Towards a Unified Theory of Replication

Mike Dahlin, Lei Gao, Amol Nayate, Praveen Yalagandula, Jiandan Zheng
Department of Computer Sciences
University of Texas at Austin
Why a Unified Theory of Replication?

(1) Better way to build replication systems

(2) Way to build better replication systems
Better Way to Build Replication Systems

Separate mechanism from policy
- Continuum of policies v. point solutions

Simpler to design and deploy
- Replication microkernel or toolkit

Integrate disparate theories/protocols
- Quorums, client-server, leases, server replication, p2p, ...

Simplify teaching
- A few principles v. a bunch of case studies

Goal: Reduce the development effort for a new replication system by an order of magnitude
A Way to Build Better Replication Systems

Synchronize palmtop to laptop

- Client-server: Limited by network to server
- Bayou: Limited by fraction of shared data (1%)

Order of magnitude improvements available!
Outline

Case for a unified theory of replication
PRACTI: A first step
Evaluation
Future directions
Case for a Unified Theory of Replication*

Current systems entangle mechanism with policy
- E.g., Coda v. Bayou
- 14 OSDI/SOSP papers in 10 years
  - New environment → new trade-offs → new mechanisms
  - Not clear new systems dominate old ones (or that 14 is “enough”)

Current literature fragmented
- Client-server v. quorums v. server replication v. p2p v. ...
- E.g., Coda and Bayou each have separate server-replication and client-server caching protocols

Impact
- Systems narrowly tailored for specific environments
- Significant effort to develop system for new environment

* Scope: “Large scale” replication
- WAN, mobile, enterprise, etc.
- File systems, tuple stores, databases, distributed objects, ...
Vision: Replication Microkernel/Toolkit

Grand Challenges:

• Each large-scale FS from OSDI/SOSP 1990-2005 as <1000-line “policy layer”

• “Universal policy” - self-tuning replication
  • Control replication to meet high level goals
  • e.g., “Minimize response time and maximize availability while providing causal consistency and less than 1 minute staleness to all replicas while using less than 2x demand-read traffic.”
Outline

Case for a unified theory of replication
PRACTI: A first step
Evaluation
Future directions
“Towards” a Unified Theory

Not there yet

• Today: PRACTI

• Unify large part of design space (almost)
  - Client-server (e.g., NFS, Coda, AFS)
  - Server replication (e.g., Bayou, TACT)
  - Object replication (e.g., Ficus, Pangea)

• Future work to incorporate
  - Quorums, general model of security, DHT-based P2P, content-keyed identifiers, …
Challenge: PRACTI Replication

Arbitrary Consistency
Provide guarantees required by application
Don’t pay for more guarantees than needed

TACT
Bayou

Partial Replication
Replicate any subset of data to any node

AFS
CODA
NFS
Pangea
GFS
WinFS (?)

Server Replication

Topology Independence
Any node can communicate with any other node

Object Replication
PRACTI Design Overview

(0) Start with Bayou
  • Log-based p2p update exchange
  • (Could also go in other direction – generalize client/server...)

(1) Separate data from metadata
  • Separate streams for invalidations and bodies
  • Challenge: Synchronize these streams

(2) Summarize unneeded metadata
  • Imprecise invalidations
  • Challenge: Track “precise” and “imprecise” data

(3) Separate mechanism from policy
  • Core: PRACTI mechanisms
  • Controller: Policy
Step 0: Start With Bayou

Updates to log
- Local checkpoint for random access

Log exchange for updates
- TI: Pairwise exchange with any peer
- AC: Prefix property, causal consistency, eventual consistency
- PR: All nodes store all data, see all updates
Step 1: Separate Data and Metadata

Separate data and metadata
- Metadata: Log invalidations
- Data: Store update bodies in checkpoint

Log exchange:
- Send invalidations separate from bodies
  → Client-server/Server-replication hybrid
**Issue: Reading Bodies**

**Mechanism: Block until data VALID**
- VALID = body matches latest invalidation

**Policy: Your choice**
- Demand read miss
  - Target is policy choice: client/server, DHT directory, original writer, random, ...
- Prefetch
  - TCP-Nice based self-tuning prefetch
Issue: Synchronization of Separate Streams

Retrieved body may be newer than metadata
→ Violate causality
→ Buffer body until apply associated inval
Step 1 Helps...

Keep good Bayou properties

• Topology independence
• Arbitrary consistency
  ▪ Prefix property
  ▪ Causal consistency
  ▪ Eventual consistency

Step towards partial replication

• Nodes only see bodies of interest
  ▪ Order of magnitude improvement!
• Nodes still see all invalidations
  ▪ Limits scalability
    - E.g., Enterprise file system in which every palmtop sees every update by any node
Step 2: Imprecise Invalidations

Nodes subscribe for
- Precise invalidations for interest sets
- Imprecise invalidations for other data

Precise invalidation
- Metadata for one write
  <object ID, accept stamp>

Imprecise invalidation
- Summary of multiple writes
  <objectSet, [start]*, [end]*>
- “One or more objects in objectSet were modified between start and end”
Imprecise Invalidations

Nodes subscribe to invalidation streams
- Specify which Interest Sets node wants to keep \textit{precise}
- \textbf{Imprecise} Interest Set
  - Replace collection of invalidations with conservative approximation
    - Recvr. treats all objects in \textit{objSet} as if invalidated between \textit{start} and \textit{end}

Bookkeeping details (see paper)
- Track which Interest Sets are missing invalidations
- Block reads to \textit{imprecise} Interest Sets
- Make interest set \textit{precise} when missing invalidations applied
Step 3: Separate Mechanism v. Policy

**Goal:** Common core mechanism

- “Replication microkernel”
- **Vision:**
  - Implement replication system for new environment in <1000 lines of policy code
Core v. Controller

**Core: Mechanism**
- **Safety**: Any message can be processed at any time
  - Asynchronous message passing style

**Controller: Policy**
- **Liveness**: Trigger messages between nodes
Controller Interface

Notified of key events
- Stream begin/end
- Invalidation arrival
- Body arrival
- Local read miss
- ...

Directs communication among cores
- Subscribe to inval or body stream
- Request demand read body

Local housekeeping
- Log garbage collection
- Cache replacement
Example: Client-Server Controller

Subscriptions

- Precise invalidations
  - Forall f in <cached files> subscribe to f from server

- Bodies
  - Forall h in <hoard list> subscribe to h from server

Local read miss on file f

if (f is imprecise)
  request metadata + body from server
else /* f is precise but invalid */
  request body from server
(read blocks until f is precise and valid)

Point of interest perhaps only to me

- Client/server crash recovery really natural/elegant
Example: EnterpriseFS Controller

Support thousands of devices
• Handful of big, geographically distributed servers
• Many desktops, laptops, palmtops, etc.

Read miss
• Use DHT to find nearest copy of data

Replication policy
• DHT tracks file popularity
  ▪ Self-tuning prefetch important updates to where they are/will be needed
• Enforce minimum replication degree for reliability and availability

Details TBD...
PRACTI Design Summary

Result: Subsume many existing mechanisms
- Client/server*: Coda, NFS, AFS, ...
- Server replication: Bayou, TACT
- Object replication: Ficus, Pangea, ...

Key ideas

1. Separate data from metadata
   - Separate streams for invalidations and bodies
   - Challenge: Synchronize these streams

2. Summarize unneeded metadata
   - Imprecise invalidations
   - Challenge: Track “precise” and “imprecise” data

3. Separate mechanism from policy
   - Core: PRACTI mechanisms
   - Controller: Policy
Additional Details

Efficient, continuous update exchange
  • Incremental log exchange

Garbage collect logs
  • Incremental checkpoint exchange using lpVV data structures

Self-tuning replication
  • Prefetch/pre-push bodies over low-priority network channel

Continuous consistency (e.g., TACT)
  • Causal consistency by default
  • Weaken: Imprecise reads (causal coherence)
  • Strengthen: Constraints layer
    ▪ Order error, temporal error, numerical error
  • Flexible conflict detection and resolution

Enforce minimum replication for availability
  • Bound invalidations

See paper for details
Outline

Case for a unified theory of replication
PRACTI: A first step

Evaluation

• Methodology
• Benefits of partial replication
• Benefits of topology independence
• Cost of supporting flexible consistency

Future directions
Methodology

How to evaluate “Unified theory”?
Partial Replication

Order of magnitude improvements:
- Both separate inval v. body AND imprecise inval
- Storage requirements see similar improvements
Topology Independence

Machines
- Laptop, palmtop, home desktop, office server

Places
- Office, home, hotel, plane
Synchronize palmtop to laptop

- **Client-server**: Limited by network to server
- **Bayou**: Limited by fraction of shared data (1%)
PlanetLabFS

Simplify running experiments
- Track current locations of files via DHT
- Flood initial data, programs from server to clients via cooperative caching
- Direct transfer of data updates among clients via cooperative caching
- Future: Self-tuning prefetching

Benchmark
- Phase 1 Disseminate:
  - Disseminate 10MB from server to all clients
- Phase 2 Process:
  - 10x pairwise exchange 1MB between random clients
- Phase 3 Post-Process:
  - Gather 1MB from each client to server
PlanetLabFS

- 3x-5x v. client-server (dissemination)
- 2.4x-9x v. server replication (process, post-process)
- 1.5x v. cooperative caching (process)
- TBD: Add self-tuning prefetching
Cost of Consistency

Tunable consistency

• Causal, causal + TACT, sequential, linearizable
• Consistent or coherent
  ▪ Consistency: Order writes across all objects
  ▪ Coherence: Order writes to individual objects

PRACTI benefits

• Semantics specified on per-read, per-write basis
  ▪ What information must a read or write wait for to complete?
    → No unnecessary read delay or write delay
• Separation of invalidations from bodies
  → Minimize delay (hence inconsistency)
Improved Consistency Trade-Offs

<table>
<thead>
<tr>
<th></th>
<th>Bayou</th>
<th>TACT-Aggressive</th>
<th>PRACTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>How</td>
<td>Batch</td>
<td>Batch</td>
<td>Incremental</td>
</tr>
<tr>
<td>When</td>
<td>Periodic</td>
<td>Frequent</td>
<td>Continuous</td>
</tr>
<tr>
<td>Invals</td>
<td>All</td>
<td>All</td>
<td>All*</td>
</tr>
<tr>
<td>Bodies</td>
<td>All</td>
<td>All</td>
<td>Self-tuning</td>
</tr>
</tbody>
</table>

Average Unavailability vs. Available Bandwidth/Write Bandwidth

Periodic (500s)
Cost of Consistency v. Coherence

Suppose I care about subset of data
- /A/* but not /B/*, /C/*, or /D/*

PRACTI
- Precise invalidations for /A/*
- Imprecise invalidations for the rest

Imprecise invalidations: “Placeholders”
- Allow future reads/writes to be consistently ordered with writes to /B/*, /C/*, /D/* if desired
  - Locally or at other nodes
- System that only guarantees coherence and never provides option of consistency could omit imprecise invalidations
- Worst case: Each precise invalidation paired with imprecise invalidation summarizing writes on which it depends
- How much overhead do these imprecise invalidations impose on nodes that don’t use them?
Cost of Consistency

Imprecise invalidations save v. all-precise
Imprecise invalidations cost v. coherence only

- Worst case 2:1 (messages)
- Locality reduces cost
Performance Summary

Better trade-offs
- Partial replication of data
- Partial replication of metadata
- Topology independence
- Minimal consistency cost

Additional benefits (see paper)
- Self-tuning replication of bodies
- Incremental checkpoint transfer
Outline

Motivation
PRACTI Protocol
Evaluation

Future Work/Conclusions
  • Towards a unified theory and practice
Questions PRACTI doesn’t answer

• Does PRACTI reduce development costs by 10x?
  ▪ Can we support 14 OSDI/SOSP papers in <1000 LOC each?
• Can we support quorums, client-server, server replication, p2p on the same substrate?
• Can we efficiently support callbacks and leases?
• How do various consistency paradigms relate?
  ▪ FIFO, causal, sequential, linearizable, etc.
    v. Reads follow writes, monotonic reads, etc.
    v. Safe, regular, atomic, etc.
• What are the “core mechanisms” for security?
• Can we support FS, tuple store, and DB on same substrate?
• Can we unify other “large scale” replication systems (e.g., cluster)?
Conclusion

Build your next large-scale replication system using PRACTI

• A better way to build replication systems
• A way to build better replication systems

Details on my web page

“PRACTI Replication for Large-Scale Systems,” M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula J. Zheng

