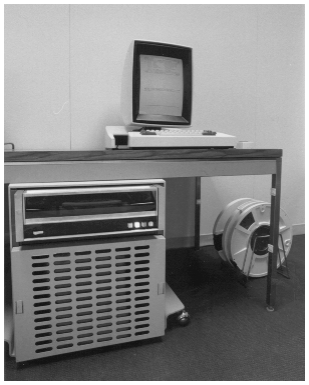
Grapevine

An Exercise in
Distributed Computing

The Xerox Alto

- First personal computer
  - Bit mapped display
  - Mouse
  - WYSIWYG
  - 128K memory, < 10MB of disk
- Enabled development of
  - ethernet
  - RPC
  - distributed mail



Grapevine

- One of the first distributed mail systems
- Reflects most of the good ideas found in today's mail systems
  - uninterpreted mail messages
    - mail for administration
    - arbitrary user interface for composing and reading
  - inbox
  - distribution lists
  - decoupled send, delivery, and receipt
  - authentication
  - access control
  - enterprise servers

Design Goals

- implement a useful distributed system for some purpose
- msg contents are irrelevant
- integrity of system not dependent on integrity of clients
  - bad clients get bad service, but good clients get good service
- reliable delivery
  - msg is delivered, or an error is returned

More Goals

- No single point of failure
  - failed server should not affect service availability
- low interactive delay
  - a few seconds
- marginal non-interactive delay
  - trade for shorter interactive delay
- decentralized administration
- scalable
  - increase capacity by adding more servers
- authentication

Why is Mail a good “starter” application?

- Mail is much simpler than distributed banking or filing
  - can occasionally lose data
    - don’t have to get everything right
  - latency is not critical
    - inherently asynchronous
  - ordering is not important
    - users already accustomed to getting mail “out of order”
    - imposes some restrictions on administrative services
  - naming is trivial
    - mailboxes
  - everything is only a hint
    - distribution lists don’t have to be perfect
    - authentication

Problems faced by designers

- networks have high latency, possibly low bandwidth
- flakey networks
- limited resources on each node
- database is not readonly
- inexperience

How it works

- Large numbers of users
- Small numbers of servers
  - registration
    - maps Rnames to entries
      - entry == [type, more Rname, authinfo, {inboxes}, connectSite]
      - connectSite refers to internet address.
    - names are hierarchical
      - Name.Registry
        - weiser.Parc@xerox.com
        - smtp gets me to xerox.com
        - grapevine gets me to registry Parc, name weiser
    - why not reflect properties directly in name space?
      - clutter up name space
      - too many queries. properties reflect nice clustering
  - message forwarding and repository

Message Acceptance

- Synchronous
- GrapevineUser establishes connection with any message server
  - for each message, GU sends passwd, fromName, recipient list
  - MS authenticates and creates a postmark
    - postmark is unique ID based on host.time of MS
    - easy retransmission detection
    - approximate message ordering

Transport and Buffering

- Asynchronous
- MS maps recipient list to individuals
- For each recipient, msg is forwarded to recipient's inbox or retained locally (basecase)
- Each message server follows the same algorithm
  - ensures that the message will be reliably broadcast to all inboxes, even if they move
    - best guess routing
    - tolerates inconsistencies in the data base
  - postmark allows duplicates to be removed

Hierarchies

- System is used recursively
  - Registration database is stored in the registry
  - all names of the form *.gv exist in all registries
    - foo.gv tells you where to find the registration information for user fred.foo
  - Use of groups
    - gv.gv -> [Rnames of groups of registries]
    - reg.gv -> [Rnames of servers that contain the registry reg]
- Why?
  - system can be queried in same way as users
  - mail transport can be used to administer the system

Resource Location Algorithm

- To locate f.r,
  - contact any grapevine server, say cabernet.gv
  - ask server to produce membership for r.gv
    - “who are the registries that know about r?”
  - contact any server s in r.gv
    - use the connectSite info in property list
  - ask s if it knows about f.r
  - 2 “short” messages to determine failure

Optimizations

- Simplifying guess
  - Any registration server contains the entry for a particular name
  - Choose “any” == “close”
- Only run “real” algorithm if guess is wrong
- Why does this work?
  - message locality
  - registry servers can cache registry information
- Message servers can cache inbox location information
  - faster transport
  - reduced load on registration servers
  - “not here” invalidation

Grapevine as a replicated database

- Supports updates and queries
- Eventual consistency
  - all copies will have the same new value following an update to one of them.
  - inconsistencies can be detected by direct query
    - but generally not a problem
    - when might it be a problem?
- Why not atomic updates?
  - all servers would have to be up
    - otherwise actions would have to be temporally ordered behind updates
      - failure of a few nodes causes long delivery delays
    - or, nodes would have to be removed from service until made consistent

Important Techniques Demonstrated by Grapevine

- replication
- caching
- hints
- decoupled service
- independent failure mode
- hierarchies
- recursive implementation
- timeouts for revocation
- authentication/encryption
- separate service from interface
- clerks and servers
- arbitrary, but consistent ordering
- forwarding pointers for relocation
- flexible service structure
- eventual consistency
- don’t solve hard non-problems
- uniform name space
- separate concepts of naming and addressing
- simple administration

General Lessons

- Global updates
- Exploit human nature
- The world is a distributed database
- Separate naming from addressing
- Follow social processes
- Caches and hints vs. “shortcut mechanisms”
- Software, not hardware is what fails
- Use what you build. Build what you use.

Questions

- Where will grapevine not scale?
- Why partition registry servers from message servers?
- Is eventual consistency suitable for all Rnames?