34 Distributed Systems

Software system executing on a set of computing devices connected by a network.

Key differences from centralized systems:

- No physically shared memory
- No global clock
- Independent component failures
- Unreliable communication subject to delays (speed-of-light) and limited bandwidth
- Individual components have only partial view of systems state/inputs
- Multiple administrative domains
- Hardware/software heterogeneity

Examples

- Internet services: DNS, NTP (and the Internet itself)
- Distributed file systems: NFS, CIFS, AFS
- WWW, CDNs
- Email (SMTP), News (NNTP)
- Multi-player games, virtual/augmented reality communities
- Corporate networks; industry federations (banking, medical); eGovernment; military
- Clusters, data centers
- Peer-to-peer: BitTorrent, BOINC, Skype

Need to revisit many OS issues

- Distributed synchronization

- Distributed scheduling
- Distributed deadlock detection
- Distributed memory management
- Distributed storage/filesystems

Plus deal with a number of new issues:

- Distributed agreement
- Distributed event ordering
- Failure recovery
- Malicious, selfish, curious components
- Heterogeneity
- Decentralized control

Why distributed systems?

- Resource sharing (e.g., shared storage/compute server/printer)
- Overcome geographic distribution (e.g., WWW, Skype)
- Increased reliability (building reliable services from unreliable components)
- Aggregate resources for scalable capacity, elasticity, economies of scale (e.g., Cloud, HPC Grid, BitTorrent, BOINC)

Challenges

- System design: how to partition responsibility among different nodes. How to coordinate them? Design distributed protocols.
- Failures (communication, hardware, software): Ideally, distributed system should appear as local system that never fails. In practice, impossible to achieve.
- Security: compromised/faulty/selfish/malicious components may intercept/manipulate messages, subvert protocols

- Performance: sometimes with additional constraints: wide-area network latencies, mobile devices/sensor networks have energy constraints, message loss, failures.
- Scalability: E.g., a cluster of file servers. What should throughput vs. number of machines curve look like? In practice you hit bottlenecks, then how should it look? Need to avoid phenomena like thrashing.

Talk about clocks, agreement, RSM as example challenges.

Topic: System Design
Example: Distributed file system. Many clients, access file server over a network, which then stores files on local disk.
Operations? Read file, Write file, Create, Move, etc.
Problem: Server becomes overloaded. Solutions?

1. Caching at the client. Read files recently written. Problem? With concurrency, cache entries may become stale. Solutions?
   - Locking + Invalidations from server. Problem: What if messages take too long to arrive at the client? Will server wait? What if client crashed? Possible solution: Leases.
   - Client checks last modified time before reading cached copy. Tradeoff? More simple and robust, less efficient. What about writes? Write-through or write-back policy?
2. Partition state among 2 servers. How? E.g. each server is responsible for half of the files. But how to ensure good load balancing? Statically? And what if suddenly some files become much more popular?

Topic: Naming
And how does client refer to remote files? Need some name that is translated into a file object. How to choose these names?
Example: A trick similar to web is to embedd server name in the file name, so a name from standpoint of client is:
/dist-fs/x.uni-sb.de/a.txt
Problem? What if location changes? Alternative: use a directory server
/dist-fs/myserver/a.txt

Directory server translates name to address of file server. What if directory server becomes overloaded? Caching helps, but can also use a hierarchical namespace. Do you know examples of hierarchical namespaces?

Topic: Fault Tolerance
Back to single server design, how to survive faults?
Solution: replicate data.
Problem: replica consistency. Don't want to delete file and let it reappear.
Problem: how to ensure failure independence?
Problem: availability in presence of a partition vs. consistency?

Topic: Security
Security challenges now have to be addressed in a distributed setting. E.g., authentication. When I give my credit card number to a company's website, how do I ensure it is really that website I am talking to? PKIs can help, but need to bootstrap trust.
Also many new problems like DDoS / botnets.

Topic: p2p
What if I want to store my files remotely (e.g., for backup) but cannot afford server infrastructure? Peer-to-peer computing can provide a cooperative, volunteer-based solution. These systems can have a huge number of nodes. Are self-organizing. Usually no distinction between clients and servers (nodes have to do a bit of both).
Basic problem: how to partition data?
Then how to locate it?
Real-world problems: e.g., free-riding, need incentives. Sybil attacks, need stronger identities or barrier to entering the system.