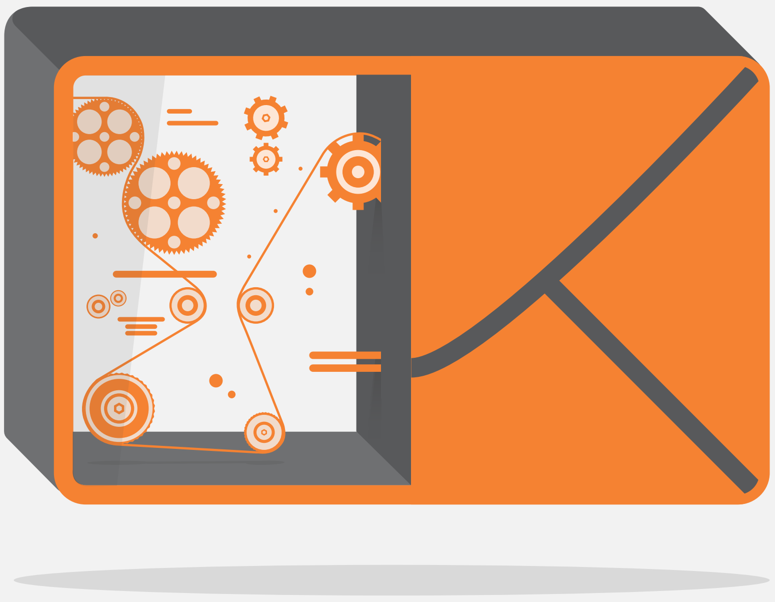
EMAIL SYSTEMS: SCALING AND HIGH AVAILABILITY USING OPEN SOURCE SOFTWARE



www.atmail.com

INTRODUCTION

This whitepaper outlines how to scale email systems using Open Source Software, from a single node up to large scale cloud deployments.

There are a number of factors to take into consideration when building a fit-for-purpose email system:

- The number of mailboxes — how many users will access the system, and when?
- Performance — what is considered acceptable performance for sending, receiving and accessing email?
- Reliability and Availability — what impact will outages and downtime have on your users or the business?
- Cost — even using 'free' open source software, hardware and software are still required to run and administer the system
- Integration and Technology — are there any limiting technical factors that require specific software or need to integrate with specific systems. What storage is available for your email integration?

These factors, how they relate to each other, and their priority can all influence how the the resulting architecture is integrated and used.

STARTING SMALL - A SINGLE NODE

In its simplest form, an email system can comprise a single node — one server that will act in all capacities to send and receive mail, and allow users to access to their mailboxes. This is very easy to setup and manage — there are numerous guides available online showing how to set up servers using MTAs such as postfix and exim, and IMAP/POP servers such as dovecot.

A single node is obviously the cheapest type of architecture to set up, run and maintain, but it has obvious downsides. If the server stops working, so does access to email. In a worst–case scenario, if the server's data has not been backed up to an external device, it could mean total loss of data. Until the server has been rebuilt or replaced, and any data restored, you will lose all access to email.

With a single server, it is best to also have some backup in place, such as a secondary MX hosted elsewhere, that will collect and queue your mail while the server is offline. If nothing has been set up, then senders may start getting messages bounced back to them since delivery is not possible.

There is no great way to scale a single server. If the service becomes too slow, the only option is to buy a larger server with faster processors, more memory and more storage.

Pros and Cons:
+ cheap
+ simple
- server outage means no mail

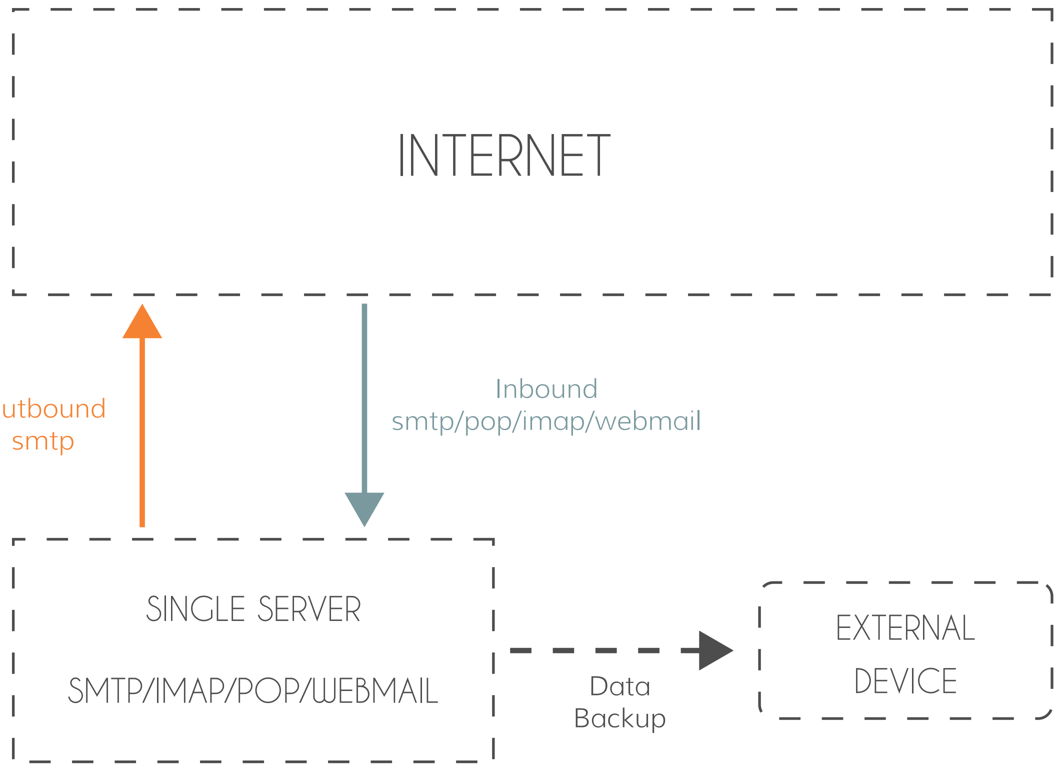


fig 1.0 *Simple, cheap and perfectly acceptable if the server is big enough to support the email needs.*

2 NODES ARE BETTER THAN 1

While a single server, with a backup, may be sufficient for some purposes, if email is considered critical, redundancy will need to be built into the system.

This could be as simple as having a second server ready on hand that can be made active if the primary server goes down. Now we have a second server, what do we do with it? A mailbox can only reside in one place at any one time, so setting up and configuring a shared or replicated filesystem will allow optimal use of a second piece of hardware.

REPLICATED FILESYSTEM

Let's look at replicated filesystem first. The second server should have an identical mail application configuration. As you can see in fig 2.0, inbound mail is delivered to the mailstore, leaving the underlying filesystem replication to ensure that the message is seen on both servers, regardless of which one initiated delivery.

With multiple servers now being used, the email architecture can be configured for High Availability, and we have started down the path of scaling out the system. In most cases when a replicated filesystem is used, if one server goes down, the user sees no impact. However, that is only the case where a single node is big enough to support the entire load.

Two servers doesn't always mean twice the capacity. Email has been, and still is, storage–intensive. Adding a replicated filesystem that is continuously replicating files between multiple nodes can affect disk performance — they are typically far slower than simply writing to a native disk, since they have more work to do.

In the open source community, glusterfs is a well known replicated filesystem. Overall, its performance can be poor, but if a High Availability email architecture is required, that performance trade-off may be seen as appropriate.

There is one other item to note. Two servers require two IP addresses and, if both servers run a public service, then both would need a public IP. A single IP address could still be used if assigned to a load balancer placed in front of both servers (more about those later). If not, a single IP address would mean the setup is an active/passive configuration between the two nodes, where the IP address 'floats' to the secondary server if the primary server goes down.

<u>Pros and Cons:</u>
+ fully redundant, high availability environment
+ multiple public IPs required in active/active scenario (unless load balancer used)
- slow disk i/o = slower performance

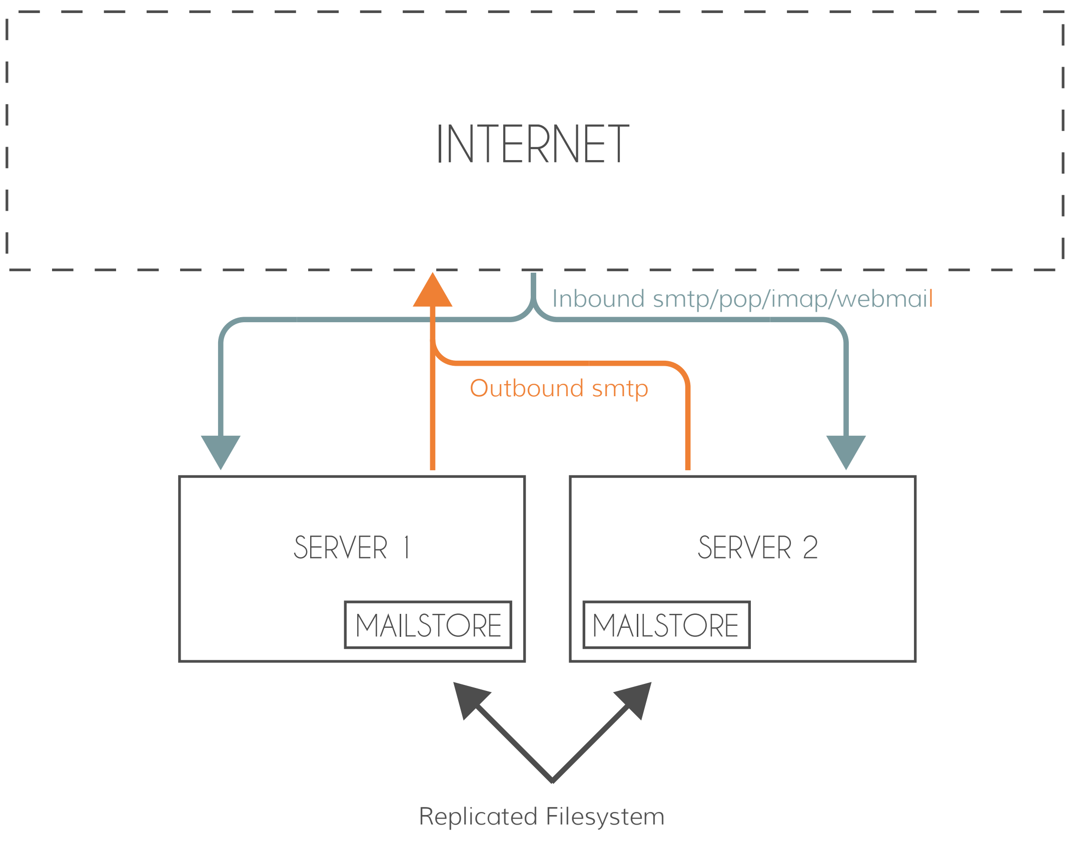


fig 2.0 With the addition of the extra node you now have an high availability environment. If one server goes down, there is no impact to users.

SHARED FILESYSTEM

The alternative to replicated filesystems is to run some form of 'shared' filesystem. This can be achieved in a couple of ways but both require some form of shared storage, and reintroduces a single point of failure in the storage device, whether it is a NAS or SAN. While the disks are in RAID, and the NAS may have multiple power supplies, there is still the possibility of hardware failure taking mail storage offline.

CLUSTERED FILESYSTEMS

Clustering is a traditional method of providing high availability, and there are many ways a mailstore could be clustered:

- An active/standby system could be setup with a non-clustered filesystem. This would be set up so that both the public IP address and storage would 'migrate' to the second node if the primary server goes down.
- An active/active system could be set up by just using clustering to create a filesystem that exists, and is readable and writable, on both nodes (eg GFS2, OCFS2)

Specialist knowledge of clustering is often required. Even so, these filesystems typically suffer the same issue as replicated filesystems, in that performance is much slower than using a standard disk array. Much simpler and better performing options exist.

+ High availability is achievable to a degree
- Requires knowledge around clustering

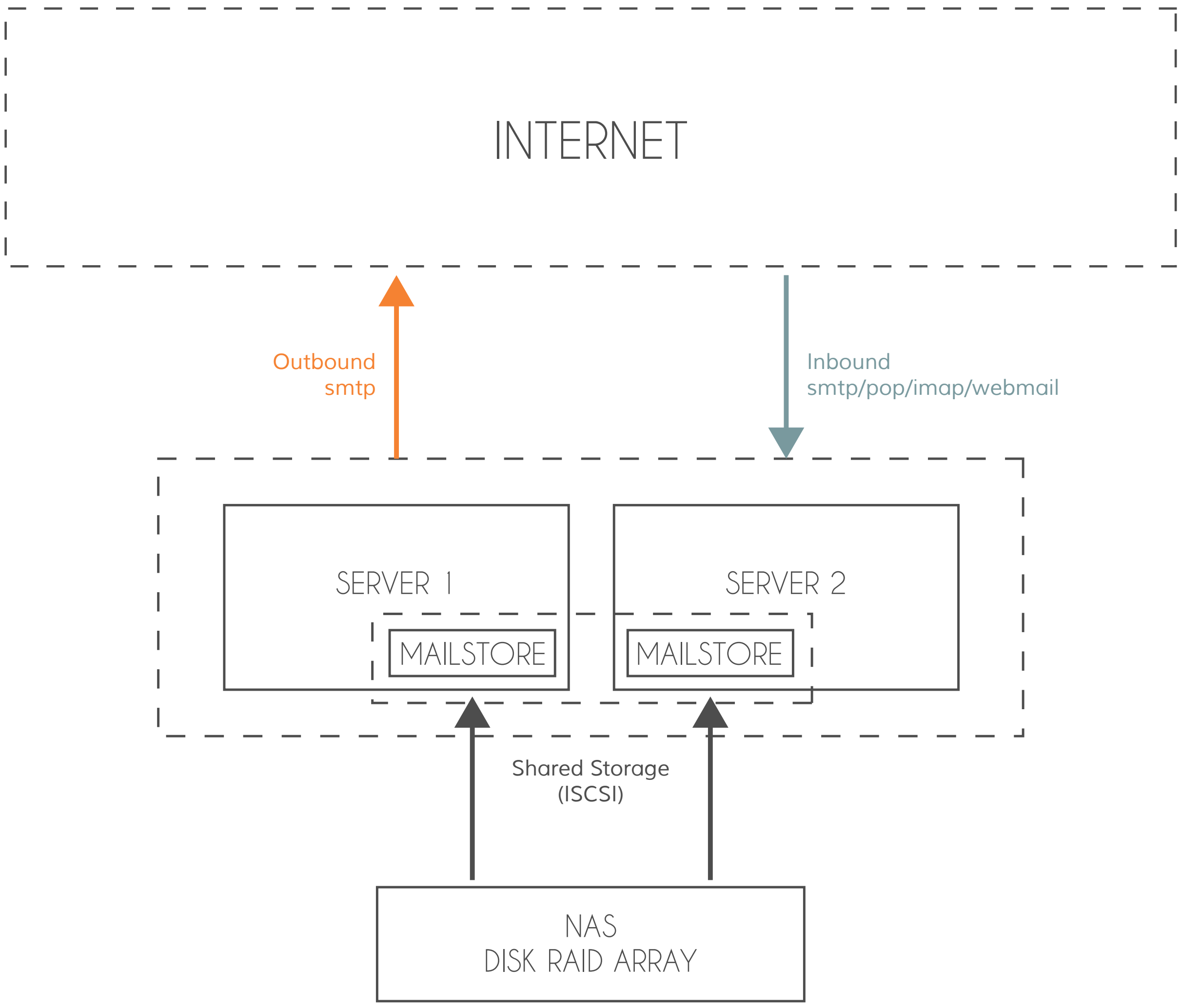


fig 2.1 *Many ways to set up a cluster from active/standby, which could have non-clustered filesystem from mailstore and be more performant to making the mailstore filesystem clustered allowing both servers to access. Requires knowledge around clustering.*

NETWORK SHARED FILESYSTEMS

A simpler, much more effective solution for shared filesystem is to use an existing network share protocol such as NFS or SMB. Through these, it is very easy to create a two–node, active/active system (see fig 2.2). Configured correctly, NFS can be very performant and scale to a number of users.

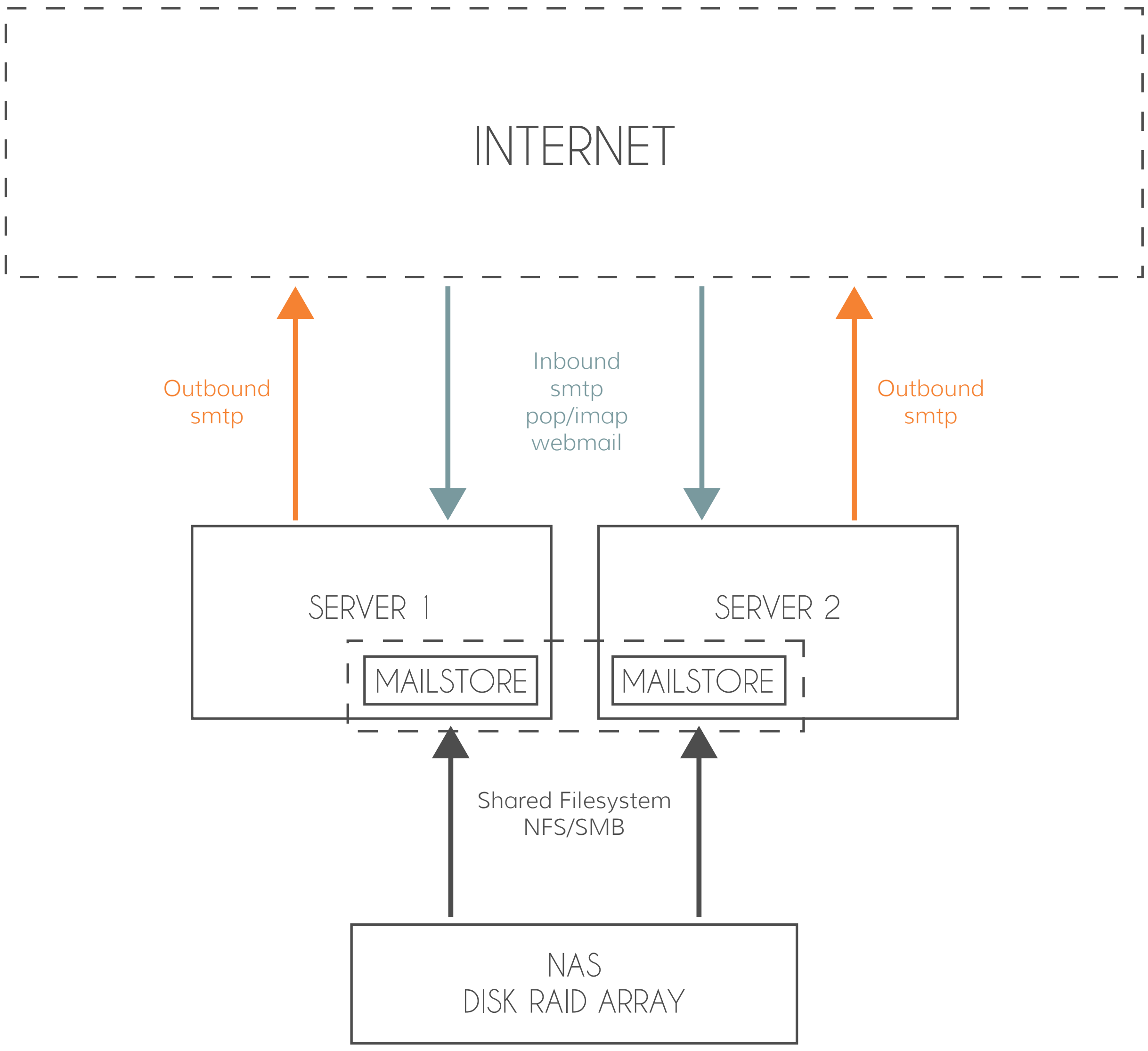


fig 2.2 Most NAS devices allow shares to be setup using NFS/SMB/ISCSI much more simpler than creating clustered filesystem

3+ NODES ARE A CROWD

So far we have kept the systems relatively small — a single mailstore — but what happens when a single mailstore no longer is enough? The system outlined above, in fig 2.2, could potentially scale much larger simply by using more NFS mailstores. However, a mechanism to share the mailboxes is needed across multiple mailstores.

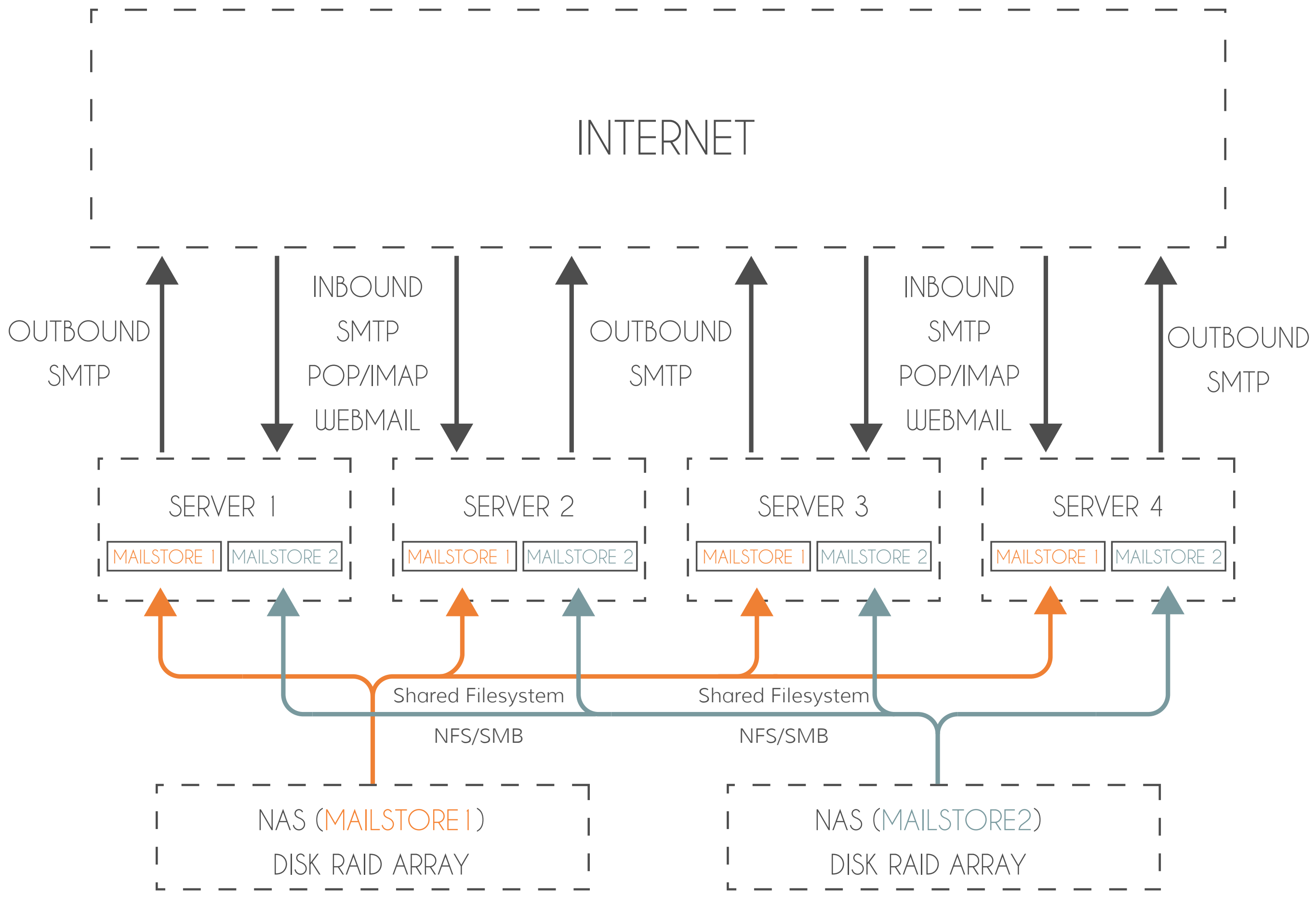


fig 3.0 *Multiple mailstores will allow scaling out to a much larger system*

The diagram above, fig 3.0, may show some obvious weak points. As more and more application servers are added, more and more public IP addresses are needed, and more servers the mailstores need to be mounted to. Also, as more and more mailstores are required, a uniform way of spreading the load is needed. If expansion has not been carefully thought out, it could lead to very messy filesystems.

A more important weak point to note in using network shared systems such as NFS, however, is what happens on the servers when a mailstore goes offline. Just because the mailstore is offline doesn't magically stop users trying to access their mail. Inbound POP/IMAP connections will still attempt to access the offline mailstore, usually until they time out, which makes connections exist for far longer. That, in turn, leads to a build-up of connections and would ultimately lead to an overload of connections. Mail deliveries to the shared drive are also no longer possible, causing errors and timeouts. Mail will gradually build up in the queue. Once the mailstore is back online, recovery time will vary depending on load and outage times.

LOAD BALANCING

What can be done to address these weaknesses? Both scalability and high availability of most services can be achieved through load balancing. Load balancing acts as a single endpoint for multiple servers, and can be implemented a number of ways, including:

- Use of Linux Virtual Servers (LVS)
- Use software proxies (such as HAProxy/Nginx)
- As a service (eg ELB if using cloud provider such as AWS)

LVS can be set up to load balance, either by proxying connections or redirecting traffic to the servers in question. The important distinction to note here is that with proxying, the servers behind the load balancer will only ever see connections as coming from the proxy server. With redirection, only the initial connection is made through the load balancer, with further communication directly between the external and application servers. This means incoming connections are seen from their point of origin, not the load balancer.

Proxying can also be done via TCP/IP proxy software such as HAProxy or, for web servers, by reverse proxies such as Nginx.

Load balancers allow a single entry point to multiple servers but doesn't that make the load balancer itself a single point of failure? It does, to a degree, which is why load balancers themselves need to be made highly available. This can be done either by some form of clustering active/standby setups (such as a floating IP address), or more simply by having more than one, and allowing DNS servers to perform a simple round robin. While the DNS route is obviously simpler, it does mean that if a load balancer was to go offline, any connection resolving to the IP address would fail. If TTLs are kept low, it would be fairly easy to create a script that would add remove 'A' records of the name from the DNS server, and keep ongoing occurrences to a minimum.

SCALING SMTP

Where web and application servers have long had the ability to maintain a link to the original connection host, via proxy and use of the X-Forwarded-For header, the same cannot be said for SMTP servers. Knowing the original IP address is vital when it comes to mail servers, given that DNS real-time blackhole list (RBL) typically used in the fight against spam. If connections are only ever seen to be coming from a proxy, these checks become impossible to perform and would have to be omitted.

Recent versions of postfix and exim, however, have added support (if only experimentally) for a proxy protocol that allows the MTA to use information about the original connection sent by the proxy, to use in RBL checks. This ability to use a proxy protocol makes using the likes of HAProxy both a simple, effective and cheap means of performing load balancing of SMTP connections.

To a degree there is already some load balancing/high availability for MX mail servers due the way DNS is used in mail deliveries. When an email is to be delivered the DNS MX records are queried and the server with the lowest priority is chosen and a connection attempt is made. If there are multiple servers sharing that priority then deliveries could be made to any one of those servers. So simply having multiple servers, each having the same DNS MX priority, resiliency and high availability are automatically achieved. Of course, this means having multiple public IPs.

SCALING IMAP/POP

IMAP/POP3 are still the standard protocols for clients to access mailboxes, but how do several thousand users access mail through the same end point? Using a proxy server that acts as the focal point to multiple backend IMAP servers, allows you to simply direct multiple endpoints through a common, single endpoint.

Further scale can be achieved by placing a load balancer in front of the proxy servers. There are a number of open source programs that allow IMAP/POP proxying such as perdition, im-approxy, nginx, and haproxy, to name a few.

SCALING WEBMAIL

This is primarily achieved in the same way as SMTP, IMAP and POP, through the use of the software load balancers that can allow multiple web servers to run simultaneously sharing connections and load among the servers.

PUTTING THIS ALTOGETHER

As you can see from fig 3.1, the load balancer has now shrunk our outside public footprint back down to potentially one IP address.

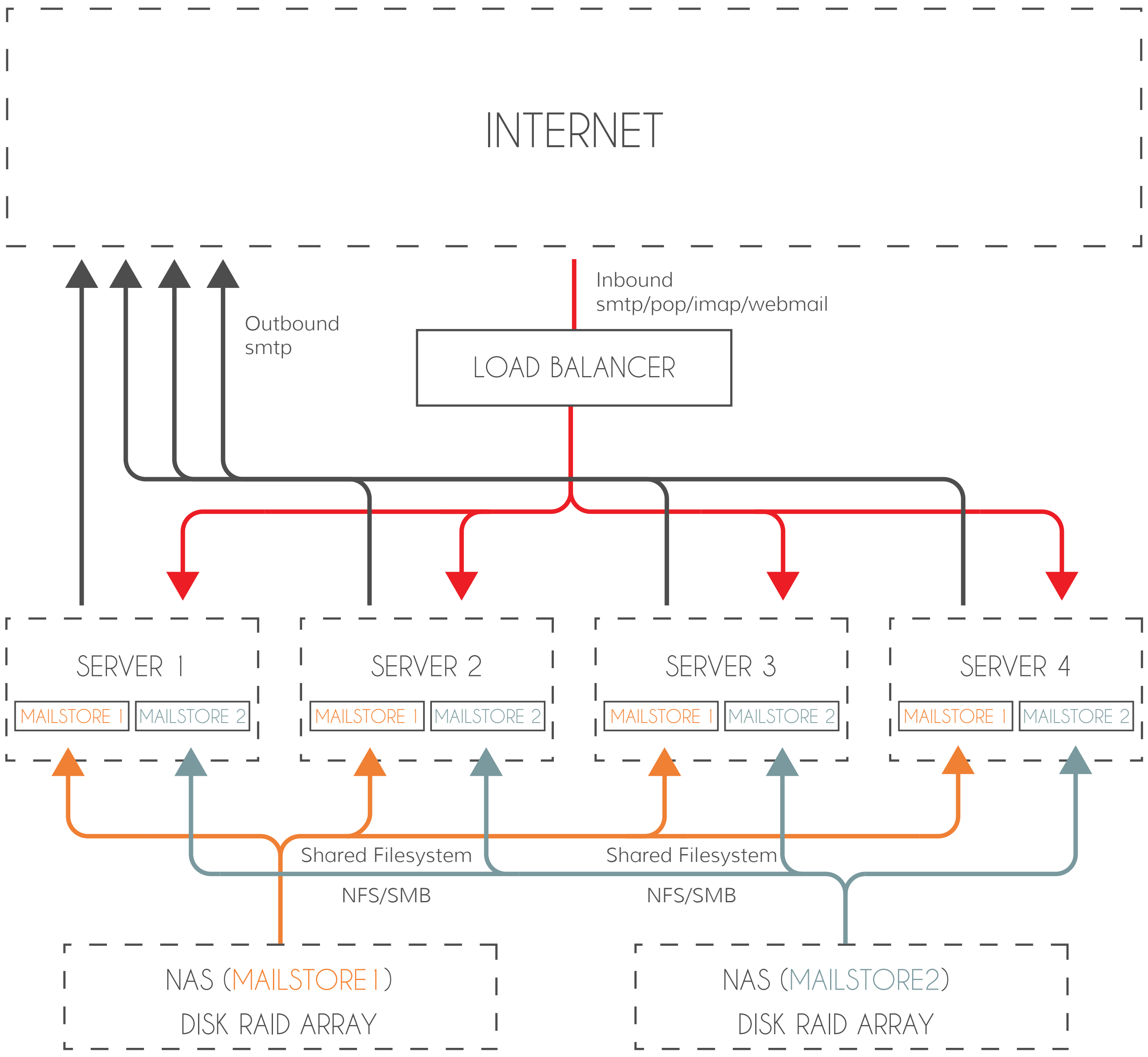


fig 3.1 *With the addition of Load Balancer the external foot print shrinks and less servers actually need direct exposure to the internet*

However, even with the load balancer, there are still an ever–growing number of mailstores, application servers and their mounts to mailstores. So how can we maintain a small public footprint, yet still deliver to multiple mailstores and gain access to them simultaneously?

Let's take the mail deliveries first. We need to add an extra function as routers take IP traffic and direct packets to the desired destination, and that's exactly what we need for email — something that will accept mail, and instead of delivering straight away, simply forward it to the desired destination.

However, we cannot simply take the mail router on its own, if we now have multiple separate mailstores, then we also need to include a POP/IMAP proxy to ensure those connections end up on the correct mailstore.

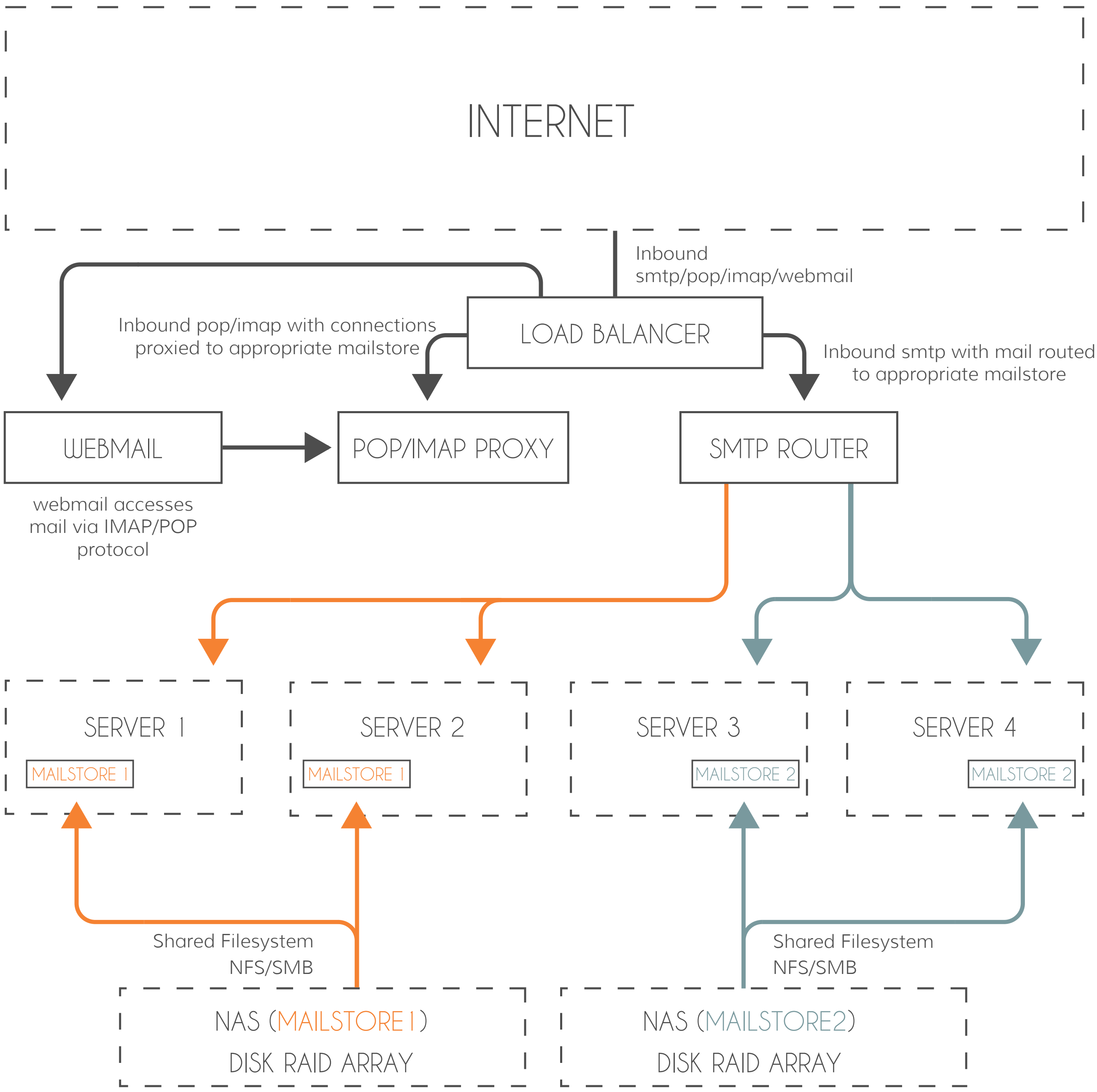

INTERNET
Inbound smtp/pop/imap/webmail
LOAD BALANCER
Inbound pop/imap with connections proxied to appropriate mailstore
Inbound smtp with mail routed to appropriate mailstore
WEBMAIL
POP/IMAP PROXY
SMTP ROUTER
webmail accesses mail via IMAP/POP protocol
SERVER 1
MAILSTORE 1
SERVER 2
MAILSTORE 1
SERVER 3
MAILSTORE 2
SERVER 4
MAILSTORE 2
Shared Filesystem NFS/SMB
Shared Filesystem NFS/SMB
NAS (MAILSTORE 1)
DISK RAID ARRAY
NAS (MAILSTORE2)
DISK RAID ARRAY

fig 3.2 *By making mailstores independent 3 additional components are required, mail router, pop/imap proxy and web client can then use imap/pop in order to connect to mailbox*

So we now have a system that, to a large degree, is scalable and highly available. There is still room for improvement. From the above, you can see that the mailstore NAS is still a single point of failure, so mounting it to multiple servers really doesn't improve anything. Also, by simply adding a second independent mailstore, the design has gradually evolved away from scaling app servers running all services, to needing to separate out and direct certain services. This leads us nicely on to scaling by service.

SCALE BY SERVICE

We started off with a single node, performing all functions, expanded that concept to gain high availability and greater scale, before gradually evolving to having separate mailstores and directing traffic to a specific mailstore where the mailbox resides. If we take this idea to the extreme, we can create a highly scalable, highly available architecture based around each service that a mail system provides.

- MX Service — responsible for handling inbound mail for delivery to known domains and mailboxes
- Client Relay Service ' responsible for allowing authenticated users and or known IP addresses to send mail
- Outbound Mail Service ' responsible for delivering mail to external domains and users
- IMAP/POP Service — responsible for allowing users to access mail via clients such as Outlook, Thunderbird or MacMail
- AntiSpam Service — responsible for scanning inbound/outbound mail for viruses and spam
- Webmail Service — responsible for allowing access to mailbox via a web application

Using this approach, each service can be scaled appropriately and made highly available. While this approach will initially require more server, and as such have a higher upfront cost, as this architecture scales up it starts to become much cheaper, as it allows for far greater control of resources where they are required. It also allows the software to be configured so it has a targeted role within the system, making them much simpler.

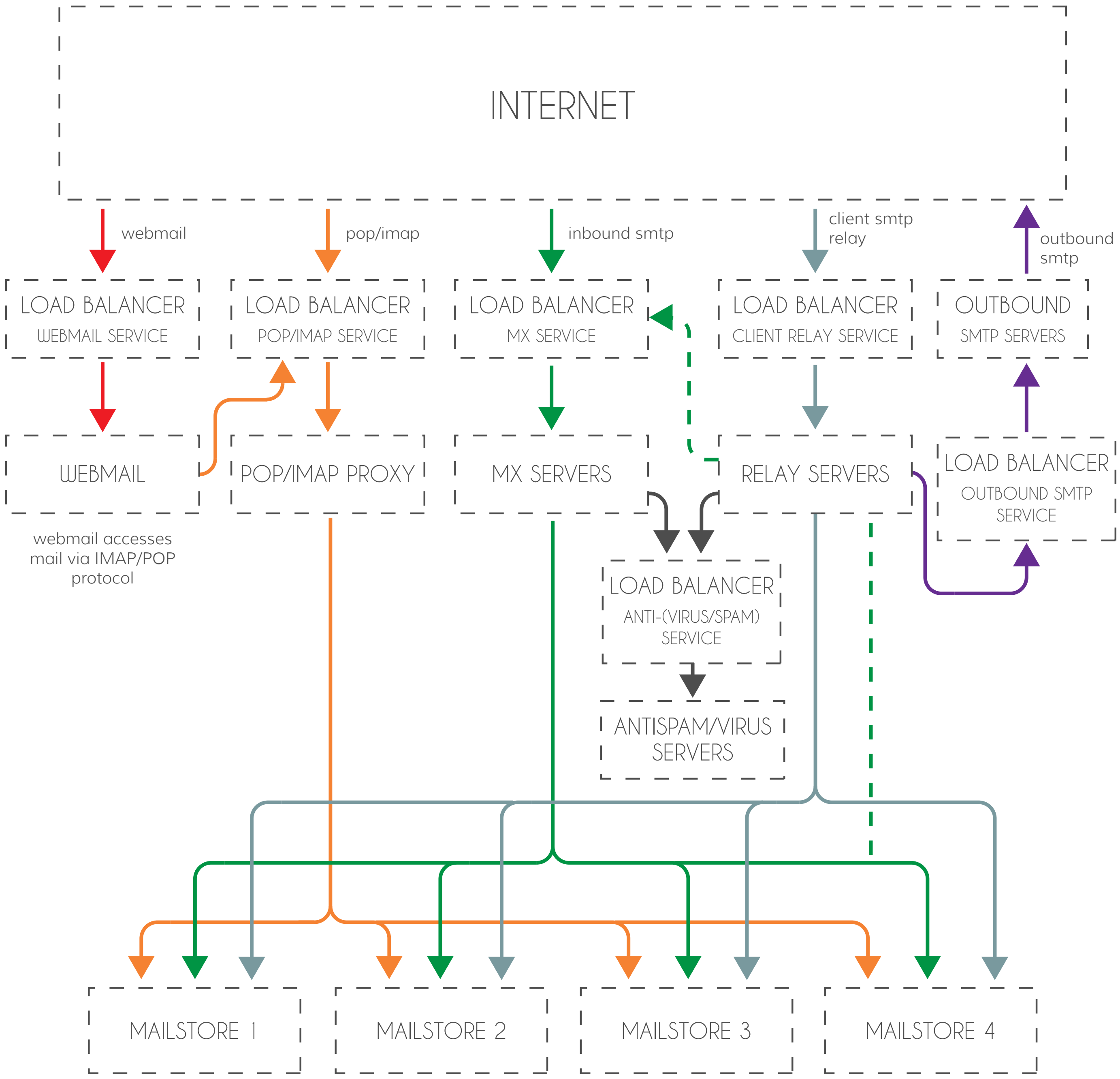


fig 4.0 *Fully scalable, high available service based architecture. The client relay could be configured a number of ways, mail for internal domains could be routed to the mx servers, or directed to the mailstore or simply to the outbound relays where a dns lookup would route them to the mx servers.*

MAKING THE MOST OF MAILSTORES

With the above architecture in place, we now have an extremely scalable and highly available system

Only by using a replication can one truly build a fully, highly available mailstore. However, there are few points to consider:

1. Performance of replicated filesystems is not great due to increased pressure on disk performance. However, without replication, more mailstores would be required.
2. Cost — By having more mailstores, there is an obvious increase in cost. More mailstores equate to more servers and additional hard disk space — remember each mailstore is multiple servers and each server must have the same amount of space to handle replication. Disk space will probably be one of the biggest expenses in such a platform.
3. Software such as DRBD could be used to replicate the disk at the block level. This would be more efficient than gluster, but again it requires setup, monitoring and doubling of disk space.
4. If real time replication is prohibitively expensive, there are alternatives that allow syncing of servers either bi-directionally, such as csync2, or uni-directionally, such as rsync . Running these can often place further pressure on the disk I/O though, which is likely already heavy.
5. If setup within a cloud environment such as Amazon Web Services or OpenStack, snapshots could be taken, or a script set up to simply detach the storage run up by another server, reattach storage and carry on.

At the end of the day, whichever avenue is taken will be a combination of cost, performance, acceptable risk of downtime, and the platform the system is built on (physical v virtual servers or cloud).

Another consideration is the flexibility required from the mailstores, such as what happens when a single server cannot cope with the requirements of a single domain, or the ability to put a mailbox anywhere so mailstores can be more evenly loaded.

Traditionally, with most MTA a list of ‘local’ domains and associated ‘users’ are configured and a single destination assigned at the domain level. This keeps mail configuration easy but does not allow for the aforementioned flexibility.

A solution is required to allow any mailbox to exist on any mailstore, regardless of domain. At the end of the day, a user really doesn’t care where their mail is stored; just that it is accessible. Microsoft call this Shared SMTP Namespace: the ability to split a single domain across multiple servers or platforms.

While this is only supported in fairly recent versions of Microsoft Exchange Server, it can be configured using open source software — it’s simply a matter of configuring a mailstore to act in some parts like a relay. This way, the mailstore can deliver to users the mail the mailstore knows about, and forward on anything else to another server that can make a decision as to where it needs to be delivered.

IF YOU HAVE ANY QUESTIONS
OUR INBOX IS **ALWAYS OPEN.**

CONTACT US

GLOBAL HEADQUARTERS
atmail pty ltd
22/224 David Low Way
Peregian Beach, 4573
Queensland, Australia