2a-0: Definitions

What is a distributed system?

- (Coulouris) “A distributed system is one in which hardware or software components communicate or coordinate their actions only by passing messages.”
- (Tanenbaum) “A distributed system is a collection of independent computers that appear to the users of the system as a single computer.”
- (Lamport) “You know you have one when the crash of a computer you’ve never heard of stops you from getting any work done.”

All of these get at different aspects of the issue ...

2a-1: Advantages of a distributed system

- Can share expensive resources or data
- Economics
  - A collection of PCs can provide better price/performance than a single mainframe.
- Speed
  - A distributed system will often have more computing power than a single mainframe.
- Inherent distribution
  - Often, your data/users/resources are geographically distributed

2a-2: Advantages of a distributed system

- Reliability
  - If one node fails, the rest of the system can continue
- Incremental growth
  - Components can be added or replaced in small increments.

2a-3: Disadvantages of distributed systems

- Software design is much more complicated.
  - Lack of appropriate tools/languages
  - Disagreement on principles: how much should users know about the system? How much the system handle on a user’s behalf?
- Potential network saturation
- Privacy and security issues
  - Allowing resources to be shared can lead to data leakage
- Extra sysadmin work

2a-4: Design Issues

- Transparency
- Flexibility
- Dependability
- Performance
- Scalability
2a-5: **Transparency**

- The goal of transparency is a *single-system image*
  - From the user's POV, it looks like a single machine.
- Types of transparency:
  - Location transparency - Users cannot tell where their resources are actually located.
  - Migration transparency - Resources can move without changing their names.
  - Replication transparency - the number of copies of a resource is hidden from users.
  - Concurrency transparency - Users can share resources without being aware of the presence of other users.
  - Parallelism transparency - A task can be run on multiple machines without the user being aware of it.

2a-6: **Transparency**

- Is transparency always a good thing? What is the downside?

2a-7: **Flexibility**

- Flexibility refers to how easy or difficult it is to change or reconfigure a system.
- The research question is how to best provide flexibility.
- In the OS world, this debate shows up in the comparison of monolithic kernels and microkernels.
  - Monolithic kernel - Provides most services on its own
  - Microkernel - Only handles a simple set of services. Most other services are implemented at the user level.
- Microkernel is very flexible and modular; services can be added, deleted, or moved without much reconfiguration.
- Monolithic kernel gives better performance.

2a-8: **Reliability**

- There are several different aspects of reliability:
  - Availability: what fraction of the time is the system usable?
  - Integrity: Data must be kept consistent. (this sometimes clashes with availability)
  - Security: Unauthorized usage must be prevented.
  - Fault tolerance: How unpleasantly does the system fail? Is data lost? Can recovery happen?

2a-9: **Performance**

- Performance is trickier than it appears.
- Lots of possible metrics
  - Response time
  - Throughput
  - System utilization
  - Network capacity
- Typically, communication costs dominate
- This leads to a *coarser-grained* parallelism that we would see in a parallel computer.

2a-10: **Distributed Programming Paradigms**

- Client/server model
- Remote Procedure Call
- Distributed Shared Memory
- Group Communication/multicast/Peer-to-peer
- Distributed objects
- Web services
2a-11: Communication paradigms

- Asynchronous: there is no bound on message delay or clock drift
- Synchronous:
  - Known upper bound $b$ on message delay
  - Every process $p$ has a local clock $C_p$ which drifts at a rate of $r > 0$ and $\forall p$ and $\forall t > t'$:
    \[
    (1 + r)^{-1} \leq \frac{C_p(t) - C_p(t')}{f_p} \leq (1 + r)
    \]
  - In English, clock drift has an upper and lower bound.
  - Also, bounds on the amount of time needed for a process to execute a single step.
- Synchronous communication allows you to implement approximately synchronized clocks, even in the presence of failure.

2a-12: Types of process failure

- In looking at different protocols and algorithms, we’ll want to know what types of failure they are resistant to.
  - Failstop (or crash): Process halts and remains in that state; failure can usually be detected.
  - Send omission: a process fails to send messages, or halts in the middle of sending.
  - Receive omission: a process fails to receive a message properly, or halts while receiving.
  - General omission: combination of send and receive omission
  - Timing failure: process commits a general omission failure, or its clock drift exceeds allowable bounds.

2a-13: Types of Communication Failure

- We also must consider failures that happen in the network:
  - Crash: a link stops completely.
  - Omission: A link fails to transmit some of its messages.
  - Byzantine: A link can exhibit any possible behavior, including generating spurious messages.
- Note: A Byzantine failure can be treated the same as an attacker/intruder.

2a-14: Summary

- There are lots of desirable properties and design issues for distributed systems.
  - Performance, scalability, reliability, flexibility, transparency
  - Often, we must sacrifice one for another
  - Some (e.g. Parallel transparency) are not possible with today's technology.
- Communication can be either synchronous or asynchronous
- Characterizing types of failure will help us identify what our algorithms and systems can and cannot stand up to.