# The Porcupine Scalable Mail Server

Yasushi Saito, Eric Hoffman, Brian Bershad, Henry Levy, David Becker
Department of Computer Science and Engineering
University of Washington
and
Bertil Folliot
Laboratoire d'Informatique de Paris 6, CNRS

porcupine@cs.washington.edu

## Abstract

This paper describes the design and preliminary performance of the Porcupine mail server, a cluster-based mail server that can handle up to 1 billion messages a day. Unlike common large-scale mail servers deployed today, there is no role separation among nodes. Each node in the cluster runs all the services supported by the cluster and balances the workload dynamically using the cluster membership information. This architecture is more available, manageable, and scalable than traditional architecture.

## 1 Introduction

Electronic mail service is one of the most valuable services offered by the Internet. E-mail traffic will continue to increase in the near future, due to the explosion in the number of users on the Internet and the increase in the number of messages each user receives. At present, sites such as AOL and Hotmail handle 10 million mail messages per day. Such giant email clearinghouses will become commonplace, and the amount of mail each of them handles will also grow in the future.

The goal of this project is to design and build an email server, called *Porcupine*, that can handle up to one billion messages a day from 10 million users, using a cluster of up to 1000 PCs. Porcupine takes advantage of today's cheap PCs and fast interconnects to achieve this goal without massive hardware or management expense. Porcupine treats the set of interconnected machines (the *cluster*) as one large server. Unlike traditional mail architectures, each node in the cluster performs a portion of all the functions that the cluster offers. For example, the mail user database is partitioned among nodes. User mailboxes are also distributed, and user messages can be stored on any node at any time. This design allows for high availability, high flexibility, and better load balancing.

In the next section, we describe the typical configuration deployed by the existing mail servers and point out its problems. In Section 3, we describe the architecture of the Porcupine mail server. Section 4 describes the system's current implementation status and studies the performance of the system using a simulator. Section 5 concludes.

## 2 Problems with Existing Large-Volume Mail Servers

Figure 1 shows the typical configuration of large-scale mail servers [Christenson 97], [DeRoest96]. SMTP and POP servers run on front-end nodes. Messages are stored on a distributed file system such as NFS, and all the details of the cluster architecture are hidden behind the file system. In addition, a database server is used to manage user information such as names, passwords, and aliases. The advantage of this design is that it is easy to install, since it uses only the off-the-shelf components. However, there are several disadvantages, as described below.

**Hardware Expense**

This architecture requires a large number of dedicated computers for storage and user database management. The computers tend to be expensive, especially if high availability and reliability are required.

**Management Expense**

One of the bigger problems is the static architecture. For example, a fixed set of users is typically assigned to each file server. Adding or removing a file server node thus requires changes to the user database and massive movement of user messages.

**Performance Problems**

Existing mail servers are slow for two main reasons. One is structure of the server software itself; e.g., sendmail [Sendmail 98], used in most of these systems, performs multiple process forks and message spools that can be avoided through use of modern operating system functions such as multi-threading. Second is that SMTP and POP (or IMAP) servers rely on file system mechanisms for mutual exclusion, because the file system is their only point of communication. For example, they use lock files to assert locks, and file creation and deletion is known to be slow.

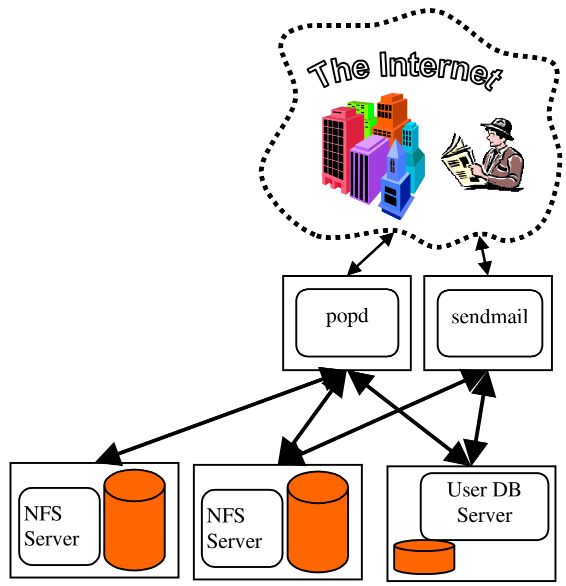


**Figure 1. Typical configuration**. *The system consists of a set of front-end nodes that run sendmail and popd, plus distributed file system servers and a user database server.*

# 3 Overview of the Porcupine Mail Server

Porcupine is a mail server designed to be scalable enough to handle one billion messages per day to ten million users, using up to 1000 PCs. It provides SMTP, POP and IMAP [IMAP98] sockets to Internet clients using DNS A and MX records.

Figure 2 shows the architecture of the Porcupine mail server. Unlike the conventional architecture, there is no role separation among nodes - each node in Porcupine is responsible for SMTP, POP, IMAP sessions, mail storage, and user database management. Each node manages a disjoint subset of users. User mailboxes are also distributed over nodes, and each node may store a portion of a user's mailbox. The user's mail data is collected and presented to the user when the user reads her mailboxes via POP or IMAP.

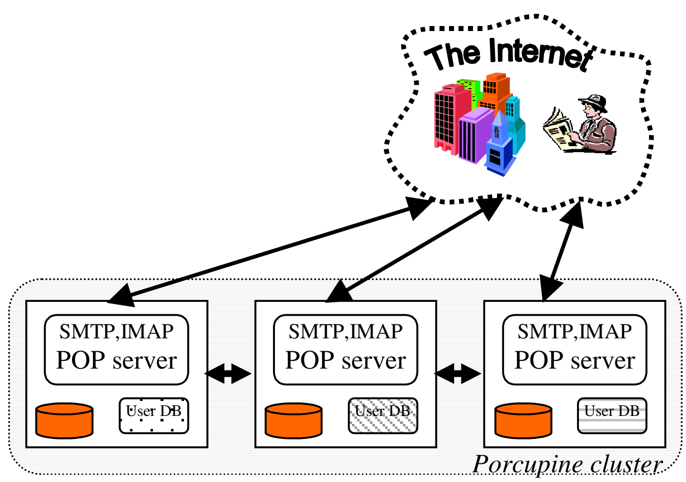


**Figure 2. Porcupine architecture.** *Porcupine cluster is symmetric.*

## 3.1 User Database Partition

The user database is partitioned by grouping users into a fixed number of equivalence classes using a hash function. A small in-memory table called the *user directory* is replicated on every node. The user directory is used to map a hashed user name to the ID of a node that manages the user names hashing to in the same bucket. In the initial implementation of Porcupine, we assume the existence of a separate centralized server, such as an NIS server, that manages user information. Each node's user database is bootstrapped from the central database. This central user database creates an availability problem, but not a scalability problem,

since the central database is used only to bootstrap the porcupine user database. Once the porcupine database is filled, the central database is no longer consulted. In the future, we plan to replicate and distribute the boot database so that we do not have to rely on a central server.
When a node fails or recovers, a distributed membership protocol [Christian95] runs to reach agreement on the cluster node set. Next, one of the live nodes (the leader) creates the new user directory by assigning an equal number of buckets to live nodes. Finally, the leader broadcasts the new user directory to the live nodes.

## 3.2 Message Distribution

A user's messages can be stored on any node at any time. To keep track of which messages are stored on which nodes, each user database entry contains a field called "*nodes,*" which maps a mailbox name to the list of node IDs that store messages belonging to the mailbox.

### 3.2.1 Message Reception and Retrieval

When a remote SMTP server wants to send a message to Porcupine, it selects a node by looking up the DNS A and MX records for the site; it then begins an SMTP session with the SMTP server on that node. When the recipient for the message is known, the Porcupine SMTP server contacts the user database server for the recipient and verifies that the user name is valid. Next, Porcupine picks a node to store the message using the following strategy. If there exists a node that already stores messages for the recipient, the new message is appended on that node; in this case the amount of disk I/O is minimized since there is no need to create a mailbox file, and mailbox reading and updating also become efficient since smaller number of files need to be opened . Otherwise, the SMTP server stores the message itself. In the future, we plan to investigate a scheme based on dynamic measures such as CPU load or disk usage. Finally, the message is stored on the node chosen, and ID of the node is added to the recipient's database entry.
When a user requests a mail retrieval using a POP or IMAP server, the server contacts the user database for the user and obtains the list of node IDs for the user's mailbox. Porcupine then initiates a parallel RPC to all the nodes in the list to fetch messages. The node list in the user database is updated if the mailbox fragment is emptied from any of the nodes.

### 3.2.2 Failure Recovery

The "nodes" field in the user database is only in the memory of the user database server. Thus, when a node crashes, all the information managed by the user database server on that node becomes invalid. After the user database repartition, a broadcast is performed to locate the user's mailboxes for user database buckets that moved. All of the mailboxes for the users that are contained in the bucket are searched at each node that receives the broadcast. This strategy balances the broadcast overhead and the disk scanning overhead.

## 3.3 Management Issues

One of the potential problems with Porcupine is administration, because the user database and mailboxes are scattered among nodes and the administrator may have difficulty relating the many fragments. Porcupine addresses this problem by providing a single-file-system view using an NFS gateway. The NFS gateway is a stand-alone daemon that maps user mailboxes and the database into a single file system via the NFS protocol. All the mailboxes belonging to a user appear in a single directory, and the user database also appears as a set of files. The administrator can use the standard operating system tools to inspect and possibly modify the system.

## 3.4 Advantages of Porcupine Architecture

**Fairer Availability Model**

If a node fails, some portion of all the users' messages become unavailable. However, all the messages that arrive after the failure are available to all the users. This is important, because new incoming mail has higher value than mail that has been read. On the other hand, in the traditional design, when a file server crashes, all the mailboxes on that server become unavailable. Users that rely on the server cannot even read new messages.

**Better Load Balancing**

Since a user's messages can be stored anywhere, work can be balanced based on any dynamic measure, such as current CPU load or current disk usage. Thus, Porcupine can better utilize the hardware capacity of the cluster.

**More Flexible Configuration**

Since there is no role separation among nodes and work distribution is always determined dynamically, adding or deleting nodes from the cluster requires only an update to DNS. (This assumes the central user database server; without it, a portion of the user database must be copied to the new node.)

**Efficient Mailing List Handling via Single-instance Store**

Mailing list messages comprise a large portion of today's email traffic. Porcupine can efficiently handle such messages by creating a single file instance for a message and creating links to all the recipients. This is more difficult and inefficient to do in a conventional architecture based on a distributed file system, because the application must maintain links manually.

# 4 Performance

We are implementing Porcupine on a cluster of PCs connected by Myrinet. We use Linux and pthreads as the development platform. Basic functions (SMTP, POP, IMAP, and NFS gateway) are implemented, but failure recovery support is still primitive. The system is currently under trial use. Because development is still underway, we show the scalability characteristics of the system through simulation, using measurements of the prototype to parameterize the simulator.

The porcupine architecture is flexible and is intended to tolerate various failures without human intervention. However, the downside is a potentially high cost of recovery procedures, since recovery involves broadcasts to locate mailbox fragments. In this section, we study the performance of the system under various circumstances and argue that the effect of failures on system performance is smaller than one might think. Two scenarios are simulated.

**Steady state** Throughput is measured when there is no failure. This simulation studies the scalability of the system.

**Network partition** The system starts from a steady state where all nodes have the user database accessible. Then, in the first second, half the nodes are cut off from the cluster and they recover after 2 ½ minutes.

A discrete event simulator was run with a synthetic input consisting of, on average, 25 SMTP requests per node per second and 10 POP requests per node per second. The mean message size was 5k bytes, and the number of users was 10,000 per node (the actual recipient distribution was fitted from the trace data, which resembles a negative exponential distribution).

These parameters were extracted from an email activity trace in our departmental mail server. Performance parameters, such as protocol processing overhead, model Linux running on Pentium Pro 200MHz PC. Each node refuses a connection if its current CPU load exceeds 80%. Only the requests that run through completion are counted (SMTP and POP sessions may abort in the middle if a node involved in the session crashes).

## 4.1 Steady State Scalability

Figure 3 shows the scalability of the cluster as a function of the number of nodes. The system scales perfectly and handles our goal of 1 billion mails per day with 650 nodes. This is understandable, because there is no single bottleneck in the system. Note that in the simulation, we assume that the network has infinite capacity, i.e., each node can push any amount of data without dropping it. In reality, this assumption may not hold. The bi-section bandwidth required in a 650-node cluster is about 1Gb/second.

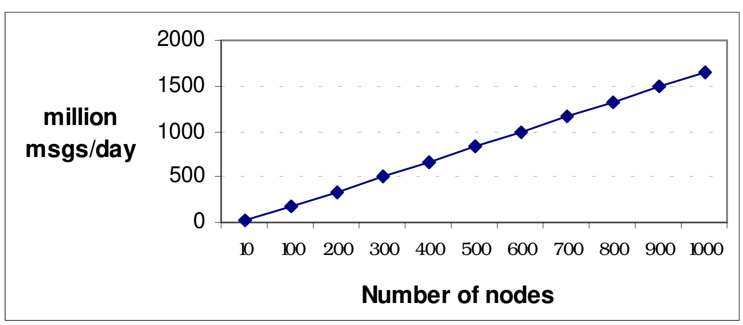


**Figure 3**. **Porcupine scalability**. *The throughput of the system scales as the number of nodes grows. The system can handle 1 billion messages/day with about 650 nodes.*

### 4.2 Network Partition

Figure 4 shows the system behavior on cluster partition. Half of the nodes in the cluster are cut off from the network in the first second and recover 2 ½ minutes later. The figure shows that the throughput drops to half during the 10-second period following partition and recovery. During this period, the membership protocol runs and the SMTP and POP servers run with the old database contents. Thus, about half of them abort. Then the system throughput drops to nearly 0 because of the broadcast storm caused by user database reconstruction. However, the throughput quickly recovers after about 10 to 20 seconds. By extrapolating from the recovery overhead shown in Figure 4, it is shown that the system can operate in about 90% of its capacity when a partition happens every hour, and that it can operate in 99% when a partition happens every three hours. Today's network is reliable enough to limit partition to every few days. Thus, we think that partition recovery overhead is negligible in Porcupine.

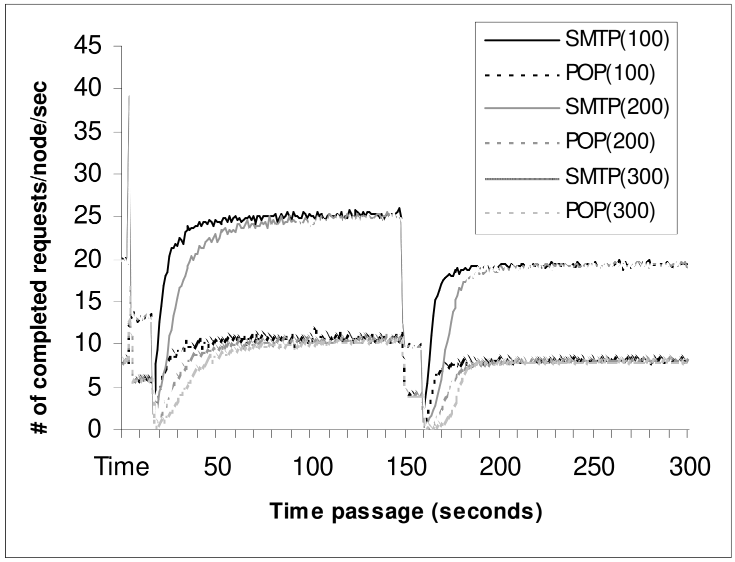


**Figure 4. System behavior upon partition**. *This picture shows the system throughput transition when half of the nodes die at time 1 and recover at time 150. The throughput drops temporarily because of broadcast messages, but it quickly recovers after 10 to 20 seconds. "SMTP(200)" in the legend means the SMTP throughput in a 200-node cluster.*

## 5 Conclusions and Future Plans

This paper described Porcupine, a clustered scalable mail server. Porcupine is designed to be cheap, scalable, fast and easy to manage. The simulation study shows that the system is indeed scalable, and failure recovery is not costly. The system is being implemented on a cluster of Linux PCs.

In the next phase of the project, we plan to replicate both the user database and user mailboxes. Replication algorithms are still being designed. In addition, we plan to introduce a recipient-specific mail processing language used for junk mail rejection or mailing list processing [Showalter98].

More information about Porcupine can be found at `http://porcupine.cs.washington.edu`.

## Reference


**[Christenson97]** Nick Christenson, et al, "A Highly Scalable Electronic Mail Service Using Open Systems", USENIX Internet Technology Symposium, Dec 1997.

**[Christian95]** Flaviu Christian and Frank Schmuck, "Agreeing on Processor Group Membership in Asynchronous Distributed Systems", Technical Report CSE95-428, Univ. of California at San Diego.

**[DeRoest96]** James DeRoest, "Clusters Help Allocate Computing Resources", http://www.washington.edu/tech_home/windows/issue18/clusters.html.

**[IMAP98]** IMAP home page, http://www.imap.org.

**[Sendmail98]** Sendmail, home page, http://www.sendmail.org.

**[Showalter98]** T. Showalter, "Sieve – A Mail Filtering Language", Internet Draft, Jan 1998, available from http://ds.internic.net/internet-drafts/draft-showalter-sieve-03.txt.