CS 5204
Operating Systems
Lecture 9

Godmar Back

Announcements

- Still working on project proposals
  - Look for reply email from me with word “approved” in it
- Out of town Oct 2-6, Oct 16-18:
  - No class on Oct 3 & 5;
  - Presentations move back; I’ve updated reading list with new tentative dates
- Midterm: Oct 17

Plan for Today

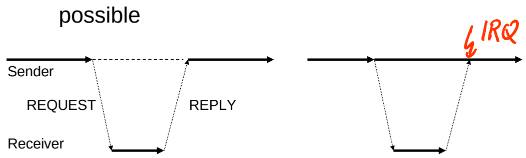
- Techniques for scalability
- Consistency models
- Openness & Flexibility
- Discussion on End-to-End argument

Continuing on Scalability

- Recall: main problem that limited scalability was centralization (in services, in data, in centralized algorithms)
- Aside from using decentralized algorithms (where possible), what else can be done to increase scalability?

Scaling Techniques

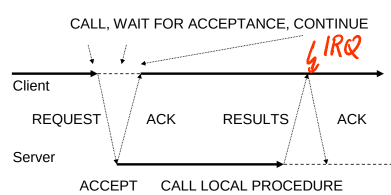
- Hide communication latencies
  - Use asynchronous communication whenever possible



synchronous vs. asynchronous

Deferred synchronous RPC

- Combines two asynchronous RPC.

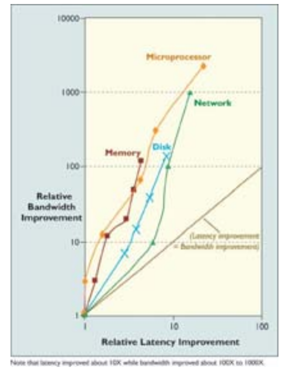


Scaling Techniques (cont'd)

- Minimize communication
  - Through distribution
  - Through piggybacking
  - Through careful placement of computation
  - Examples of these?
- Note shift in focus over time
  - as bandwidth becomes cheaper stronger focus on avoiding relative latency penalty

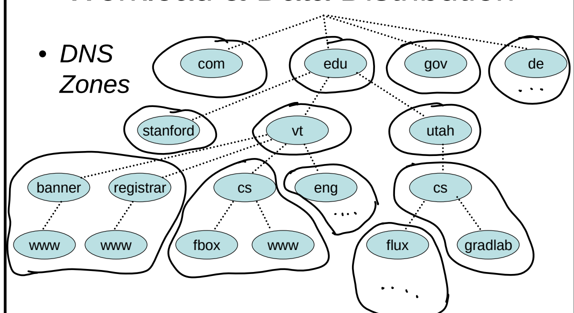
Latency lags Bandwidth

- Patterson [2004]
- Answers:
  - Caching
  - Replication
  - Prediction



Workload & Data Distribution

- *DNS Zones*



Consistency Models

- Scalability goal when using caching/replication:
  - minimize synchronization requirements
  - use relaxed consistency models when possible
- Consistency Models
  - Strict consistency
  - Sequential consistency; linearizability
  - Causal consistency
  - FIFO consistency
  - Weak consistency
    - Refinements: Release consistency, Entry consistency

Strict Consistency

- Any read on a data item x returns the value most recently written to x.
- Ideal model for programmers
  - Requires global clock (example: leases)

P1:	W(x)a	
P2:		R(x)a

(a)

P1:	W(x)a		
P2:		R(x)NIL	R(x)a

(b)

Sequential Consistency

- The result of the execution is the same as if reads and writes were executed in some sequential order; reads and writes of each process are executed in program order within that sequence

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)b	R(x)a

(a)

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)a	R(x)b

(b)

Sequential Consistency (cont'd)

- Note that sequential consistency requires
  - Maintaining constraints by program order
  - Data coherence within global sequence ("history")
- Updates must be synchronous
  - Write update vs. write invalidate
- Performance: it has been shown that r+w > t where r: read time, w: write time, t: message time
  - Optimizing writes makes reads slower & vice versa

Causal Consistency

- Not all processes need to see all writes in the same order
  - Causal consistency – only if writes are causally related (as in happens before relship)

P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

This sequence is causally consistent, but not sequentially or strictly consistent

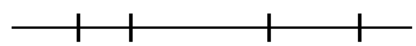
Causal Consistency (II)

- Example of a violation: W(x)a happens before W(x)b, so P3 and P4 must see results in same order

P1:	W(x)a				
P2:		R(x)a	W(x)b		
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

Weaker Consistency Models

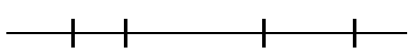
- Idea: don't propagate all updates, only propagate consistent state between updates to distributed synchronization variables



- Provide sequential consistency, but only with respect to sync points

Release Consistency

- Propagate writes when releasing a distributed synchronization variable



- Can be done eagerly or lazily
- Also possible: entry consistency
  - Only update those that will be accessed after entry

E2E (cont'd)

- Note that endpoint != application
  - Endpoint can also be a layer
  - How to identify the endpoints?
- Reasons for violating E2E:
  - Performance
  - Cost
  - Software engineering/Code Reuse (?)
- E2E is only a guiding principle, a type of "Occam's Razor"

Summary

- Transparency goal
- Techniques for scalability
- Consistency models
- Fault tolerance approaches & results