Boxwood: Abstractions as the Foundation for Storage Infrastructure

John MacCormick, Nick Murphy, Marc Najork, Chandramohan Thekkath, Lidong Zhou,Microsoft Research

Presented by: Robert Robinson, CS848

Overview

- Motivation for Boxwood
- Overview of Boxwood
  - Architecture
  - Major system components
- Performance results
- Boxwood in use - BoxFS
- Conclusions & observations

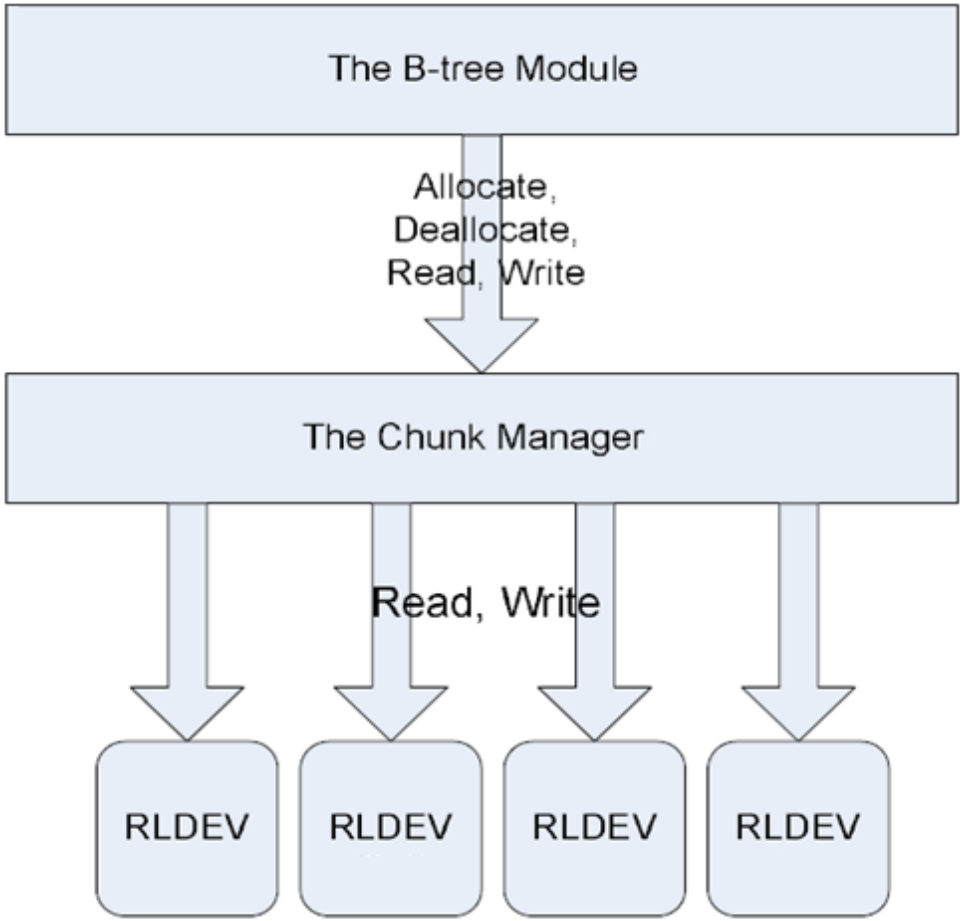
Motivation

- Writing distributed & reliable storage systems is hard
  - Examples: File systems, database systems
  - Issues: Consistency, fault tolerance, scalability, management
- Each implementation handles these issues internally
  - Increases complexity
- Idea: Create a high-level abstraction to hide these issues
  - Utilize layering to simplify implementation

Introduction

- Create a distributed and reliable storage infrastructure
  - Provide additional abstractions on top of this storage system
  - Application writers don't have to worry about the details
- Similar to the Google papers discussed previously
  - Distributed locking (Chubby)
  - Shared metadata storage (Chubby)
  - Replicated file contents exposed via a chunk interface (GFS)
- A lot of basic assumptions made:
  - Deploy in a highly-connected environment (datacenter)
  - Security is not needed
  - Failures will have fail-stop behaviour

System Architecture



Source: Boxwood: Abstractions as the Foundation for Storage Infrastructure, J. MacCormick et al.

System Components

- Paxos service
  - Consensus, storage of global state
- Distributed locking service
- Failure detection
- Replicated block device (RLDev)
  - Reliable byte storage
- Chunk manager
  - User-visible chunk storage abstraction
- B-tree module
  - User-visible B-tree abstraction

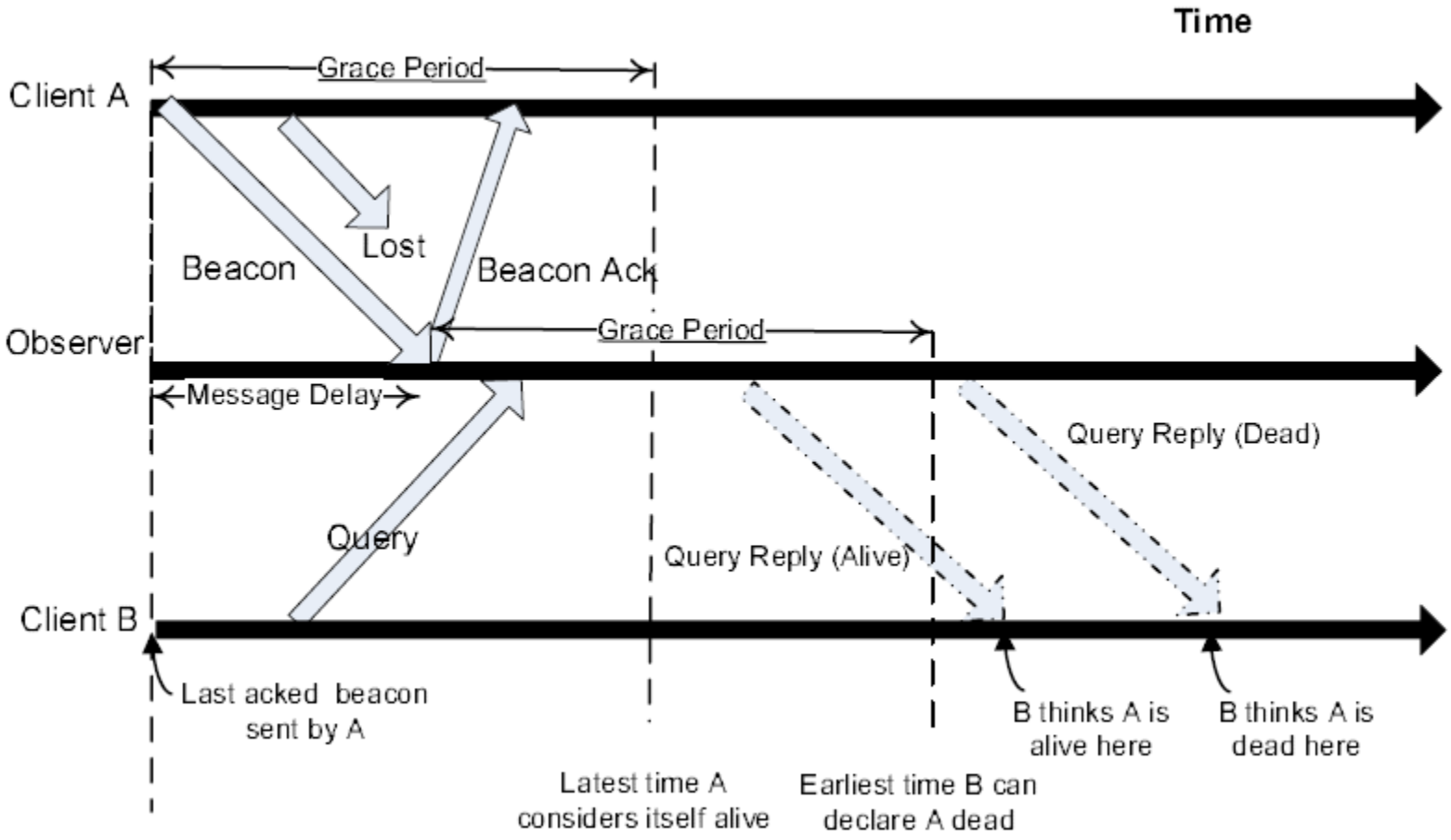
Paxos Service

- Stores global state across multiple machines
  - Uses Paxos to cause state changes to occur in the same order
  - Stores number of clients, number of RLDevs, etc
  - Can tolerate *k* failures on 2*k*+1 machines
- Used by the distributed lock service
  - Stores lock master identity, client identities
- Is not involved in reads & writes, just system changes
  - Prevents overloading Paxos hosts

Failure Detection

- Designed to maintain 2 invariants:
  - If a machine fails, it will eventually be detected as dead
  - If the service tells a host that another host is dead, it is dead
- Each machine sends keepalives to a group of observers
  - A host is failed only if a majority of observers think it has failed
  - Synchronous clocks are not required
- A host which don't receive keepalive acks will kill itself
  - Guarantees a machine is dead if its observers thinks it is
- A host queries the observers to determine liveness
  - Presumably observer addresses are stored using Paxos

Failure Detection



Source: Boxwood: Abstractions as the Foundation for Storage Infrastructure, J. MacCormick et al.

Distributed Lock Service

- Provides reliable reader/writer locks for multiple clients
  - Used by RLDevs, the chunk manager, and the BoxFS server
  - Has a single master and multiple backup instances
  - The master server, and all clients, are stored using Paxos
  - Failure detector used by backups to identify a failed master
- Locks are used as degenerate leases
  - Failure detector identifies failed clients and frees their locks
- Only a single master is used
  - It is believed additional scalability is not needed
  - A single lock will only ever be implemented by a single server

Replicated Logical Devices (RLDev)

- The key component of Boxwood
  - Behaves like a typical block device
  - Uses chained declustering replication
  - All information about RLDevs is stored using Paxos
    - List of RLDevs, primary and secondary hosts for each RLDev, etc.
- Provides a low level replicated storage interface
  - Simplifies the upper layer implementations
- Each RLDev is replicated on multiple machines
  - Currently only 2 copies exist, on a primary and a secondary

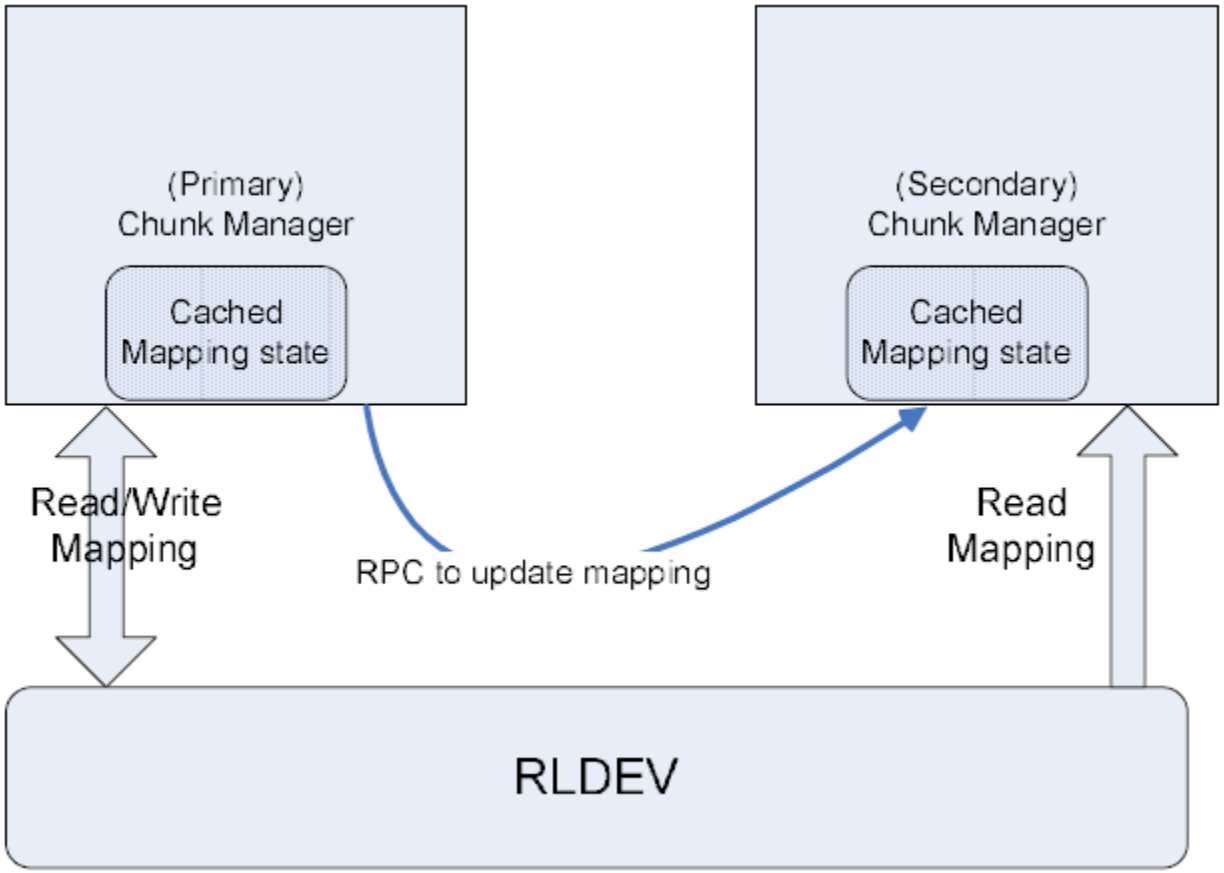
Replicated Logical Devices (RLDev)

- Clients write to the primary, reads from either
  - Writes block until replicated to the secondary
  - During failure, the other host will accept *degraded mode* writes
- Degraded mode writes are saved to a log file
  - Simplifies reconciliation when the other host comes back
- Primary also maintains a log of all in-flight writes
  - Dirty region log, simplifies recovery from transient failures
  - Clients can disable this log, but then must handle consistency
- Load balancing can be obtained by migrating RLDevs
  - Not a whole lot of detail on this in the paper
  - Who makes the decision?

Chunk Manager

- A chunk is the basic unit of user storage in Boxwood
  - Sequence of consecutive bytes allocated on a RLDev
  - Each chunk is uniquely identified with an opaque handle
- 4 supported operations
  - Allocate, free, read, and write
- Chunk managers are run in pairs for fault tolerance
  - Only the primary does alloc and free, either can read & write
  - Each chunk manager only manages chunks on a set of RLDevs
  - Mappings from handles to chunk offsets are stored on a RLDev
  - Updates to the mapping table are protected by the *map lock*

Chunk Manager



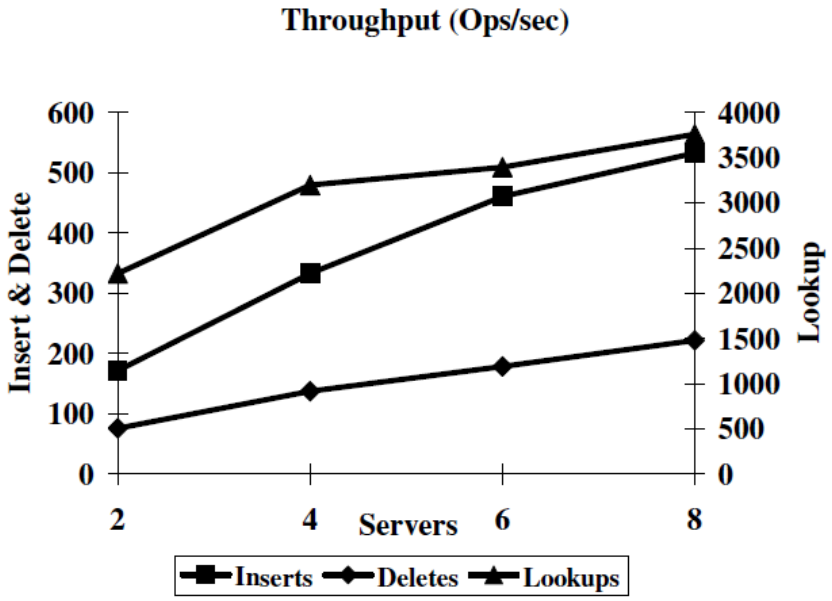
Source: Boxwood: Abstractions as the Foundation for Storage Infrastructure, J. MacCormick et al.

B-Tree Service

- B-trees are commonly used to implement dictionaries
  - This B-tree module is the first of many envisioned in Boxwood
  - B-trees are commonly used in file systems – useful for BoxFS
- Implements a distributed Sagiv B-link tree
  - Locking is much simpler in Sagiv than alternatives
  - A global lock is used to synchronize shared access
  - Operations on a single B-link tree provide ACID properties
  - Clients must enforce ACID properties on multiple trees

Performance Results

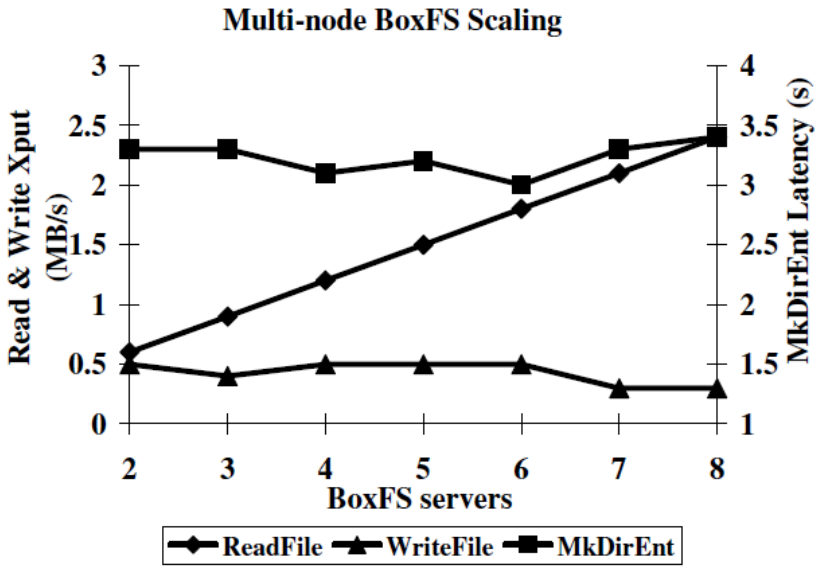
- RLDev results
  - Throughput increases as packet size increases, but disk utilization decreases
- Chunk Manager results
  - Batching chunk allocation requests greatly reduces latency
- B-tree results
  - Scales well when operating on many independent trees
  - Contention on a single tree reduces scalability noticeably



Performance operating on a shared tree

Putting it all together: BoxFS

- Builds a typical file system on top of Boxwood
  - Uses the B-tree service to implement the file system hierarchy
  - Files are directly stored using the chunk manager
- Locking is fine-grained
  - Multiple clients can lock different chunks in the same file



Conclusions

- Created the tools needed in a distributed storage system
  - Distributed consensus, locking, replicated data store
- Layering provides a platform to add new services
  - Demonstrated with the B-tree service
- Scaling to small numbers of machines is possible
  - Larger configurations unknown
- Developing actual user services is straightforward
  - BoxFS demonstrates reasonable performance

Points to Consider

- Why has this approach not taken off?
- Will application writers trust the provided infrastructure?
- How many times are storage algorithms written?
  - Kind of silly to have a whole framework when only 2 things in the world will use it...
- Why are chunks not protected by a checksum?
  - GFS made a point of including this
- Comparisons to LVM?