

Agenda

- Last time (Tues Apr 24 -- Google day)
 - The Anatomy of a Large-Scale Hypertextual Web Search Engine (Computer Networks, 1998)
 - Google File System (SOSP 2003)
- Thurs Apr 26: no class
- Tuesday May 1
 - Finish off Google (map-reduce)
 - Class Wrap-up (and HW#5 Due)
- Fri Apr 11 (2-5pm): Final (closed books, closed notes)

CS451: Distributed Systems (Spring 2007)

Questions for CS451

- Is it good for other applications?
 - Is this going to replace NFS?
- Why was this paper accepted?
- Pros/cons of the centralized master?
- Why does Google not use the standard Linux file system?
- Why does Google not use the POSIX API?
- What does Google lose by having its own file system?
- Did the paper say where exactly this is used within Google?

CS451: Distributed Systems (Spring 2007)

GFS vs. NFS vs. AFS

- So why did the google person justify the need (to his/her boss) for GFS?
 - What was their file system before this?
 - They indicate that they use Linux a lot, so it was probably NFS.
 - Was there something about NFS that wasn't "working"?
 - Hmmm... this is tricky. Consider: use patterns (how they read/write data)
 - Is the non-POSIX API justified?
 - Note: it's additive (existing Linux apps will run -- they just won't utilize the "extra" functionality of snapshot and record_append)
 - Bottom line: Why do they need a GFS?
 - Most likely reasons: speed, atomic append... other reasons?
- Why is the google data replicated? For access speed? For reliability/availability? (Can google just regenerate its file system in a few days?)
- Is GFS useful for other environments? Hmmm....

CS451: Distributed Systems (Spring 2007)

Paper #2: MapReduce

- Some computations cannot be parallelized (e.g., Fibonacci -- why?)
- Some computations are "easy" to parallelize
 - A large amount of data that must be processed (e.g., master/worker)
- MapReduce: an abstraction that allows Google engineers to perform simple computations while hiding the details of parallelization, data distribution, load balancing and fault tolerance.

CS451: Distributed Systems (Spring 2007)

Map/Reduce

- Map/Reduce
 - Programming model from Lisp (and other functional languages)
- Many problems can be phrased this way
- Easy to distribute across nodes
- Nice retry/failure semantics

CS451: Distributed Systems (Spring 2007)

Map in Lisp (Scheme)

- (map f list [list₂ list₃ ...])
- (map square '(1 2 3 4))
(1 4 9 16)
 - Unary operator
- (reduce + '(1 4 9 16))
(+ 16 (+ 9 (+ 4 1)))
30
 - Binary operator

CS451: Distributed Systems (Spring 2007)

Map-Reduce

- **Functional programming vs. imperative programming**
 - Imperative: manipulation of *state* ("now set your variable named *X* to 15...")
 - Functional: "what I want done", not "how to do it"
- **Is Map-Reduce more toward functional programming or imperative programming?**
- **How broadly applicable is map-reduce?**
- **Should I have given a programming assignment in Map-Reduce?**

CS451: Distributed Systems (Spring 2007)

Class Wrap-up: Goals of this class

- **Balance theoretical aspects with practical**
- **Practical: Programming Assignments**
 - PA#1: IM client via sockets
 - PA#2: Replicated IM server
 - PA#3: Web services tutorial (.NET FW 3.0, Apache Axis2, gSOAP)
 - PA#4: P2P file sharing system
 - [PA#5: just book questions]

CS451: Distributed Systems (Spring 2007)

What did we do in CS451?

<u>Architectures</u> Client/Server P2P	<u>Programming Models</u> Sockets RPC (e.g., SUNRPC) Dis Objects (e.g., Java RMI) Web Services (RPC) Map/reduce (??)	<u>Architectures</u> Client/Server P2P
<u>Fundamentals / Underpinnings</u> Security Networks (emph: TCP) Time Failure Models / Fault Tolerance Name Services (e.g., DNS) Transactions Replication	<u>"Other"</u> DFS	<u>Algorithms</u> Clock sync Logical clocks Global states Snapshot – Chandy/Lamport Dis Mutual Exclusion Elections Multicast Consensus
	<u>Next Semester?</u> Mobile devices? Multimedia? Async Web services? Dis Transactions?	

CS451: Distributed Systems (Spring 2007)

Before Midterm

- **Chapter 1: Characterization of Distributed Systems**
- **Chapter 2: System Models**
- **Chapter 3: Networking and Internetworking**
- **Chapter 4: Interprocess Communication**
- **Chapter 5: Distributed Objects and Remote Invocation**
- **Chapter 7: Security**
- **Chapter 8: Distributed File Systems**
- **Chapter 9: Naming**

CS451: Distributed Systems (Spring 2007)

Chapter 19: Web Services

- **XML**
- **SOAP**
- **WSDL**
- **UDDI (?)**
- **URI, URL, URN**

CS451: Distributed Systems (Spring 2007)

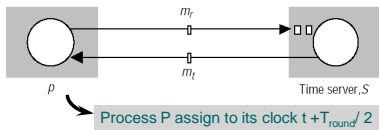
Chapter 10: Peer-to-Peer

- **Why P2P?**
 - High-availability, fault-tolerance, scalability
- **Applicability**
 - Sharing of content, sharing of storage, sharing of CPU time
- **Case study: Gnutella**
 - Messages: Ping, Pong, Query, QueryHit, Push (firewalls)
 - TTL
 - Problems: limited horizon, scalability, DOS attacks, Privacy attacks

CS451: Distributed Systems (Spring 2007)

Chapter 11: Time and Global States

- NOT 10.6 (Distributed debugging)
- Synchronizing physical clocks
 - Internal vs. external
 - Christian's method (external, not synchronous system)



- Berkeley algorithm (internal)
- NTP (Multicast, procedure-call, symmetric mode)

CS451: Distributed Systems (Spring 2007)

Chapter 11: Time and Global States

•Logical time and logical clocks

•"happened-before"

HB1 If \exists process P_i : $a \rightarrow_i b$, then $a \rightarrow b$

HB2: $\forall m$, $\text{send}(m) \rightarrow \text{receive}(m)$

HB3: If a , b and c are events such that $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

- Two events a and b are concurrent if $a \not\rightarrow b$ and $b \not\rightarrow a$
- Implementing logical clocks: via timestamps
 - How do we get total ordering? (the same timestamp can be assigned to different events): ANS: use "conventional ordering of processes"

CS451: Distributed Systems (Spring 2007)

Chapter 11: Time and Global States

•Global states

- Why? Distributed GC, Distributed deadlock detection, Distributed termination detection, Distributed debugging

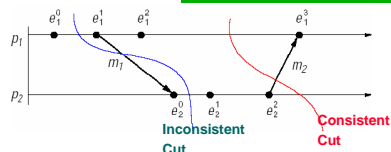
CS451: Distributed Systems (Spring 2007)

Global States

- $\text{History}(P_i) = h_i = \langle e_i^0, e_i^1, e_i^2, \dots \rangle$

$h^k = \langle e_i^0, e_i^1, e_i^2, \dots, e_i^k \rangle$: finite prefix of Process' history

- **Cut**: subset of the global history of a system's execution $C = h_1^k \cup h_2^k \cup \dots \cup h_n^k$
- Cut C is **consistent**, for each event it contains, it also contains all the events that happened-before that event:



- A global consistent state corresponds to a consistent cut

CS451: Distributed Systems (Spring 2007)

Global States: Snapshot algorithm of Chandy/Lamport

•What's the basic idea?

•Neither processes nor channels fail; FIFO ordering of message delivery; strongly connected processes; any process can start global snapshot; processes act "normally" during snapshotting;

•On receiving marker on Channel C:

- if (P_i has not yet recorded its state) then
 - [1] it records its process state now;
 - [2] records the state of C as the empty set;
 - [3] turns on recording of messages arriving over other incoming channels;

• else

- [1] P_i records the state of C as the set of messages it has received over C since it saved its state.

• Marker sending rule:

- After P_i has recorded its state, for each outgoing channel C, P_i sends one marker message over C (before it sends any other message over C).

- Does it always generate a consistent global state? Why?
- Does it always terminate?

CS451: Distributed Systems (Spring 2007)

Chapter 12: Coordination and Agreement

•NOT 11.4.3 Ordered Multicast

•Distributed Mutual Exclusion

•Requirements: Mutual exclusion, progress, no starvation (aka Safety, Liveness, and Ordering)

•Algorithms: Central Server, Ring-based, multicast w/logical clocks, Maekawa's voting algorithm (deadlock?)

•How do we compare algorithms?

- Implementation complexity, # messages, latency for entering critical section, # processes that must agree, robustness/problems

•Election Algorithms

•Ring-based, Bully

•Group Communication

•Basic multicast, reliable multicast

•Consensus

•Properties to satisfy: termination, agreement, integrity

•Byzantine Generals

CS451: Distributed Systems (Spring 2007)

Chapter 13: Transactions and Concurrency Control

- **Basic idea: all-or-nothing, isolation, serial equivalence**
- **Concurrency Control Protocols**
 - Locking (read locks, write locks; two-phase locking)
 - Optimistic Concurrency Control
 - What's the basic idea in each? What are the pros/cons of each?

CS451: Distributed Systems (Spring 2007)

Chapter 15: Replication

- **Replication of data: Maintenance of copies of data at multiple computers.**
- **Goals:**
 - Enhanced performance.
 - Increased availability.
 - Fault tolerance.
- **Some potential requirements:**
 - Replication transparency.
 - **Consistency:** if a copy is modified, how and when the others are updated determines "the price of replication"
- **System models**
- **Fault tolerance basics**
 - Passive replication vs. active replication

CS451: Distributed Systems (Spring 2007)

Google Day

- **Basic problem statement of google (Indexing/Searching the Web)**
- **Architecture of google**
- **Google File System (GFS)**
 - Use-case
 - Architecture
- **Map-Reduce**
 - Basics

CS451: Distributed Systems (Spring 2007)

Final Exam!

- **Friday, May 11, 2pm-5pm, this classroom**
- **CLOSED BOOKS, CLOSED NOTES**
- **Structured the same basic way as the Midterm**
- **~2/3 Post March 20**
- **~1/3 Pre-March 20**
 - <http://www.cs.virginia.edu/~humphrey/cs451/Assignments/MidtermPrep.html>

CS451: Distributed Systems (Spring 2007)