Practical Replication

The Dangers of Replication and a Solution
(SIGMOD'96)

The Costs and Limits of Availability for Replicated Services (SOSP'01)

Presented by: K. Vikram, Cornell University

Why Replicate?

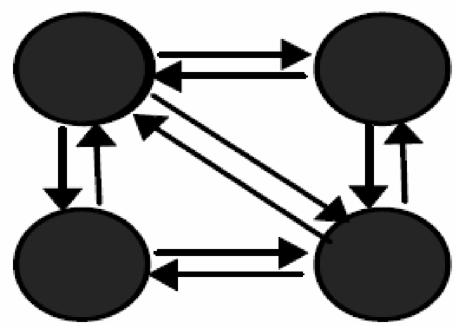
- Availability
  - Can access resource even if some replicas are inaccessible
- Performance
  - Can choose the replica that gives high performance (eg. closest)

Data Model

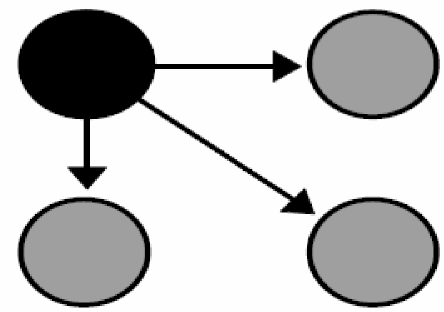
- Fixed set of objects
- Fixed number of nodes
  - Each has a replica of all objects
- No hotspots
- Inserts, Deletes $\rightarrow$ Updates
- Reads ignored
- Transmission and Processing delays ignored

Dimensions

- Eager vs. Lazy
- Group
  - Update anywhere



- Master
  - Only the primary copy can be updated



Comparison

Table 1: A taxonomy of replication strategies contrasting propagation strategy (eager or lazy) with the ownership strategy (master or group).

Propagation vs. Ownership	Lazy	Eager
Group	N transactions N object owners	one transaction N object owners
Master	N transactions one object owner	one transaction one object owner
Two Tier	N+1 transactions, one object owner tentative local updates, eager base updates	

Eager Replication

- Update all replicas at once
- Serializable Execution
- Anomalies converted to waits/deadlocks
- Disadvantages
  - Reduced (update) performance
  - Increased response times
  - Not appropriate for mobile nodes

Waits/Deadlocks in Eager Replication

- Disconnected nodes stall updates
  - Quorum/cluster enhanced update availability
- Updates may still fail due to deadlocks
- Wait Rate:

$$\frac{TPS^2 \times Action\_Time \times (Actions \times Nodes)^3}{2 \times DB\_Size}$$

BAD!

- Deadlock Rate:

$$\frac{TPS^2 \times Action\_Time \times Actions^5 \times Nodes^3}{4 \times DB\_Size^2}$$

Waits/Deadlocks in Eager Replication

- Can we salvage anything?
- Assume DB increases in size

$$\frac{TPS^2 \times Action\_Time \times Actions^5 \times Nodes}{4 \times DB\_Size^2}$$

- Perform replica updates concurrently
  - Growth rate would be quadratic

Lazy Replication

- Asynchronously propagate updates
- Improves response time
- Disadvantages
  - Stale versions
  - Reconcile conflicting transactions
  - Scaleup Pitfall (cubic increase)
  - System Delusion (inconsistent beyond repair)

Lazy Group Replication

- Use of timestamps for reconciliation
  - Objects have update timestamps
  - Updates have new value + old object timestamp
- Reconciliation Rate: $\frac{TPS^2 \times Action\_Time \times (Actions \times Nodes)^3}{2 \times DB\_Size}$
- Cubic increase still bad
- Collisions when disconnected

$$\frac{Disconnect\_Time \times (TPS \times Actions \times Nodes)^2}{DB\_Size}$$

Lazy Master Replication

- Each object has an owner
- To update, send an RPC to owner
- After owner commits, source broadcasts the replica updates
- Not appropriate for mobile applications
- No reconciliations, but we may have deadlock
- Rate: $$\frac{(TPS \times Nodes)^2 \times Action\_Time \times Actions^5}{4 \times DB\_Size^2}$$

Simple Replication doesn't work

- "Transactional update-anywhere-anytime-anyway"
- Most replication schemes are unstable
  - Lazy, Eager, Object Master, Unrestricted Lazy Master, Group
- Non-linear growth in node updates
  - Group and Lazy Replication (N^2)
- High deadlock or reconciliation rates
- Solution: Restricted form of replication
  - Two Tier Replication

Non-transactional replication schemes

- Abandon serializability, adopt convergence
- If connected, all nodes eventually reach the same replicated state after exchanging updates
- Suffers from the *lost update* problem
- Using commutative updates helps
- Global serializability still desirable

An ideal scheme should have

- Availability and Scaleability
- Mobility
- Serializability
- Convergence

Probable Candidates

- Eager and Lazy Master
  - No reconciliation, no delusion
- Problems
  - What if master is not accessible
  - Too many deadlocks
- How do we work around them?

Two-Tier Replication

- Base Nodes
  - Always connected (owns most objects)
- Mobile Nodes
  - Usually disconnected (originates tentative Xns)
  - Keeps two versions: local & best known master

Two-Tier Replication

- Two types of transactions
  - Base (several base + at most one connected-mobile node)
  - Tentative (future base transaction)
- Mobile → Base
  - Propose tentative update transactions
  - Databases synchronized

Two-Tier Replication

- Tentative Transaction might fail
  - Acceptance Criterion
- Originating node is informed on failure
- Similar to reconciliation but
  - Master is always converged
  - Originating nodes need to contact just some base node
- Lazy Replication w/o System Delusion

Analysis

- Deadlock rate is N^2
- Reconciliation rate is zero if transactions commute
- Differences between results of tentative and base transaction needs application specific handling

To Conclude

- Lazy-group schemes simply convert deadlocks to reconciliations
- Lazy-master is better but still bad
- Neither allow disconnected mobile nodes to update
- Solution:
  - Use semantic tricks (timestamps + commutativity)
  - Two-tier replication scheme
  - Best of eager-master-replication and local update

Availability is the new bottleneck

- Too much focus on performance
- Local availability + network availability
- Caching and Replication
- Consistency vs. Availability
- Optimistic Concurrency
- *Continuous Consistency*
- Availability depends on
  - Consistency level, protocol used for consistency, failure characteristics of the network

Continuous Consistency

- Generalize the binary decision between
  - Strong Consistency
  - Optimistic Consistency
- Specify exact consistency required based on
  - Client, network and service characteristics

Continuous Consistency

- Applications specify maximum distance from strong consistency
- Exposes consistency vs. availability tradeoff
- Quantify Consistency and Availability
- Help system developers decide on how to replicate
  - Given availability requirements
- Self-tuning of availability

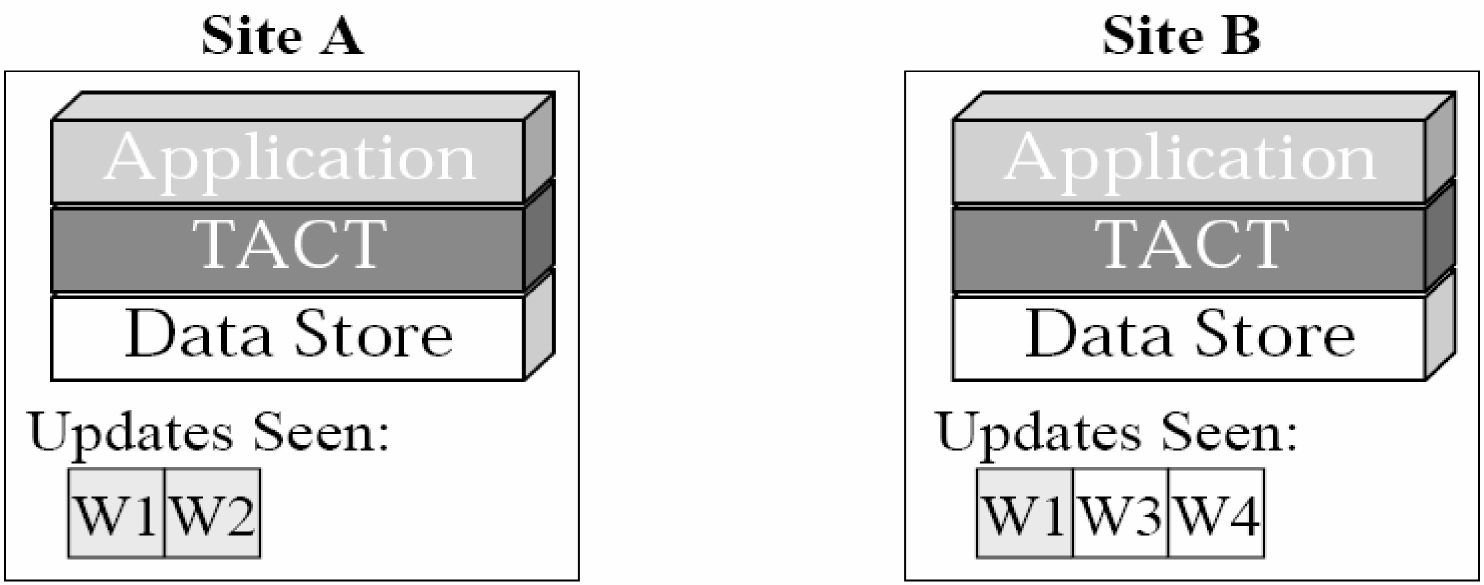
The TACT Consistency Model

- Replicas locally buffer a maximum number of writes before requiring remote communication
- Updates are modeled as procedures with application specific merge routines
- Update carries application-specific weight
- Updates are either tentative or committed

Specifying Consistency

- Numerical Error
  - Maximum weight of writes not seen by a replica
- Order Error
  - Maximum weight of writes that have not established final commit order (tentative writes)
- Staleness
  - Maximum time between an update and its final accept

Example



NE = 2 (from W3, W4)
OE = 0

NE = 1 (from W2)
OE = 2 (from W3, W4)

(Assume Serialization Order = W1 W2 W3 W4)

System Model

- Model replica failures as singleton network partitions
- Assume failures are symmetric
- Processing and network delays ignored
- Submitted client accesses
  - Failed, rejected or accepted
- $Avail_{client}$ = accepted/submitted

  = $Avail_{network} \times Avail_{service}$

Replication

Service Availability

- Workload
  - Trace of timestamped accesses
  - Accesses that reach a replica
- Faultload
  - Trace of timestamped *fault events*
  - Fault events divide a run into *intervals*

Bounds on Availability

- $\text{Avail}_{\text{service}} \leq \mathcal{F}$ (consistency, workload, faultload)
- Upper bound on availability
- Independent of consistency maintenance protocol
- Gives system designers a baseline to compare their availability against

The Intuition

- Consistency protocol answers *questions*
  - Which writes to accept/reject from clients
  - When/Where to propagate writes
  - What is the serialization order
- For upper bound, optimal answers are needed
- Exponentially many answers
  - How do we make this tractable?

Methodology

- Partition into $Q_{offline}$ and Q_{online}
- Use pre-determined answers to $Q_{offline}$ to construct a *dominating algorithm*
- Given a workload and faultload, P_1 dominates P_2 if
  - P_1 achieves same/higher availability than P_2
  - P_2 achieves same/higher consistency than P_2
- Upper bound is the availability achieved by $\mathcal{P}$ that dominates all protocols

Methodology

- Some inputs to the dominating algorithm exist which make it dominate all others
- Search answers to Q_{online} to get an optimal dominating algorithm
- Maximize $Q_{offline}$ to keep it tractable

Numerical Error and Staleness

- Pushing writes to remote replicas always helps
- Thus, write propagation forms $Q_{offline}$
- Write acceptance form Q_{online}
- Exhaustive search on possible sets of accepted writes intractable
- Aggressive write propagation allows a single logical write to represent all writes in a partition – reduces search space
- Reduces to a linear programming problem

Order Error

- Aggressive write propagation coupled with remote writes being applied only when they can be committed
- Write commitment depends on serialization order
- Domination relationship between serialization orders
- Three sets of serialization orders
  - ALL, CAUSAL, CLUSTER

Example

- Replica 1 receives W_1 and W_2, Replica 2 receives W_3 and W_4
- $S = W_1W_2W_3W_4$ dominates $S' = W_2W_1W_3W_4$
- CAUSAL = W_1 precedes W_2 and W_3 precedes W_4
- CLUSTER = $W_1W_2W_3W_4$ or $W_1W_2W_3W_4$
- CLUSTER > CAUSAL > ALL

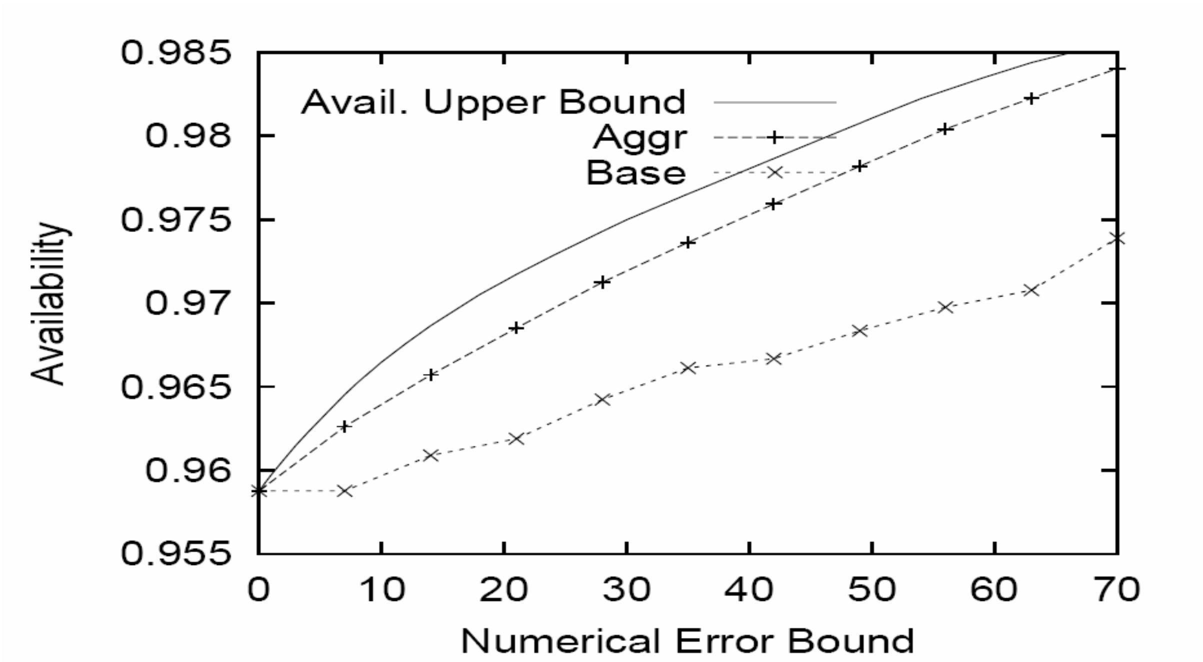
Complexity

- Exponential in worst case
- Linear programming approximated
- Serialization order enumeration was found tractable in practice

Evaluation

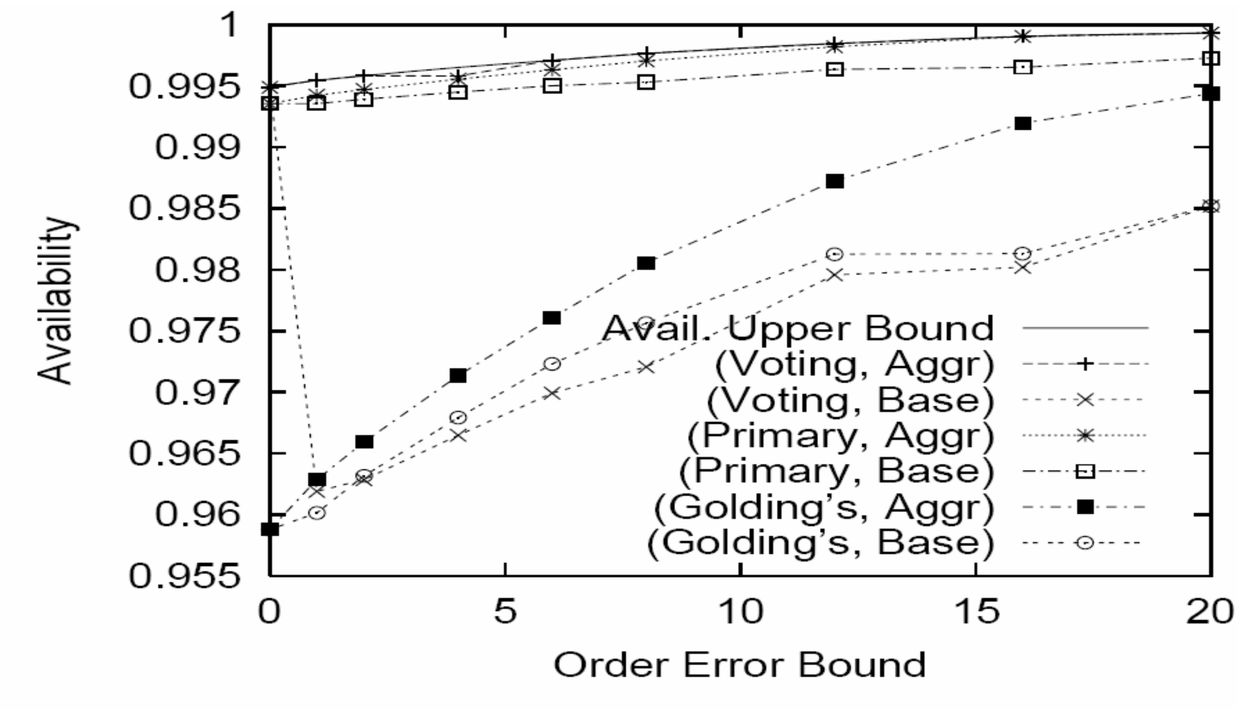
- Construct synthetic faultloads with varying characteristics
- Various consistency protocols
- Write Commitment
  - Primary Copy
    - Write is committed when it reaches the primary copy
  - Golding’s algorithm
    - Each write assigned a logical timestamp
    - Replica maintains a version vector
  - Voting
    - Serialization order decided through a vote

Availability as a function of numerical error bound



Pushing writes aggressively enhances availability

Availability as a function of order error

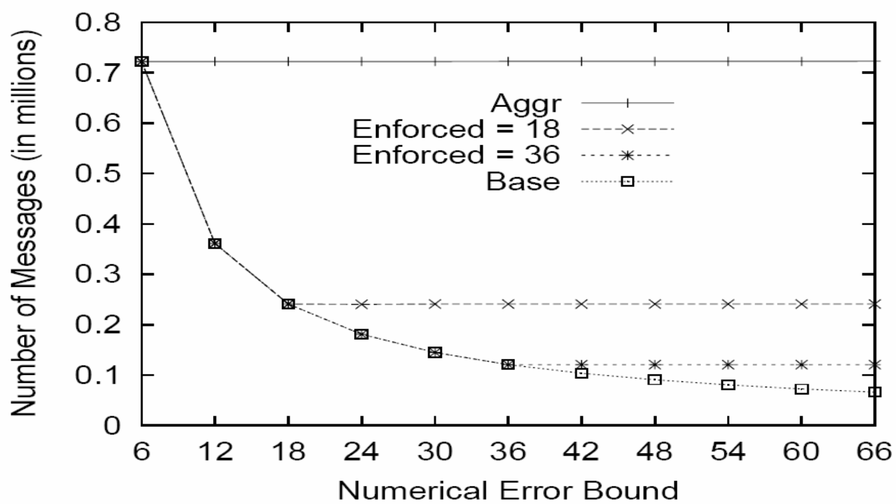


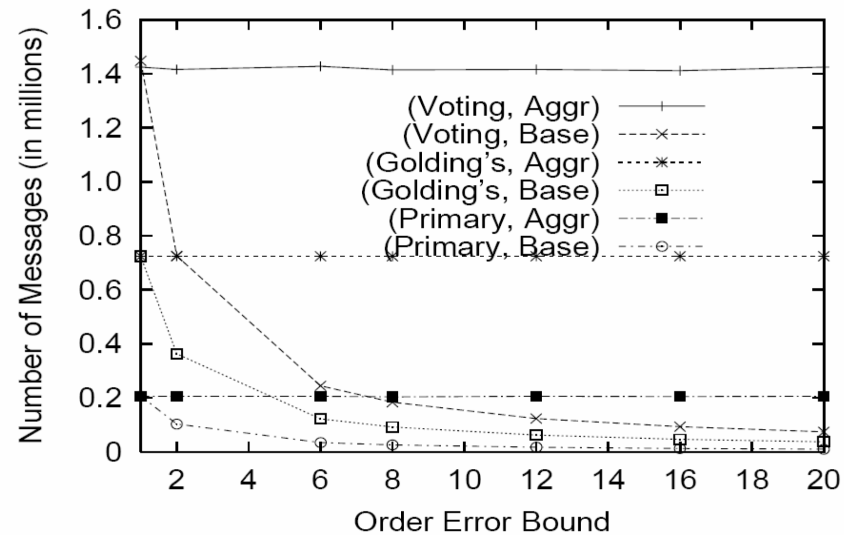
- Primary copy has highest level of availability
- With aggressive order error bounding, voting achieves highest availability

Evaluation

- Other faultloads yielded similar results
- Theoretical bounds were reached because
  - All partitions were singleton partitions
  - For most failures, the system transitions from fully connected to singleton partition and back
- Faultloads without these properties cannot reach the bounds
- However, properties are somewhat consistent with the Internet

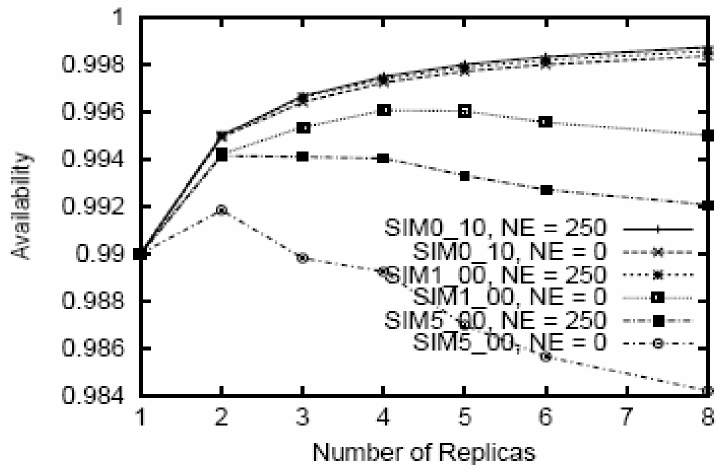
Availability vs. Communication

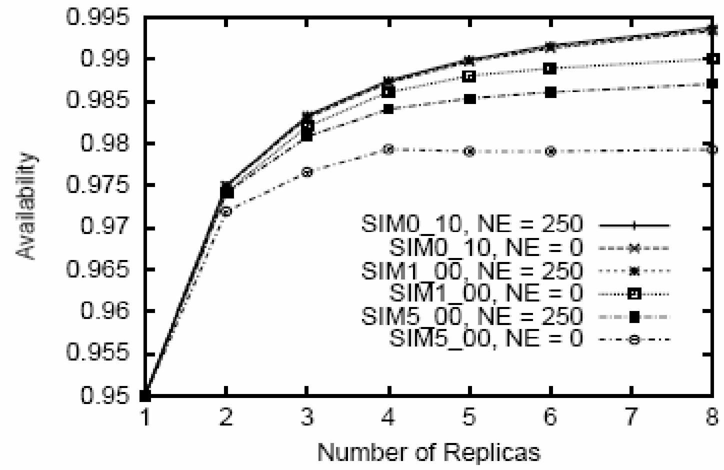




Achieving maximum service availability with a relaxed consistency model can Entail increased communication overhead

Effects of Replication Scale





There is typically an optimal number of replicas

Conclusion

- Simple optimizations to existing consistency protocols can greatly improve availability
- Voting and primary copy achieve best availability
- Additional replicas are not always useful
- Higher availability can be achieved only by relaxing consistency