NinjaMail: the Design of a High-Performance Clustered, Distributed E-mail System

J. Robert von Behren, Steven Czerwinski, Anthony D. Joseph, Eric A. Brewer, and John Kubiatowicz
Computer Science Division
University of California, Berkeley
{jrvb,czerwin,adj,brewer,kubi}@cs.berkeley.edu

Abstract

In today's Internet era, electronic mail is replacing telephony and postal mail as the primary means of communication for hundreds of millions of individuals. Free e-mail services, such as Microsoft's Hotmail and Yahoo's Yahoo! Mail, each have tens of millions of subscribers. However, these and other current e-mail systems unfortunately are not capable of handling the scale of Internet e-mail use, while still providing reliable, high performance and feature-rich services to users. This limitation is the result both of suboptimal use of cluster computing resources, and of highly variable performance of wide-area connections over the Internet. This paper presents NinjaMail, a novel geographically distributed, cluster-based e-mail system built on top of UC Berkeley's Ninja cluster architecture and OceanStore wide-area data storage architecture. NinjaMail is unique in that it uses a collection of clusters distributed through the wide-area to provide users with highly available, scalable and feature-rich services via a wide variety of access methods.

1 Introduction

E-mail functionality has progressed a long way from its original roots of simple text-based access to sophisticated, feature-rich, multimedia access[1]. At the same time, the number of users with e-mail access has grown exponentially. For example, Hotmail alone has over 61 million active users [8].

E-mail systems can be evaluated in several dimensions: feature set or functionality, scale (in terms of users and per-user data storage limits), wide-area performance, data persistence (i.e., explicit or implicit time limits for data storage), consistency model across multiple clients, and fault-tolerance (in terms of server reliability and reachability). Existing systems make tradeoffs along these dimensions. For example, the popular free web-based services support large numbers of users, but they provide limited features and data storage, poor wide-area performance, limited persistence (an indirect side effect of limited storage capabilities), and limited fault-tolerance (overall relatively high for servers, but poor for reachability). To date, systems have not been able to deliver the best aspects of all dimensions, primarily because of limitations with current cluster computing models, and poor tools for dealing with wide-area scalability. Thus, administrators are forced to make difficult tradeoffs.

Our solution to these problems is NinjaMail, a novel distributed cluster-based e-mail system built on top of UC Berkeley's Ninja cluster and OceanStore wide-area data storage architectures. NinjaMail is designed to reliably provide millions of users with high-performance access to a rich set of e-mail handling and storage features.

In the rest of this section, we present a taxonomy of current e-mail systems, in Section 2 we categorize design tradeoffs and discuss the Ninja and OceanStore architectures, in Section 3 we present the design for NinjaMail, followed by a discussion of its implementation and evaluation in Section 4. Finally, in Sections 5 and 6, we present related work and conclusions.

1.1 Current E-mail Systems

Today's e-mail systems range from small office servers with a few hundred users, to server farms that handle web e-mail accounts for the likes of Yahoo! Mail or Hotmail[2]. E-mail server designs are the byproduct of a complicated set of tradeoffs between user access methods, server performance, fault tolerance, and configurability. To better under-

[1] In this paper, we consider the issues associated with user access to e-mail through an endpoint system, and not system to system delivery of e-mail, as addressed by systems and protocols, such as Grapevine [1] and SMTP [9].

[2] http://mail.yahoo.com and http://www.hotmail.com

stand the design decisions that face e-mail system developers, it is useful to categorize e-mail servers according to the ways in which they allow users to access mail. These access methods and the restrictions they impose on a system are described below.

1.1.1 Store and Forward Servers

Historically, most Internet mail servers have offered only store and forward mail service[3]. The canonical example is a Post Office Protocol (POP) [6] server, in which messages are kept on the server only until the user first accesses and downloads them[4].

This simple design offers two advantages. It simplifies performance prediction (as discussed in [10]), since each message has a limited lifetime on the server and goes through very little processing. Additionally, since users download and store messages on their local machines, they are able to access old messages even when the mail server is unavailable due to a server failure or a network partition.

Unfortunately, storing messages on the client machine has a number of undesirable side effects. When users access the server from multiple client machines, they must manage and administer inconsistent views of their separate local mail repositories. Furthermore, the administrators of those machines must be separately responsible for backing up and restoring any important messages. When these machines are laptops or home computers, this choice really means that users' mail repositories are not backed up at all, and would be completely lost if the client machine were to fail or be stolen. Another disadvantage of store and forward systems is that all of the interesting mail operations (such as searching, automatic filing, etc.) are performed on the client machine after messages have been downloaded, placing a heavy computational burden on the machines of users who have large mail repositories. Finally, store and forward servers do not provide users any way to access new messages if they are separated from the mail server by a network partition.

1.1.2 Server-Only Mail Repositories

A clear alternative to store and forward message access is to simply keep all of the messages on the server. This is the approach used by web based mail servers, such as Hotmail and Yahoo! Mail, and by traditional enterprise mail severs, such as Microsoft Exchange and Lotus cc:Mail. These servers provide a central repository for a user's mail, which can be accessed from many different client machines.

[3]Note that store and forward e-mail service is an end user service and thus is different from store and forward delivery of e-mail between servers (e.g., using SMTP [9]).

[4]Although the POP protocol does allow for messages to be left on the server, access to older messages on the server is very cumbersome.

This approach offers several advantages over store and forward mail servers. Firstly, by accessing all mail through the server, users always have a consistent view of their mail repository from any client machine. Secondly, central management of the message repository enables centralized backup or replication of data by the e-mail service provider. A centralized mail repository also enables the addition of features (full text indexing and searching, for example) that may be difficult for the client to perform itself. This capability is particularly useful for supporting small clients, such as PDAs. Finally, server-only repositories have the opportunity to provide other features — such as pager notifications for important messages — that would be impossible from a disconnected client.

The drawbacks of server-only mail repositories are twofold. Firstly, they are heavily dependent on the network connection between the client and the server, and a failure of this connection will prevent users from accessing either new or old mail. Similarly, poor client-server network performance renders these systems virtually unusable. Secondly, most of the work is performed by the server, making it difficult to design a server-only repository that can scale to large numbers of users and still provide a wide variety of features. For example, office e-mail servers such as Exchange provide rich feature sets at the expense of scalability. At the opposite extreme, web-based e-mail systems such as Yahoo! Mail optimize for scalability at the expense of feature richness.

1.1.3 Client-Side Caching Systems

The final option employed by present-day email systems is to combine a client-side message cache with a permanent message repository on the server, the approach taken by Internet Message Access Protocol (IMAP) [2] servers[5].

Client-side caching systems capture many of the advantages of both store and forward systems and server-only mail repositories. Like store and forward systems, client-side caching systems support disconnected access to mail. Like server-only repositories, they provide users with a consistent message store view from different clients, can be centrally managed and backed up, and can include sophisticated server-side features, such as full text searching.

The primary disadvantage of these systems is the complex client-server synchronization protocol required to support disconnected clients, which increases development time and may introduce bugs. Additionally, the overhead of synchronization operations may reduce runtime perfor-

[5]Hotmail is currently offering a beta service that supports client-side caching. A quick calculation based on Hotmail's per-user storage limits and an estimate of how many messages users receive per day indicates that Hotmail users can probably only store about 3 months worth of mail. This storage limitation forces users to use Hotmail as a store and forward service.

mance. Finally, client-side caching systems inherit the performance tradeoff of server-only repositories: the more features that are added to the server, the smaller the user population that can be supported.

2 NinjaMail Overview

NinjaMail is designed to address two of the main shortcomings of present-day e-mail systems: the tension between scalability and features, and the inability to deal well with problems in the wide-area network. The single-cluster NinjaMail architecture provides a scalable and fault tolerant service, without sacrificing user features or limiting the available mail access modes. This offers significant advantages over current single-cluster mail systems, in the number of features that can be supported. Additionally, with the wide-area features discussed in Section 3.2, NinjaMail can provide both high availability and performance, even in the face of poor wide-area network performance or network partitions. The differences between NinjaMail and present systems are highlighted in Figure 1.

Server Type \ Properties		Scalable	Feature Rich	Data Persistance	Consistent View Across Clients	Tolerate WAN Failures
Store & Forward (eg POP)		Yes	No	No	No	Yes (old mail only)
Server Only	Hotmail	Yes	No	2MB limit	Yes	No
	Exchange	No	Yes	Yes	Yes	No
Client Cache (eg IMAP)		No	Yes	Yes	Yes (eventual consistency)	Yes (old mail only)
NinjaMail		Yes	Yes	Yes	Yes (eventual consistency)	Yes

Figure 1. E-mail system comparison

Scaling well within a single cluster presents a number of interesting challenges, making it no accident that current e-mail systems (and other systems) tend to trade featurerichness for scalability. Providing service to a large user pool places extreme demands on all hardware and software components of a system, including CPUs, disk systems, networks, and data consistency modules. Current systems limit the number of supported features to help reduce their overall complexity. The primary design challenge for NinjaMail was to find a way to effectively manage this complexity.

Providing highly available services in the face of widearea failures also poses significant technical challenges. For example, allowing modifications to a user's message store during network partitions introduces the need for data synchronization, an especially complex requirement when multiple parties can simultaneously modify a user's message store.

NinjaMail addresses these problems by leveraging technology being developed at UC Berkeley, using Ninja's cluster computing architecture to create a local-area high performance mail server, and OceanStore's storage management architecture to connect multiple Ninja clusters across the wide-area. The tools provided by Ninja and OceanStore are discussed below.

2.1 UC Berkeley's Ninja Project

The Ninja project is developing a cluster-based platform for highly scalable, available, and fault-tolerant Internet services (http://ninja.cs.berkeley.edu). NinjaMail builds on top of the Ninja architecture (see Figure 2), leveraging its scalability, fault-tolerance, and cluster maintenance, and freeing the NinjaMail designers to focus on building a feature-rich e-mail service.

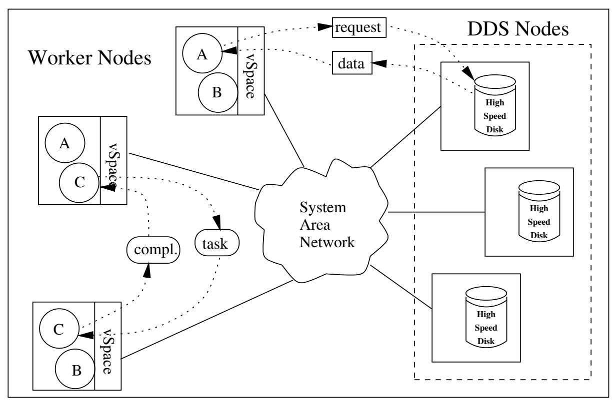

Figure 2. Ninja Architecture Overview

Persistent storage in Ninja is provided by Distributed Data Structures (DDSs), which transparently stripe and replicate data over subsets of the cluster's nodes. The DDSs solve the complications of cluster-wide data replication and synchronization once, and provide a clean persistent storage abstraction, both for metadata used by the Ninja infrastructure and for services built on top of Ninja. The initial Ninja DDS, in the form of a distributed hash table, has undergone extensive testing, performance analysis, and optimization [4].

Ninja is implemented in Java, and each node (or in some cases each CPU) runs a single JVM that houses the management infrastructure unit (a vSpace), and all of the applications on that node. Distributed applications are constructed of "worker" objects, similar to the approach used in [3]. Ninja automatically starts and stops workers on different nodes, both to provide load balancing and fault tolerance.

Ninja uses an asynchronous task/completion messaging layer for communication among workers and between workers and the distributed data structures. Workers are written as single-threaded finite state machines that respond to messages (tasks and completions) passed to them by the infrastructure. The lack of blocking calls within the workers allows for a high degree of concurrency without a large number of threads. This structure has been shown to be very effective at improving throughput in systems such as web servers that must serve a large number of requests per second [5, 7].

2.2 OceanStore

The OceanStore project is developing a global-scale information storage and retrieval utility (http://oceanstore.cs.berkeley.edu). Data placed in the OceanStore is replicated across multiple servers. This provides data persistence in the face of failures of individual OceanStore nodes — a feature particularly important for valuable email data. To protect the security of this data, it is stored in an encrypted format, such that only the owner of the data (or a trusted agent possessing the owner's key) can decrypt it.

OceanStore provides applications with a set of standard data access APIs. These APIs allow typical actions such as read and write, as well as more specialized access modes such as atomic file append operations. The OceanStore infrastructure handles wide-area replication and conflict resolution to provide eventual consistency for data that is modified from widely distributed parts of the OceanStore. Additionally, OceanStore can automatically migrate data close to clients that are currently accessing it, to avoid the latencies associated with accessing data across a wide-area link. This data migration also improves overall data availability, since "nearby" data is less likely to become unavailable due to a network partition.

NinjaMail builds on OceanStore to provide an effective mechanism for distributing a user's email data across the wide-area. This allows for improved access to the email service from anywhere in the globe, even in the face of slow or partitioned networks.

3 NinjaMail Design

For our initial explorations, we have designed a single-cluster version of NinjaMail. This will allow us to evaluate the performance of the Ninja cluster architecture on email processing workloads, as well as to explore the consistency requirements of email transactions. Additionally, we have begun the design of a wide-area email service, based on components of both Ninja and OceanStore. This wide-area system addresses some of the difficulties of wide-area network performance, and further improves scalability by distributing email computation around the globe, rather than across a single cluster.

3.1 Single-Cluster Email Services

Within a cluster, NinjaMail is composed of several (potentially replicated and distributed) modules that communicate via simple APIs (see Figure 3). At NinjaMail's core, the MailStore module handles storage operations, such as saving and retrieving messages, updating message metadata, and performing simple per-user message metadata searches. A message's metadata represents its mutable attributes (folder and flag information) and significant headers. Access modules support specific communication methods between users and NinjaMail, including an SMTP module for pushing messages into the MailStore and POP, IMAP and HTML modules for user message access. NinjaMail's extension modules enable the addition of features, such as message filters or message body searching.

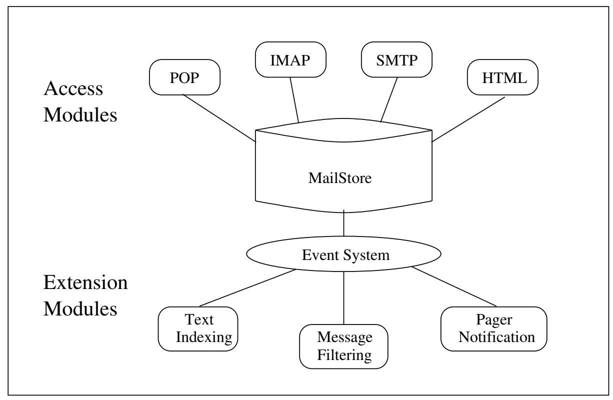

Figure 3. NinjaMail cluster design

3.1.1 The MailStore Module

The MailStore module controls all changes to a user's message repository, using the following operations:

- addMessage() - commits a message and associated metadata to a Ninja DDS.
- getMessage() - retrieves a message from the DDS
- getMessageIDs() - retrieves a list of IDs for messages that match a search specification. For example, message folders are built on the fly, by searching for all messages with the "folder" tag set to a particular value.
- updateMessage() - updates a message's metadata.
- deleteMessage() - remove a message and its associated metadata from the DDS.

Note that no message modification controls are provided. All additional information about a message should be added to its metadata.

NinjaMail's local-area scalability is based upon the MailStore's distribution model. To avoid conflicts during data updates and to make maximum use of data caching, the MailStore routes all traffic for a given user through the same "master node". This choice greatly simplifies the design by eliminating race conditions on message index updates. Moreover, the mapping between users and master nodes can be changed dynamically, avoiding potential hot spots that might occur with a static assignment of users to nodes, and providing users with fault tolerance if their current master node fails. Messages are stored in a distributed hash table, and are retrieved by IDs that are generated when the messages are first added to the MailStore.

Metadata operations use an in-memory database. When a user first interacts with the NinjaMail cluster, the metadata index for all of their messages is loaded into the memory of their current master node[6]. This approach enables fast searching through the user's message repository. Metadata updates are applied first to the in-memory database, then flushed individually to the DDS. A periodic checkpoint flushes the entire database to the DDS, reducing the number of updates that need to be re-applied to recover from a crash.

3.1.2 Access Modules

Access modules enable clients to interact with the MailStore using standard Internet protocols. The SMTP module accepts incoming messages and adds them to the MailStore. The POP and IMAP modules allow users to interact with the MailStore through existing e-mail client software, such as Eudora or PINE. Finally, the HTML interface provides web access to a user's mail, in a similar style to Hotmail or Yahoo! Mail.

3.1.3 Extension Modules

To add new functionality, NinjaMail will use a flexible event notification system currently being developed for the Ninja environment. When the MailStore performs actions such as adding a new message or modifying message metadata, it posts an event to the event notification system. Extension modules subscribe to event types of interest, and take action accordingly when they receive events. For example, to provide full-text searching, an indexing module would subscribe to message addition events. When the module later receives such an event, it retrieves the associated message body and indexes it. Other components, such as the IMAP and HTML modules, would interact with the indexing module to provide search capability to end users.

[6]MailStore performance could be improved by loading only the most likely to be accessed message indices (e.g., those in the user's inbox), decreasing the memory requirements of the index in the common case. Additional indices would then be brought into memory as required by the user's actions.

3.2 Extending to the Wide-Area

Our design for a wide-area email system combines elements of a single-cluster NinjaMail service with the wide-area data replication and storage facilities of OceanStore. The central theme of our model is that all data interactions between clients and servers, or between different mail servers should take place via the OceanStore infrastructure, rather than through specialized protocols such as SMTP, POP, IMAP, etc. This is illustrated in Figure 4.

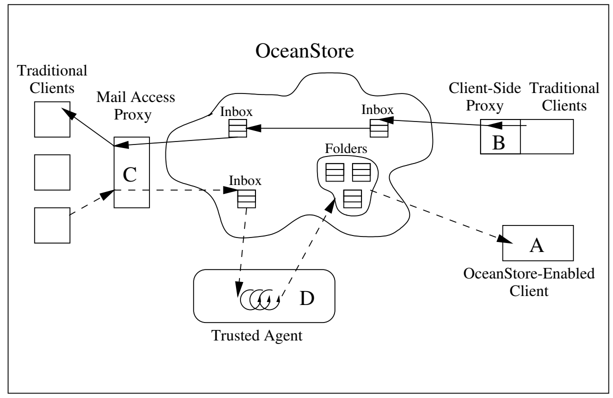

Figure 4. Wide-area Mail Architecture

Sending a message in this architecture simply entails appending that message to a user's mail inbox file in the OceanStore. The inbox file is automatically transfered across the wide-area through OceanStore's replication mechanisms, where it can be read locally by the user's email client software (A in Figure 4). Traditional POP, IMAP, and SMTP interactions can be supported by either client-side or infrastructure proxies to the OceanStore (B and C). Trusted agents (D) in the infrastructure can perform message filtering and refiling, pager notifications, and other actions on behalf of users.

NinjaMail's modularity lends itself quite naturally to this architecture. Because all message storage operations occur in the MailStore module, only this module needs to interact with the OceanStore. NinjaMail's access modules map directly to the protocol proxies depicted in Figure 4. Similarly, the extension modules and event model discussed in Section 3.1.3 can operate as trusted infrastructure services, to perform operations on messages as they are added to a user's inbox, or as they are filed into other folders.

4 Implementation and Evaluation

We have implemented an initial single node version of NinjaMail and are currently working on a clustered (based upon Ninja version 2) and distributed (based upon OceanStore) implementation. Both Ninja v2 and

OceanStore are still under development, hence development of NinjaMail components is proceeding in parallel with these efforts. Thus, our performance experiments rely on the single node design combined with preliminary performance numbers for Ninja v2.

Ninja has been implemented entirely in Java, as has NinjaMail. Our test cluster consists of 64 Linux server machines (2 500MHz Pentium III CPUs, fast SCSI disks, and 512 MB of memory), interconnected via a switched 100 Megabit Ethernet network.

NinjaMail's maximum performance is constrained by three limiting factors: CPU performance, the cluster's internode communication bandwidth, and disk bandwidth / latency. By appropriately partitioning user data across clusters, most e-mail workloads will be cacheable within a cluster's 32 GB of memory, eliminating the influence of disk characteristics. Distributed hash table performance tests show the impact of the other limiting factors:

- For get and put operations under 2500 bytes, the maximum per-node throughput is 1000 requests/second for gets and 250 requests/second for puts. These numbers remain roughly constant as the size of the cluster is varied, showing that the limiting factor in this case is CPU time; not network bandwidth.
- For reads and writes above 2500 bytes, the byte copying overhead begins to adversely limit throughput. Above 8000 bytes, performance is limited by each node's link.

The throughput of small request/response pairs between Ninja workers is comparable to small read hash table performance. Hence, inter-worker communication will likely be CPU limited to 1000 requests/second per node.

4.1 Message Throughput

As a sanity check of our design, we calculated NinjaMail's performance as a store and forward mail server, where there are essentially two stages in a message's lifecycle. Firstly, the system receives the message and commits it to stable storage. Secondly, sometime later, the user connects to the system and downloads the message. In NinjaMail, this requires the following operations:

- Hash table write for the message as it is received.
- Inter-worker communication to send the message metadata to the user's current master worker.
- Hash table write to commit metadata to stable storage.
- Hash table read to fetch the message metadata.
- Hash table read to fetch the message.
- Hash table write to save modified metadata (i.e., to mark the message as read).

As in [10], we assume an average message size of 4.7 KB. Since the size is less than 8000 bytes, performance will be CPU limited (due to the time required to set up and process messages). Thus, using our observed performance numbers, we expect per-node performance of approximately 52 store/retrieve cycles/second. To put this into perspective, in March 1999, Yahoo! Mail served 3.6 billion mail messages to its 45 million users [11]. At 52 messages/second, NinjaMail would be able to handle this load with a 27-node cluster. More conservatively, a slightly larger cluster (e.g., 35 nodes) would likely provide good service for non-uniform loads. Preliminary tests show that Ninja clusters scale well up to at least 64 nodes; thus, a single cluster should be able to handle store and forward services for nearly 100 million users.

Since the full NinjaMail system will support a richer set of user interactions than a simple store and forward system, we expect final system throughput to be lower, based largely on how users interact with the system. Conservatively, assume that each message is viewed an average of 2 times. Thus, the system should be able to handle around 38 messages/second per node, requiring at least 36 nodes to handle the Yahoo! Mail traffic mentioned above. Since extension modules such as SPAM filtering could cause many messages to be viewed zero times, actual performance could be better. However, the actual usage patterns remain to be seen.

4.2 System Responsiveness

At maximum throughput levels, the hash table get/put latencies are around 2 milliseconds (ms) and 4 ms respectively. Since client to NinjaMail latencies will likely be greater than 20 ms, multiple inter-cluster requests will not significantly affect overall client latency. In particular, the additional hash table lookup overhead for message reading should be at most 15 ms.

4.3 Wide-Area Scalability

With a large enough set of widely distributed nodes, we expect the OceanStore to easily handle the replication and distribution of petabytes of data. This is enough storage to allow every person on the planet hundreds of megabytes of email data. Moreover, this data is automatically backed up by the OceanStore's wide-area data replication scheme.

By geographically distributing multiple Ninja clusters to act both as access points for traditional email client software, and as user agents to perform filtering, and other functions, we can effectively create a widely distributed email system. Since the email activities of individual users are largely independent of one another, we expect these services to scale well as more Ninja clusters are added to the system. This should allow NinjaMail and OceanStore to provide a truly global-scale email service.

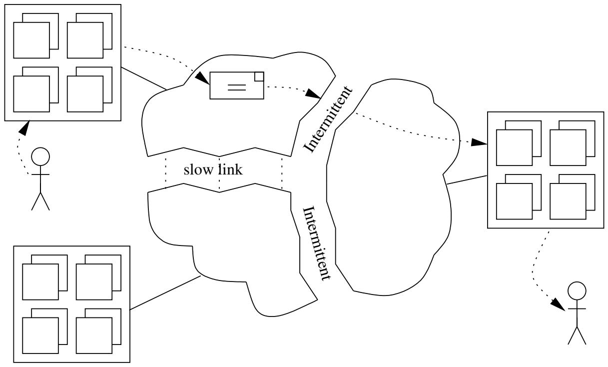

Figure 5. NinjaMail - 1000ft view

4.4 Tolerance of Wide-Area Network Problems

The conventional solution to mail access during intermittent network partitions is support for disconnected operation, allowing users to access any locally cached messages, but not providing access to new messages. In particular, since all messages must go through a central server, even new messages between users *in the same network partition as each other* will be inaccessible until the network is back up. Similarly, poor wide-area network performance (e.g., high roundtrip network latencies) may render the user's mail service useless.

Combining Ninja's high-performance cluster-based computing with OceanStore's wide-area data replication and migration creates a powerful synergy. During a network partition, if a user can reach *any* server, they will be able to use OceanStore's distributed data location service to locate and access new messages that are generated from within the same partition. Additionally, in an intermittently connected network, new messages are continuously being propagated to the user's current location. Thus, even if the network is partitioned when the user next accesses their mail, they will still have access to messages created prior to or during the time the network was available. Finally, data migration reduces wide-area network latencies by enabling users to always communicate with a mail server that is "close" to them in the network topology. Because OceanStore is designed to migrate data dynamically as the user's access patterns and locations change, NinjaMail will provide high performance even for users who change location frequently.

5 Related Work

There is very little current research about e-mail systems. The one exception we are aware of is the University of Washington's Porcupine, a scalable e-mail system, based at present on store and forward functionality. Porcupine uses a single cluster model and is built primarily of off the shelf components, simplifying the initial development. NinjaMail, however, is built on Ninja and OceanStore platforms, decisions that we believe will provide better performance and extensibility. We are not aware of any distributed cluster-based e-mail systems.

6 Conclusion

In this paper, we have presented a design for a novel two-tiered distributed e-mail infrastructure. At the system area cluster level, we use concepts from the Ninja v2 system in order to improve scalability and fault tolerance. In the wide-area, we will use OceanStore in order to propagate data between these clusters for improved fault tolerance and to migrate data closer to the client applications. We believe this two tier model will enable us to create an email infrastructure that will better support planet sized email populations. Additionally, we will be able to incorporate richer functionality through our event model, allowing a wide variety of applications to tie into the email system for such features as mail notification and filtering.

Currently, this system is still in the early stages of development. Most of the base infrastructure for the cluster area system has been created. Over the next several months we will be incorporating the OceanStore system, as its development proceeds.

References

[1] A. Birrell, R. Levin, R. Needham, and M. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25:260–274, 1982.

[2] M. Crispin. *Internet Message Access Protocol — Version 4rev1*. Internet RFC 2060, Dec. 1996.

[3] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Cluster-based scalable network services. In *Proc. of the 16th Symposium on Operating Systems Principles (SOSP)*, St-Malo, France, Oct. 1997.

[4] S. Gribble, April 2000. http://www/~gribble/distrib_hash/index.html.

[5] M. F. Kaashoek, D. Engler, G. Ganger, and D. Wallach. Server operating systems. In *SIGOPS European Workshop*, pages 141–148, September 1996.

[6] J. Myers and M. Rose. *Post Office Protocol — Version 3*. Internet RFC 1939, May 1996.

[7] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the Annual USENIX Technical Conference*, June 1999.

[8] PC World Communications, April 2000. http://www.pcworld.com/pcwtoday/article/0,1510,16045+1+0,00.html.

[9] J. Postel. *Simple Mail Transfer Protocol*. Internet RFC 821, Aug. 1982.

[10] Y. Saito, B. Bershad, and H. Levy. Manageability, availability and performance in porcupine: A highly scalable internet mail service. In *Proc. of the 17th Symposium on Operat-*

ing Systems Principles (SOSP), Kiawah Island Resort, South Carolina, Dec. 1999.

[11] Yahoo! Inc., April 2000. http://docs.yahoo.com/docs/pr/1q00pr.html.