Distributed File Systems

Distributed File Systems

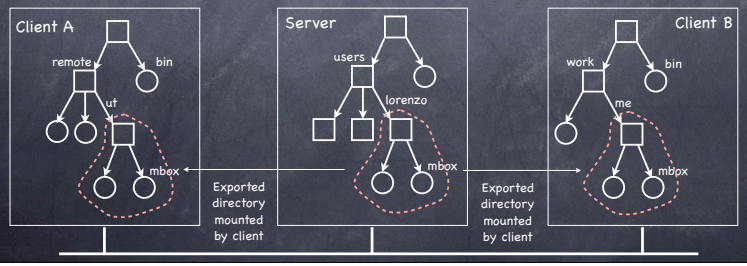
- Provide transparent access to files stored on remote disks
- Issues
  - Naming: How do we locate a file?
  - Performance: How well does a distributed file system perform as compared to a local file system?
  - Failure handling: How do applications deal with remote server failures?
  - Consistency: How do we allow multiple remote clients to access the same files?

Naming

- Two approaches
  - Explicit naming: <file server: file name>
    - e.g. windows file shares
  - Implicit naming
    - Location Transparency: file name does not include name of server where file is stored
    - Location Independence: file name does not change even if file location does

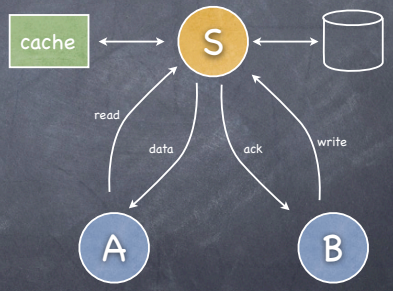
But how is a server identified?

- Most common solution
  - static, location-transparent mapping
  - NFS Mount protocol
    - mount/attach remote directories as local directories
    - maintain a mount table with directory >> server mapping
      - /home/lorenzo >> zathras:/vol/vol0/users/lorenzo



Performance: the simple view

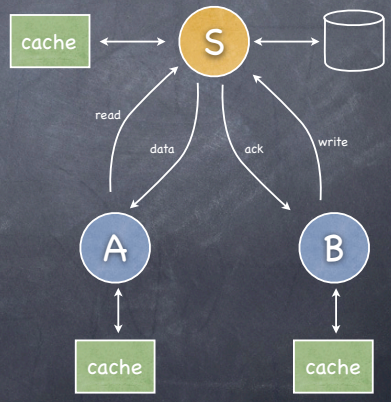
- Straightforward RPC
  - use RPC to forward every file system request to remote server
  - server executes request locally
  - responds back
- Advantage
  - clients see consistent view of file system
- Disadvantage
  - poor performance
    - network traffic
    - server a bottleneck



Client caching!

SUN's NFS (Networked File System) 1984

- Use caching to reduce network load
- Advantage
  - clients can execute reads, writes, etc locally
    - reduce network load
    - improve client performance
  - cache blocks, headers, etc. at both server and clients
    - in memory
- Disadvantages?



SUN's NFS Issues: Consistency

- What if multiple client share the same file?
  - easy if both are reading...
  - what happens on writes?
- Client-initiated weak consistency protocol
  - clients poll servers periodically to check if file has changed
  - when file is changed, server is notified
    - data in server's memory is lost
  - when clients notices file has changed at server, it retrieves new version
  - what if multiple clients write to the same file?
    - generally, using a delayed write-back policy
  - HTTP uses essentially the same protocol!
- Alternative: server-initiated protocol

SUN's NFS Issues: Failures

- What if server crashes?
  - data in server's memory is lost
  - shared state across RPCs is lost
    - what happens if server crashes after seek?
  - commands may be retried
- What if client crashes?
  - data in client's cache is lost

The eternal sunshine of the stateless server

- Server is stateless
  - keeps no state about clients or open files
    - except as hints to improve performance
  - each file request provide server with all info needed to complete operation
    - ReadAt(inode,position), not Read(openFile)
- Operations are as much as possible idempotent
  - "at least once" semantics
  - what happens on "remove"?
- Server failures are transparent to clients
  - when server fails
    - hangs until server recovers or
    - eventually crashes
  - why not return an error?

NFS: Summary

- Key features
  - location transparent naming
  - client-side and server-side caching for performance
  - stateless architecture
  - client-drive weak consistency
- Advantages
  - simple
  - highly portable
- Disadvantages
  - lack of strong consistency

scaling

Consistency

- Distribute
  - partition data and computation across multiple machine
    - Java applets, DNS, WWW
- Replicate
  - make copies of data available at different machines
    - mirrored web sites, replicated fs, replicated db
- Cache
  - allow client processes to access local copies
    - Web caches, file caching

The Model

- Shared data is kept in a data store
  - a register, a file system, a database...
- Clients access the data store through read and write operations
- Consistency Semantics: a contract between the data store and its clients that specifies the results that clients can expect to obtain when accessing the data store

Coherence vs. Staleness vs. Consistency

- Coherence: restricts order of reads and writes to one location
- Staleness: bound maximum (real-time) delay between writes and reads to one location
- Consistency: restricts orders of reads and writes across locations

Coherence

- A read should return the result of the latest write to a memory location

```
P1:
for (ii = 0; ii < 100; ii++) {
   write(A,ii);
}
```

```
P2:
while(1){
   printf("%d", read(A);
}
```

- A memory that is not coherent may return
  - 1 2 3 3 3 4 8 10 9 11 12 13
- How could this happen?
  - writer sends updates via Internet; updates are reordered in route
  - reader switches between two servers (e.g. redirected to a different Akamai node)

Staleness

```
P1:
while(1) {
   sleep(1000ms);
   write (T, A, "At %t price is %d\n");
}
```

```
P2:
while(1){
   sleep (1000ms);
   printf("%s", read (A);
}
```

- Assume perfectly synchronized clocks and a real time OS
- Reads may return

```
At 1:00:00 price is 10.50
At 1:00:01 price is 10:55
At 1:00:02 price is 10.65
At 1:00:02 price is 10:65
At 1:00:05 price is 13:18
....
```

- Why staleness?
  - NFS polling interval
  - cache update/invalidation delayed by network

Consistency

- Restricts order of reads and writes across locations

```
P1:
for (ii = 0; ii < 100; ii++) {
   write(A,ii);
   write(B,ii);
}
```

```
P2:
while(1){
   printf("(%d, %d)", read(A), read(B));
}
```

- A memory that is not consistent may return
  - (0,0),(0,1),(1,2),(4,3),(4,8),(8,9),(9,9),(9,10),(10,10),(11,10),(11,11),(12,12)
- Is there also incoherence?
  - ...

Sequential Consistency

- "The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program" (Lamport, 1979)
- In other words:, create a total order that includes all the operations of the execution, such that:
  - the total order respects the local history of each process
  - every read returns the result of the latest write, according to the total order (data coherence)

Consistency

- Another example

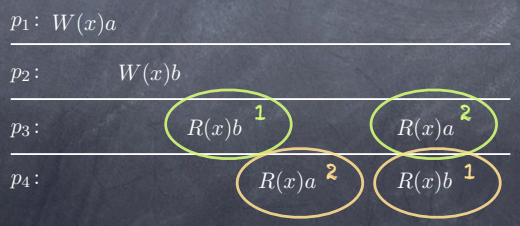
```
P1:
write(A,0);
...
write(A,1)
if(read(B) == 0) {
    printf("P1.");
}
```

```
P2:
write(B,0);
...
write(B,1)
if(read(A) == 0) {
    printf("P2.");
}
```

- Which outputs are legal under strict coherence? Under sequential consistency?
  a) "P1." • •
  b) "P2." • •
  c) "" • •
  d) "P1.P2." •
  e) "P2.P1." •

Sequential Consistency

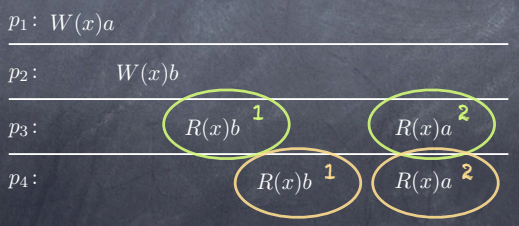
- "The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program" (Lamport, 1979)



Is this data store sequentially consistent?

Sequential Consistency

- "The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program" (Lamport, 1979)



Is this data store sequentially consistent?

Linearizability

- "The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program.

  In addition, if $ts_{OP1}(x) < ts_{OP2}(y)$, then operation $OP1(x)$ should precede $OP2(y)$ in this sequence (Herlihy & Wing, 1991)

```
P1:
write(A,1);
write(B,1)
```

```
P2:
sleep(1 year)
...
read(a),read(B)
printf("A=%d B=%d",A,B);
}
```

- "A=0 B=0" is legal under sequential consistency, but not under linearizability

Limitations of Strong Consistency

- So, linearizability it is, then?
  - implementing strong semantics has intrinsic costs
    - Lipton and Sandberg: in sequential consistency can have either fast reads or fast writes, but not both
  - CAP theorem: it is impossible in a distributed system to provide simultaneously
    - Consistency: all nodes see the same data at the same time, even in the presence of updates
    - Availability: every request receives a response
    - Partition Tolerance: the system properties hold even when the system is partitioned
  - one can only have two properties out of three...

Weaker Consistency: Causal Consistency

- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)

Weakening Sequential Consistency: Causal Consistency

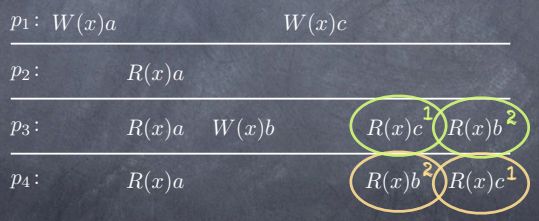
- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)

p_1:	$W(x)a$			$W(x)c$	
p_2:		$R(x)a$			
p_3:		$R(x)a$	$W(x)b$	$R(x)c$	$R(x)b$
p_4:		$R(x)a$		$R(x)b$	$R(x)c$

Is this data store sequentially consistent?
Causally consistent?

Weakening Sequential Consistency: Causal Consistency

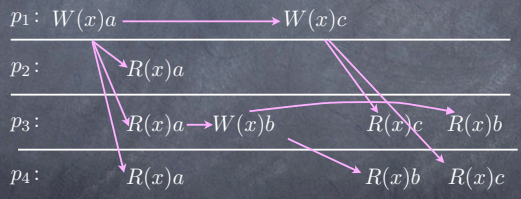
- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)



Is this data store sequentially consistent? No
Causally consistent?

Weakening Sequential Consistency: Causal Consistency

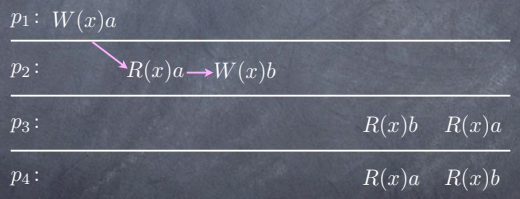
- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)



Is this data store sequentially consistent? No
Causally consistent? Yes

Weakening Sequential Consistency: Causal Consistency

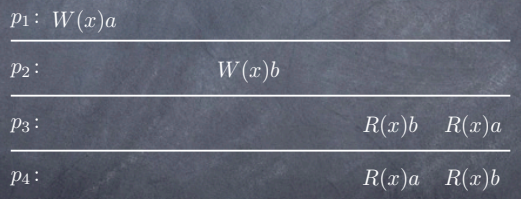
- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)



Is this data store sequentially consistent? No
Is this data store causally consistent? No

Weakening Sequential Consistency: Causal Consistency

- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)



Is this data store causally consistent? Yes

More Weakening: FIFO Consistency

- "Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes" (PRAM consistency, Lipton and Sandberg 1988)

More Weakening: FIFO Consistency

- "Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes" (PRAM consistency, Lipton and Sandberg 1988)

p_1: $W(x)a$				
p_2:	$R(x)a \rightarrow W(x)b \rightarrow W(x)c$			
p_3:		$R(x)b$	$R(x)a$	$R(x)c$
p_4:		$R(x)a$	$R(x)b$	$R(x)c$

Is this data store causally consistent? No
Is this data store FIFO consistent?

More Weakening: FIFO Consistency

- "Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes" (PRAM consistency, Lipton and Sandberg 1988)

p_1: $W(x)a$				
p_2:	$R(x)a$	$W(x)b \rightarrow W(x)c$		
p_3:			$R(x)b$ $R(x)a$	$R(x)c$
p_4:			$R(x)a$ $R(x)b$	$R(x)c$

Is this data store causally consistent? No
Is this data store FIFO consistent? Yes

The Joys of Programming

Process p_1

$x := 1$
if $(y = 0)$ then
  kill (p_2)

Process p_2

$y := 1$
if $(x = 0)$ then
  kill (p_1)

Initially, $x = y = 0$

What are the possible outcomes?

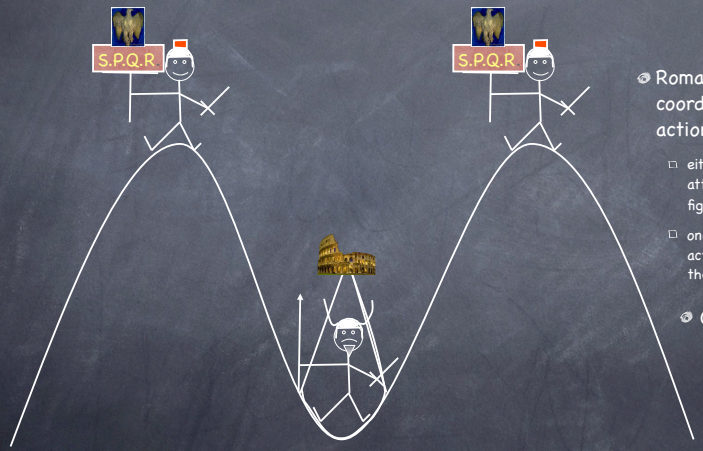
Andrew File System

CMU, 1987

- Key features
  - callbacks
    - server keeps list of which client has which file
  - write-through on file close
    - when file changes, server notifies all clients with a copy
  - consistency: updates visible only on file close
    - more precise semantics

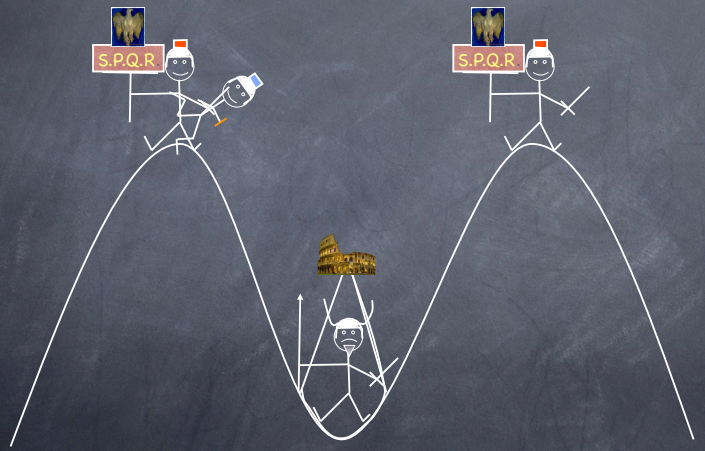
      as if all reads on open, all writes on close
  - client local disk used for caching
    - server load is further reduced
  - server is stateful
    - on server failure, need a recovery protocol to rebuild state
    - can ask clients
    - but what if client crashes?

Two Generals' Problem



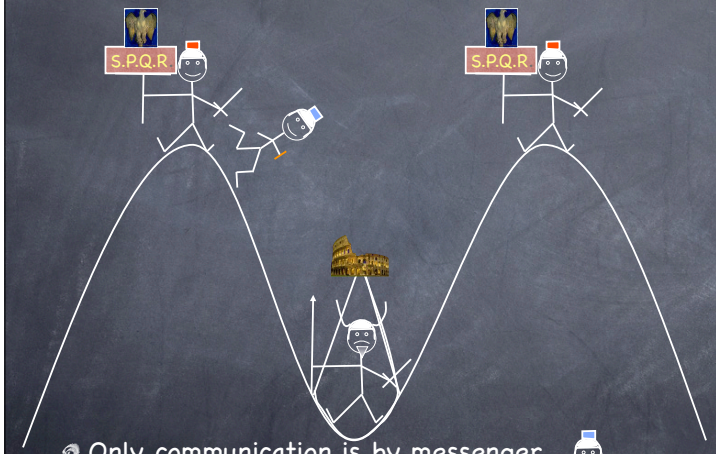
- Romans must coordinate their actions
  - either both Generals attack or both retreat to fight another day
  - once they commit to an action, they cannot change their mind
- Otherwise, Barbarians win

Two Generals' Problem



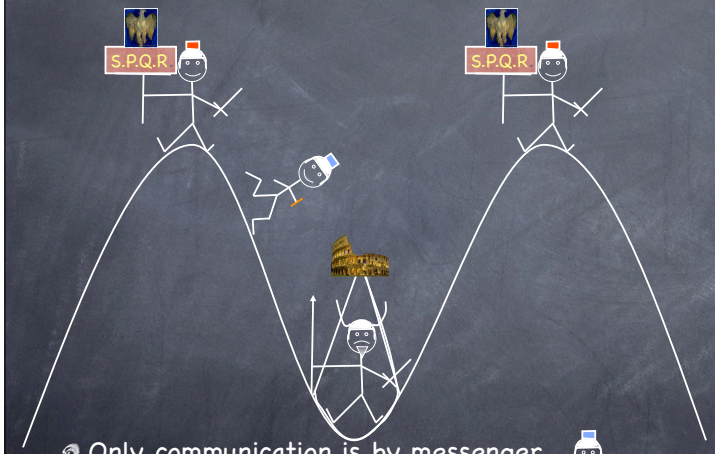
- Only communication is by messenger

Two Generals' Problem



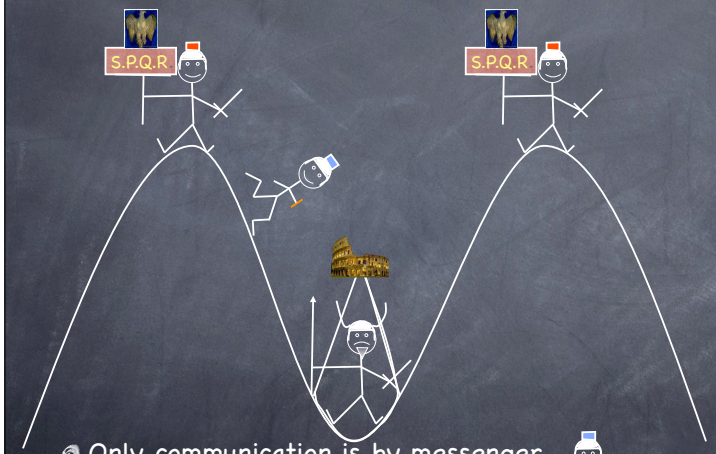
- Only communication is by messenger
- Messengers must sneak through the valley

Two Generals' Problem



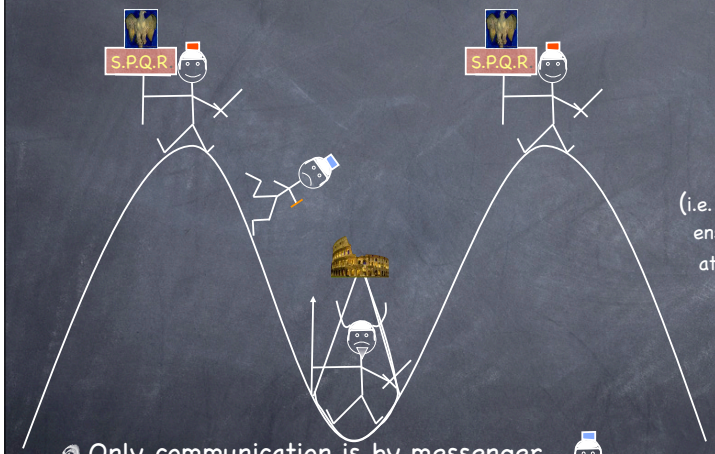
- Only communication is by messenger
- Messengers must sneak through the valley

Two Generals' Problem



- Only communication is by messenger
- Messengers must sneak through the valley
- They don't always make it

Two Generals' Problem



Problem:
Save Western Civilization
(i.e. design a protocol that ensures Romans always attack simultaneously)

- Only communication is by messenger
- Messengers must sneak through the valley
- They don't always make it

Two General's Problem

Claim: There is no non-trivial protocol that guarantees that the Romans will always attack simultaneously

Two General's Problem

Claim: There is no non-trivial protocol that guarantees that the Romans will always attack simultaneously

Proof: By contradiction

- Let n be the smallest number of messages needed by a solution
- Consider the n-th message m_{last}
  - The state of the sender of m_{last} cannot depend on the receipt of m_{last}
  - The state of the receiver of m_{last} cannot depend on the receipt of m_{last} because in some executions m_{last} could be lost
  - So both sender and receiver would come to the same conclusion even without sending m_{last}
  - We now have a solution requiring only $n-1$ messages – but n was supposed to be the smallest number of messages!

Two General's Problem

Claim: There is no non-trivial protocol that guarantees that the Romans will always attack simultaneously

Proof: By contradiction

- Let n be the smallest number of messages needed by a solution
- Consider the n-th message m_{last}
  - The state of the sender of m_{last} cannot depend on the receipt of m_{last}
  - The state of the receiver of m_{last} cannot depend on the receipt of m_{last} because in some executions m_{last} could be lost
  - So both sender and receiver would come to the same conclusion even without sending m_{last}
  - We now have a solution requiring only $n-1$ messages – but n was supposed to be the smallest number of messages! Contradiction