Lecture 2: Intro to DS structure and Grapevine

1. Terminology
    a. Internet == interconnected LANs. Only connects one company, not whole world
2. Notes from reviews:
    a. Contributions:
        i. list of lessons
    b. Flaws:
        i. failures scales with size of system – more nodes, more likelihood of failure
            1. they note this with link length
        ii. Writing? need more illustrations?
        iii. Was consistency model too weak? Does it need strong consistency?
        iv. Scaling by dividing roles (messaging vs naming) limiting
        v. Not discuss security – is this a writing flaw or a technical flaw?
        vi. Need admin to tolerate some flaws, such as corrupt disk
3. My flaws:
    a. Nested groups make some things harder, like checking membership. Also need to prevent loops
    b. Should provide stronger consistency guarantees to admins, such as session guarantees that they will see the effect of their edits or tell them they may not (e.g. do binding)
    c. Use of ACL caching makes distribution lists hard to use for security, since cannot revoke access immediately (e.g. when you fire someone)
    d. Remailing solution (make user handle it) could be better – could provide efficient bulk transfer for moving an entire mailbox.
        i. shows challenge of relying on high-level abstractions for a use outside their goal; they can be too expensive

Overview of distributed systems

e. Desirable Properties
    i. **Fault-Tolerant**: It can recover from component failures without performing incorrect actions.
    ii. **Highly Available:** It can restore operations, permitting it to resume providing services even when some components have failed.
    iii. **Recoverable:** Failed components can restart themselves and rejoin the system, after the cause of failure has been repaired.
    iv. **Consistent:** The system can coordinate actions by multiple components often in the presence of concurrency and failure. This underlies the ability of a distributed system to act like a non-distributed system.
    v. **Scalable:** It can operate correctly even as some aspect of the system is scaled to a larger size. For example, we might increase the size of the network on which the system is running. This increases the frequency of network outages and could degrade a "non-scalable" system. Similarly, we might increase the number of users or servers, or overall load on the system. In a scalable system, this should not have a significant effect.
    vi. **Predictable Performance:** The ability to provide desired responsiveness

in a timely manner.

  vii. **Secure:** The system authenticates access to data and services [1]

f. Failure Types
  i. **Halting failures:** A component simply stops. There is no way to detect the failure except by timeout: it either stops sending "I'm alive" (heartbeat) messages or fails to respond to requests. Your computer freezing is a halting failure.
  ii. **Fail-stop:** A halting failure with some kind of notification to other components. A network file server telling its clients it is about to go down is a fail-stop.
  iii. **Omission failures**: Failure to send/receive messages primarily due to lack of buffering space, which causes a message to be discarded with no notification to either the sender or receiver. This can happen when routers become overloaded.
  iv. **Network failures**: A network link breaks.
  v. **Network partition failure**: A network fragments into two or more disjoint sub-networks within which messages can be sent, but between which messages are lost. This can occur due to a network failure.
  vi. **Timing failures:** A temporal property of the system is violated. For example, clocks on different computers which are used to coordinate processes are not synchronized; when a message is delayed longer than a threshold period, etc.
  vii. **Byzantine failures**: This captures several types of faulty behaviors including data corruption or loss, failures caused by malicious programs, etc. [1]

g.

Eight fallacies: Generally, LAN conditions don't always exist

a. The network is reliable.
  i. What if network is 5 nines reliable – 99.999%. If you send a gigabit of data
  ii. Network can fail for a variety of reasons: backhoes, operators, software failures
  iii. How often is our department cut off from the Internet, or are you at home?

b. Latency is zero.
  i. "*But I think that it's really interesting to see that the end-to-end bandwidth increased by 1468 times within the last 11 years while the latency (the time a single ping takes) has only been improved tenfold. If this wouldn't be enough, there is even a natural cap on latency. The minimum round-trip time between two points of this earth is determined by the maximum speed of information transmission: the speed of light. At roughly 300,000 kilometers per second, it will always take at least 30 milliseconds to send a ping from Europe to the US and*

back, even if the processing would be done in real time."

ii. *Matters most when waiting for a response (round-tripd elay)*

c. Bandwidth is infinite.
   - iii. Getting better, definitely
   - iv. Problem comes not from a single client, so much, but from many clients acting simultaneously (e.g. refreshing every X minutes)
   - v. Wide-area bandwidth limited by TCP/IP and losses
     1. At 40 msec RTT and 0.1% (1 in 1000 packet) loss, TCP/IP capped at 6.5 Mbps.
     2. To reach 500 Mbps, need 3×10^{-7} error rate

d. The network is secure.
   - vi. Example: FTP sends password, username in cleartext
   - vii. E.g. MS Windows RPC did not validate format – assume correct. Malformed packet would crash server
   - viii. Attacks:
     3. IP injection
     4. Snooping
     5. Denial of service
     6. Dictionary attacks
     7. Malware on client desktops (see Google in China), means firewalls aren't enough

e. Topology doesn't change.
   - ix. Machines move, to different networks, different routes
   - x. Can't statically say how to route things, where servers are, etc.
   - xi. So: use names for indirection (e.g. dns, not ip addresses)
   - xii. So: use discovery: ask a service for the best server to use

f. There is one administrator.
   - xiii. Cannot change everything at once
   - xiv. Cannot change everything at all – e.g. could change server settings but not all client settings

g. Transport cost is zero.
   - xv. Network is not free – provided in this department, but for real systems someone must pay for it

h. The network is homogeneous.
   - xvi. Latencies, reliability, distances vary
   - xvii. E.g.: DSL, dialup, LAN clients, mobile
   - xviii. Different speeds, latencies, prices

4.

Grapevine

a. Name server & email system
   - i. Like Windows Active Directory + Exchange email system
     1. Actually, almost identical design

        ii. Name server maps user names to mailboxes and groups to members (other groups or users)
            1. Key piece of almost any distributed system – how do you name the entities on the system so programs & users can refer to them
            2. What a name means is an important question
                a. It it an address where you can send messages
                b. Is it an indirection that can be used to look something up?
                c. What are uniqueness properties? E.g. within a domain or globally unique?
                d. Can they change?
                    i. What if used for access-control lists?
                    ii. What if used in distribution lists?
                    iii. If they can change, typically have an unchanging internal version (e.g. user ID in Unix, SID in Windows)
        iii. Email system buffers and delivers messages to inbox, where user downloads them
        iv. Note: these are separate services that can be used independently
            1. name system can use email message delivery for replication
            2. email can use name system for looking up where to deliver things
            3. File systems can use name system for access control & authentication
        v. Separate admins in each registry (domain)
    b. What is the scale of this system? An enterprise
        i. 10,000 users
        ii. 30 servers
        iii. A dozen sites
        iv. QUESTION: Is it reasonable to assume a limit to how big you want something to scale?
    c. QUESTION: What is envisioned environment?
        i. Internet (interconnected LANs) at a single large organization with single management
        ii. Mix of high speed (lan, 56 kb, 8kb links)
    d. What are goals?
        i. Scale to many users
        ii. Scale by adding machines, not bigger machines
            1. "fixed size" does not mean never buy larger, but not assume you can scale by buying bigger machines
        iii. User can always send a message
        iv. Tolerate failures of any machine
        v. Decentralized administration
        vi. Large range of user sets – small to very large
    e. QUESTION: What problem does this solve?
        i. How to scale a service horizontally (adding more machines) rather than

vertically (bigger machines)

ii. How?
  1. Replicate for reliability
  2. Hierarchically partition data (names, users) to different machines
  3. Avoid global state
  4. Avoid rigid consistency
     a. Write things in one place, replicate later
     b. Don't replicate message contents
     c. QUESTION : Why does this work?
        i. Humans involved can handle inconsistency
        ii. Programs must be coded to not expect inconsistency
        iii.

iii. What state doesn't do this?
  1. Set of machines – replicated globally
  2. Originally, members of a distribution list
     a. Why not?
     b. Interactions of users are not local – more users indicates bigger lists, bigger groups of users

iv. As system grows, what makes it scale?
  1. Admin must decide how to partition things, where to add servers, how to incorporate hierarchy into big groups
  2. Admins may relocate users, add new servers, etc.

v. How make it reliable?
  1. Replicate services
  2. Focus on high availability when needed: sending messages
     a. Not everything is high availability – all mail may not always be available
  3. QUESTION: Is replication enough?
     a. ANSWER: No: need capacity to tolerate added load after a failure, or need to shed load (drop requests) to make up for lost capacity

f. QUESTION: What do they give up to make this work?
  i. Guarantees: latency can be long
  ii. Consistency: may make an update that is not immediately visible, may have duplicate messages delivered

g. QUESTION: What are the lessons for distributed systems?
  i. Load can scale beyond capability of a single machine (or a set) leading to congestion collapse
  ii. All state should be partitioned/replicated, avoid global state or having to have all of anything somewhere
  iii. Remote monitoring, local logging helps debug distributed problems
  iv. Makes things fail in an understandable way
     1. Example: file system. Better to have whole directories unavailable

than some files in a directory. Lets users work around
   2. Example: reload/stop button on web browser

v. Replication help hardware reliability but not software reliability
   1. Single software bug can get passed to all machines – client repeatedly connects to servers and infects them
   2. Similar to Amazon datacenter collapse last year – software had a retry bug causing congestion collapse.

vi. Design for on-line maintenance
   1. In a high-availability system, cannot take system offline for debugging.
   2. Should be able to repair a single machine while rest of system keeps working.

h. High-level solutions

   i. Relax consistency
      1. can have actions that were just performed disappear if go to a different replica
      2. Guarantees eventually, given a quiescent functioning system, all values will converge
      3. Why does this work?
         a. Humans can work around
      4. What other stronger (slightly) could you make?
         a. Session guarantees: while connected to a server, all requests will observe the effect of all previous requests. (local causality)
         b. Causality: anybody you connect to will see the effect of your previous requests, no matter where they were sent (causal consistency)

   ii. Partition work
      1. E.g. hierarchical groups
      2. In distributed systems, some knowledge is global. Handle by:
         a. Keep it small and static (slowly changing), loosely consistent

   iii. Replicate for availability
      1. Full copies of naming database
      2. Handling conflicting updates:
         a. Last-writer-wins most of the time – fits model of how humans think in this environment
         b. Not always works – creating a new name
            i. want first-creator wins

   iv. Move data closer to work – not done, but proposed
      1. When sending message to group with users stored on remote computer, could move message to a close message server and do fast, local communication instead of remote communication

        2. For expanding large groups; make it multiple groups on different machines that do local expansion (layer of indirection)
    v. Caching
        1. Used for repeated access checks – don't need to perform same user/group lookups
    vi. Alternate data structures for more efficient access
        1. Store flattened version of nested groups
        2. Like index in database
        3. Partitioning
            a. Database of users is split into registries, each registry can be stored on different machines
            b. Names partition users based on registry name
    vii. Spreading load
        1. Put users mailboxes on different machines to avoid hot spots
        2. Put backup mailboxes for users on a single primary on different machine to avoid overload on failure
        3. PROBLEM: need to manually rebalance users in some case
        4. WHAT IS THE ISSUE:
            a. Need to move their entire mailbox - - lots of state
            b. Issue: can be hard to recover from a bad placement choice, and moving data impacts ongoing load
            c. SO: don't move users just on a failure; only if machine really dies or need to really rebalance.
    viii. Locality / traffic patterns
        1. Try to put users that share email on a single machine
        2. Try to optimize delivery to send messages over slow links only once
        3. Try to put users of a group near the servers hosting that group
    ix. Idempotent operations
        1. Allow duplicate messages; avoids cost of expanding groups completely in delivery
        2. Postmark in message allows duplicate detection in mailbox; adding same message multiple times doesn't do anything more
        3. Can avoid expensive sorting, duplicate detection algorithms
    x. Delta encoding
        1. Distribution lists originally propagated entirely then merged
            a. WHY?
                i. All other objects were done that way, distribution lists were objects
        2. Changes to group memberships sent as deltas (add / remove member) instead of new membership (entire list of members)
        3. Removes need to merge large data objects; just apply change
    xi. Expose internals

        1. Naming convention dictates whether a name on an ACL is a user or group; allows for faster lookups because don't need to expand users
    xii. Input throttling to reduce overload
        1. Servers reject messages when disks are full
        2. Can lead to deadlock
    xiii. Indirection:
        1. Add layer of indirection for group membership to make it faster – not have to lookup all members.
        2. PROBLEM: what does this make hard?
            a. ANSWER: need to prevent loops (not easy)
            b. ANSWER: slow to figure out if a user is a member of a group
                i. so cache flattened list

i. How address the fallacies?
    i. Pretty much took into account every one