1   Distributed Systems

What are distributed systems? How would you characterize them?

- Components of the system are located at networked computers
- Cooperate to provide some service
- No shared memory
- Communication (sending messages): delays, unreliable
- No common clock (makes coordination much more difficult)
- Independent node failure modes
- Hardware/software heterogeneity (e.g., little/big endian)
- Concurrency (an issue that was already present in an OS). Means that we can scale the capacity of the system by adding more resources.

Examples of distributed systems?

- Web
- Email
- DNS
- Peer-to-peer systems (file sharing, CDNs, cycle sharing)

- Distributed File Systems (NFS, ...)

- Sensor Networks

- Akamai CDN

Why do we care about them? What is their purpose? What do we gain from them?

- Resource sharing (e.g., shared file server)

- Overcome geographic separation (e.g., web)

- Increase reliability (build reliable systems from unreliable components)

- Aggregate resources for high capacity (cycles: SETI@home, bandwidth: BitTorrent, disks: Freenet, Amazon's S3)

Challenges.

- System design – how to partition responsibility among different nodes. How to coordinate them? Design distributed protocols.

- Failures – communication, hw, sw. Ideally, distributed system should appear as local system that never fails. In practice, impossible to achieve.

- Security – adversary may compromise machines, manipulate messages, subvert protocols

- Performance – sometimes with additional constrains: wide-area network latencies, sensor networks have energy constrains, message loss, failures.

- Scalability – E.g., a cluster of file servers. How should throughput vs. number of machines curve look like? In practice you hit bottlenecks, then how should it look like? Need to avoid phenomena like thrashing.

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? a. 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. b. 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 onto 2 servers. How? E.g. different users can be assigned different servers. 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 also 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

Most of the problems were mentioned in last lectures, but now have to be solved in a distributed setting. E.g., authentication. When I give my credit card number to a company's website, how do I ensure it's really that website I'm talking to? PKIs can help, but need to bootstrap trust.

Also many new problems like DDoS / bot nets.

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 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 I partition data?

Then how do I locate it?

Real-world problems: e.g., free-riding, need incentives. Sybil attacks, need stronger identities or high barrier to entering the system.