BLUETAIL™

Mail Robustifier Whitepaper

H. Millroth

hakanm@bluetail.com

Overview

The Bluetail Mail Robustifier improves reliability, scalability and managability of standards-based third-party mail servers. That is, it makes your favorite mail server system more robust and manageable.

The mail robustifier is a software-only product. It works together with almost all conceivable mail architectures: separate machines for the different mail protocols, all mail protocols on each machine, etc.

Here are some examples of what it does:

- Makes it easy to add, remove or upgrade machines or individual mail servers without service outages.
- Masks server failures by automatically redirecting mail sessions to other servers.
- Dynamically load balances between mail servers in a better way than round-robin DNS.
- Smoothly handles overload situations by throttling connection attempts from new mail clients and by prioritizing clients with existing connections over new clients.
- Makes it easy to implement differentiated service classes (for example "gold class" users, who are given special privileges), by determining target server based on information in external databases.
- Supports spam filtering and admission control based on information in external databases.

Architectural overview

As can be seen from Figure 1, the basic architectural principle is to put a new layer between the mail clients and the mail servers. This layer manages the mail traffic to and from the servers.

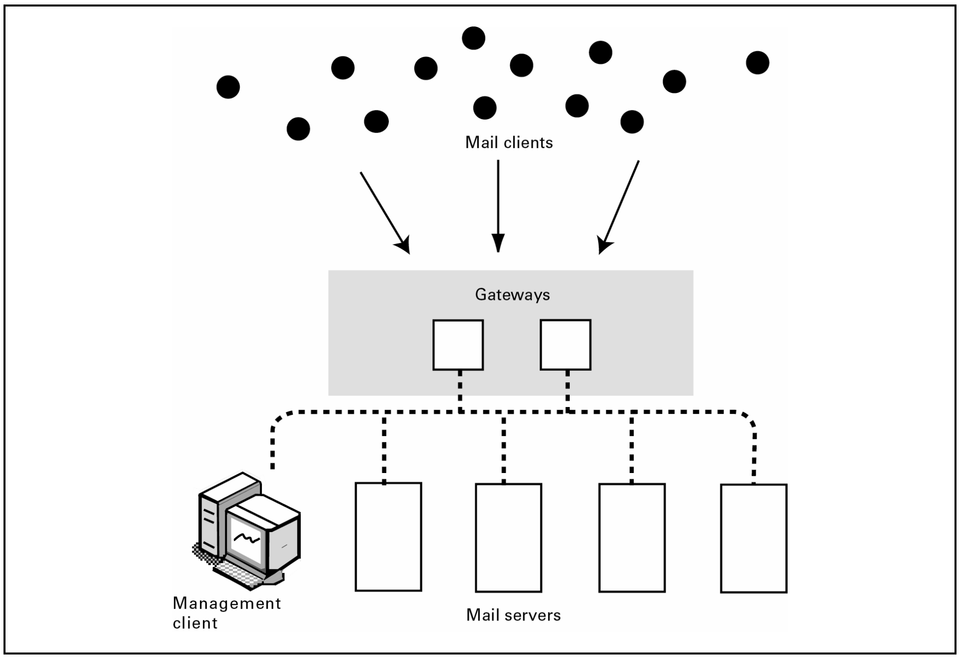

Figure 1: *Typical mail architecture using the Mail Robustifier.*

The robustifier connects the machines in clusters. If one machine in a cluster fails, then the other machines in the cluster cooperate to provide the functionality of the machine that has failed. Individual machines in a cluster may be taken out of service in a controlled manner without interrupting system operations. Additional machines can be added to a cluster without interrupting operations.

Some of the concepts discussed in this overview are described in greater detail in the *Configuring the System* and *Operating the System* chapters.

Gateways and mail servers

Incoming mail traffic is routed through *gateways* which perform traffic control and dispatch the traffic to the appropriate mail servers.

The system supports three types of gateways and mail servers: POP3, IMAP4, and SMTP. For each protocol type we can have one or more gateways and mail servers in the system.

DNS is set up to point to the gateways, not the mail servers. If there are several gateways for a protocol, use DNS round-robin to distribute requests.

The mail servers are not part of the Mail Robustifier product. Any mail server that conforms to the protocol standards works with the product. It is also possible to mix different servers for the same protocol in the system. For example, you can have one *Sendmail* machine and one *qmail* machine and load-balance between them. You can also migrate from, say, *Sendmail* to *qmail* (or vice versa) without service interruption, one machine at the time.

The management client and the management server

The robustifier is managed from a *management client* that runs on a standard PC or workstation. There are two management clients: one with a graphical user interface and the other with a command-line interface.

The management client communicates with the *management server* that handles system configuration and operational requests. If the server machine crashes, a new management server is started on another machine. The old server's IP address is re-mapped to the new machine and the management service is resumed, so this failover is transparent to the management client. That is, the client will have contact with the system as long as there is at least one machine up and running.

Multiple management clients can be connected simultaneously to the system.

Frontend and backend machines

The gateways and the management server run on *frontend* machines. The mail servers run on *backend* machines.

Nodes

The mail robustifier software runs on *nodes*. A node is a virtual machine that executes within a single operating system process. Each node can be configured to perform single or multiple tasks.

Each frontend and backend machine has one node (*frontend nodes* and *backend nodes*). The backend nodes measure the load on the backend machines and report this information to the gateways. Thus there are two kinds of entities running on backend machines: backend nodes and mail servers.

All nodes in the system are connected and check each other's status through a low-level *heart beat* mechanism. This means that the system will quickly notice if any node stops working. In that case the machine is considered dead – see the section *Machine Failures* on page 6 for a description of which actions are taken in the event of failure.

Figure 2 shows two frontend machines, each running an SMTP and IMAP gateway, and three backend machines. The SMTP servers run on all three backend machines, while there is an IMAP server on only one backend machine. There is also one load-reporting node on each backend machine.

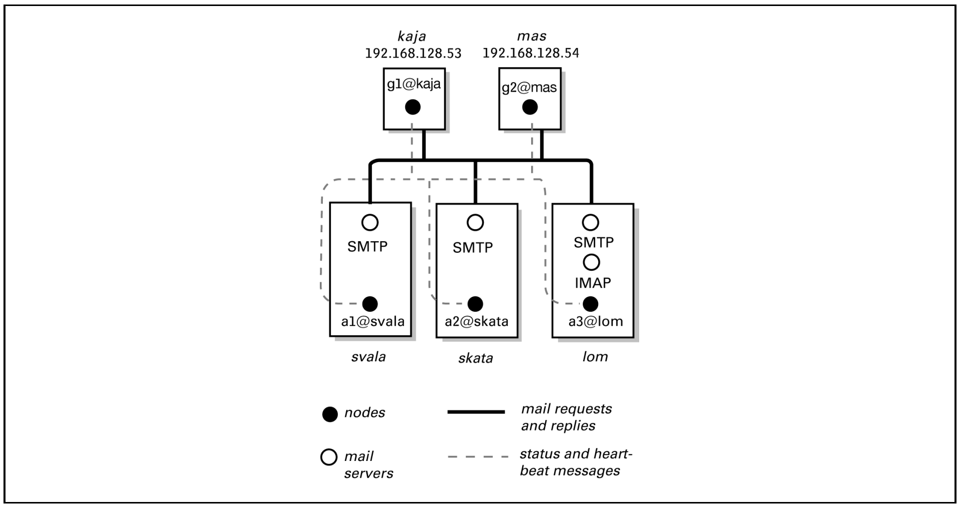

Figure 2: *Configuration with two frontend machines and three backend machines.*

In another scenario, a single node might run all three gateways as well as the management server. In a third scenario, gateways and the management server run on a backend node – this is useful mainly for failover and testing purposes (it follows from this that a machine can be both a frontend and a backend machine, although this is not normally the case).

Clusters

A *cluster* is a collection of nodes and mail servers that work together. A *frontend cluster* is a cluster running gateways and/or the management server. A *backend cluster* is a cluster running mail servers and backend nodes.

The purpose of frontend clusters is to define failover sets: gateways fail over to other nodes in their cluster. Frontend clusters can be heterogenous: a particular cluster may run gateways of different types as well as the management server.

Backend clusters, in contrast, are used for purposes of load balancing and differentiated service classes. Backend clusters are homogeneous: a particular cluster can only run a single type of mail server.

Figure 3 shows a system with one frontend cluster and two backend clusters.

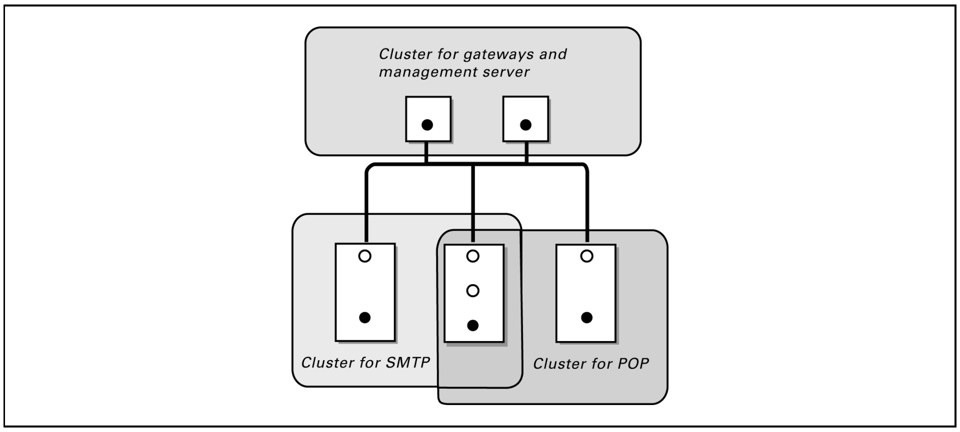

Figure 3: *System with one frontend cluster and two backend clusters.*

Monitors

Each machine running a gateway or mail server also runs a *monitor*. The monitor regularly "pings" the gateway or server using an appropriate protocol-specific request. This is used both to collect response times and to determine gateway/server failure.

Thus, the heart-beat mechanism check the status of *nodes* and monitors check the status of gateways and mail servers.

Load thresholds

Each node has a default *load threshold* that indicates the load at which the machine should be considered overloaded.

An alarm is generated if the average load of the nodes in a backend cluster reaches their average load threshold. In addition, the overload control algorithm (see the *Overload control* section on page 9) is activated.

A single backend machine participating in several clusters has one load threshold per cluster.

For example, consider the two backend machines in Figure 4 which implement two clusters. Both the standard POP3 cluster and the gold customer POP3 cluster consist of nodes A and B. To prioritize gold customers, we set a higher load threshold in the gold cluster than in the standard cluster. This way gold customer get a larger share of the machine resources.

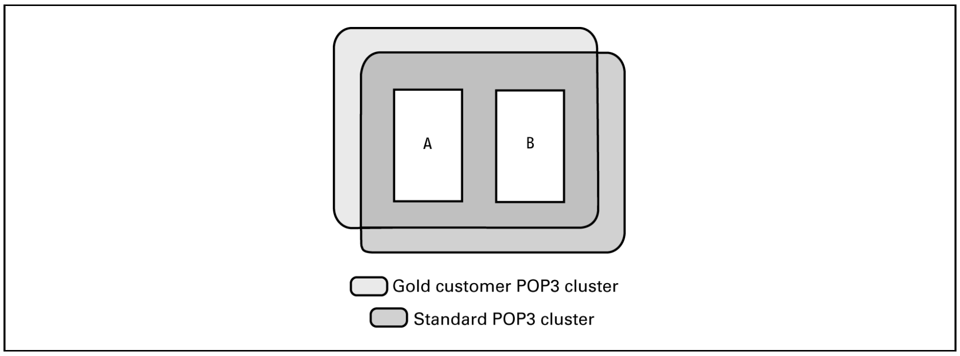

Figure 4: *Configuration with two backend clusters*

Machine failures

The mail robustifier can detect a machine crash in two ways:

- Node failures can be discovered by the heart-beat mechanism.
- A gateway or mail server may fail to answer its local monitor.

If a frontend machine is considered dead, the gateways and/or management server running on it fail over to other frontend nodes. The IP addresses of the gateways/management server are re-mapped to the new machines. When the failed machine comes back up, the gateways normally migrate back to it. Gateways can be configured to be *sticky*, that is they stay on the failover node when the crashed machine returns. The management server, for example, is sticky.

If a backend machine is considered dead, new requests from the gateways are redirected to another backend machine, until the first backend machine recovers.

Interfaces

Each gateway uses a default *interface* for network communication. If this interface is broken, or if the gateway fails over to another node, it uses one of the available interfaces on the node (these interfaces are called *failover interfaces*).

Data tables

Cluster selection and admission control / spam filtering can be based on information stored in external databases.

The information is preprocessed and then loaded into the system and stored on the local disks of every frontend machine. This allows for orders of magnitude faster searches than using an external database server on a separate machine.

POP3 and IMAP4 gateways each use a single table type for both admission control and cluster selection. SMTP gateways use four types of tables for spam filtering.

Features

Online system maintenance

Mail servers can be *blocked* – meaning that new requests to the server are directed to another server – and later *deblocked*. Using this feature, maintenance on backend machines can be done without disturbing normal operations. The procedure is:

- Block the mail server. Now the server no longer accepts new requests; these are directed to another server instead.
- Wait until most existing mail session on the server terminate (as determined, for example, by looking at the statistics monitors in the GUI).
- Stop the mail server. (Client sessions that still use the server will now be disconnected. If the clients reconnect, they will be connected to another server.)
- Do whatever maintenance that needs to be done: install a new mail server, upgrade to a new OS version, replace a broken interface card, replace the machine, etc.
- Start the mail server.

- Deblock the mail server. The server now accepts requests again.

To avoid the capacity loss of taking one machine out of service, a temporary backend machine can be added to the system before starting this procedure.

Software release handling

The mail robustifier is upgradable while in operation. To achieve this functionality it includes a sophisticated software release handling system for its own software.

Once installed, the mail robustifier takes care of managing different releases of its software: unpacking and installing new releases as well as rolling back to old releases. This is done by the system itself, since all frontend and backend machines must run the same release of the software to ensure correct behavior.

If a new release is installed while a machine is down, the release will be automatically installed when the machine returns. If the installation of a new release fails, the system automatically rolls back to the earlier release on all machines.

New releases of the mail robustifier software can normally be installed while the system is running, without disturbing normal operations. (However, major new releases may require that some or all nodes are stopped before the software is installed.)

Load balancing

The mail robustifier supports load balancing of backend machines on a per cluster basis. That is, the gateways dispatch requests to mail servers so that all backend machines within that cluster have the same load (more precisely, their load-to-threshold ratios are the same), so machines with different capacity handle different loads.

The load of machines belonging to different clusters is not balanced. (However, if the clusters overlap the load will automatically be balanced between these clusters. For example, the three machines in Figure 3 on page 5 will be load balanced, since the clusters have one machine in common.)

The load of a machine is measured by CPU load as reported by the Unix *uptime* command.

If more than one gateway is used, then each gateway is given a separate IP address and DNS round-robin is used to share the load between the gateways. A problem with DNS round-robin – stale IP addresses in DNS caches and clients – is avoided, since the IP address of a gateway is re-mapped if the gateway fails over to another machine.

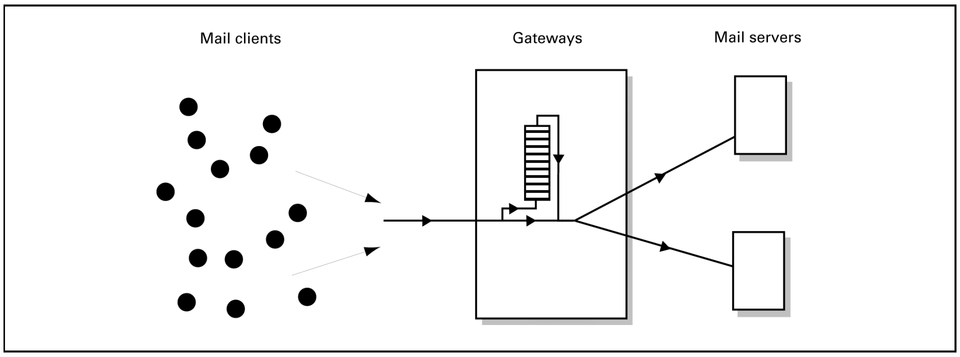

Figure 5: *Overload control.*

Overload control

In order to retain a high service level under heavy load, the robustifier can reject new client requests when the load exceeds the given threshold. Initially, only a small number of requests are rejected. If the overload situation continues, the rejection rate is increased until the overload situation is resolved. POP3 and IMAP4 clients that have been rejected can be queued in the gateway: when they are first in line, they are admitted even if the overload situation is not yet resolved (see Figure 5).

POP3 and IMAP4 clients that have been given access to a mail server are given a *prioritized time slot* during which they are are assigned a "virtual session" during which their next command is prioritized over commands from new clients in overload situations.

prioritized in overload situations when they send new commands to the server. This is to ensure that a mail session made up of a sequence of individual commands is either accepted in its entirety or rejected before it even begins.

Rejected POP3 and IMAP4 clients are notified by a "connection rejected" notification, which is usually reported to the user in a popup dialog. Rejected SMTP clients or mail transfer agents get a "temporary error" return code indicating that they should try again later.

Admission control and spam filtering

New client requests can be rejected based on external table data. For example, we can specify that users matching a table entry should be blocked from our IMAP service.

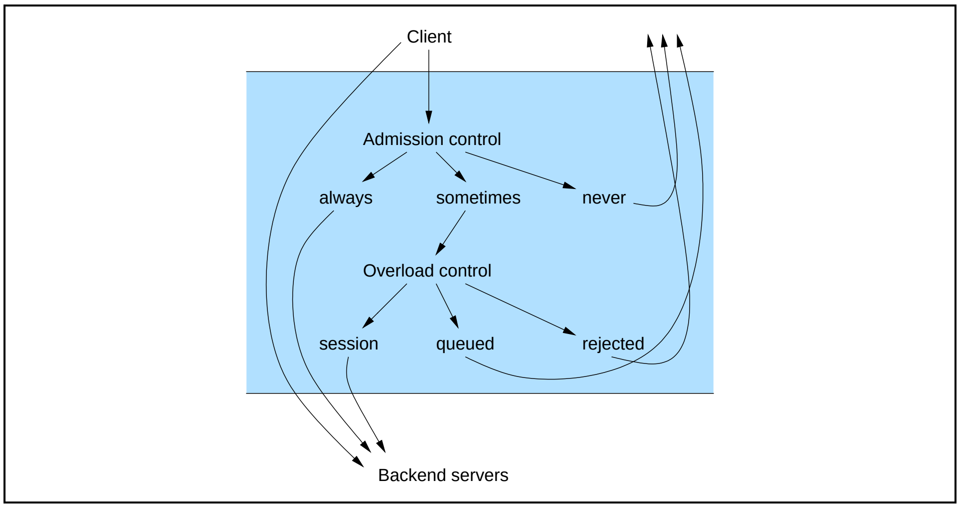

Figure 6: *Overload control.*

Spam filtering by the mail robustifier is done by SMTP admission control following the recommendations of RFC 2505. The mail robustifier supports the following table-based filtering methods:

- *Peer-name blocking.* The connecting host can be accepted or rejected based on its domain name or its IP address.
- *HELO blocking.* Mails from particular senders, as identified by the HELO command, can be accepted or rejected.
- *MAIL FROM blocking.* Mails from particular senders, as identified by the MAIL FROM command, can be accepted or rejected.
- *RCPT blocking.* Mails to particular recipients can be accepted or rejected.

The possible outcomes of overload control and admission control, and the relationships between them, are described in Figure 6.

Table-based cluster selection

POP3 and IMAP4 user requests can be dispatched to different servers based on external table data as shown in Figure 7.

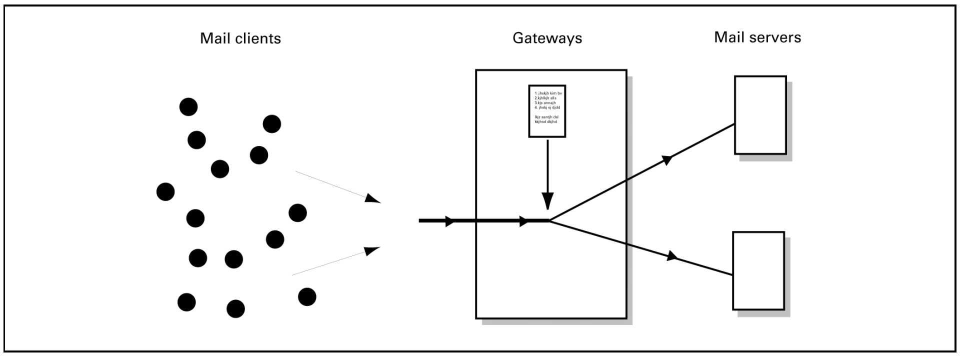

Figure 7: *Table-based cluster selection.*

For example, we can specify that POP3 users in *Table42* should be dispatched to the POP3 cluster *Cluster42*. An application of this feature is to dispatch premium customers to one backend cluster and regular customers to another.

Handling mail server failures

If a mail server crashes, the frontend gateway will automatically send new requests to other servers. If and when the server recovers, the gateway will automatically send it new requests.

Ongoing SMTP transactions are transparently moved to a new server if the current server crashes (the transaction state is stored in the gateway). That means that the message can be accepted although the current SMTP server crashes.

Handling gateway crashes

If a gateway crashes, it migrates to another frontend machine. As part of the migration, its IP address is re-mapped to the new machine. This means that the IP addresses handed out by DNS (and possibly cached by mail clients) will still work after a crash.

Existing connections are terminated if a gateway crashes; however, the clients can immediately reconnect and be served by the migrated gateway.

Example Configurations

This chapter describes four examples of mail system architectures that use the BLUETAIL Mail Robustifier.

Configuration files for the examples can be found in the *examples* directory of the program distribution.

A basic redundant system

The first example is a minimal architecture that exploits most benefits of the robustifier. It consists of one frontend machine running a combined POP3/IMAP4/SMTP gateway, and two backend machines running POP3, IMAP4 and SMTP servers (see Figure 8).

There is one frontend cluster consisting of all three machines, with the frontend machine as the primary node for the gateway. This means that the gateway will run on the frontend machine unless that machine is down; in that case the gateway will failover to one of the backend machines. Thus, in case the frontend machine crashes or is taken out of service, one of the backend machine will work as host for both the gateway and its mail servers.

The management server normally runs on the frontend machine. It will failover to a backend machine if the frontend is out of service, just as the gateway does.

There are three backend clusters, each consisting of both backend machines. This means that the two machines share the load for all three types of traffic (POP3, IMAP4 and SMTP). If one of the backend machines is out of service, the other machine handles all traffic. Possible overload due to reduced server capacity is handled by the overload control mechanism in the gateway.

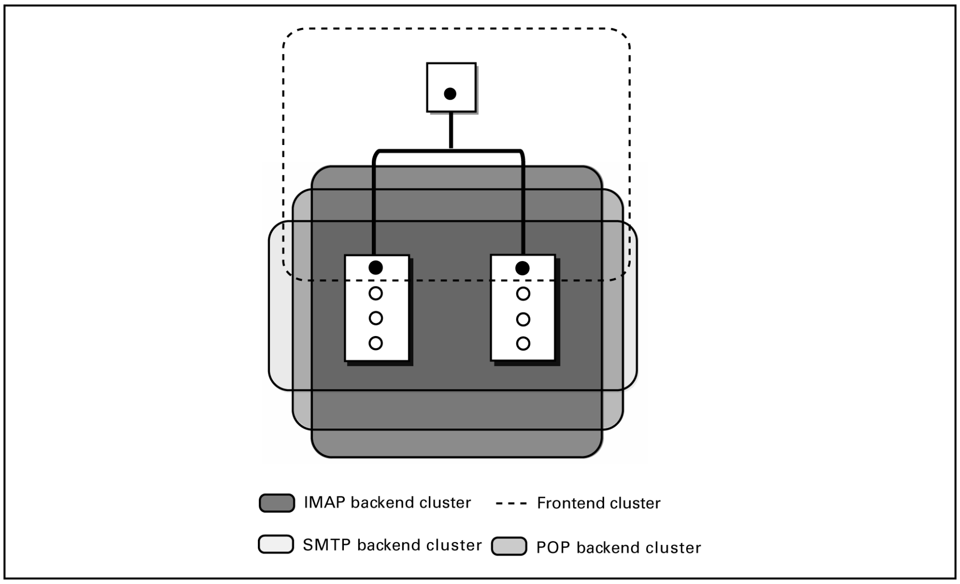

Figure 8: *A basic redundant system (basic.conf).*

This basic architecture can be varied in a number of ways, for example:

- More frontend machines can be added to increase gateway capacity.
- More backend machines can be added to increase mail server capacity.
- If there are several frontend machines, they can all run the same type of traffic, or each can be dedicated to one type of traffic (for example, one POP3 gateway and one SMTP gateway).
- Similarly, the backend machines can be partitioned in different ways: all machines can run all protocols or each machine may run only one or two of the protocols.
- For each type of traffic (POP3, for example), the backend machines can be partitioned to handle several service classes. For example, "free email" customers can be directed to a specific server while business customers can use all available servers and, in addition, have priority on the machine used for "free email." This can be done without static configuration of client software; all clients use the same host name.

In the following examples we illustrate some of these options.

A large coarse-grained system

There are two basic ways to build a large-scale mail server system:

- A coarse-grained architecture with a few big server machines. Each machine runs several types of traffic.
- A fine-grained architecture with many small server machines. The servers are partitioned so that each machine only runs one type of traffic.

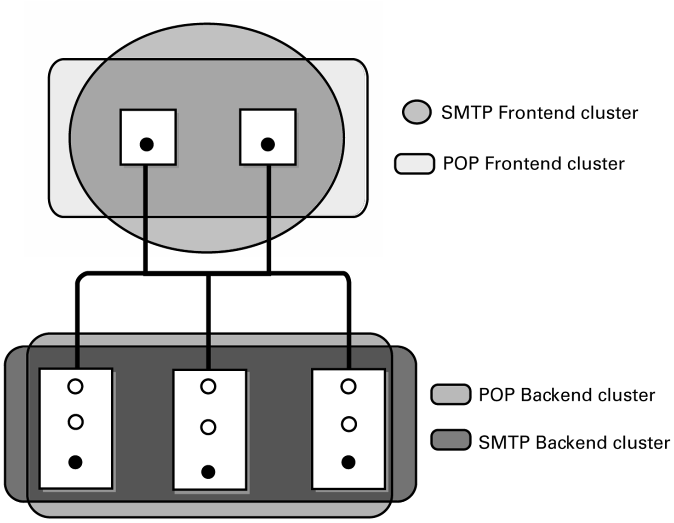

Figure 9: *A coarse-grained system (coarse.conf)*

Figure 9 shows a system of the former type. This system supports only POP3 and SMTP. It consists of two frontend machines and three big backend machines.

One frontend machine runs a POP3 gateway and the other runs an SMTP gateway. There are two frontend clusters, one for POP3 and one for SMTP. Each cluster consists of both machines – the POP3 gateway is also a standby for the SMTP gateway and vice versa.

Each of the three backend machines runs both POP3 and SMTP servers. The load thresholds for POP3 and SMTP can be differentiated, to prioritize one of the protocols over the other on each machine.

A large fine-grained system

Figure 10 shows how a large-scale mail system can be realized with a fine-grained architecture. The frontend cluster is the same as in the previous example.

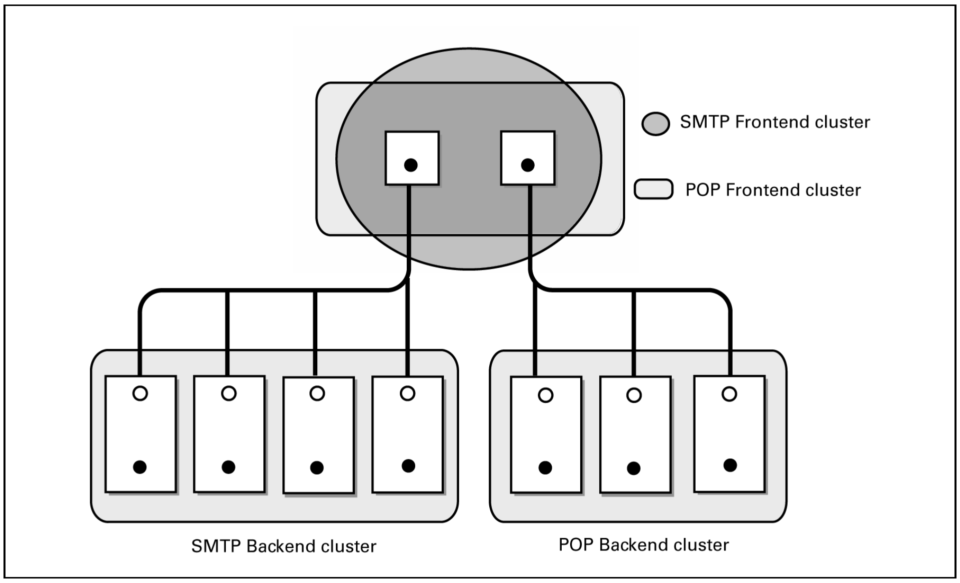

Figure 10: *A fine-grained system (fine.conf).*

The backend tier consists of four dedicated SMTP machines and three dedicated POP3 machines. The SMTP traffic is load-balanced within the SMTP cluster and the POP3 traffic within the POP3 cluster.

A fine-grained architecture like this has some advantages: there are small failure zones, and the system can grow incrementally by adding more small machines.

A system with differentiated service classes

The system shown in Figure 11 implements three different service classes for IMAP4 access, using three backend machines. Each service class is realized as a separate backend cluster:

- The "free email" cluster consists of one server.
- The "normal" cluster consists of the "free email" server and one additional server.

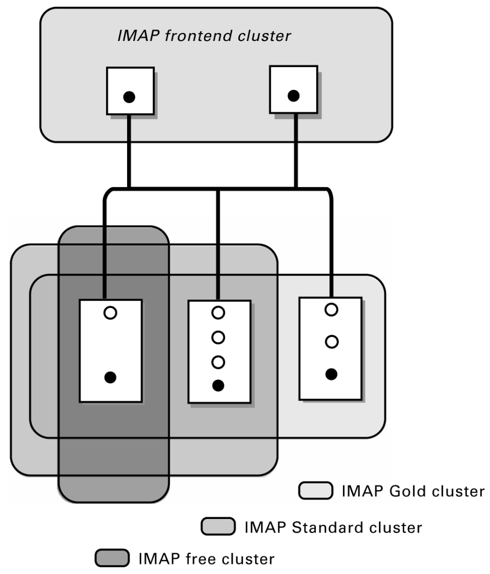

Figure 11: *A system with differentiated service classes (cos.conf).*

- The "business customer" cluster consists of all three servers.

This clustering scheme ensures that higher service classes have higher total capacity, and more reliability through redundancy. In addition, higher service classes can be further prioritized by having higher load thresholds than lower service classes on the shared machines (see the *Overview* chapter, page 6).