

Sinfonia: a new paradigm for building scalable distributed systems

Aguilera et. al.

Presented by Gary Jackson

Introduction

- Goal: enable infrastructure applications
- Principle 1: reduce operation coupling to obtain scalability
- Principle 2: make components reliable before scaling them

Outline

- Architecture
- Application
- Experiments
- Conclusion

Assumptions

- Entire deployment is within a single datacenter
- Latencies are low
- Partitioning doesn't happen
- Node and disk failures happen infrequently

Components

- Client nodes: where the application runs
- Memory nodes: where the store is kept and accessed
- Management node: performs reliability functions
- Not mentioned in the basic components section or the diagram

Minitransactions

- Collect information for transactions together in to one message
 - Comparisons, reads, writes
- Send the message to one or more involved memory nodes
- Any one failure causes the minitransaction to fail

Minitransactions

- Lets all transactions happen with two messages from the client
 - Submit
 - Confirm
- Constrains what we can do, but not enough to be useless

Examples

- Swap
- Compare-and-swap
- Atomic read of many data
- Acquire leases
- Change data if lease is held
- What can't they do?

Fault Tolerance

- Application crashes never affect data since minitransactions are atomic
- When the application crashes, outstanding transactions are cancelled by a recovery coordinator on the management node

Optional Fault Tolerance

- Mask independent failures in memory nodes
- Use transaction logging for recovery
- Node replication for failover
 - Uses lights-out management to prevent split brain when replica assumes control

Optional Fault Tolerance

- Preserve data on correlated failures
- Restore data on disasters

Failure recovery

- Client node crash
 - Outstanding minitransactions are cancelled
- Memory node crash
 - Node plays log
 - Then contacts co-participants for outstanding minitransactions
 - If any participant has not voted, then the vote is no

Failure Recovery

- Widespread Failure
 - Management node detects massive failure
 - Individual node recovery same as before
 - But does not contact other nodes for transactions not in the decided log
 - Uses a heuristic to detect failure

Backups

- Phase 1: Lock all addresses on all nodes
 - This temporarily stops all minitransactions
- Phase 2: Release lock and start backing up
 - Nodes write to memory until backup is complete

SinfoniaFS

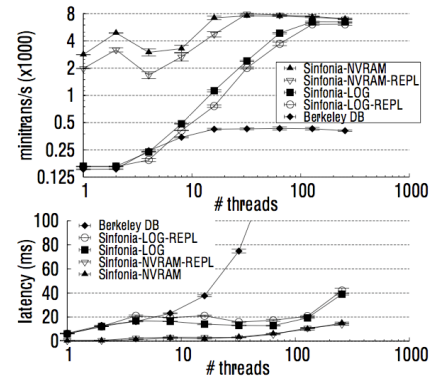
- Laid out like many other filesystems
- Caching at the application level for things like inodes
- Encourages node locality

SinfoniaGCS

- Claim that group communication is hard to implement
- They implemented on top of Sinfonia
- Nifty mechanism for maintaining the cooperative data structure

Vs. existing system

- Sinfonia on a single node vs. Berkeley DB
- Berkeley DB flakes due to page level lock



Different Transaction Schemes

- Unbatched multmessage transactions
- Batched multmessage transactions
- Batched minitranaction:
- Unbatched minitransactions

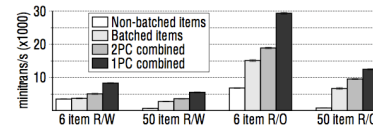


Figure 13: Performance with various combinations of techniques.

Scalability

- Their efficiency measure is effectively parallel speedup

system	system size							
	2	4	8	16	32	64	128	246
Sinfonia-NVRAM	7.5	11	19	37	73	141	255	508
Sinfonia-NVRAM-REPL	7.9	5.9	11	23	44	74	130	231
Sinfonia-LOG	2.4	2.3	5.1	10	21	37	73	159
Sinfonia-LOG-REPL	1.9	1.3	2.1	5	11	21	41	71

Table shows thousands of minitrans/s

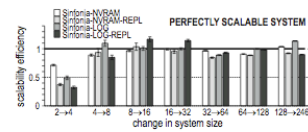
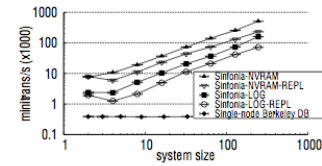
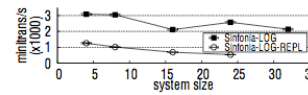


Figure 14: Sinfonia scalability.



Conclusion

- Minitransactions are novel
- Not a fully decentralized system
- Scalability limited by datacenter assumption
- Is the comparison to Spread fair?