Final Notes

Kurtis Jantzen

Definition A collection of independent computers that appears to its users as a single coherent system.

Goals of a Distributed System

1. Connecting users to resources (availability & security)
2. Transparency
3. Openness
4. Scalability

Transparency

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be replicated in different places
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Scalability

- Size-scalability
- Geographic-scalability
- Administrative-scalability

Scaling Techniques

1. Hiding Communication Latencies
2. Distribution
3. Replication

Architectural Styles

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

Lamport's Timestamp Algorithm

```
P_i: (initially TS_i= 0)
    On event e:
        Case e is send(m), where m is a message
            TS_i= TS_i+ 1
            m.TS = TS_i
        Case e is receive(m), where m is a message
            TS_i= max(TS_i, m.TS)
            TS_i= TS_i+ 1
        Case e is any other event
            TS_i= TS_i+ 1
        e.TS = TS_i  // timestamp e
```

Vector Timestamp Algorithm

```
P_i: (initially VT_i= [0, ..., 0])
    On event e:
        Case e is send(m), where m is a message
            VT_i[i] = VT_i[i] + 1
            m.VT = VT_i
        Case e is receive(m), where m is a message
            for j = 1 to N:  // vector length
                VT_i[j] = max(VT_i[j], m.VT[j])
            VT_i[i] = VT_i[i] + 1
        Case e is any other event
            VT_i[i] = VT_i[i] + 1
        e.VT = VT_i  // timestamp e
```

Attiya and Welch's Totally Ordered Broadcast

- Use Lamport's TS
- Associate a priority queue (by timestamp) with each site
- Broadcast (including self) the update
- When update is received enqueue it
- Broadcast (except to self) ack if update is not from self
- Apply an update locally, only if it is at the head of the queue and it was ack'ed by every body else

Distributed Computing Models

Synchronous Distributed Computing Model

- Synchronous model:
  - Process execution speed: bounded
  - Message transmission delay: bounded
  - Clock drift rate: bounded
- Useful for analysis of algorithms
- Can be built if processes can be guaranteed
  - Enough CPU cycles and N/W capacity
  - Clocks with bounded drift rates
- Can make use of time-outs to detect failures

Asynchronous Distributed Computing Model

- Asynchronous model:
  - Process execution speed: not bounded

  - Message transmission delay: not bounded
  - Clock drift rate: not bounded
- More realistic (e.g Internet)
- Harder
- More general
- Cannot make use of time-outs to detect failure

Critical Section Correctness

- Mutual Exclusion: safety
- Deadlock-Freedom: progress
- Starvation-Freedom: fairness

Dependability Measures

- Availability: A system is ready to be used immediately
- Reliability: A system can run continuously without failure
  - A system goes down 1ms/hrhas an availability $> 99.99\%$, but is unreliable
  - A system that never crashes but is shut down for a week once every year is 100% reliable but only 98% available
- Safety: System fails, nothing serious happens
- Maintainability: How easy it is to repair a system
  - System should be able to fix itself

Fault Types

- Transient (appear once and disappear)
- Intermittent (appear-disappear behavior)
- Permanent (appear and persist until repaired)

Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure - Receive omission - Send omission	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval

Type of failure	Description
Response failure - Value failure - State transition failure	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure (Byzantine failure)	A server may produce arbitrary responses at arbitrary times