4 extensions. Available via SSL/TLS or clear text transport, and also with asynchronous I/O for server scalability.
- **iCalendar –** Zimbra is based on the emerging Internet calendaring standard for scheduling events, busy time, to-dos, etc. embodied in the following RFCs:
    - RFC 2445 - Internet Calendaring and Scheduling Core Object Specification (iCalendar)
    - RFC 2446 - iCalendar Transport-Independent Interoperability Protocol (iTIP)
    - RFC 2447 - iMIP (iTIP over MIME)

  Zimbra also publishes calendar data (such as in vCal format) for read-only access from clients like Apple iCal.
- **RSS, REST & SOAP –** All Zimbra functions are also securely published on appropriate XML endpoints for other applications or command-line utilities (including migration tools) to leverage.
- **"Over the Air" Synchronization to Mobile Devices –** Zimbra's sweet spot smart phones that provide native Personal Information Management (PIM) software that includes messaging, address book, and calendaring applications. With the appropriate device synchronization software (no Zimbra software is required on the handset!), Zimbra Mobile can today synchronize contacts, appointments, and email with Nokia, Motorola, Samsung, Treo, and Blackberry smart phones. (Only the Blackberry devices require any additional server software—the rest sync straight from the device to the ZCS server.) Moreover, since intranet and Internet applications can access the ZCS server over open/standard protocols, there is now an easy way to extend applications to mobile devices via ZCS: such as exporting customer contacts from the CRM system, exporting a travel itinerary from your procurement application, or simply sending an email request to visit a particular URL from the browser.

Mash-ups and Zimlets. Zimlets, which are the topic of their own whitepaper, represent a widget architecture for both UI and server-side integration (again, as shown in Figure 1). The term "mash-up" has been coined to describe the aggregation and customization of multiple web interfaces/services to eliminate context-switching between existing systems. Since the sum can indeed be greater than the parts, mash-ups can deliver surprisingly richer user experience as well as improved productivity.

The Zimbra Zimlet architecture enables developers to declaratively integrate information from disparate sources—such as the Internet and enterprise information systems—into just about any content within ZCS, including email messages, contact fields, and calendar appointment notes. In fact, ZCS comes preconfigured with several of these Zimlet XML templates—including (just for example) integration with Google/Yahoo Maps, Amazon, Salesforce.com, and VoIP systems, but the real upside is the viral proliferation of Zimlets as the model is embraced by the open source community.

The architecture of Zimlets consists of a client and server framework, as well as a set of web services and JavaScript APIs. Zimlets are exposed to the end user via the ZCS Ajax client. The Zimlet server infrastructure provides the template for lifecycle management such as deployment, configuration, access control, and "undeployment". In addition, the Zimlet server framework provides the ability to integrate with the ZCS search engine (such as for indexing specially recognized objects like customer identities), while the Zimlet client framework provides support for context menus on panel items, content objects, drag and drop support, and the embedding of remote content into ZCS. A key goal of the architecture is to enable the implementation of a broad range of these Zimlet templates, from those requiring little or no custom code (i.e. entirely declarative) to those that use JavaScript to tightly integrate rich UI behavior within the ZCS client, or Java extensions to do the same on the server-side. Most of the example Zimlets, however, required no coding whatsoever—their behavior is fully specified via XML template file which accesses the generic web service proxy on the server (since Ajax client applications make requests only to their dedicated server and are generally precluded from accessing arbitrary websites directly). For a deeper dive on Zimlets, please see the dedicated whitepaper and gallery of Zimlets on the Zimbra website.

Postfix, anti-spam, and anti-virus. The Zimbra Collaboration Suite also includes Postfix, which serves as the Zimbra Mail Transfer Agent (MTA) for sending and receiving email across the Internet. Postfix is “embedded” within Zimbra in that the installation, configuration, and operation of Postfix is (optionally) transparent to the Zimbra administrator. Postfix interoperates the ZCS Mailbox Server over industry-standard SMTP/LMTP. Alternative MTAs that support SMTP/LMTP can be used in place of Postfix.

In addition to being the Zimbra MTA, Postfix also provides a well-tested integration point (in Zimbra’s case, via *amavisd-new*) for third party anti-spam (AS) and anti-virus (AV) solutions. Indeed, this is precisely how we have integrated the SpamAssassin and ClamAV solutions, which are also bundled within ZCS. (Of course, external anti-spam/anti-virus solutions may also be employed with Zimbra simply by routing the incoming mail through such hosted services as Postini.) Postfix is also a natural integration point for message policy enforcement systems. All Zimbra messages (internal and external) flow through Postfix. For larger scale deployments, Postfix should be configured on separate servers than those running the ZCS Mailbox Servers (more on this below).

4. Zimbra Server Architecture

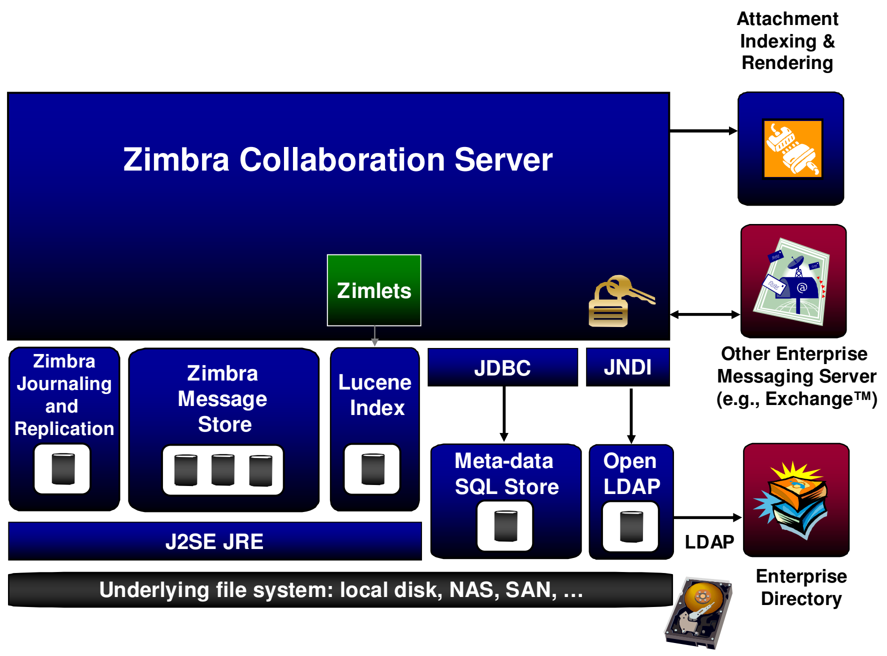
Attachment Indexing & Rendering
Zimbra Collaboration Server
Zimlets
Other Enterprise Messaging Server (e.g., Exchange™)
Zimbra Journaling and Replication
Zimbra Message Store
Lucene Index
JDBC
JNDI
Meta-data SQL Store
Open LDAP
LDAP
Enterprise Directory
J2SE JRE
Underlying file system: local disk, NAS, SAN, …

Figure 2: Back-End Architecture

Server architecture. Just as we are convinced that JavaScript/Ajax was the right choice for the ZCS native client, so did we find Java to the right choice for the principal server-side programming environment for Zimbra:

- Java is well-integrated with the Web - For systems software, the richest, most mature bindings to web technologies like XML and web services, are in C# (for the Microsoft-centric camp) and in Java (for everyone else).
- Integration of components in "managed code" - The Zimbra server includes numerous open source subcomponents as well as a substantial amount of new code. Combining many of these subcomponents into a single process image is substantial simplified in a managed-code environment like Java's—the Zimbra Collaboration Suite is an easier product to build, test, and troubleshoot, as a result of the Zimbra Community having to worry less about garbage collection, memory reference errors, reentrancy, and so on.
- Performance - You may not have expected to see performance in this list, but for I/O intensive applications like transaction processing and messaging, Java offers performance advantages over the alternative languages. Java has very good concurrency and asynchronous I/O, allowing Zimbra to multiplex inputs and outputs more efficiently within a single process image, and hence avoiding the additional overhead of process swapping and interprocess communications (IPC is typically three orders of magnitude more expensive than *intra-process* local procedure calls). The proof of course is in the pudding—Zimbra offers superior performance in head-to-head benchmarking against the fastest C-based alternatives. (Similarly, Java-based transaction systems like WebLogic and WebSphere are now competitive with C-based alternatives like Tuxedo and

Encina/CICS.) And so while Java may never compete with C in terms of instructions executed per second (by the way, Java *is* now competitive objected-oriented code written in C++), for multi-component, network I/O-intensive applications like Zimbra, Java is the better choice.
- Safe choice - Managed code is foundational to the server-side Internet architectures of the largest software companies—C#: Microsoft; Java: IBM, Oracle, SAP, BEA, and so on. Moreover, Java runs huge business-critical workloads within enduser IT today: e.g., there are individual corporations with more than 10,000 CPUs of production server-side Java environments, and application workloads in the many thousands of transactions per second.

Message store. The Zimbra message (or blob) store is built on the underlying Unix/Linux file system. The mapping is one file per message—we actually write the RFC822 MIME message representation directly to a file. This has a number of benefits:

- Once written, messages are immutable in Zimbra, and file systems are very efficient at storing and retrieving immutable unstructured/semi-structured blocks of data.
- Garbage collection is provided natively by the operating system, which offers both greater efficiency and lower administrative overhead.
- Independent utilities, tools, and scripts can easily recognize and process the Zimbra message representation on disk.
- Native operating system capabilities such as indexing/search (*e.g.,* Mac Spotlight), compression, and security.
- Last but not least, multi-level caching—Zimbra caches data itself, but Zimbra also benefits from the underlying operating system caching. Since enterprise messaging systems are generally gated on read and write performance, effective caching is one of the critical foundations of a scalable messaging implementation.

Indeed, we experimented by building a "thicker" blob store in which we laid out messages on disk "by hand" (analogous to the approach incorporated within many existing messaging servers), but found that Unix/Linux files systems are now sufficiently optimized that we could not do better, even with careful handcrafting of the disk layout code. In retrospect, this is not surprising: how could we hope to beat the investment that has gone into mature file system hardening and performance? (Moreover, by relying on top of standard file systems, the Zimbra server can support a broader range of network-attached storage.)

For each instance of the Zimbra mailbox server (one instance per machine), we store one copy of each message including all its attachments, even if that message is destined for multiple mailboxes. This obviously can lead to substantial savings in disk if large attachments are often sent to multiple recipients on a server. (Some messaging servers only provide single copy per mailbox or single copy per storage group, which can lead to significant redundancy in more expensive managed storage with no improvement in availability/fault tolerance.) For operating systems that support them, we utilize hard links to the single copy to efficiently ensure that a file is garbage collected after all users have deleted it from their mailbox metadata (more on this below).

While we have no strict prerequisites in terms of the Linux file system, we recommend EXT3, a highly-reliable journaling file system incorporated into Linux. We chose EXT3 for its mix of reliability (we run with *dirsync* enabled), performance, and its fast restart (*fsck*) via control-data journaling. (Zimbra does not require data journaling from the file system.) Linux itself is included within Zimbra software appliance downloads, but not downloads targeted for a particular operating system release (*e.g.*, Red Hat Enterprise Linux, SuSE, Mac OS, *etc.*).

Metadata SQL store. While messages themselves are immutable, the meta-data associated with a message goes through frequent state changes as it is used. Message meta-data includes such items as

- What folder is the message in?
- What tags does the message have?
- Is the message read or unread?
- What conversation is the message part of?
- And so on

The Zimbra Collaboration Suite includes an embedded relational database for managing mailbox meta-data, and it is reshipped within each Zimbra distribution. We choose to use a relational database for Zimbra meta-data because

- Once again, we sought to leverage the hardening and performance investment of smart engineers before us—For efficient searching and reliably updating highly structured meta-data, it is hard to beat relational databases.
- Caching — Unlike immutable messages, Zimbra's meta-data is both frequently read and written. Relational databases provide very efficient caching for both reads and writes. Update caching efficiency comes out of combining multiple updates, even those spanning transactions, into a single disk write (this optimization, of course, leverages the database's sequentially-written transaction log to ensure that no changes are lost in the event of an outage).
- The SQL interface provides the Zimbra Community with investment protection in that the ZCS mailbox server could be very easily ported to other relational databases from the MySQL implementation that is currently included in ZCS.

We selected MySQL in particular because

- MySQL was compatible with our open source licensing and distribution.
- MySQL has a large community that the Zimbra Community seeks to leverage.
- MySQL is easily embedded, so that it can be made transparent to those installing, configuring, and managing Zimbra deployments.

Search. Just as it is with Google Gmail and Apple Mail (via Spotlight), search is fundamental to the Zimbra Collaboration Suite. Typical Zimbra users tend to rely on a small number of (or even a single) archive folder and then use search to almost instantaneously lay their fingers on whatever information they are looking for. All the meta-data associated with a message (folder, read/unread, tags, and so on) is available as search criteria, as are indexes into the content of the messages and the content of any attachments to the messages. Zimbra provides a simple, yet rich grammar for specifying searches, and then offers a graphical search builder for constructing advanced searches without having to learn that grammar.

Like Gmail but unlike Apple Mail/Spotlight, Zimbra search is conducted on the server side. The advantages of server-side search are access to full mailboxes (including archived content), reliance on faster disk/CPU for increased performance (even factoring in the network latency), and the elimination of a substantial amount of redundant computation associated with indexing attachments sent to multiple recipients.

Zimbra leverages Apache Lucene to manage the index that enables fast searching, and Zimbra uses third-party software from Verity to extract text from attachments for indexing within Lucene.

Saved searches. Once constructed, searches can be saved within the normal folder hierarchy. A saved search acts like a virtual folder in that new messages that meet that search criteria will be displayed when you click on the saved search. The typical use scenario for saved searches is to provide "views" or "virtual folders" of your inbox or full mailbox, such as for new "external" messages (those coming from customer/partners outside of your domain), new "management" messages (those coming from your boss or your direct reports), mailing list messages, and so on.

Cross-mailbox search. All of the comprehensive Zimbra search capabilities are also available to administrators (subject to configurable authorization) for use across a specific set of mailboxes, such as for use in a compliance- or human resources-related discovery (see also archiving discussion below).

Lucene Index. Lucene is a high-performance, full-text search engine from Apache. Lucene works by generating a full "segment" index for all words/tokens in a particular message, and optionally, its attachments. This segment is then merged into the receiving users' existing index (one index per user). The index itself is represented in flat files. Search simply traverses these optimized file structures, often in parallel.

Search is very fast—users perceive it to be nearly instantaneous. The preprocessing required to construct an index, on the other hand, grows linearly with the size of the text. Attachments tend to be the overriding factor in this overhead. Attachment indexing is a function of user's class of service, and so can be turned on or off based on server scaling requirements. Message text indexing is more essential to the user experience (as well as significantly lower-overhead due to the smaller datasets)—it is a big part of the reason why many Zimbra users become comfortable with dropping their complex, time-intensive folder hierarchies in favor of a single "Stash" folder, because powerful search frees them from the worry that they will not be able to find what they are looking for.

We have found this Lucene index to typically be about 20% of the size of the text being indexed. Simply by compressing messages and their attachments prior to storage, we can make up the space required to store the Lucene index. As a user's index grows larger, the savings increase, since there is greater reuse of tokens. Garbage collection is done on the indices only after expunging messages (emptying the trash), so that trashed messages can still be found. (Many Zimbra users choose to use their trash as a secondary archive folder, since no message data need be expunged from the trash until the user’s quota is reached.) Should a more catastrophic failure lead to an index corruption, there is also an administrative interface for regenerating Lucene indices (which are, of course, idempotent).

Attachment indexing and rendering in HTML. Zimbra leverages Verity to provide its ability to index most every message attachment type found on the Internet. The Verity Server (it runs as a separate process as shown in the diagram above) also provides the capability to render all of those attachment types in HTML—which is useful for providing read-only access to documents for if the appropriate client-side software is unavailable (such as on a home system), as well as for limiting potentially harmful binary downloads to the client. This risk of viruses and worms can be mitigated by opening attachments (including attachments nested in zip files) on a secured Unix server and then deliver them to the client rendered in benign HTML. These features are configurable, both system-wide in real time (for responding to a virus attack) and user by user (as a class of service).

Journaling. The Zimbra journal is like a database transaction log in guaranteeing that no data is lost in the event of a failure and that the system can restart quickly. First and foremost, the Zimbra journal ensures consistency across the three Zimbra databases—the message/blob store, the meta-data store, and the Lucene-based index. For example, the Zimbra journal logs sufficient logical data to ensure that if a message has been committed to disk within the message store, then subsequent failures could not prevent the meta-data and index from also being appropriately updated. Delete, archive, and other operations work similarly. In this way, the Zimbra Journal serves as a “meta” transaction log to ensure that all the underlying state maintained by Zimbra is consistent and correct.

Zimbra’s journaling subsystem is also the foundation for hot backup, smart recovery (individual mailbox, point-in-time, etc.), and replication (such as for site failover and disaster recovery) on commodity hardware (more on replication below.)

Hierarchical Storage Management (HSM). Zimbra natively supports volume management and the auto-aging of messages from fast spindles to slower ones, such as via HSM hardware. The fact that all Zimbra data-storage is file based (as opposed relying upon raw partitions) makes HSM integration relatively straightforward (just as it simplifies supporting arbitrary network-attached storage).

5. Scalability via Partitioning and Distribution

Zimbra was designed from the ground up to scale to meet the needs of businesses with 100,000s of users and service providers with millions and even tens of millions of users. The Zimbra architecture inherits from distributed systems expertise that was gleaned building messaging systems that today host more than 100 million mailboxes world-wide and Java/Web systems that have thousands of production server CPUs within single large-scale deployments.

Of paramount importance to scaling is partitioning. Effective partitioning depends upon leveraging "locality of reference" for both processing and data—if certain servers can be specialized to solve some subset of the bigger problem, then the essential code and data are more likely already to be in memory or close at hand on fast disk. Partitioning techniques include the *vertical* partitioning of functional tasks and the *horizontal* partitioning of data and the associated processing (more below).

Partitioning is augmented by other well-honed distributed systems techniques like automated replication, data dependent routing, load balancing, and failover. Overall, these techniques have proven (e.g., Google, Yahoo!, Amazon, etc.) to scale well beyond the reach of more centralized architectures that tend to rely on stateless processing and very-large databases.

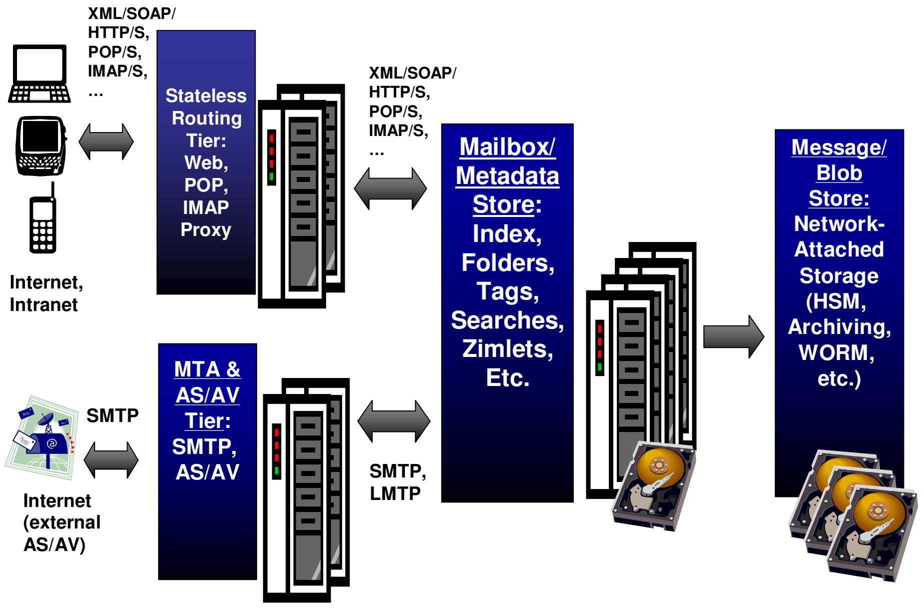
XML/SOAP/
HTTP/S,
POP/S,
IMAP/S,
...
Stateless Routing Tier: Web, POP, IMAP Proxy
Internet, Intranet
XML/SOAP/
HTTP/S,
POP/S,
IMAP/S,
...
Mailbox/ Metadata Store: Index, Folders, Tags, Searches, Zimlets, Etc.
Message/ Blob Store: Network-Attached Storage (HSM, Archiving, WORM, etc.)
SMTP
MTA & AS/AV Tier: SMTP, AS/AV
Internet (external AS/AV)
SMTP, LMTP

Figure 3: Distributed Architecture and Partitioning

Vertical partitioning allows complex processing tasks to be divided into subtasks that can be independently optimized, managed, and debugged. Vertical partitioning within Zimbra primarily consists of off-loading the computationally-expensive **security tier** (which interconnects ZCS with the Internet), from the ZCS mailbox servers (which manage user data—messages, appointments, contacts, etc.). This security tier includes Postfix—the ZCS Mail Transfer Agent (MTA) included within Zimbra for mail routing, policy, *etc.*, as well as any on-premises anti-spam and anti-virus. (Zimbra includes the leading open source AS/AV solutions in SpamAssassin, DSPAM, and ClamAV, but is also compatible with commercial AS/AV.)

Zimbra's security tier is effectively stateless—the SMTP protocol provides for the automatic redelivery of unacknowledged messages, and Zimbra doesn't *ack* until the message has been transactionally stored within the user's mailbox. This allows you to independently size your ZCS MTA server farm based on aggregate security processing workload, but still have Zimbra manage all the distributed subcomponents as well as the automated routing of communications to and from the ZCS mailbox servers (via SMTP & LMTP). The ZCS MTA tier (or optionally any other MTA used in conjunction with Zimbra) routes inbound requests by looking up the user(s) in the LDAP administrative repository (more below).

Horizontal partitioning is far more crucial for large-scale deployments, since there is generally a lot more data than tasks. All medium and large Zimbra deployments are horizontally partitioned across servers (and the attached storage) by end-user mailboxes. An end-user's mailbox includes his or her messages, calendar(s), address book(s), documents, and so on, which are all collocated for very efficient context switching and search across applications. So ZCS servers are inherently stateful—each serves as the *primary* home for a subset of the aggregate mailboxes. This requires that each ZCS server have the smarts to reroute a protocol request (via XML/SOAP/HTTP/S, IMAP/S, POP/S, RSS, iCal, *etc.*) to the appropriate primary server in the event that an in-bound load balancer makes the wrong decision.

Automated replication and failover is also essential for large-scale deployments. For example, LDAP configuration data (which includes end-user mailbox home locations) can be fully replicated/partitioned across as many replica servers as are required to meet performance and availability requirements. LDAP replicas may be collocated with ZCS mailbox servers, MTAs, or "vertically partitioned" to dedicated administrative servers.

Mailbox data, on the other hand, can be transparently replicated within the underlying storage system (such as by using RAID or mirroring) for availability only. (It simply does not make sense to replicate mailboxes for scalability, given how frequently state data is updated, and the overhead of ensuring transactional consistency between multiple mailbox copies.) *Clustering* technology is used to automatically failover from the primary server to a preconfigured secondary (or tertiary) server, which then assumes the role of primary for that mailbox. (*I/O fensing* and associated technologies ensure that the former primary no longer has "write" access to the mailbox in order to avoid "split-brained syndrome"). In the future, Zimbra will also support mailbox replication on "vanilla" storage (such as for multi-site disaster recovery) by replaying the ZCS transaction or change log (used to guarantee consistency between the message and meta-data stores) on a secondary server (more below).

Meta-data optimization/partitioning is one of the less frequently discussed scaling techniques employed within the Zimbra architecture. *Meta-data* for a mailbox is generally all of the data required for *navigating* to the appropriate message or meeting. Zimbra meta-data includes ZCS's very efficient, Lucene-based index into all the text contained in every message, meeting, contact, document, attachment, and so on. Zimbra meta-data also includes the structured meta-data that captures folders, tags, dates, read/unread status, etc. Zimbra uses an off-the-shelf SQL database for optimizing structured meta-data queries and updates.

Meta-data is, of course, horizontally partitioned by user into the appropriate ZCS mailbox server, but the key additional insight is that this meta-data can also be partitioned from the target data (message body, meeting, document) to ensure very efficient search, UI painting, and so on. Separating the meta-data and target-data makes it far more cost-effective to keep the meta-data on fast disk, allowing sophisticated search and navigation to be nearly instantaneous even across multi-gigabyte mailboxes. Latency in access to the message body itself (which could, for example, reside in an HSM system) is not nearly so problematic to the user experience as latency or inefficiency in accessing the meta-data. Partitioned meta-data also allows potentially expensive operations such as compliance-related cross-mailbox discovery to be handled efficiently (via simply composing the appropriate horizontally-partitioned search results; more on this below).

Ajax client impact on scalability. As the Ajax model is relatively new, questions continue to be raised regarding Ajax's impact on Web 2.0 application scalability. (Of course, Google, Yahoo!, and MSN have all demonstrably proved out Ajax scalability within their Web portals, albeit with the proviso that the impact on their back-end servers has not been publicly disclosed.)

Applications broadly consist of UI logic, business logic, and data (access & update) logic. In the old days (and at the risk of dating the author), all of this programming logic ran on a centralized or "mainframe" server, which means that for each user operation, the server is responsible for doing all of the work. Traditional fat client applications, turn this model on its head by off-loading all of the UI logic and most of the business logic and data manipulation from the server to the client (modulo database stored procedures and triggers). Fat client applications can nevertheless hammer their servers simply by not being sophisticated about how much and how often data was being "shipped" to the client for processing—that is, *data shipping* to the client can actually be more expensive on the server than *function shipping* to that server (with stored procedures, triggers, et al). With a reasonably smart design, however, fat client applications typically use more client and less server CPU per operation than a corresponding server-centric application.

In terms of its broader application architecture, the Web platform looks quite a bit more like a mainframe application: Yes, the 3270 green screen has been replaced by the far richer browser and HTML, but the UI and virtually all of the business logic (modulo Javascript field validation) is done on the Web application server. Of course, the fat client model simply does not work on the Web: Organizations can ill afford to ship their business-critical, confidential data off for untrusted client applications to access and update. Nor can they afford to share their differentiable business logic with competitors and hackers. At the same time, we should recognize that in general Web applications are inherently *more expensive* in terms of server load than traditional client server, because the server code is also responsible for the computationally-significant UI work of constructing an HTML response for each HTTP request.

Now to Ajax. Ajax can be described as a *husky* client architecture, because the network boundary between client- and server-side code is defined by the application programmer!—that is, somewhere between "thin" and "fat", but with all the deployment advantages of "thin". The client processes an XML response to a Web application service (via Service-Oriented Architecture or SOA).[2] Arguably, the most significant benefit of Ajax and SOA is in the re-use, security, and maintainability that comes along with these Web application services, but there are significant performance ramifications as well. Specifically, compared to a traditional Web application, Ajax/SOA app's consume *less* server-side CPU, because the UI processing is off-loaded to the client (try watching your CPU meter when Ajax is getting crunched within your browser).

No doubt, this is a generalization: The Ajax/SOA architecture requires that the client generate XML invocations for the server and then parse the XML (or interpret JSON) responses. Since XML processing is computational expensive on the server as well as the client, it is definitely possible to craft poorly performing Ajax applications. But in general, the performance hurdles in realizing Ajax are browser-based rather than server-based, simply because the UI computation has been off-loaded to the client. In fact, Ajax applications tend to go to the server more often than typical Web applications, since the cost of the server invocation to, say, get my appointments when I mouse-over a date, is far cheaper on both the server and the network than generating a whole new page (especially when the result set is cached for subsequent access). That finer granularity of access does require that Ajax server logic be more stateful than some Web applications (although the broad trend is for statefulness in Web applications as well). However, in terms of the server-side workload, our claim (borne out by experience to date) is that the typical Ajax application will have a smaller footprint than an analogous Web application. The same is true of network bandwidth—Ajax applications generally consume less than traditional Web app's, with the caveat that the initial download of the Ajax client code can be a significant hit if you are not on a broadband connection.

Most importantly, however, is the fact that the interface between Ajax client and server application (whether it is written in Java, C#, PHP, or Ruby on Rails) is chosen by the application designer. Making an deliberate effort to choose that boundary well—for modularity, for re-use/mash-up, for security, for performance, and for longevity—and managing server workload will be far less significant challenge.

---

[2] Indeed, this is precisely the modular separation that Web application architects have been advocating for years, even when all of the actually code was running on the server-side. One of the big frustrations application architects continue to have with the typical Internet application is that while the architects advocated a strong separation of UI logic (JSP) from business logic (JavaBeans, EJB Session Beans) and data logic (JDBC, EJB Entity Beans), many applications violate modular design practice by, for example, sprinkling JDBC calls directly into their JSPs.

6. Quality and Open Source

In this section, we hope to make two points: (1) to justify that in the pursuit of software quality—reliability, ease of use, security, and so on—open source is an *asset* rather than a liability; and (2) to discuss the quality assurance (QA) regimen of the Zimbra Collaboration Suite.

If the first point remains controversial, it is in part because of the fear, uncertainty, and doubt raised by vendors that are competing with open source. The reality, of course, is that most software (open source as well as proprietary) is of insufficiently high quality from the end-user's perspective. After all, the QA investment required to get from innovative software project to widely-adopted, business-critical software product (with fault tolerance, security, integration, compliance, performance, scalability, usability, ease of configuration, ease of administration, …) is high enough that few software projects (open source or proprietary) ever get there.

But some open source software clearly has cleared this hurdle. Take Apache, Linux, and Firefox for example, each of which is of defensibly higher quality than its proprietary alternatives. Perhaps the intended criticism being made by the proprietary software vendors is simply that open source software is cheap enough (free) that the associated QA regimen must somehow be inadequate. Closing this is a gap is precisely what the commercial open source (not an oxymoron) vendors (e.g., Red Hat, Mozilla Corp., Zimbra, but also IBM, Oracle, and Apple) have as their *raison d'etre*! But more importantly, this argument discounts the QA contribution of thousands of talented community members that are testing and troubleshooting the software in advance of end-user commercial deployment.

Of course, not all open source software reaches such quality "critical mass", but for the ones that do, open source is an asset in the pursuit of software quality. The following open source benefits have all contributed substantially to the quality of Zimbra:

- **Scrutiny of the source code -** To ensure software quality, there is nothing like the added developer accountability that results from the public scrutiny of (and commentary on) his/her code base. Products that pass muster are likely to have cleaner, more elegant internal architectures that will make them a better long-term investment. (The reverse is also true—vendors of proprietary products often fear that open source will air their architectural dirty laundry, as much as they fear it will cannibalize their license revenues.) It turns out that this is true even for security—public scrutiny of the source code has actually helped Linux, Apache, and Firefox strengthen their network security far more than it has helped attackers identify potential weaknesses.

  The source code for the Zimbra Collaboration Suite is available via the Zimbra public website www.zimbra.com for all to download, use, and create their own derivative works thereof.

- **Transparency -** Open source projects/vendors are far more likely to open up their bug databases, provide open communication between developers and users (via forums), as well as to open their development, build and test regimen. By inviting users into the "sausage factory", open source is at least welcoming greater participation and feedback on quality.

Developers and administrators that are interested in participating in the Zimbra Community can join via www.zimbra.com/community/, which includes access to the Zimbra bug and enhancement database, forums, collaborative Wiki (for documentation, localization to other languages, etc.), Zimlet and "skins" gallery, and so on.

- **Continuous quality assurance (QA) -** Most open source software is developed in public in that new code for the project is typically posted to the Internet daily or weekly. The fact that community members (as well as anyone else with an Internet connection) can download and use this forward-development code demands a higher-level of automated QA on the nightly and weekly builds. In this way, quality assurance is built into the development process rather than left to the end of the "new feature" coding cycle.

  The Zimbra QA regimen is founded upon comprehensive automated testing of both the ZCS client and server applied to each ZCS build. The client-side QA regimen uses scripting in Mercury QTP to simulate and validate most every human interaction with the Zimbra Ajax client across browsers. For the server-side, Zimbra has developed a test fixture called *Soapgen* that can simulate tens-of-thousands of concurrent users performing virtually every operation on their mailbox. Soapgen is a Java-based client application that relies on the mapping from user actions to the XML/SOAP calls that would be made by the ZCS Ajax client (as well as the synchronization calls from, for example, the ZCS MAPI provider for Outlook) to the server. Some of these tests focus on individual operation correctness, while others focus on aggregate performance, scalability, and fault tolerance.

- **Intellectual property (IP) hygiene -** Most software (both open source and proprietary) is now built from existing components. There is arguably now less risk of IP contamination (that is, does your vendor own the IP rights to the software they are giving you?) with open source than with closed source alternatives, because all such open source dependencies are transparent.

  We at Zimbra are zealots about protecting the IP "cleanliness" of the Zimbra code base. We are very careful about incorporating third-party contributions and technologies (generally using only state-of-the-art open source projects with compatible licensing). Moreover, we require IP assignment before we incorporate any potential contributions from the community within the ZCS code base. (This is increasingly common practice for open source organizations like Apache and the Free Software Foundation, as well as companies like Zimbra.) Together these policies protect the Zimbra community and end-users from any hidden dependency on encumbered IP.

- **Community QA -** Last and most importantly, successful open source projects are able to draw large developer and testing communities for more economically than proprietary vendors can. (It is after all a trade—open source projects share their core software for free precisely to draw the value-add of this greater community.) The collective validation and testing investment made by the respective communities are what make Linux, Apache, and Firefox arguably the most secure and highest quality technologies in their respective categories.

Zimbra enjoys an active, thriving open source community, one that is now the largest in its category (for example, Zimbra is the community choice award winner on Source Forge). However, an even larger quality benefit accrues from Zimbra's re-use of existing open source components within the ZCS server. If Zimbra had endeavored, like many other messaging efforts before us, to build our own blob store (we instead use the Linux/Unix file system), our own mailbox store (we use a relational database), our own web container (we use Apache), our own MTA (we use Postfix), our own index (we incorporate Lucene), and so on, then we could never have applied the same QA regimen to our homegrown versions of these components than that which each has independently received from the open source community. This allows the Zimbra team to focus on innovation instead of reinventing (and re-QA'ing) the wheel.

In summary, software quality varies more between projects/products than it does within categories (OSS versus proprietary). So users must continue to do due diligence in order to ensure that the quality of the candidate software is sufficient for their needs—the more business-critical the deployment, the greater the need for picking winning technologies whose risk is low because quality assurance is broadly amortized. However, between comparably successful proprietary and open source alternatives, open source is likely to both provide higher quality for the price and, given the above, higher quality in general!

7. Clustering, Replication and Archiving

Clustering. Zimbra is integrated with operating system-level clustering in order to provide rapid failover in a typical LAN/MAN topology. For Zimbra, clustering means "single copy" failover, in that there is one image on disk (modulo underlying RAID redundancy) for which a secondary server "takes over" from the primary. Zimbra, for example, is integrated with the Linux-/Unix-based cluster suites (such as the Red Hat Enterprise Linux/RHEL Cluster Suite) in order to allow the secondary server to assume control over a set of mailboxes associated with a failed primary server. In a typical configuration, the designated Zimbra failover server(s) would also host primary mailboxes of their own in order to avoid the additional expense of idle hardware.

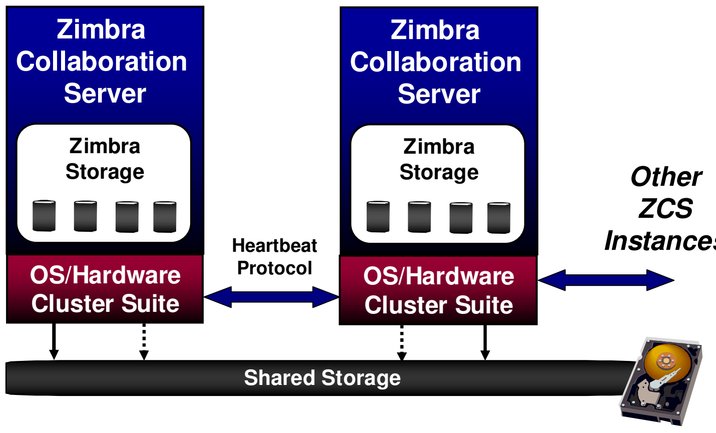
Zimbra Collaboration Server
Zimbra Storage
OS/Hardware Cluster Suite
Heartbeat Protocol
Zimbra Collaboration Server
Zimbra Storage
OS/Hardware Cluster Suite
Other ZCS Instances
Shared Storage

Figure 4: Clustering Architecture

In such a clustered configuration, Zimbra relies upon network-attached storage, and optionally using RAID (if configuring RAID, we recommend RAID 0+1 for write performance) to allow the secondary to assume control of the volume(s) formerly used by the primary. Upon mounting the partition for read/write from the secondary, the file system journal is used to repair any partial writes to disk, and then the Zimbra secondary simply replays the transaction log written by the primary in order to ensure no loss of data integrity and consistency.

The operating system cluster suite provides the heartbeat protocols essential for detecting failure as well as the I/O barriers (or I/O "fensing") for avoiding "split-brain" syndrome—i.e., for ensuring that only a single instance of a ZCS mailbox server is able to write to a specific user mailbox at a given time. When the cluster suite detects a failure, it also invokes the appropriate Zimbra initialization scripts. During recovery, the Zimbra secondary also generally assumes the IP address of the Zimbra primary.

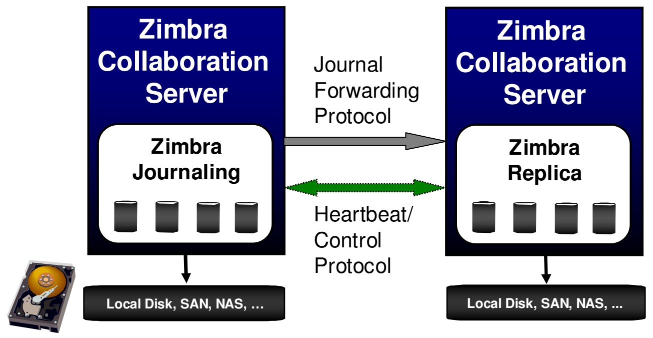
Zimbra Collaboration Server
Zimbra Journaling
Journal Forwarding Protocol
Heartbeat/ Control Protocol
Zimbra Collaboration Server
Zimbra Replica
Local Disk, SAN, NAS, ...
Local Disk, SAN, NAS, ...

Figure 5: *Future:* Native Replication Architecture

Future: Native Replication. Zimbra clustering can be characterized as "single copy" failover in the sense that, from perspective of ZCS, there is only a single disk image to worry about (RAID mirroring is hidden to Zimbra). The Zimbra Collaboration Suite will in the future also provides "dual copy" failover that does not rely on underlying shared disk. ZCS native replication is likely the better fit for failover across a WAN, such as for multi-site disaster recovery wherein shared network attached storage may be infeasible. Zimbra replication is also well suited to commodity server and storage hardware.

The Zimbra replication architecture is master/slave, and is based on the same Zimbra journaling subsystem used for fault tolerance, back up, and recovery. The master is responsible for journaling operation records both to the primary and spooling those same records to the secondary. An efficient binary protocol is used to steam log data from primary to secondary. Records must be committed on both primary and secondary before the primary confirms operations/receipt of email to ensure no loss of data in the event of a catastrophic failure. The secondary Zimbra server is then responsible for asynchronously "replaying" its local copy of the journal to perform all successful updates on its local databases, and as such will tend to run a few seconds behind the primary (but, of course, with zero chance for any loss of data).

Under the Zimbra replication architecture, each server has their own copy of all database state, so "split brain" syndrome is only a risk if both the Zimbra primary and secondary have active clients, such as could occur in the event of a network partition. To avoid split brain syndrome for ZCS replication, if the secondary is unable to contact the routing infrastructure (in order to assume the primary's IP address), then a network partition is presumed to have occurred, and no failover takes place (the assumption being that split-brained execution and the complexity of subsequent reconciliation is worse than an end-user experiencing a temporary outage).

Archiving. Archiving for enterprise messaging has emerged as a critical vehicle for implementing retention policies, such as may those required for compliance with Sarbanes-Oxley (SOX), the Health Insurance Portability and Accountability Act (HIIPA), human resources regulations, and so on. The most typical architecture for email archiving is two distinct systems: an email server (such as the Zimbra Collaboration

Server or Microsoft Exchange) and the archive server (such as EMC Legato or Veritas KVS) employing WORM (write once, read many) storage. Zimbra supports such a configuration, and the Postfix MTA included within the Zimbra Collaboration Suite can easily be configured to act as the forwarding agent, as illustrated in Figure 6.

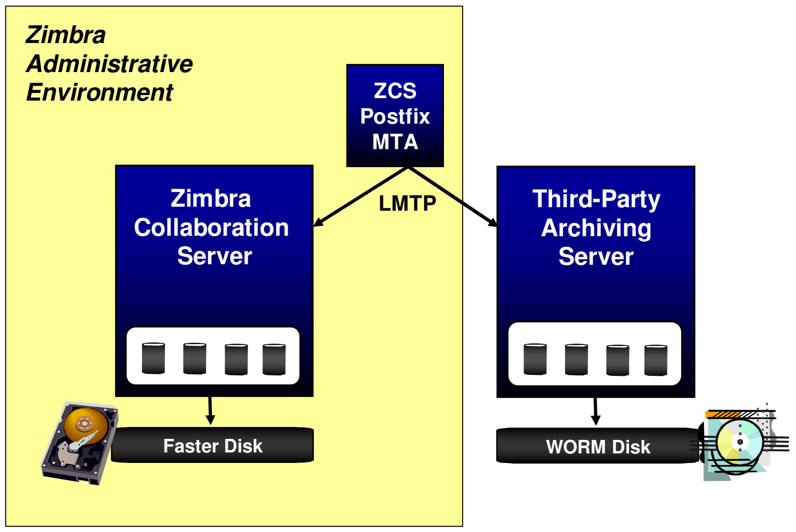
Zimbra Administrative Environment
ZCS Postfix MTA
LMTP
Zimbra Collaboration Server
Third-Party Archiving Server
Faster Disk
WORM Disk

Figure 6: Third-Party Archiving Architecture

There are liabilities associated with this archiving architecture, however:

- Implementing and enforcing consistent retention policies (such as "seven years and out") between the messaging and archiving system is problematic.
- All messages and large attachments are typically stored twice (or else there is a painful, error-prone integration between the on-line email server and the archive), and redundant copies are maintained until either the user deletes and expunges or the archival retention time limit kicks in.
- Once a user inadvertently deletes messages from their primary, there is no easy way to give him or her access to the archive.
- Two disparate systems and the associated storage need to be configured, secured and managed.

So while Zimbra does support the typical archiving configuration above, ZCS was designed to also offer an alternative approach in which Zimbra manages both the primary (fast disk) and the archive. Under this model, all of Zimbra's rich user search capabilities are made available to administrators for cross-mailbox discovery (subject, of course, to appropriate security policies). An integrated archiving approach allows messages to be automatically "aged" out of the primary mailbox into the archive—in this case, the primary meta-data is updated so that the user still has access to the message (at least until such time as a system-wide or user-specific retention age limit is reached). From the user's perspective, message migration to the archive is transparent modulo some potential additional read latency from a slower volume or HSM system. This integrated approach also permits the end-user to directly access the archive, such as to examine deleted but not yet expired messages.

8. Configuration Management

Zimbra stores all configuration data, including per-user class of service (CoS) data and the name of the primary/secondary mailbox server (used by the MTA and proxy tiers for routing) within an LDAP directory. OpenLDAP is embedded within Zimbra in order to manage all of this Zimbra-specific configuration data without reliance upon external directories. At the same time, Zimbra supports the proxying of user login and access to the Global Address List (GAL) to an existing enterprise directory such as Microsoft Active Directory or other LDAP-compliance directories. Indeed, while we support ZCS configurations in which (1) non-Zimbra configuration data is stored within Zimbra's embedded OpenLDAP, and (2) Zimbra configuration data is stored within an independent enterprise directory, the sweet spot seems to be (3) store Zimbra-specific configuration data within the Zimbra-managed, embedded OpenLDAP and store Zimbra-independent enterprise configuration data within the existing enterprise directory.

While the slowly changing configuration data within the LDAP directory should never become a scaling bottleneck, OpenLDAP does support replication and partitioning in order to accommodate very large Zimbra deployments.

9. Security

The Zimbra Collaboration Suite employs a classic network security model in that the server-side systems must generally be trusted, but the client-side need not be. Within a Zimbra deployment, as with other web and database systems, the servers should be physically protected (*e.g.*, within a data center, locked machine room, or closet). Moreover, administrative access to the Zimbra servers and the underlying operating systems should be limited to your designated administrators and from your secured intranet (except as necessary via secure transport from trusted external sources, such as Zimbra's support organization).

ZCS was designed to support highly-secure deployment on intranets as well as on the Internet. By opening up ZCS servers to access over the Internet, businesses and organizations can deliver secure messaging and collaboration to their employees/members even on their home computers, public Internet kiosks, and so on. Moreover, they can do so without having to deploy a costly Virtual Private Network (VPN). (A VPN is, of course, the end-user's choice, since Zimbra works as well with VPNs as without.)

Network privacy and server authentication. The Zimbra security technology that makes secure access over public networks possible is the same as that which has been in use for a decade on the Internet: privacy and server authentication via (optional, but recommended) SSL/TLS encryption for all network communications. The Transport

Layer Security (TLS) encryption and authentication protocol is the web *defacto* standard (signified by "*https:*" URLs). (TLS is the incremental successor of Secure Sockets Layer/SSL, but we follow common practice in using the two terms interchangeably.) In typical use (as within Zimbra), only the server is authenticated, but all network communications (from browser to server and server to browser) are strongly encrypted for privacy. TLS encryption has never been broken outside of artificial laboratory settings. In addition to protecting the XML/JSON HTTP communications to/from the Zimbra Ajax client, TLS can be used to protect communications to/from other messaging and collaboration clients via standard protocols like POP/S, IMAP/S, SMTP/S, and so on (where the "/S" signifies that POP, IMAP, SMTP, *etc.* are respectively running over a secure TLS transport).

Client authentication. Since client systems cannot be physically secured like server systems, the client is not trusted in the ZCS architecture (just as it is not within other web systems). For ZCS, all client/server invocations (other then login, of course) require an authentication token to execute. That *auth-token* contains a cryptographically secure representation of the user's individual and machine/network identify, as well as an expiration time. This auth-token prevents classic data-injection attacks on the server wherein, for example, a malicious user changes the id/username embedded in a URL or a hidden form in order to gain access to services to which they are not so authorized. The fact that the auth-token is cryptographically secure means that would-be attackers cannot glean sufficient information from snooping on authenticated communications or on the PC client disk to break into the system. (A network snooper could, however, potentially read someone's email if the ZCS server in question does **not** require network privacy via TLS, which is in part why TLS is recommended.)

Anti-virus software/hardware. Any secure ZCS configuration should also necessarily be running up-to-date anti-virus (AV), anti-worm, and anti-phishing software (typically, all three are now grouped together within AV solutions). Zimbra administrators are free to choose the open source ClamAV solution embedded within Zimbra or a third-party alternative(s) that may either be offered as a hosted service or else via "on premises" software or appliances. For "on prem" AV solutions, the typical integration for third-party software is via the *amavisd-new* plug-in framework for Postfix, or instead by "daisy chaining" other MTAs in front of the Zimbra Postfix, which is the right approach for email security appliances.

AV software is most important for traditional messaging and collaboration clients (Microsoft Outlook) that download and manipulate messages, but ZCS web client users still have the option (unless it is turned off for their class of service) to download potentially malicious attachments if they were not trapped and quarantined by the AV software.

Zimbra also allows attachments to be opened on secured Linux/Unix servers (which are much less vulnerable to attack than Windows servers), and then rendered to the less secure client in HTML. While server-side attachment rendering is a security benefit, it does raise the possibility that the software used to extract indexing text from or to render an attachment (ZCS uses Verity) could itself be vulnerable to a virus/worm. While Linux/Unix viruses are substantially rarer than their Windows counterparts today, it nevertheless makes sense to view the AV solution as protecting the server as well as the client.

Ajax security advantages. While they rely on the standard web platform technologies, Ajax applications do come with additional security considerations (discussed below). Less often discussed are inherent security *advantages* of Ajax clients:

- **Dynamic Ajax client download** – Ajax client code is downloaded on demand from the trusted ZCS server after a particular user logs-in. No software is left on the client for a malicious person to tamper with.
- **No persistent client caching** – A substantial exposure with traditional web mail clients is that they cache HTML data that includes message contents, addresses, *etc.* on the client disk during normal operation. This can be significant security vulnerability for access from public kiosks or other shared computers. Ajax applications like the ZCS client do no data caching on disk. (While the ZCS application code may well be cached on disk during use, no user data ever is; see below)
- **Server-side control of intranet and Internet mash-ups –** Zimlets and other mash-ups are precluded from accessing arbitrary services on the Internet, and must instead (like Java applets) make all invocations back to the originating server (in our case, the ZCS server). This means the ZCS server can act as a secure, proxy gateway for accessing intranet applications, and can govern which externally web services (if any) are accessible for mash-up within the Zimbra Ajax client.

Ajax security considerations. The most obvious security issue for Ajax applications is that the associated source code is inherently downloaded to the browser for interpretation. This is a concern for any application code that contains intellectual property that the author does not want to share with the world: while obviscation and white-space removal can certainly render JavaScript much harder to read, developers should consider Ajax applications to be like HTML in that others will be able to examine your code. The only real answer is to either (1) limit log-in to the application to trusted users/partners using TLS for privacy (see below); or else (2) keep closely-held algorithms on the server-side, and simply invoke them via web services from the Ajax client.

All this is not an issue for the Zimbra Ajax Client, since it is, after all, open source. But privacy of the code does highlight another critical benefit of using TLS: with TLS, the Ajax application itself (JavaScript, Cascading Style Sheets, etc.) cannot be tampered with in route to the user's browser, and if access is limited to trusted users, then only they get potential access to the Ajax source code.

Subject to the additional precautions enumerated below, Ajax applications can otherwise be made as highly-secure as the web technologies upon which Ajax is based. Zimbra provides the following additional guarantees to further secure ZCS Ajax deployments over even public networks:

- **No server-side interpretation of JavaScript or other client-submitted code** – ZCS receives vanilla XML requests from the browser client that are validated and then processed by ZCS server-side Java code. No JavaScript flows from client to server, and there is no server-side interpretation of any application data (message bodies). In fact, there is no JavaScript execution on the server period. This ensures that there is no way for even a hostile Zimbra Ajax client with an authentication credentials to inject malicious code for execution on the server-side.
- **Limited client-side interpretation of JavaScript within user data** – The Zimbra Ajax client is, of course, an application that runs within the confines of the web browser. There is an additional risk, then, to displaying the contents of rich HTML messages that themselves contain JavaScript, in that this JavaScript could make malicious calls to the ZCS server (it is generally precluded from making invocations to other sites). Zimbra does attempt to filter/block any "risky" HTML, first on the server and then again on the client, but such filtering is like AV software in that it cannot be guaranteed 100% effective. When a rich HTML message contains any suspect JavaScript, better to err on the side of caution.
- **Benign URLs** – All Zimbra's GET-based REST and URL-based APIs are read-only and do not modify data. This ensures that users (with pre-validated security credentials) cannot be fooled into clicking on a malicious link that someone sends them in an email or posts on their external website that would then, say, delete or forward your email.
- **Mash-ups/Zimlet validation** – Zimbra's mash-up architecture does provide the opportunity to introduce server-side Java code (most Zimlets run on the client side), but the introduction of Zimlets requires server administration privileges. Zimbra recommends only deploying Zimlets that have been certified by Zimbra or else which have been vetted by your security architects internally. The key is that trustworthy Zimlets do not expose additional attack points for malicious users.

The end result is that while an attacker with appropriate security credentials could certainly damage their own mailbox, there is no way for them to compromise other mailboxes or otherwise tamper with the Zimbra Ajax client code other users are running (unless such a hacker manages to obtain administrative access to the ZCS server). (If a particular user's mailbox where to get "hacked", any missing data can be easily restored from the administrator from the continuous back-ups.)

End-to-end encryption. While we recommend TLS to protect Zimbra client/server communications, ZCS is also compatible with client-side encryption technologies, such as existing messaging client plug-ins (such as for Microsoft Outlook or Apple Mail) that implement Secure Multipurpose Internet Mail Extensions (S-MIME) or Pretty Good Privacy (PGP). The advantage of S-MIME or its alternatives is that the message content is encrypted "end to end"—from the machine of the originating sender all the way to the recipient's machine. The downside of end-to-end encryption is that it makes the content of the message opaque to the ZCS server as well as potential hackers. When message contents are opaque to a ZCS server, it prevents Zimbra from running server-side Zimlets and from indexing and searching messages either for individual users or for compliance/cross-mailbox discovery (more on this below).

Most Zimbra deployments instead achieve end-to-end encryption by chaining best-fit encryption technologies together. TLS encryption combines strength with being very light-weight (no client-side key management required), and is hence an ideal fit for protecting client/server communications. For messages bound for the Internet, technologies like S-MIME can instead be used for server to server communications, with the Public Key Infrastructure (PKI) management relegated to the server-side as well. By chaining TLS and S-MIME together, all of the Zimbra server-side features that require access to message contents can be realized and moreover, the total cost of ownership is lower because there is no expensive client-side PKI. Zimbra does not today provide native S-MIME support (it is on the roadmap), but ZCS is compatible with third-party S-MIME solutions that plug-in to the MTA tier (Postfix).

Yet another approach to end-to-end encryption relies requesting or requiring Postfix to use TLS when communicating with other MTAs over the Internet. In this way, while messages may be stored in clear-text on disk, they are private while crossing public or private networks. While this appears to be an even lighter-weight approach to realizing network privacy than S-MIME, *et al.*, the downsides of relying upon TLS end-to-end are that not all MTAs provide TLS today, and more importantly, that this model presumes trust of intermediary MTAs—*i.e.,* there is no standard mechanism to annotate and validate messages to ensure that TLS was indeed used for all network communications. Nevertheless, it is still a good idea to configure the Zimbra MTA to use TLS when communicating with similarly-enabled MTAs: while clearly not full proof, MTA-to-MTA TLS is more secure than clear text.

If the end-user also desires encryption of the contents of a ZCS mailbox while it is stored on disk (for protecting against access from unauthorized users, applications, or scripts that have access to the data center), then we would recommend hardware/OS cryptography that is transparent to the ZCS server: the ZCS server gets assigned a key that provides it access, but data on the disk is opaque to other users/applications. Such hardware-based “crypto” solutions are available for leading network-attached storage platforms, and offer substantial performance advantages over a ZCS software-based encryption on/off of disk (which is not provided today).

10. Compliance and Discovery

The email archiving solutions market is growing explosively with the proliferation of retention and compliance policies, often motivated by the increased regulatory overhead of Sarbanes Oxley (SOX), Health Insurance Portability and Accountability Act (HIPAA), and so on. While email archiving is frequently grouped with more general-purpose archiving and data warehousing solutions (designed for files and databases), the underlying requirements for messaging and collaboration solutions are highly specialized. Moreover, we are convinced that the following factors are going to drive **convergence** between the on-line messaging and collaboration server (for users) and the decision support compliance and discovery system for administrators!

Need for "real-time" compliance. Compliance policies can be essential to the operations of a business or organization: consider the mandatory fiduciary isolation of competing business units or assurance of information privacy in health care. The enforcement of such policies cannot be left to an auxiliary system, but must rather become seamlessly integrated with the core messaging and collaboration solution itself.

Low-overhead of discovery. Many ad-hoc decision support applications must be targeted at data warehouses simply because the performance overhead on the associated on-line system (which are generally not optimized for ad-hoc queries) would be prohibitive. However, for messaging and collaboration systems the aggregate workload of all the decision support processing (*e.g.*, cross-mailbox search, compliance-related discovery) is negligible relative to that of the normal on-line functioning of the messaging and collaboration system.

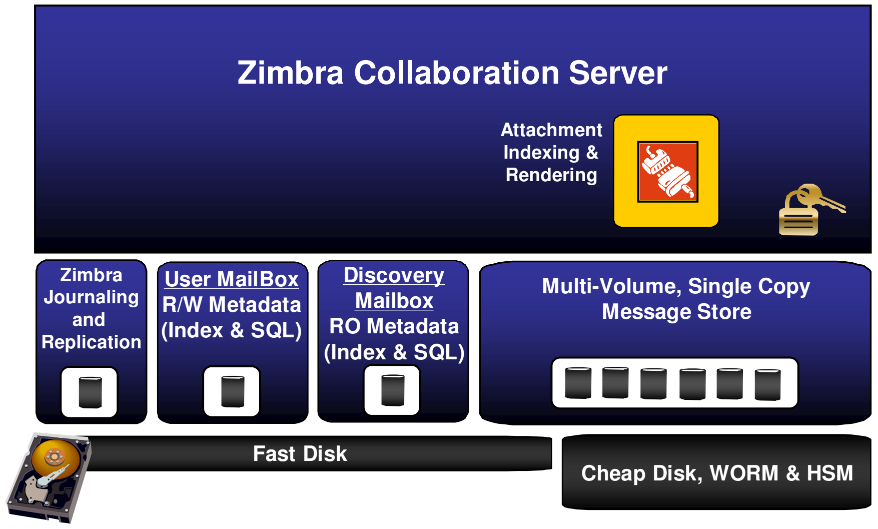
Zimbra Collaboration Server
Attachment Indexing & Rendering
Zimbra Journaling and Replication
User MailBox R/W Metadata (Index & SQL)
Discovery Mailbox RO Metadata (Index & SQL)
Multi-Volume, Single Copy Message Store
Fast Disk
Cheap Disk, WORM & HSM

Figure 7: *Future:* Native Zimbra Compliance Architecture

Single-copy immutable blob storage. The aggregate storage requirements for email/messaging solutions are large and growing quickly (multiple gigabytes per user for information workers). Moreover, email (as well as instant messages/IMs, voicemails, and so on) are write-once, read-many. So it is best to store each message only once (modulo any "implicit" redundancy for RAID, auto back-up, disaster recovery, etc.) rather than pay the cost of maintaining copies in both an on-line (end-user) system as well as an administrator-only archive system. Policies allow the auto-aging of messages from faster spindles to slower ones (such as within hierarchical storage management systems) as well as ensuring that messages/documents are retained as long as necessary but no longer. For regulatory-intensive environments, that storage can be Write-Once, Read-Many (WORM) to ensure that no tampering with content whatsoever could have taken place.

Mutable and immutable meta-data. As we have already remarked, search has become *the* tool for navigating large datasets. Zimbra end-users can search their mailboxes based on virtually any syntactic property of their email, contacts, appointments, documents, and so on. In fact, such arbitrarily rich syntactic search should be available both to individual users (so that they can lay their fingers on the right email as quickly as possible) as well as to administrators (to enable *the* most efficient, accurate cross-mailbox search for discovery and compliance).

The key new insight is that while *one* immutable message store can work for both end-users and administrators doing cross-mailbox discovery, the mailbox meta-data stores (RDBMS and index) must be different, at least for more highly regulated environments. As illustrated in Figure 7, administrators may optionally want to enable a read-only or "pristine" view of all of a particular user's meta-data. This *shadow* or *discovery* mailbox leverages identical ZCS code, but does not allow users (or administrators) to log-in or make updates. Discovery mailboxes are thereby guaranteed to include **all** of the messages, documents, contacts, appointments, and so on, that belong to a particular user for the interval during which this shadow mailbox was enabled.

Our claim: the combination of above factors is already driving convergence between today's disparate on-line messaging and archiving solutions. There is simply insufficient justification for the increased total cost of ownership (TCO) of

- Multiple message stores;
- Multiple meta-data/index stores;
- Multiple query models;
- Multiple server configuration and administration;
- Multiple security models; and
- Multiple scalability and fault tolerance solutions.

Instead, we are convinced that future messaging servers are going to **natively** provide

1. Rich intra-mailbox search based on meta-data and indexing [available in ZCS now];
2. Rich cross-mailbox search & discovery via the same meta-data/indices [available now];
3. Volume and hierarchical storage management (for auto-aging message bodies) [available now];
4. Automated management of read-only or *pristine* user meta-data [ZCS future];
5. Real-time policy enforcement [future]; and
6. Certified write-once/read-many (WORM) hardware integration [future];

Innovation and competition at reducing the TCO for email archiving will be one of the most critical factors in picking the future winners in enterprise messaging and collaboration. In the mean time, of course, it is essential that solutions like ZCS easily integrate with existing email archiving solutions. We in the Zimbra Community believe we have a modest head start in identifying and delivering this convergence, but believe the benefits are sufficiently compelling that the greater market will ultimately follow this path.

11. Compatibility

In addition to integrating with LDAP-compliant directories, Zimbra is focused on providing as much in the way of compatibility with existing enterprise infrastructure as is feasible:

- Support for the broad spectrum of deployed messaging clients, including Outlook via MAPI with support for cached mode, Apple Mail, iCal, and Address Book, Evolution, Thunderbird, and so on;
- Migration tools for moving email, address book, calendar, and so on from existing messaging servers, like Microsoft Exchange, Lotus Domino, or IMAP servers, to Zimbra;
- Easy scripting and standard/REST bindings to the Zimbra Collaboration Suite, to accommodate custom migrations and integrations;
- Single sign-on interoperation with existing security solutions (e.g., Windows Domain Authorization via Active Directory);
- Existing servers and storage systems;
- Future: free/busy sharing with other enterprise collaboration solutions;
- Existing archiving solutions; and, last but not least
- Existing hosted and on-premises anti-spam and anti-virus solutions.

Moreover, the Zimlet framework (documented in depth elsewhere) provides ZCS users with unprecedented customization and integration with other intranet/Internet applications.

12. Conclusions

As we stated at the outset, technical architecture is the foundation for both the existing and future capabilities of any software product. For Internet enterprise messaging and collaboration solutions, we argue you should look for the following:

1. **Optimized use of disk.** For the last couple of decades, processor performance has dramatically outstripped disk I/O performance: Processor performance has been increasing at a rate of 60% per year (Moore's Law), while disk access times (limited, of course, by mechanical movement of the disk arm) have improved far less quickly. Moreover, that limited annual improvement in disk seek times is getting consumed by additional disk capacity (disks are doubling on average each year, so disk performance is lucky to be relatively flat).

   Of course, disk writes are essential for persistence, but there are still email servers that use disk I/O for pipelining interprocess communications! Given this ever widening gap between processor and I/O performance, the only interprocess communication that should leverage disk writes is that which (1) is essential for correctness in the face of failures; or (2) takes place across machine boundaries (such as between the MTA and messaging servers in a typical configuration). Otherwise, the most efficient architecture for enterprise messaging is the use of highly-optimized file systems (for the storage of immutable, byte-oriented messages) and highly-optimized relational databases (for the storage of mutable, record-oriented meta-data). By leveraging such standard, thoroughly-tested code paths, the overall solution can also provide superior caching and compatibility.

Last, but not least: storage may be cheap, but managed storage is not cheap enough not to be concerned about the cost of delivering multi-gigabyte mailboxes to all of a business or service-provider's users. The winning solutions will not only use storage efficiently, but leverage the same storage for archiving and compliance as well as the on-line message and collaboration functions.

2. **Use of and integration with web technologies.** The suite of technologies embodied within the Internet are extremely relevant to enterprise messaging and collaboration. Indeed, email servers will either embrace existing and emerging Internet standards for collaboration or else provide proprietary alternatives to those standards. We encourage betting on Web technologies such as
   - Multi-browser AJAX for rich, zero-administration Web clients and a tightly-integrated Web experience for the user (including multi-site or multi-application content "mash ups");
   - Web security — SSL/TLS, Single sign-on, etc.;
   - VOIP/SIP for unified messaging (UM) interoperation with voicemail and Internet telephony systems;
   - Standards-based "over the air" synchronization with mobile devices (sans additional software installs on those devices);
   - RSS subscriptions to changing information;
   - iCalendar/CalDAV support;
   - Wiki/Ajax authoring overlaid on public folders;
   - XML/SOAP and JSON transport between Ajax client and server;
   - XML/SOAP integration from enterprise applications to messaging server;
   - XML/SOAP integration from the messaging solution to enterprise and Internet applications, which allows the "mashing up" of messaging/collaboration content to supporting applications and services; and
   - Integrated instant messaging (IM) based on XMPP.

3. **Server architecture based on managed code.** Messaging and collaboration increasingly involve the intra-process integration (as per #1) of multiple, largely I/O bound subsystems, often from different sources (including open source subcomponents). At the same time, messaging and collaboration solutions must be better integrated with Internet technologies (as per #2). Managed code environments like Java are key, we believe, to robustly and quickly marrying these requirements.

The greater Zimbra Team and Community thank you for listening!