FuzzyLog

A Partially-Ordered Shared Log

Dec. 17th, 2018

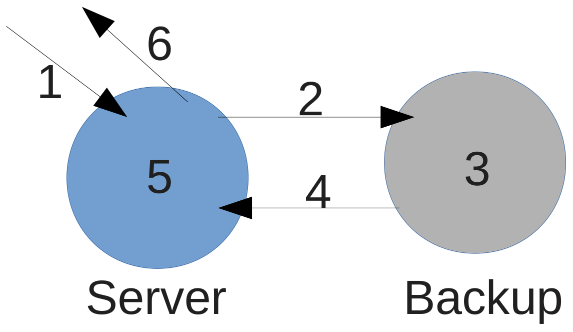
Background: Control Panel Services

- Control panel services (coordinator, schedulers, filesystem namenodes, …)
- States are complex (in-memory data structures)
- Typically implemented on a single server
  - GFS: “*Having a single master vastly simplifies our design and enables the master to make sophisticated chunk placement and replication decisions using global knowledge.*” [2]

Background: Problems

- Single-point failure
- Doesn’t scale well
- Distribution of state can be difficult

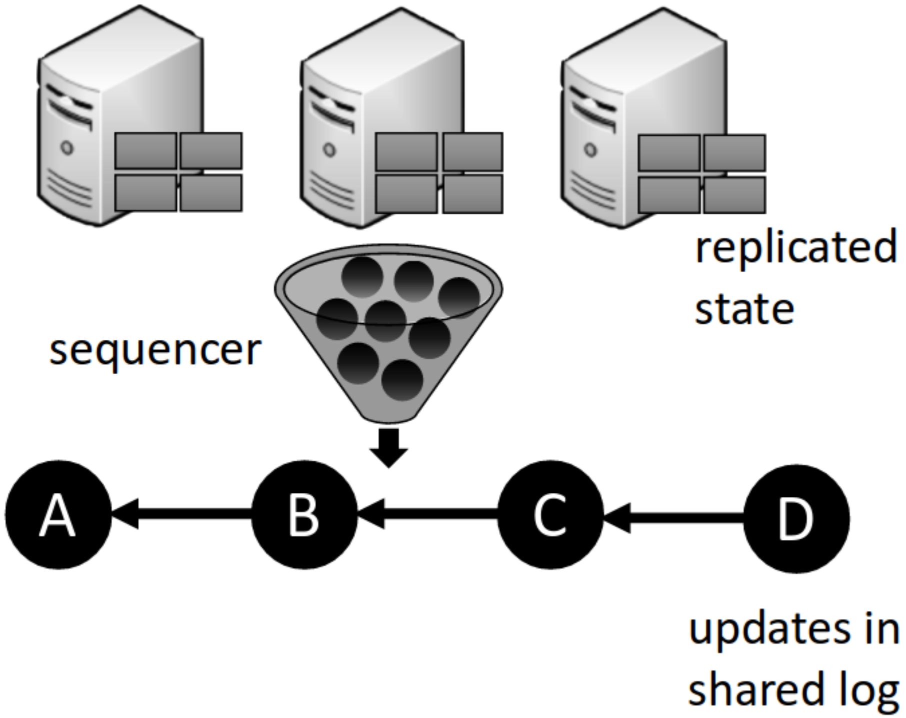
Background: Existing Solutions



- Real-time backup
  - Switch to backup server on failure
  - e.g., GFS [2]
  - Simple, inherently consistent
  - Doesn’t scale well
- Distributed Protocols
  - e.g., Paxos, 2PC
  - Complex, inefficient, difficult to merge into our own platform

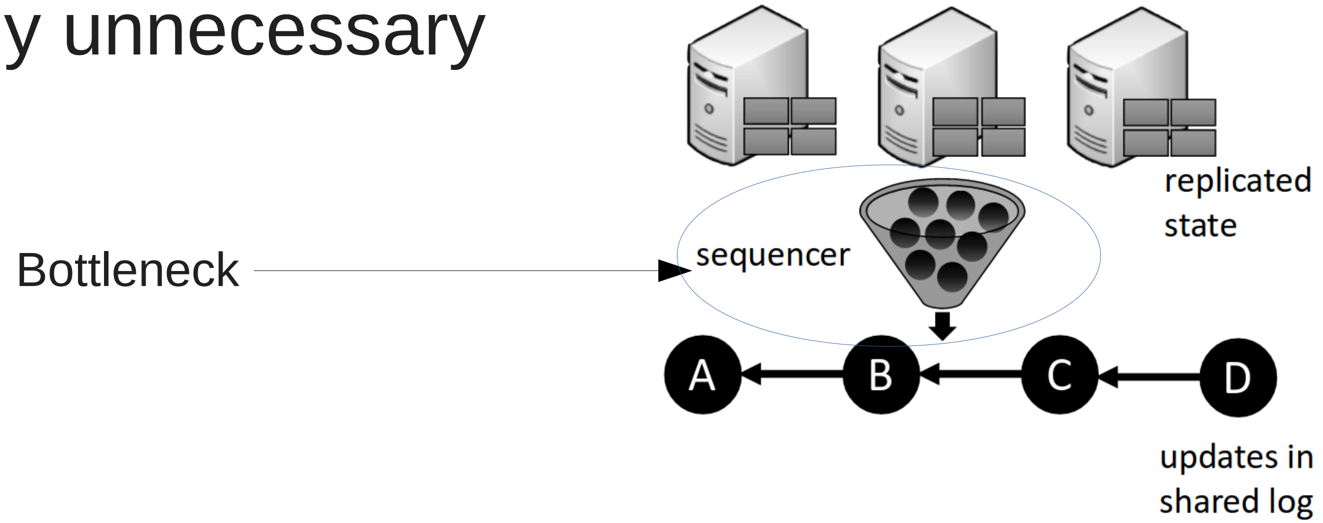
Background: Shared Log

- Another solution
- A simple layer that maps higher-level operations to appends/reads on the log



(Conventional) Shared Log

- Imposes a global total order on all nodes (to maintain consistency)
  - Always expensive
  - Often impossible
  - Typically unnecessary

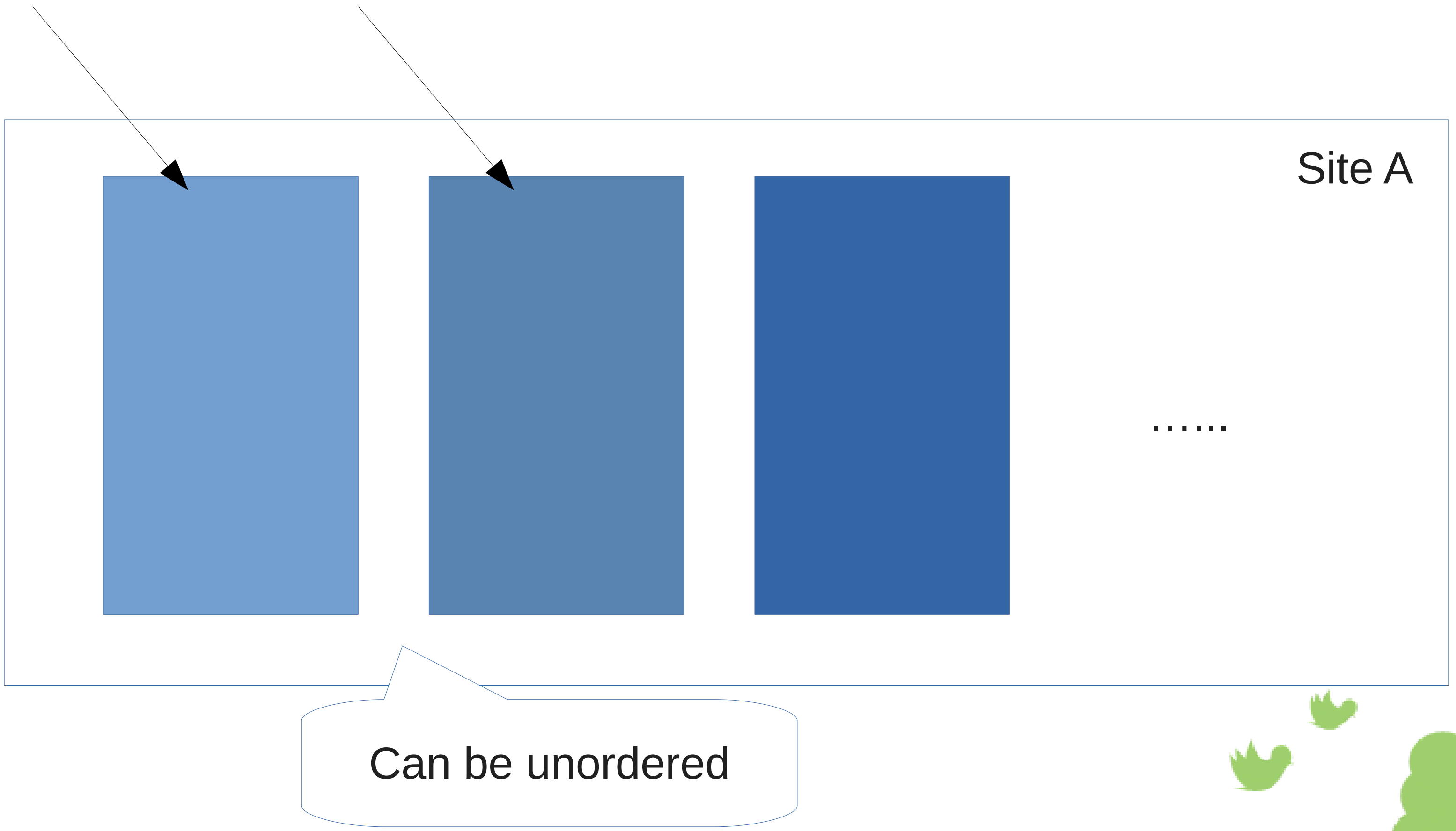


Can we provide the simplicity of a shared log without imposing a total order?

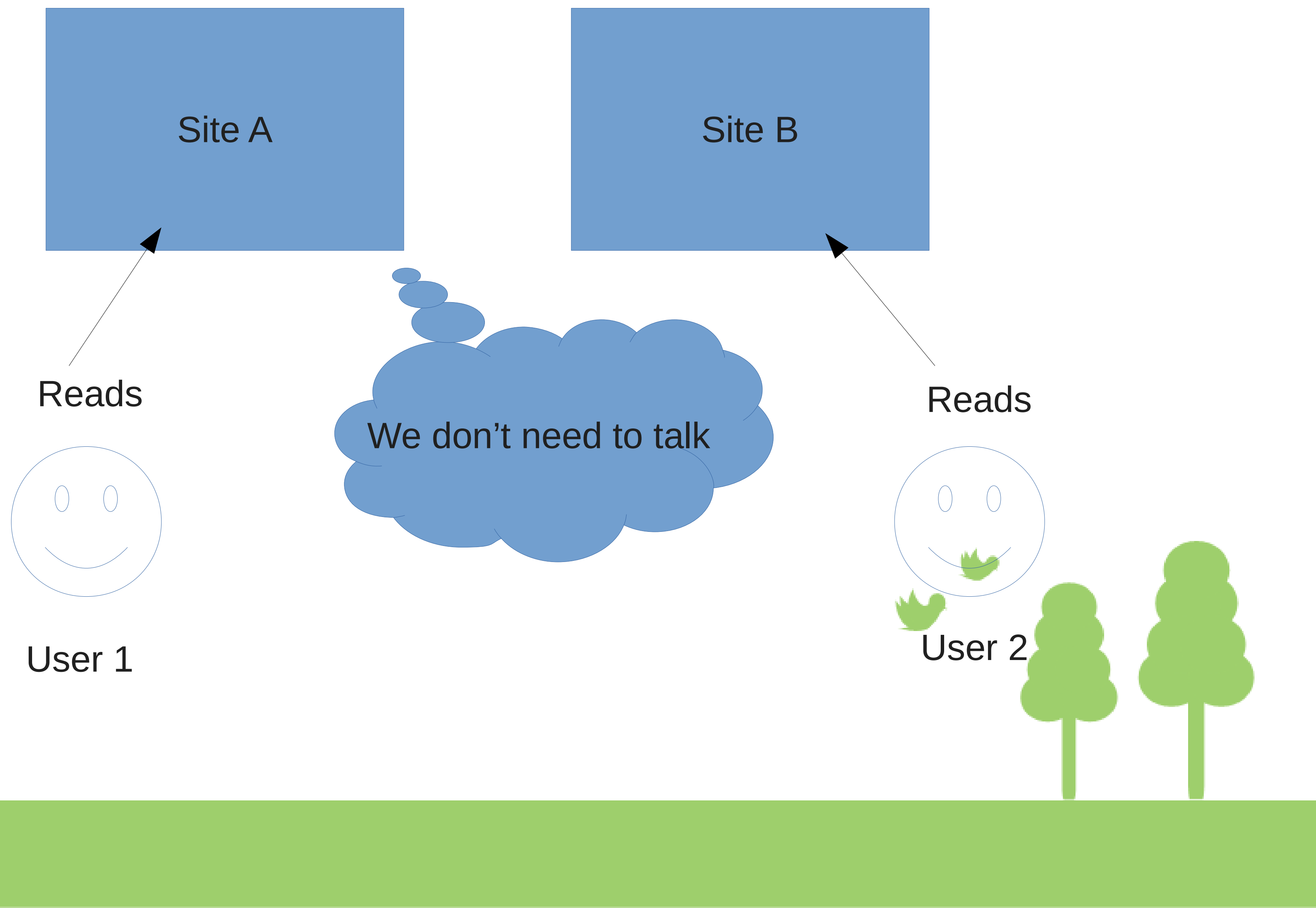
Introducing FuzzyLog

- A *partially-ordered* shared log
- ‘Fuzzy’
- Two sources of ‘partial order’
  - Sharding
  - Geo-replication

Motivation: Sharding



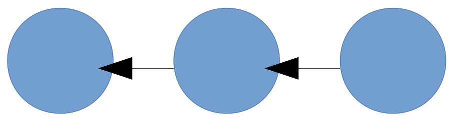
Motivation: Geo-replication



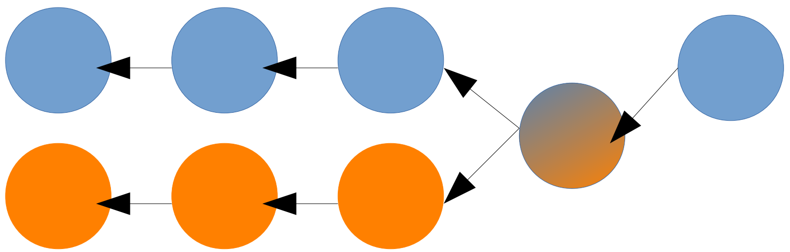
Representation

- DAG — partial order
- Color — shard
- Chain — log of each site
  - Replicated on every site
  - Each chain totally ordered
- “B ← A” — A happens-after B

(a) Single site, single shard

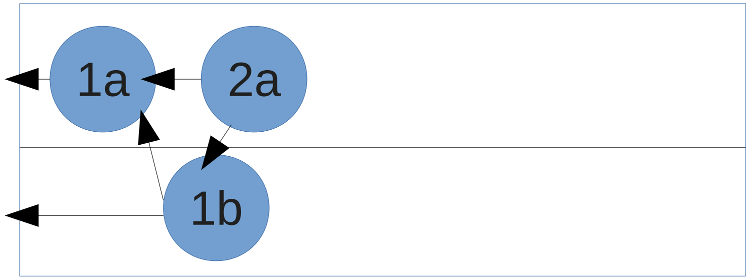


(b) Single site, multiple shards



(c) Multiple sites, single shard

Site A

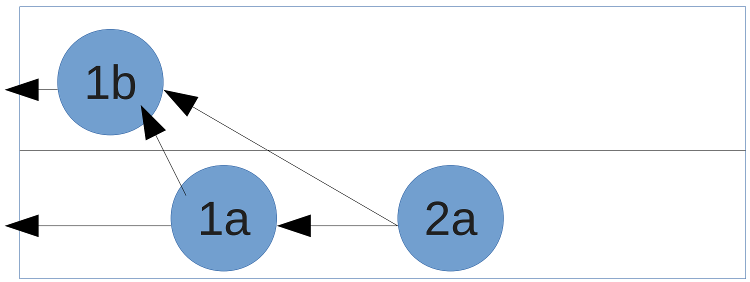


$1a \leftarrow 1b \leftarrow 2a$

(d) Multiple sites, multiple shards

(TODO) fig-b + fig-c

Site B



$1b \leftarrow 1a \leftarrow 2a$

(Causality maintained when append in this manner)

API

- handle = new_instance(colorID, snapshotID)
  - Create a new shard on a site
  - Based on snapshotID
- append(handle, data, nodeColors)
  - Append a new entry
- sync(handle, callback)
  - Client syncs with FuzzyLog
- trim(handle, snapshotID)
  - Reduce log size

Why does it work?

- (not explicitly answered in the paper)
- By nature, it is shared log, which is already known to work (despite its inefficiency)
- Casual consistency guarantee (on a single shard)
- Serializability (each chain)

- Conclusion: operations to all shards on every site can be *correctly* ordered

Implementation

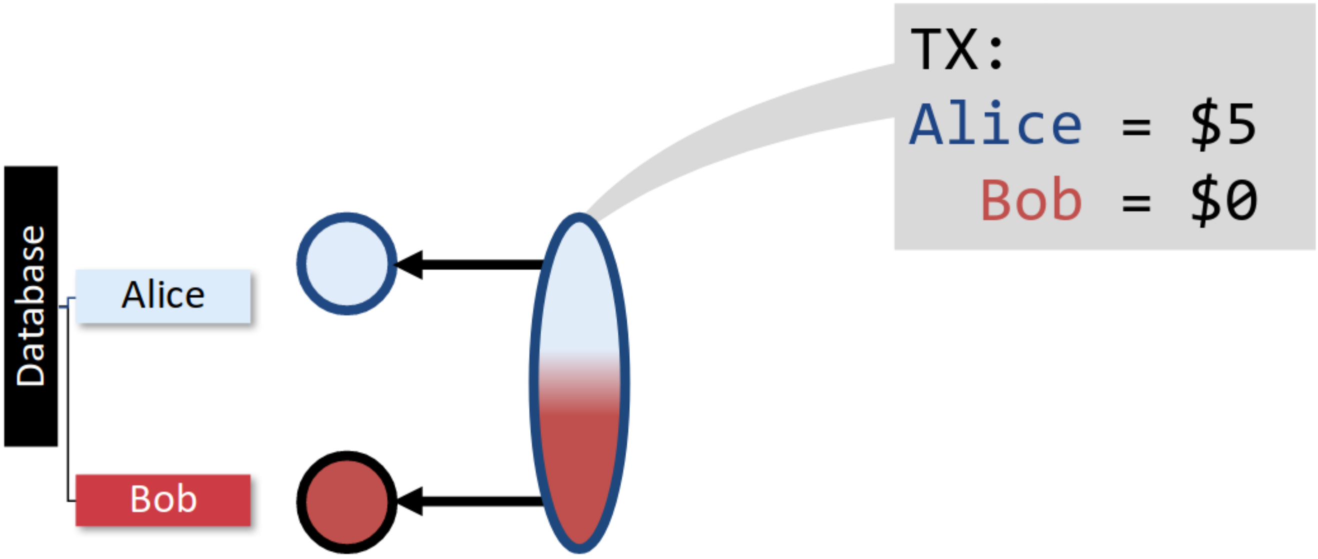
- Dapple
  - A FuzzyLog server / platform
  - On which new applications can be developed
  - Scalable, space efficient, high performance
  - https://github.com/JLockerman/FuzzyLog
    - Coded in Rust

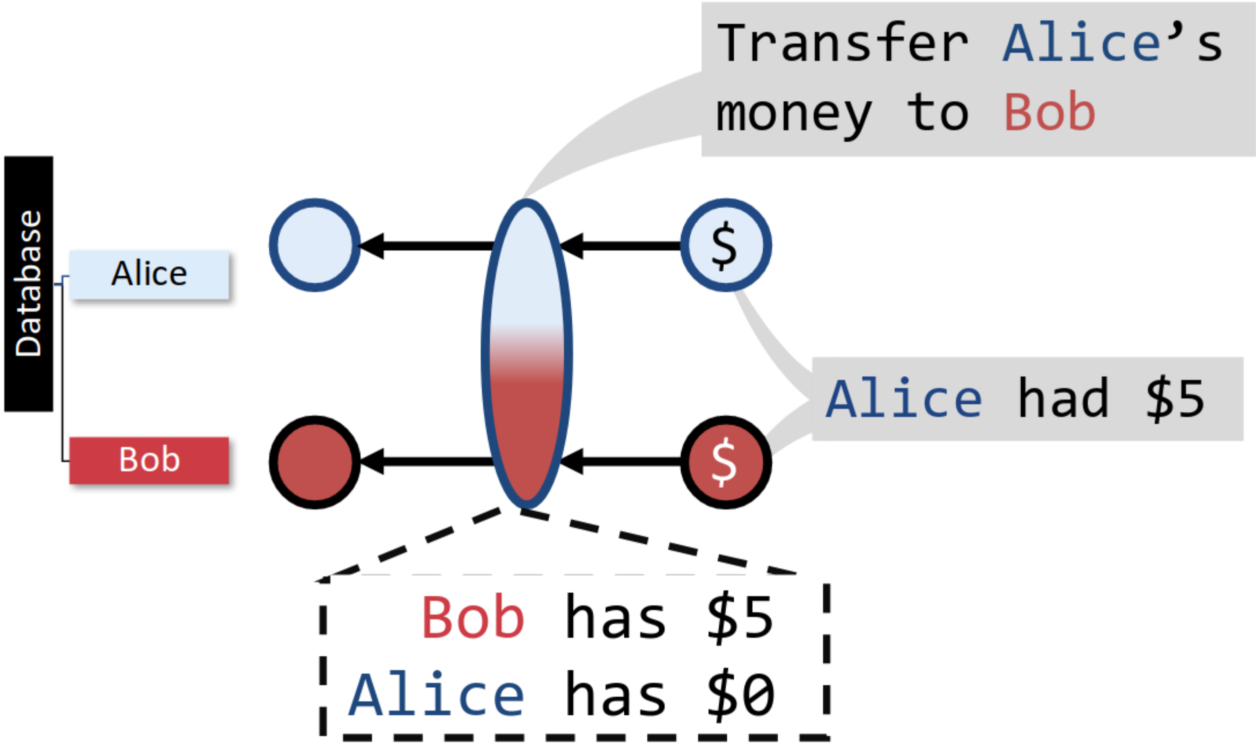
Implementation

- At each site,
  - Logs from different sites are stored separately (as a chain)
  - Chain replication
  - Remote states are retrieved periodically (chainserver itself acts as a client)
- Multiple color operation
  - Skeen’s algorithm
  - Formal verification in Coq
  - Usually 2 phases; 3 phases when a client crashes

Application: AtomicMap

- **Goal**: atomic consistency
- **Scenario**: single site, multiple shards
- **How**
  - Write: Append entries to all chains even if the colors differ
  - Read: Append entries only to the corresponding color
- **Observe**
  - All shards share a common write history
  - Reads always correspond to some linearizability
  - Strict serializability!


TX:
Alice = $5
Bob = $0
Database
Alice
Bob


Transfer Alice's money to Bob
Database
Alice
Bob
$
$
Alice had $5
Bob has $5
Alice has $0

Application: CRDTMap

- **Goal**: causal consistency
- **Scenario**: multiple sites, single shard
- **How**: use one color for all operations
- **Why**
  - The operations for one color are causally consistent
  - Therefore, all operations are causally consistent

Application: CAPMap

- **Goal**: best-effort consistency
  - Strong consistency
  - Causal consistency during network partitions
- **Scenario**: multiple sites, single shard
- **How**
  - Server appends to primary site; syncs until it sees ‘put’ itself
  - When partitioned, appends to local history; after the partition healed, throw away local state and replay primary logs + local logs
- **Observe**
  - Primary guarantees serializability, thus strong consistency
  - When the partition heals, primary syncs and replays all logs since last sync, (all nodes converge to the same state eventually), thus causal consistency

Application: RedBlueMap

- **Goal**: RedBlue consistency
- **Scenario**: multiple sites, single shard
- **How**
  - Single color
  - Red ops routed to the primary site
  - Blue ops performed on the secondary site
- **Why**
  - Red ops are totally ordered against each other
  - Blue ops commute with each other
  - This indeed is RedBlue consistent

Evaluation: Latency

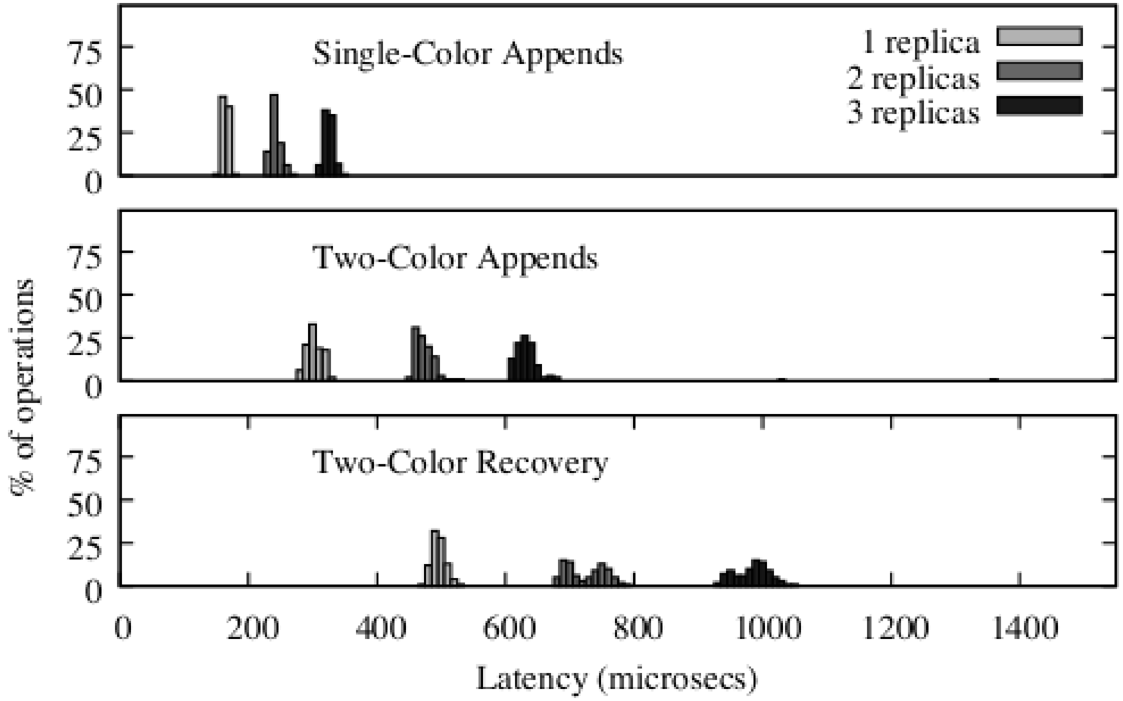


Figure 6: *Dapple executes single-color appends in one phase; multi-color appends in two phases; and recovers from crashed clients in three phases.*

Evaluation: Scalability

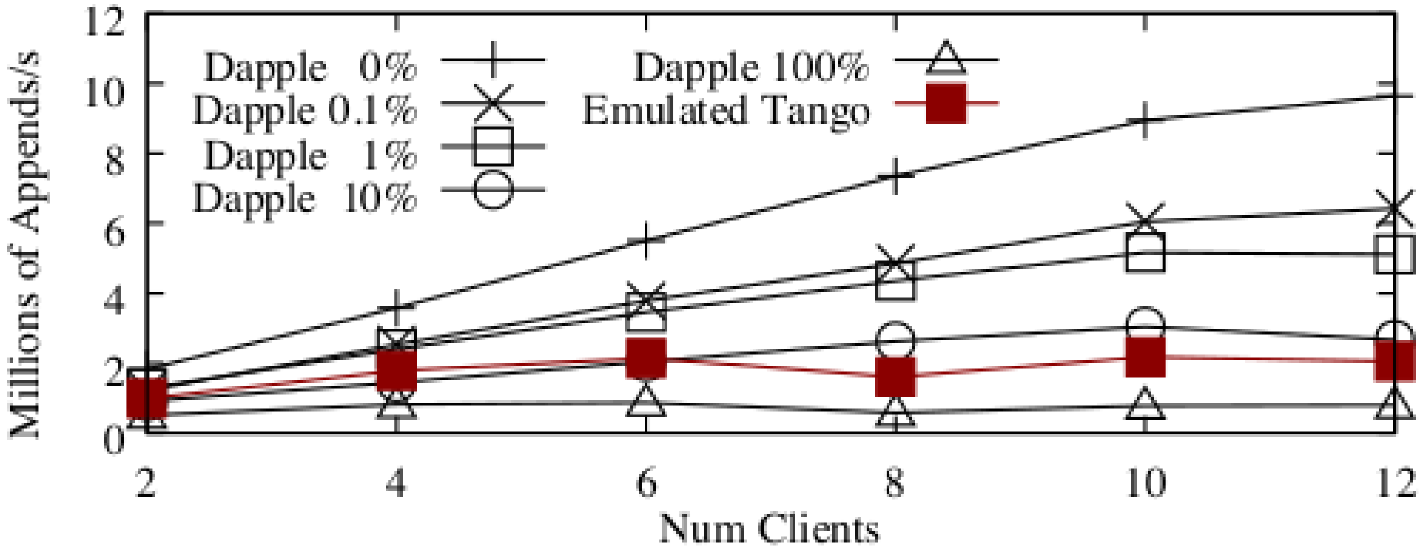


Figure 7: *Dapple scales with workload parallelism, but a centralized sequencer bottlenecks emulated Tango.*

Shared log systems scale with #clients badly
FuzzyLog systems scale well when multi-shard 'put's are rare

Evaluation: Scalability

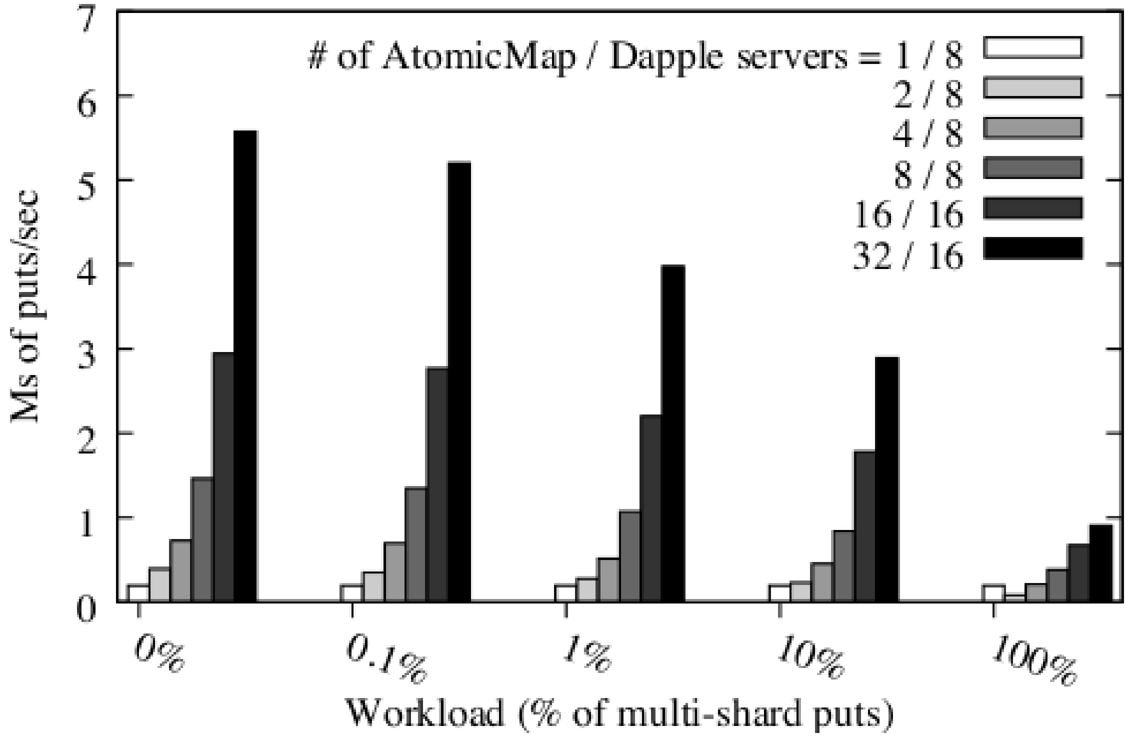


Figure 8: *AtomicMap scales throughput while supporting multi-shard transactions. Each bar labelled N / K shows throughput with N AtomicMap servers running against a K-server Dapple deployment.*

FuzzyLog systems scale well with #servers

Evaluation: Weaker Consistency

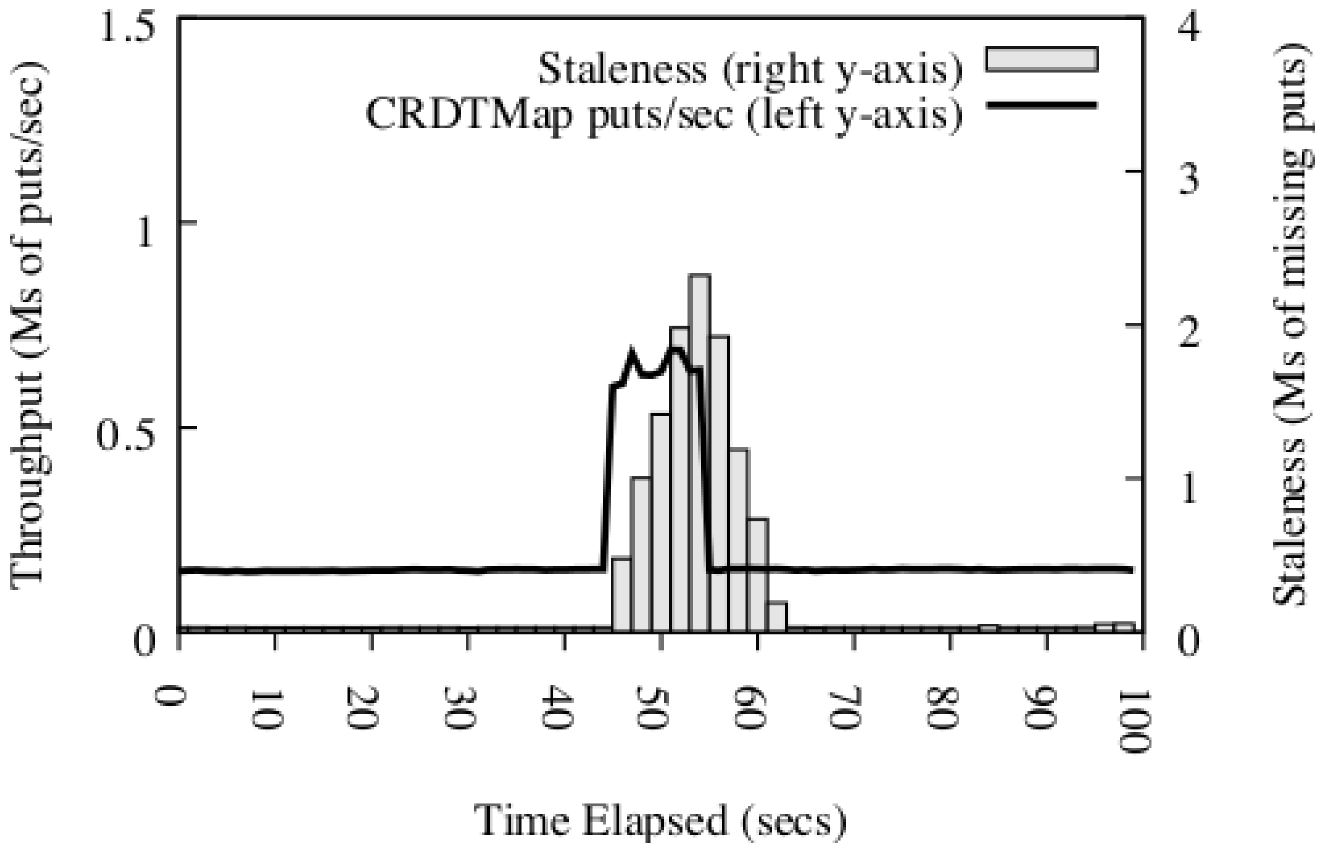


Figure 9: *CRDTMap provides a trade-off between throughput and staleness.*

Evaluation: Partition Tolerance

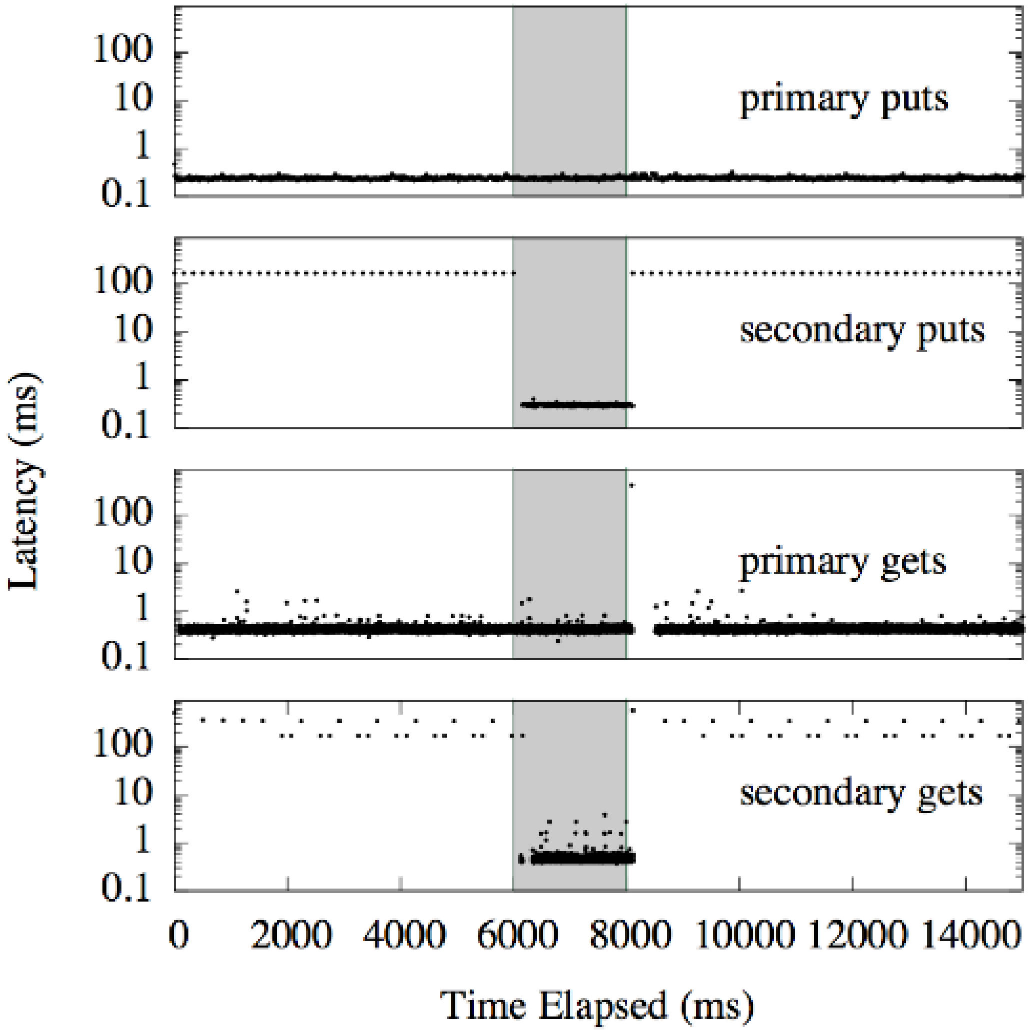


Figure 10: *CAPMap switches between linearizability and causal+ consistency during network partitions.*

Conclusion

- Simple and intuitive
- Powerful and flexible
- Performant
- Easy to build applications atop
  - All of these applications only require several hundreds of lines of code!
- Different levels of consistency guarantees
- Handle network partitions gracefully

References

1. The FuzzyLog: A Partially Ordered Shared Log

   (referenced throughout the slides)
2. The Google file system