Experience with Grapevine: The Growth of a Distributed System

Michael Schroeder, Andrew Birrell, and Roger Needham

presented by: Ryan O'Connor <rjo@cs.ubc.ca>

November 4^{th}, 2009

Grapevine

Birrell et al.:

> *"Grapevine is a distributed, replicated system that provides message delivery, naming, authentication, resource location, and access control in an internet of computers"*

Grapevine provides two services:

- **message service:** accepts and delivers mail to individuals and lists
- **registration service:** provides everything else using a *registration database*

Registration Database

Contains information about users, machines, services, distribution lists, and ACLs

- Grapevine partitions the registration database into multiple registries
- Each registry maps names to groups or individual entries (*RNames*)
- RNames use two-level hierarchical naming scheme (*name.registry*)
- Entire registries are replicated between servers – an update can be processed by any server that has a copy of the given registry

Current State

Grapevine has grown considerably since its inception:

- **1981:** 5 servers, 1500 individuals, 500 groups, 2500 messages per day
- **1983:** 17 servers, 4400 individuals, 1500 groups, 8500 messages sent (but 35000 messages delivered!)

Scalability Experience

Goal: Increase system capacity by adding servers, not upgrading

- This works! (kind of...)
- Grapevine will probably scale to designed maximum size of 30 servers

What Works:

- registration database partitioning mostly works – a larger user base is handled with *more* registries, rather than *larger* registries

Problems:

- some configuration information must be stored on all servers. This information is proportional to the number of servers and registries.
- Grapevine assumes direct connectivity between servers
- **Distribution lists:** some lists are subscribed by a fixed percent of the total population

Solutions to the Distribution List Problem

The authors note that large distribution lists must be accommodated.
Quick fix: add another layer of indirection!

- divide distribution list by registry
- deliver message to one server for each registry, which then delivers message normally.

Another "fix": newspapers have editors that filter content.
Therefore mailing lists must be moderated "before they can become universal"

Configuration Problems

Mailbox Location:

- need to keep inboxes close to users, but split between servers to balance load
- however, splitting inboxes between servers reduces message sharing
- even worse for secondary mailboxes: server failure could overload backups

Registry Partitioning:

- registries currently correspond to geographical locations
- however, some distribution lists cause a large registry to be available in tow locations
- solution: add organizational registries as well.

Transparency and Distribution Problems

Eventual Consistency

- update delays can be bothersome for users
- Solution: updates are often related, so client programs should connect to same server while performing updates.

Remailing

- when a mailbox was moved its contents were remailed
- this could cause a large amount of mail to be remailed
- remailing was a bad idea

More Transparency and Distribution Problems

Other Problems

- admins do not know when a mailing list is unused
- deletion delays cause mail to be delivered to non-existent users
- list expansion can take a very long time

Transparency Problems:

- if there was a problem, users usually want to know what it was
- however, the state of a distributed system is harder to determine than the state of a single system

Load-related Problems

Update Replication

- updates are replicated by sending the new entry
- this resends large entries when a minor modification is made (i.e. adding a user to a group)
- solution: just send the update

Authentication

- Groups cause a problem again.
- group expansion can be very slow, and is used for ACLs.
- Solution: 'flatten' nested groups

Reliability Problems

Spare Resources

- spare resources are consumed quietly as the system grows
- lack of spare resources is only noticed during a failure

Inbox Placement

- secondary inbox placement can cause overload during failure
- you won't notice poor secondary inbox placement until it is too late!

Grapevine is Used by Many Applications

- cannot provide service guarantees for specific applications

Conclusion

Cool Stuff:

- Eventual Consistency is tolerable in practice
- Scalability can be achieved by adding more servers

Looking Back:

- disks weren't free back then
- two-level naming not enough
- a bit ambitious regarding what goes in to grapevine registry

Questions

- Is transparency a good thing?
- Does this really scale?