CS 525
Advanced Distributed Systems
Spring 2013

Furquan Shaikh

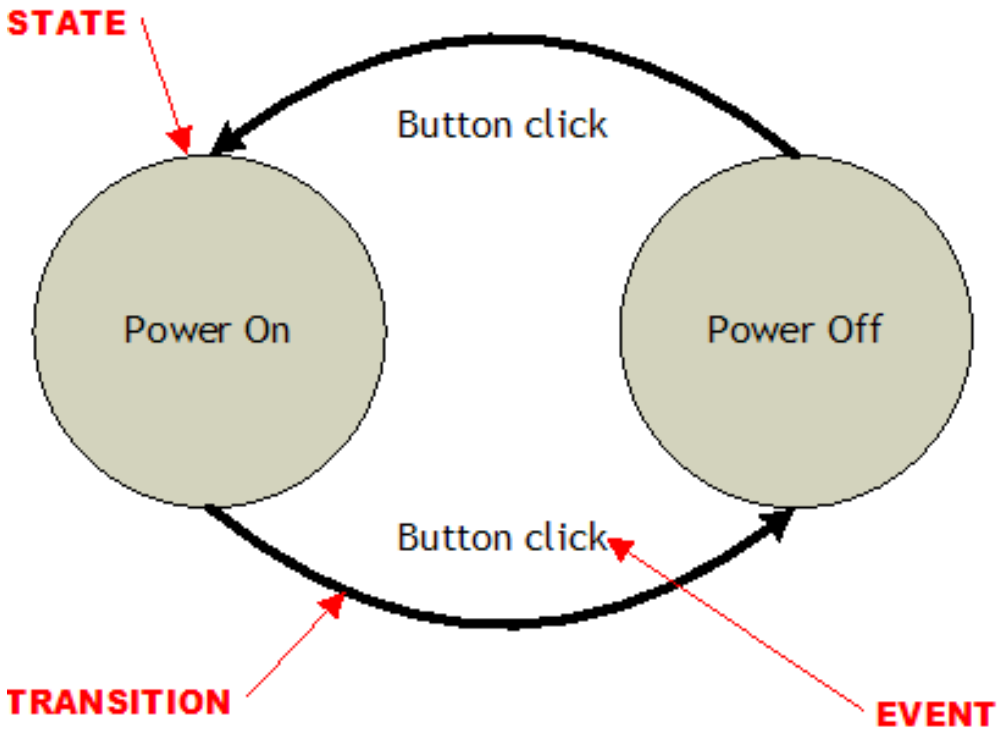
Paxos Replicated State Machines as the Basis of a High-Performance Data Store

February 12, 2013

Slides Reference:
http://static.usenix.org/events/nsdi11/tech/slides/bolosky.pdf
Implementing_FaultTolerant_Services_Using_the_State_Machine_Approach_A_Tutorial

BACKGROUND AND MOTIVATION

State Machine



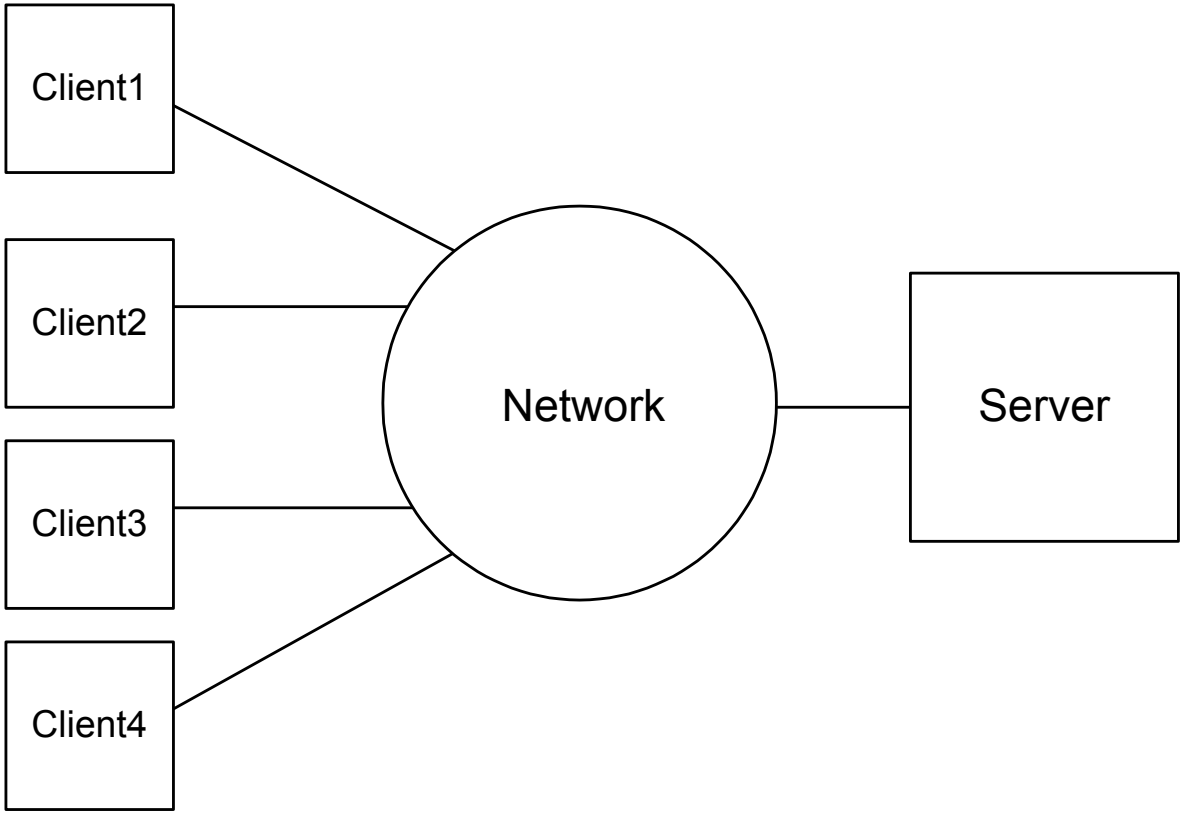
Faults

- **Fault:** System behavior inconsistent with the specification
- **Failure models:**
  - Byzantine Failures
  - Fail-stop Failures
- **t-fault tolerant:** system that satisfies specification provided no more than t of its components become faulty

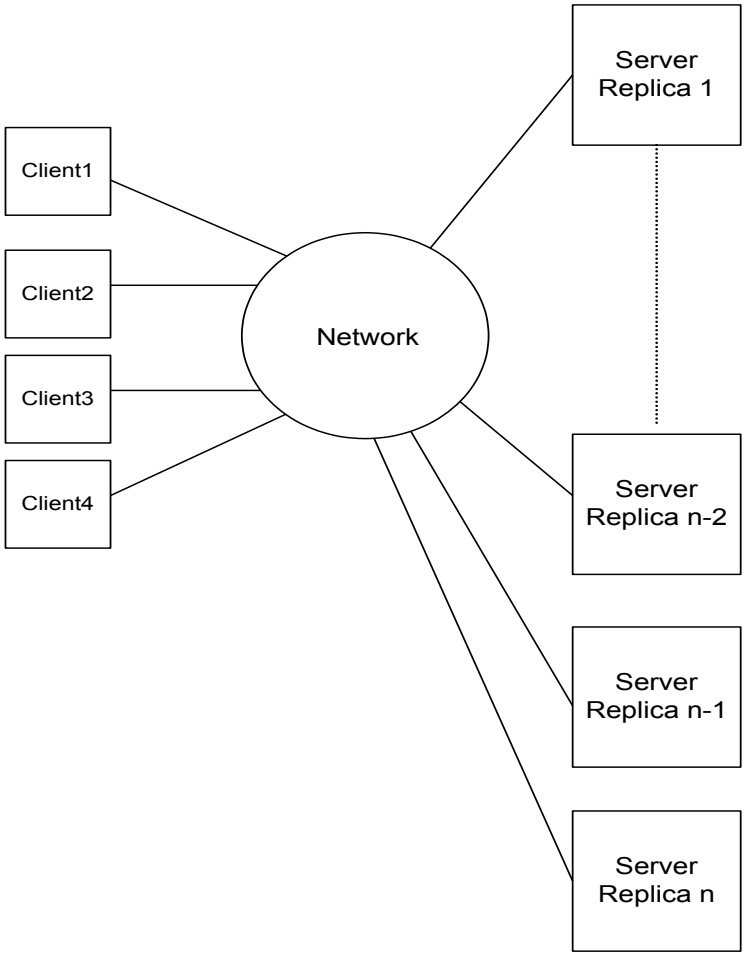
Replicated State Machines

- A fault tolerant state machine can be implemented by replicating that state machine on each of the processors in a distributed system.
- Requirements of Replica
  - Same initial state
  - **Execute the same request in the exact same order**
  - Perform the same thing and produce the same output

Client-server model



Fault tolerance using Replicated State Machines



REPLICATED STATE MACHINES FOR MAKING DATA STORES FAULT TOLERANT

Conventional Systems

- **Primary-backup Replication**
  - Run at slower rate of primary, may rely on some kind of clock synchronization
- **Google File System**
  - Append mostly files, weaker consistency, sacrifices efficiency for overwrites
- **Google's Chubby Lock Service**
  - Relies on clock synchronization
- **Storage Area Networks (SANs)**
  - Costlier, use special hardware (eg. Battery-backed RAM)

Breaking the Trend

- Is it possible to build a fault-tolerant high-performance data store from commodity parts? Without compromising on semantics or relying on special hardware or clock synchronization?

- The answer is PAXOS Replicated State Machine

Paxos Replicated State Machines

- Features:
  - Sequentially consistent
  - Persistent
  - Fault tolerant
  - Don't rely on clock sync for correctness
  - Considered slow (???)

Applicable Scenarios

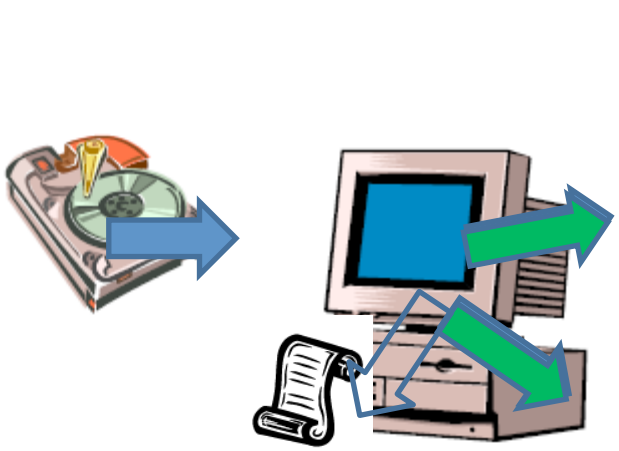
- For data center systems:
  - Consistency and availability is preferred over partition tolerance (CAP theorem)
  - Operation latencies ≥ Network Latencies
- Paxos Replicated State Machines:
  - Almost (or better than) base performance
  - No need for compromise

GAIOS and SMARTER
William Bolosky et al (Microsoft Research)

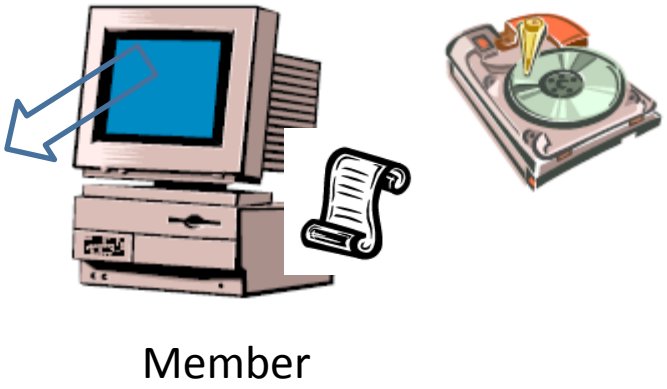
How is PAXOS used?

- Paxos is a protocol for guaranteeing input ordering even with:
  - Multiple clients
  - Unreliable networks
  - No synchronized clocks
  - Unlimited machine reboots
  - Some permanent stopping faults (i.e. disk losses)
  - But not Byzantine faults

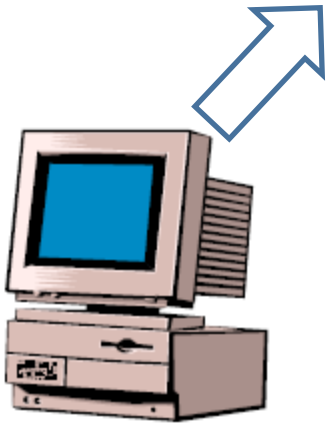
Basic Operation



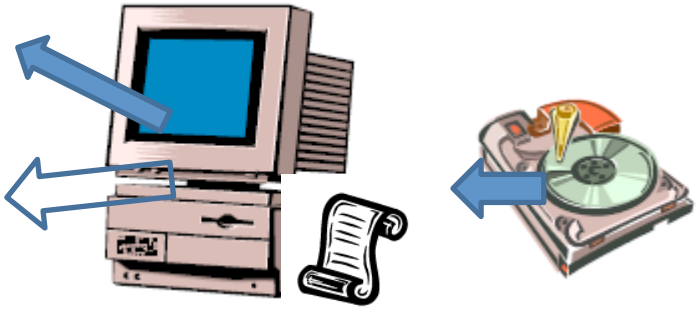
Leader



Member



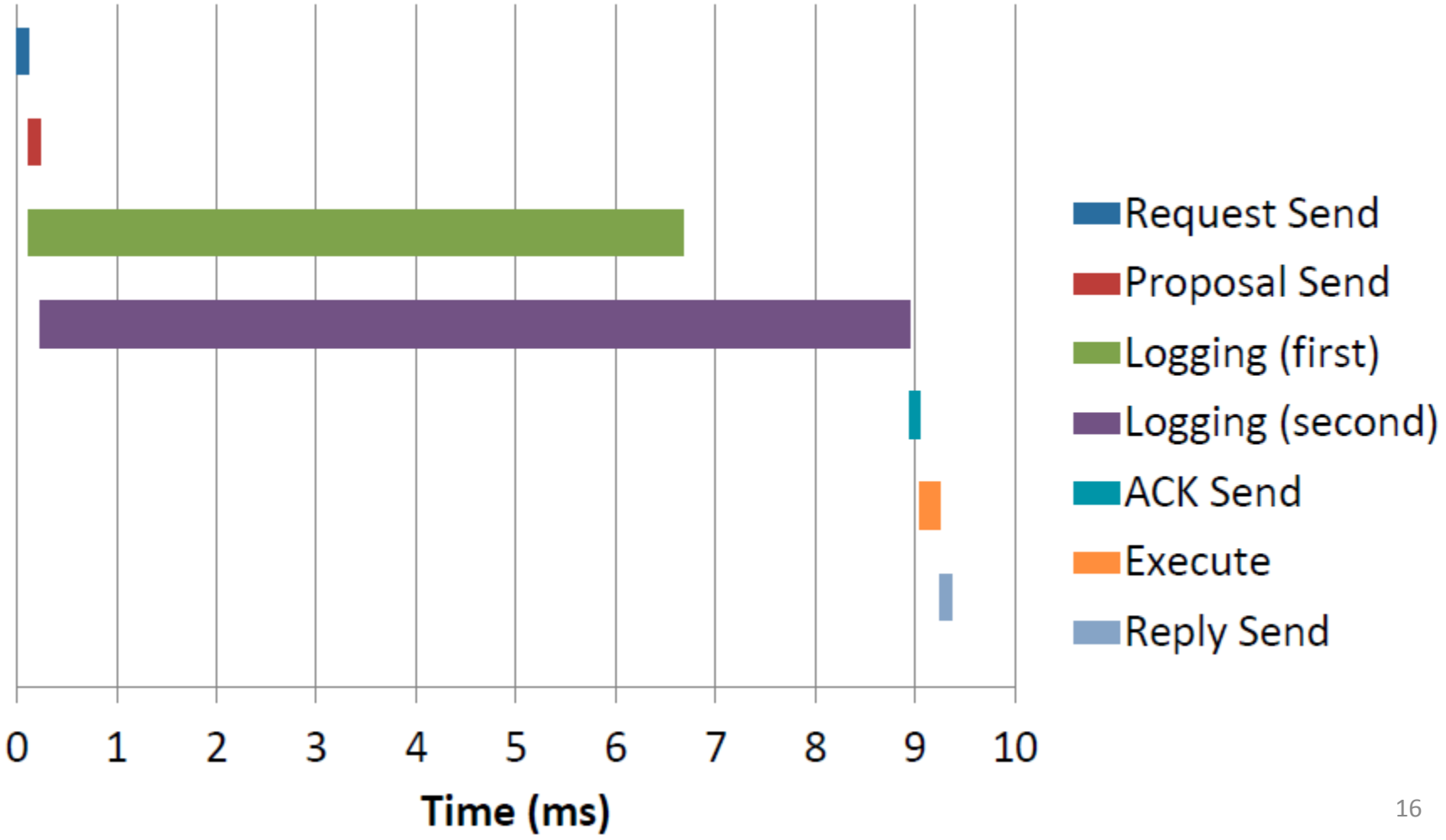
Client



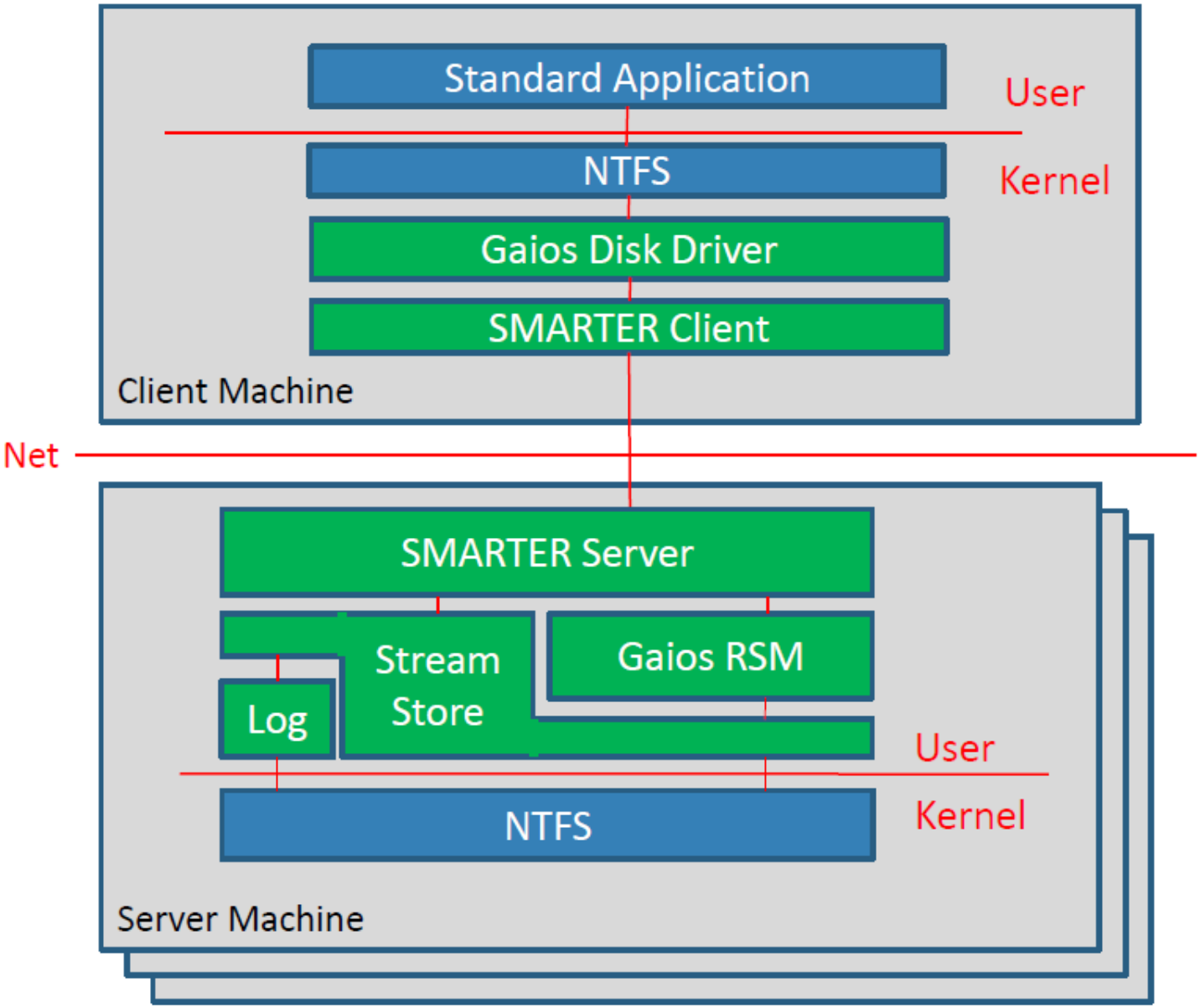
Member

Prepare + ACK

4K Write Latency Timeline



Gaios Architecture



How is efficiency achieved?

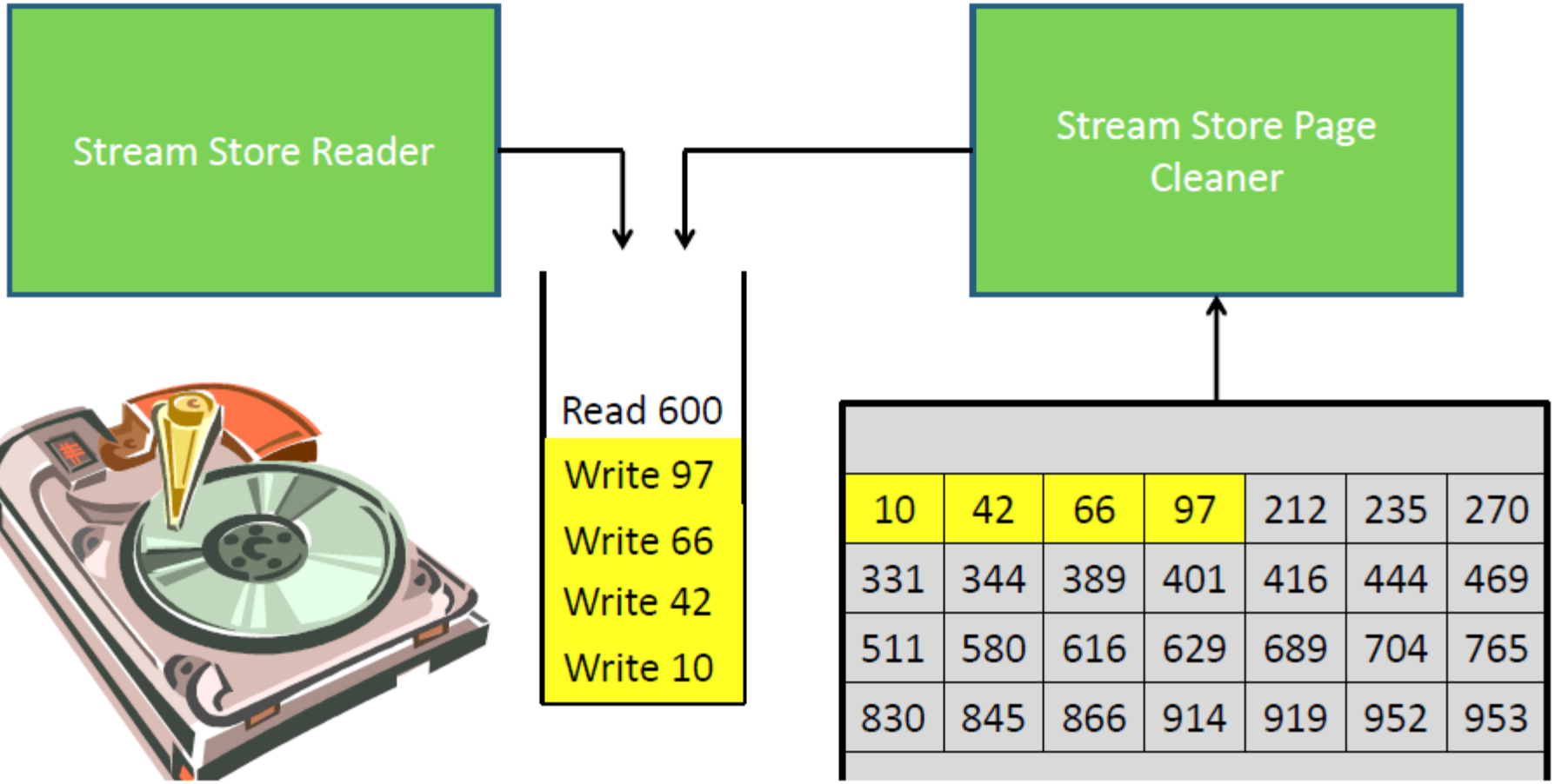
- Pipelining
- Batching client requests
- Batched write behind
- Overlap fetch with logging

- Novel read-only operation protocol that allows consistent reads from any node

Read Consistency Property

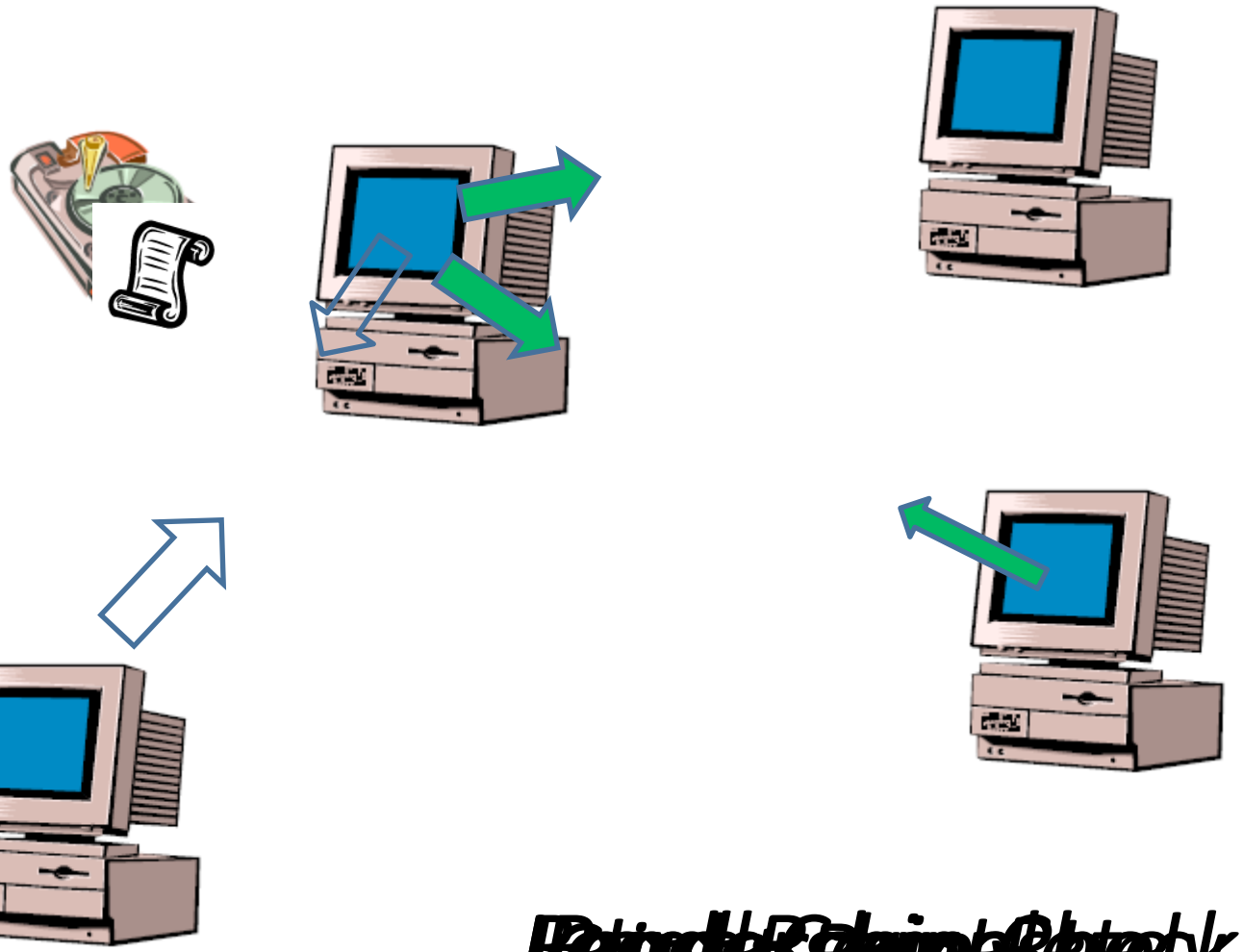
- When a read-only request R completes, it reflects any data known by any client to be written at the time R was sent.
- Only need to run in one place
- Using all disks can enhance performance (load-balancing)
- Dynamically selecting location helps
  - Avoid nodes that are writing (avoid read write contention)

Read Write Contention

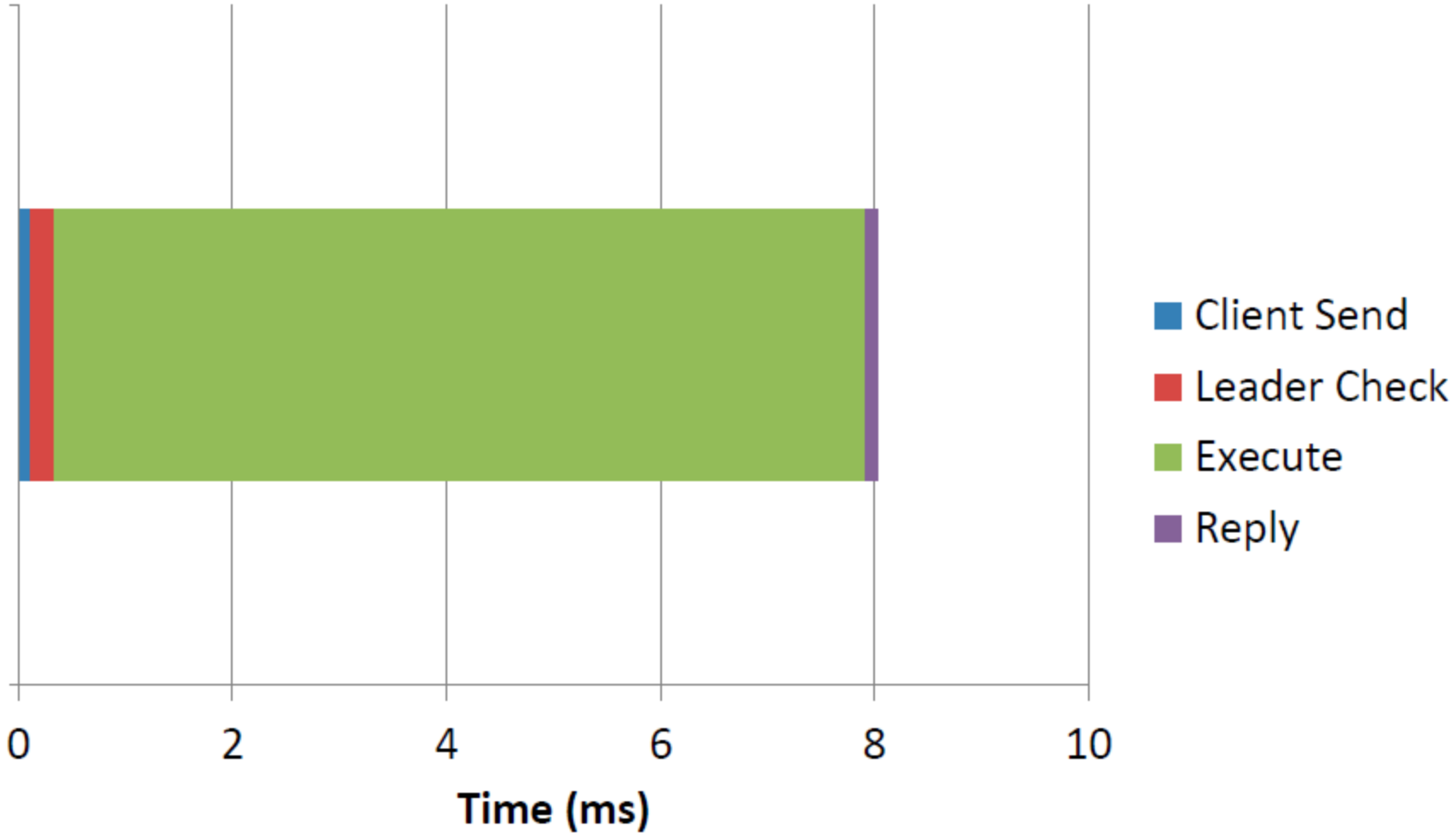


Randomize Checkpoint timing across nodes

Read-only Operation



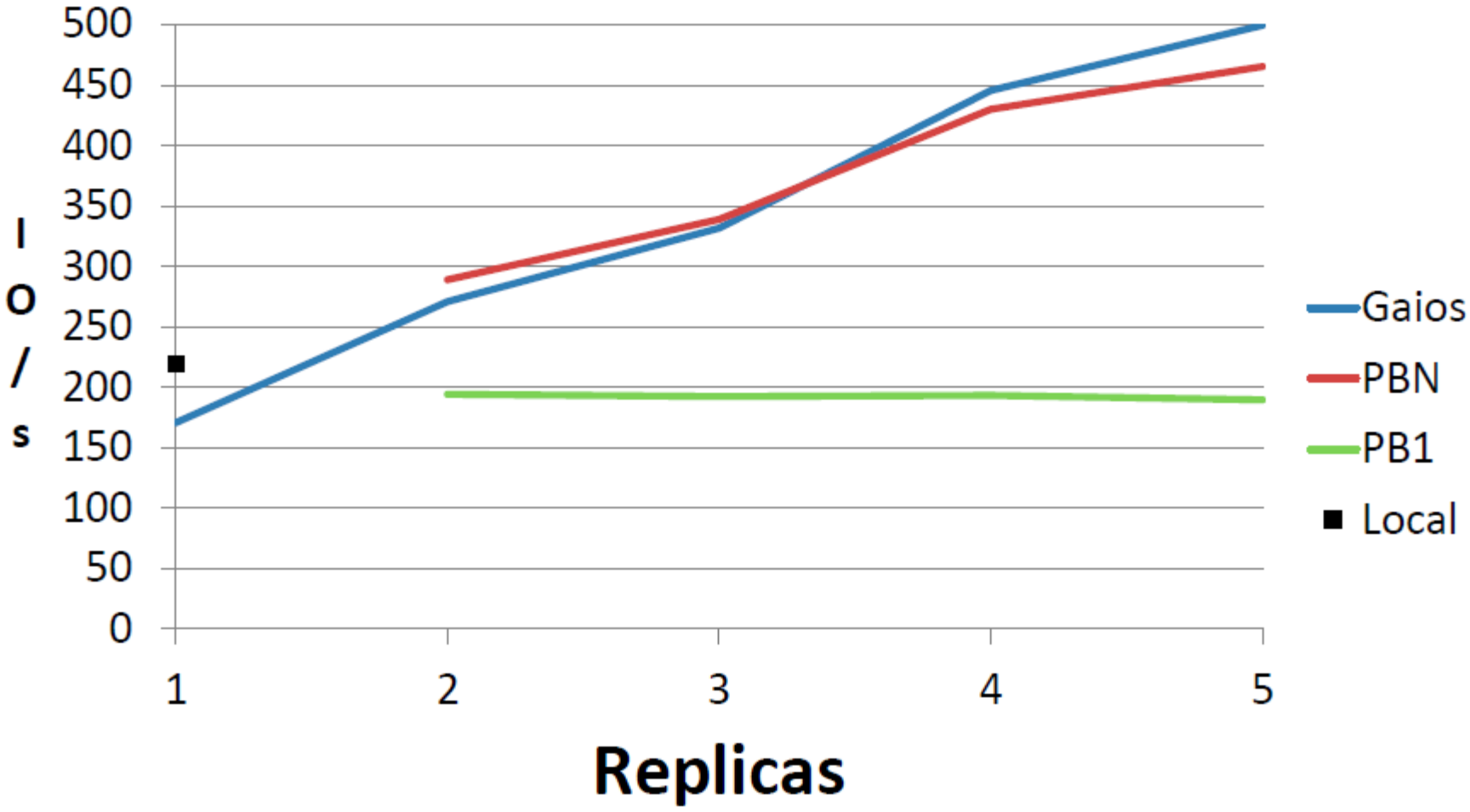
4K Read Latency Time



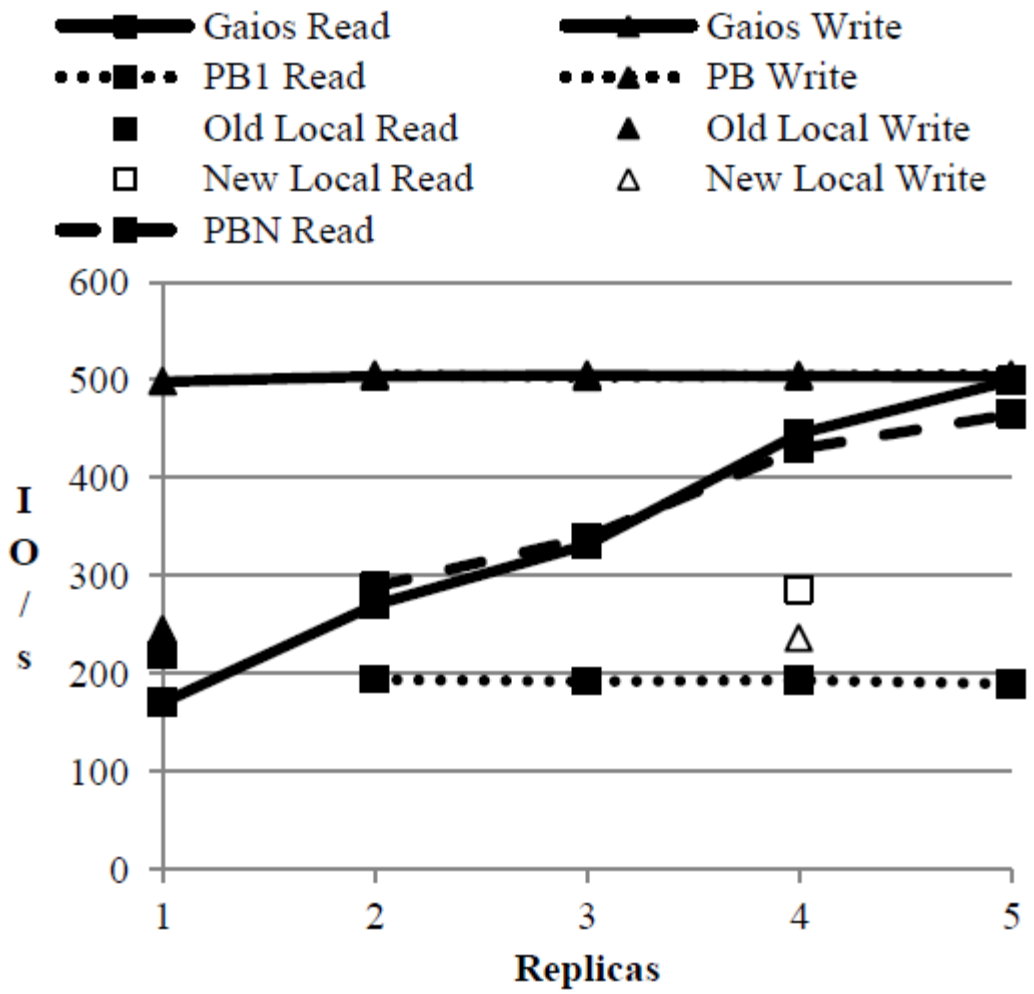
Performance Evaluation

8K Random Read Throughput

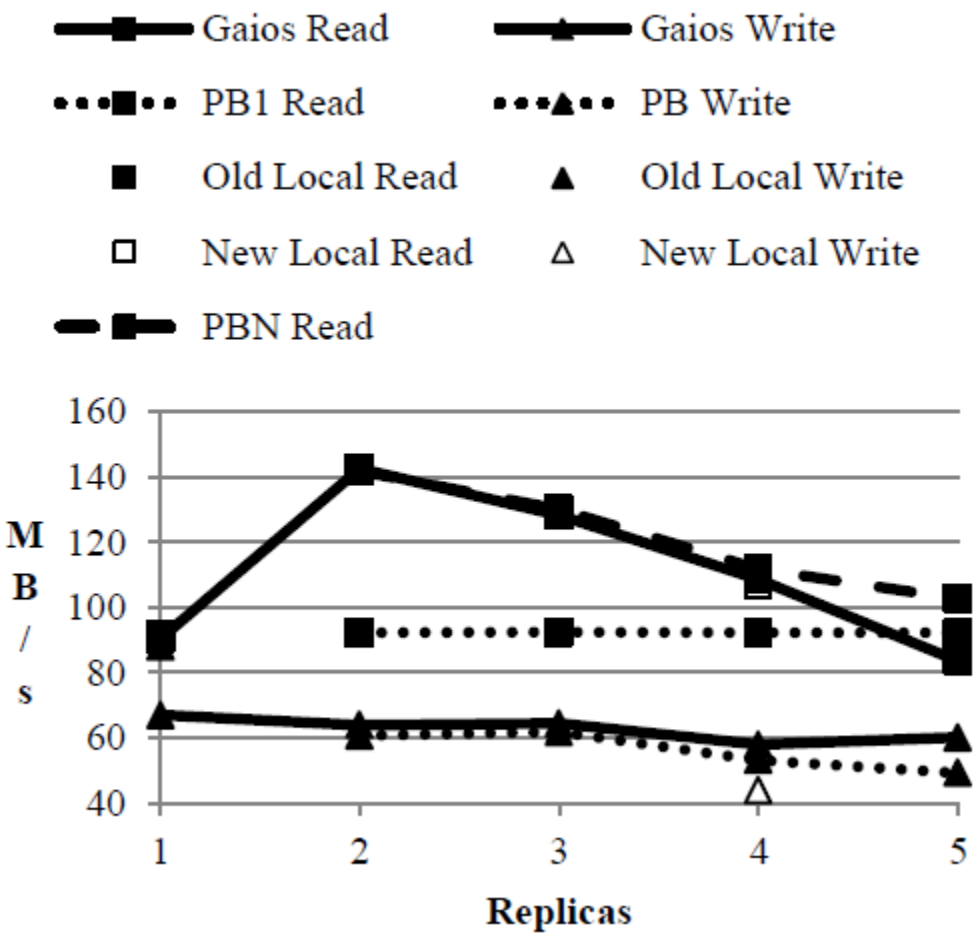
(Lots of outstanding requests)



Random IO performance



Sequential IO performance



OLTP Performance

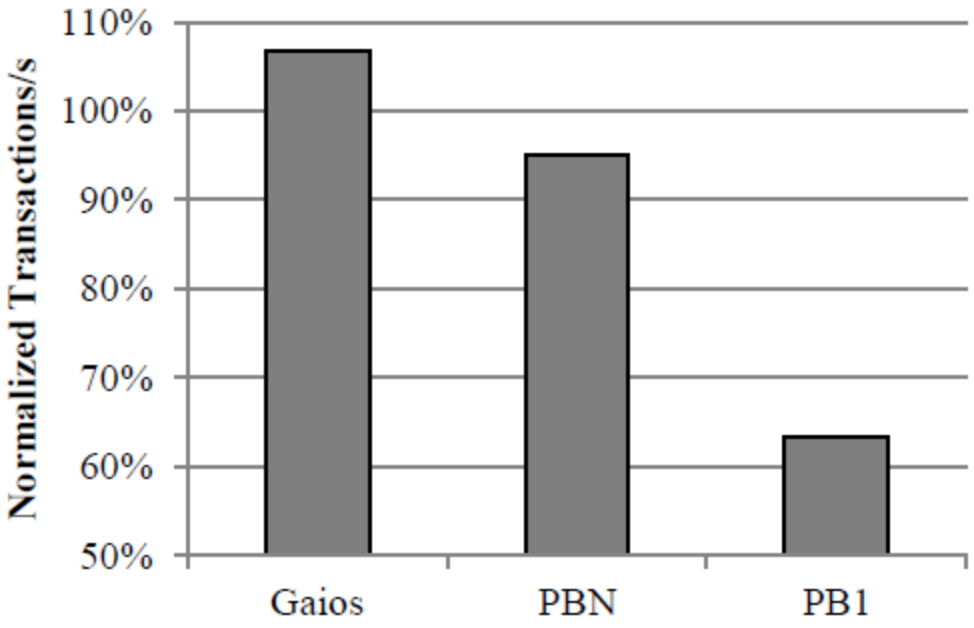


Figure 7: OLTP Performance

Summary

- Paxos RSMs work perfectly fine for high-performance disk-based applications
- No need to compromise on semantics, or buy special hardware, depend on clocks, etc.

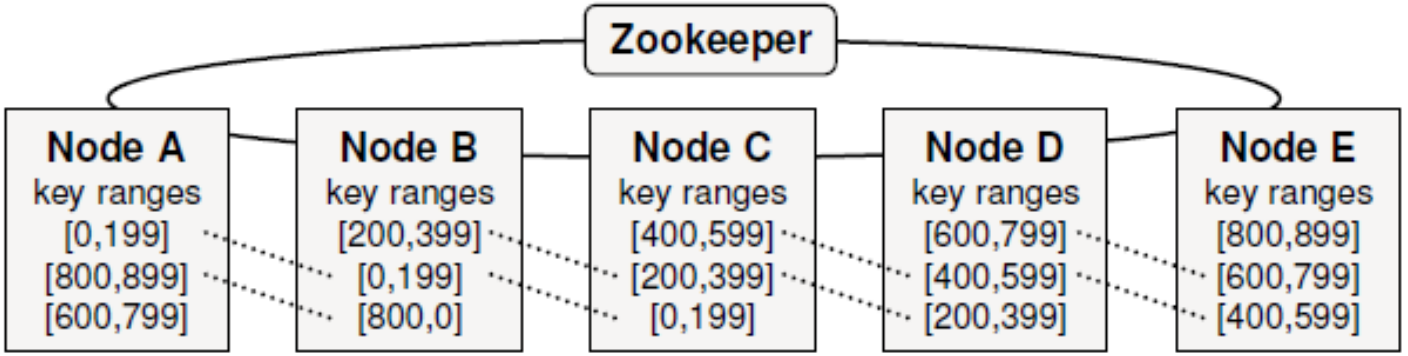
Points to ponder

- What happens in case of partition?
- Is this suitable for WANs?
- What is the performance in view of faults?
- Can this system be run completely out of memory? (i.e. moving logs out of disk)
- What will happen with new advancement in disk technology? SSD, flash?

Spinnaker

- Experimental datastore
- Runs on large cluster of commodity servers in data center
- Key-based range partitionining
- 3-way replication
- Optional strong /timeline consistency

Spinnaker cluster



API

get(key, colname, consistent)

Read a column value and its version number from a row. The setting of the 'consistent' flag is used to choose the consistency level. Setting it to 'true' chooses strong consistency, and the latest value is always returned. Setting it to 'false' chooses timeline consistency, and a possibly stale value is returned in exchange for better performance.

put(key, colname, colvalue)

Insert a column value into a row.

delete(key, colname)

Delete a column from a row.

conditionalPut(key, colname, value, v)

Insert a new column value into a row only if the column's current version number is equal to 'v'. Otherwise, an error is returned.

conditionalDelete(key, colname, v)

Like conditional put but for delete.

Conclusion

- Paxos based replication systems for guaranteeing CA
- Suitable for data center like systems
- Traditional thought that Paxos slows down system not true under the light of higher latency operations

The Chubby lock service for loosely-coupled distributed systems

Mike Burrows

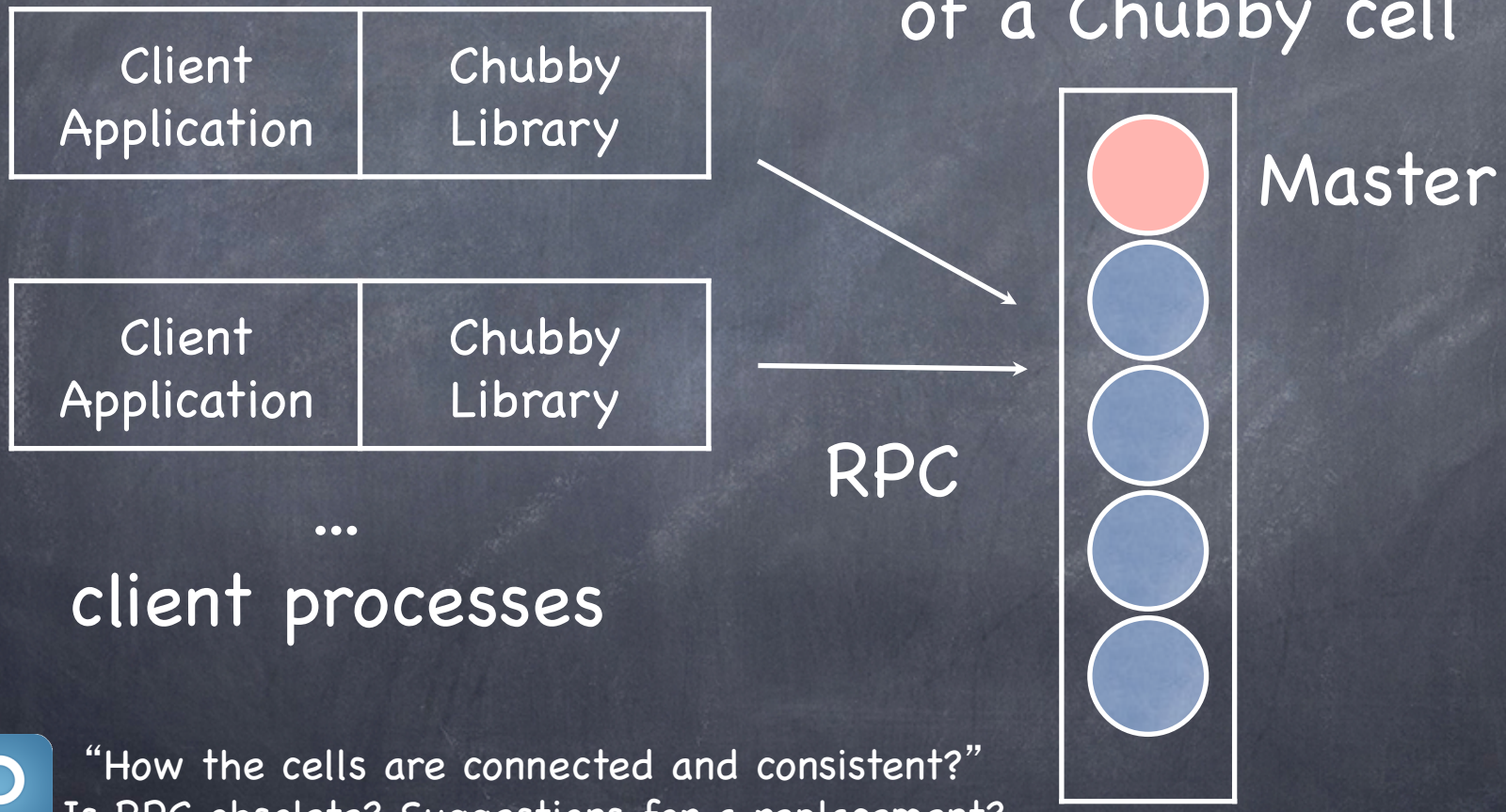
Presented by Fangzhou Yao
CS 525 Feb 12th, 2013

Introduction

- Chubby
  - Lock service in a loosely-coupled distributed system e.g. 10k 4-processor machines connected by 1Gbit/s Ethernet
  - Client interface similar to a simple file system that performs whole-file reads and writes
- Objectives: availability and reliability
- Used for: GFS, Bigtable
- Meta-data filesystem, name service, etc.

Design

5 servers (replicas) of a Chubby cell



"How the cells are connected and consistent?"
Is RPC obsolete? Suggestions for a replacement?

System Structure

- A Chubby cell consists of a small set of servers (usually five) known as replicas
- Replicas use a distributed consensus protocol (Paxos) to elect a master
  - Master lease: time interval that replicas will not elect a new master (majority in votes), usually a few seconds
  - Upon failure, a new election will be initiated after the lease

System Structure

- Replicas maintain copies of a simple database
  - Only the master reads and writes to it
  - Others update from the master
- Clients send read and write requests only to the master
  - Finding the master by sending master location requests to replicas listed in DNS
  - Clients talk to the master though the Chubby library

System Structure

- Write requests
  - The master sends it to all replicas via the consensus protocol (Paxos, again)
  - Replies after the write reaches a majority of replicas
- Read requests
  - The master satisfies the read alone
  - This is safe because of the master lease - no other masters at the same time

System Structure

- If a replica fails and does not recover for some time, like a few hours
  - A fresh machine from the free pool is selected to be a new replica as a replacement
  - Updates the DNS tables, replacing IP
  - Obtains a recent copy of the database
  - The current master checks DNS periodically for new replicas

Files, Directories, and Handles

- UNIX-like filesystem interface
  - /ls/foo/wombat/pouch
  - ls: prefix for Chubby lock service
  - foo: the Chubby cell, resolved via DNS lookup
  - /wombat/pouch, resolved inside of the cell
- Supports for use as master changes

Files, Directories, and Handles

- Supports a strict tree of files and directories
- Supports traditional file operations e.g. create, delete, open, write... but no path-dependent operations, like move
- Supports advisory reader and writer lock on a node - it only prevents others from getting an advisory lock on the same file, but does not actually prevent the read or write.
- Meta-data on nodes (files and dirs) includes: Access Control List for Read, Write and Change of ACL; 4 increasing 64-bit numbers (instance, content generation, lock gen, ACL gen); 64-bit file-content check sum

Locks and Sequencers

- Locks (each file and directory will behave)
  - Advisory and not mandatory
  - Mandatory locks makes object inaccessible
- Sequencers (A byte-string describing the state of the lock after acquisition)
  - Name, mode (exclusion or shared), number
  - Server validates this and handle requests

Events

- Clients can subscribe to many events
  - File contents modified - monitor the location of a service advertised via the file
  - Child node added, removed, modified - mirroring
  - Master failed over - warning for clients
  - Handle becomes invalid - usually communication problem
  - Lock acquired - if a primary has been elected (rarely used)
  - Conflicting locks - allows the caching of locks (rarely used)

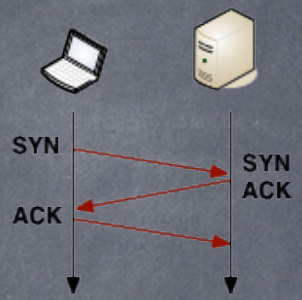
APIs

- Open() creates handles, takes a node name
- Close() destroys handles; Poison() causes fail directly
- GetContentsAndStat() returns content and meta-data; GetStat() for meta-data; ReadDir() for children
- SetContents() sets all contents to a file; SetACL()
- Delete() deletes the node if it has no children
- Acquire(), TryAcquire(), Release() for locks
- GetSequencer(), SetSequencer(), CheckSequencer() for sequencers

Caching

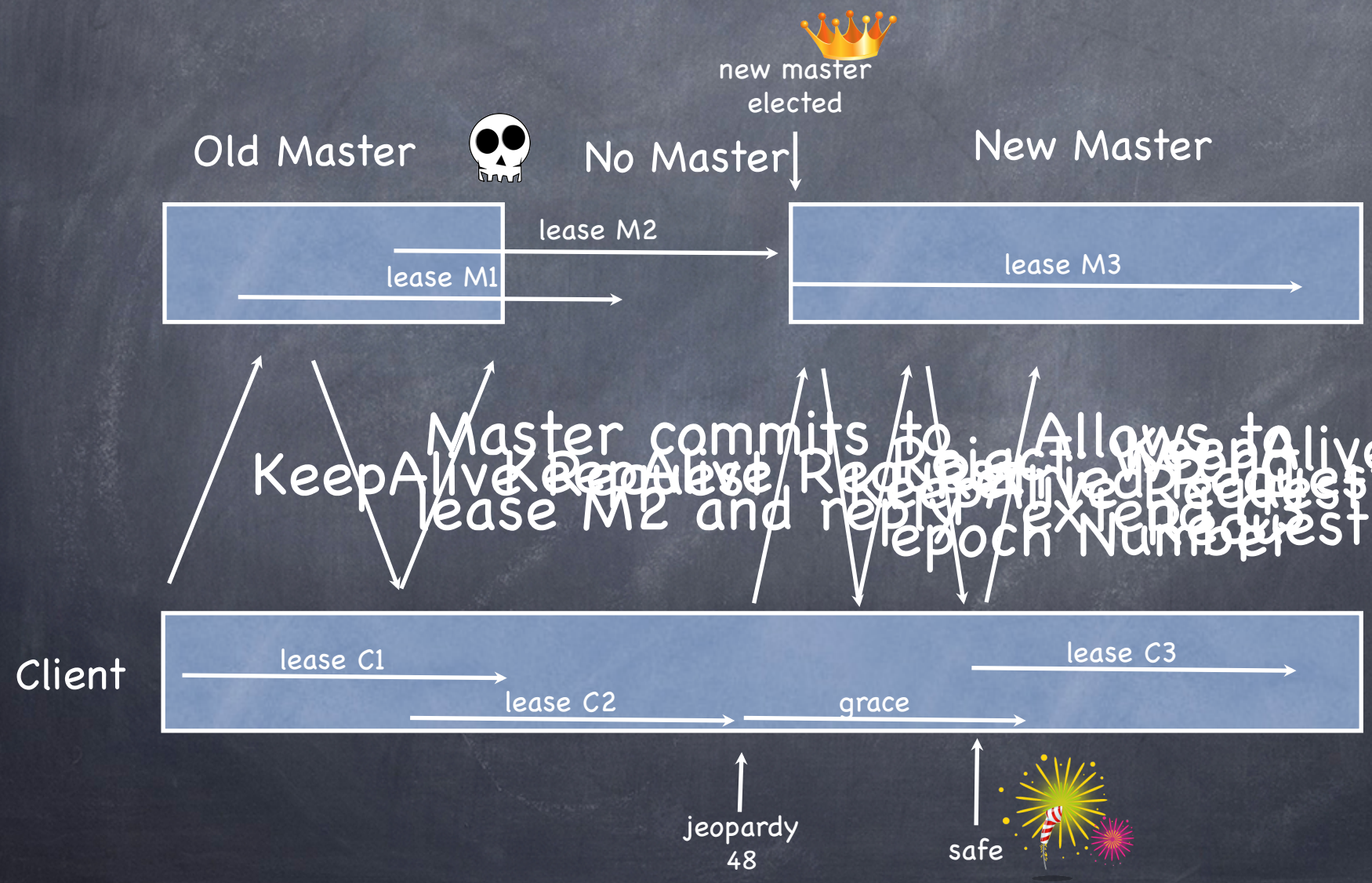
- Objective: To reduce the read traffic
- File data and meta data are cached on clients
  - Consistent, write-through
- Master will invalidate cached copies when a write request is initiated
  - List of what clients may have cached
  - Sends invalidations on top of KeepAlive to clients
  - Clients flush the changed data
  - Servers starts the modification

Sessions, KeepAlives

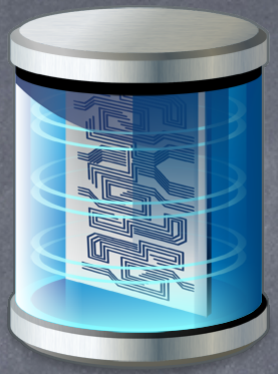


- Session maintained through KeepAlives (Periodic handshakes)
  - A client sends KeepAlive requests to a master
  - A master responds by a KeepAlive response
- Clients maintain a local timer to estimate the session timeouts, which are not perfectly synchronized
- If local timer expires (session in jeopardy), wait for a grace period (45s) before ending the session

Fail-overs


new master
elected
Old Master
No Master
New Master
lease M2
lease M1
lease M3
KeepAlive
KeepAlive
KeepAlive
KeepAlive
Master commits to
lease M2 and
Allows to
epoch Number
Client
lease C1
lease C2
grace
lease C3
jeopardy
48
safe

Other Specs



- Database Implementation
  - Berkeley DB - logging and snapshotting
- Backup
  - The master writes a snapshot of its database to a GFS server
- Mirroring
  - A collection of files can be mirrored from one cell to another, e.g. configuration files

Mechanisms for Scaling

- 90,000 clients communicate with a single cell
  - Server machine is the same with clients, so the master may be overwhelmed
- Regulate the number of Chubby cells
- Reduce communication
- Increase lease time
- Client caching
- Protocol-conversion servers

Mechanisms for Scaling: Proxies

- Passes requests from clients to the cell
- Reduces server load by handling both KeepAlive and read requests, but not for write traffic, which passes through the proxy's cache
  - Writes are just a small portion
- Needs additional RPC to writes and first-time read

Mechanisms for Scaling: Partitioning

- Partitioning of namespace between servers
- N partitions, each has a set of replicas and a master
- Node D/C in D is in P(D/C) == hash(D) mod N
- Meta-data for D may be different
- Intends to enable large Chubby cells with little communication between partitions

Use and Behaviors

time since last fail-over	18 days
fail-over duration	14s
active clients (direct)	22k
additional proxied clients	32k
files open	12k
naming-related	60%
client-is-caching-file entries	230k
distinct files cached	24k
names negatively cached	32k
exclusive locks	1k
shared locks	0
stored directories	8k
ephemeral	0.1%

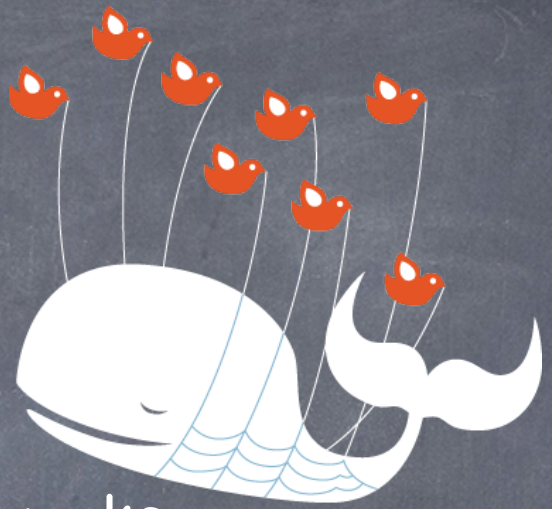
stored files	22k
0-1k bytes	90%
1k-10k bytes	10%
> 10k bytes	0.2%
naming-related	46%
mirrored ACLs & config info	27%
GFS and Bigtable meta-data	11%
ephemeral	3%
RPC rate	1-2k/s
KeepAlive	93%
GetStat	2%
Open	1%
CreateSession	1%
GetContentsAndStat	0.4%
SetContents	680ppm
Acquire	31ppm

- What can we see from this table?

Use and Behaviors

- Many files are used for naming
- Configuration, access control and meta-data files are common
- Negative caching is significant
- Around 10 clients use each cache file
- Few clients hold locks and shared locks are rare
- RPC traffic is dominated by session KeepAlives

Outages



- Recorded 61 outages over a few weeks
  - 700 cell-days of data in total
  - Most 15s or less, and 52 of them were under 30s
- Main causes: network congestion, maintenance, overload, and errors due to operators, software, and hardware.

Summary

- What did we get from Chubby?
  - Lock service with Paxos protocol
  - Design for Loosely-coupled distributed system
  - UNIX-like filesystem interface
  - Reliability and availability

Discussion

- "Fine-grained locking could be ignored" v. "design of a two tier locking system supporting both coarse and fine lockings"

- "Chubby is a single system that provides multiple services. This could mean that if one of the service requirements is changed, the whole system needs to be updated or even re-written."

- What can we do about KeepAlives?
- Other Questions?
- Thanks for your attention!

Reference: http://www.cs.cmu.edu/~chensm/Big_Data_reading_group/slides/shimin-chubby.ppt