SoS: Samba on (a large) Scale: exploring ctdb Alternatives

Ralph Böhme, Samba Team, SerNet

2023-05-10

Outline

net use //thecloud

Distributed Databases: ctdb et al.

Benchmarks

Conclusions

Q&A

Outtakes: Distributed Databases

net use //thecloud

The Goal

```
$ net use \\thecloud
```

- Highly scalable Opensource Cloud SMB with Samba
  - hundreds of nodes
  - hundreds of thousands of clients
- Migrate data to the cloud while keeping applications working
- Elasticity: adding/removing nodes must be cheap
- Availability: multi-datacenter, multi-region
- Build cloud SMB like Azure SMB . . .

The Goal

`$ net use \\thecloud`

- Highly scalable Opensource Cloud SMB with Samba
  - hundreds of nodes
  - hundreds of thousands of clients
- Migrate data to the cloud while keeping applications working
- Elasticity: adding/removing nodes must be cheap
- Availability: multi-datacenter, multi-region
- Build cloud SMB like Azure SMB . . . with Samba

Off-the-shelf Building Blocks

Samba Cloud SMB Building blocks

- Clustered Filesystem
  - CephFS, GPFS, GlusterFS, Lustre, ... ?
- Distributed Database
  - `ctdb`, ... ?
- This time we only look at the database component

Distributed Databases: ctdb et al.

ctdb limitations

- ctdb has consistency and scalability limitations
  - Data is not replicated, SMB3 Persistent Handles can't be implemented
  - Use case is high-performance NAS in a single DC
  - Not suited for cloud SMB at scale
- Real world scalability: production max 16 nodes, 50k clients
- Elasticity: adding or removing a node => hell freezes
- Availability: no multi-region / multi-datacenter support

The idea

- There are many scalable Open Source distributed databases out there
- Can any of those fit the bill?

Zoo of Distributed Databases

CockroachDB, Zookeeper, Google Spanner, Ceph, Cassandra
etcd, Azure Table, Scylla, Riak, FoundationDB

Azure CosmosDB, Apache Hbase, TiKV, Yugabyte, Google Bigtable

Requirements: Consistency

Consistency

- Samba needs a database with strong consistency quarantees
  - for K/V-databases this means linearizability
  - for transactional databases this means strict serializability
  - to implement locks we need transactions or atomic compare-and-set
- This is required for data consistency and to implement locking
  - locking is needed to serialize and isolate access to two resources: filesystem and database

Requirements: Performance

Performance

- Due to its non-replicating design `ctdb` has a very high throughput and low latency
- For many workloads low latency is not first priority:
  - remote office collaboration opening an .docx file: takes 200 ms longer to open? Does it matter?
- Assume SMB workload with mostly non-concurrent file access
  - the resulting DB access pattern is also non-concurrent access to different records
  - depending on the database this might allow good horizontal scalability
- Expect simple PAXOS or RAFT based databases to not scale well
  - the leader is a single threaded bottleneck
- Expect databases which avoid a leader bottleneck to scale better
  - there are three candidates: FoundationDB, TiKV and Apache Cassandra 5 (which is not yet released)

Distributed Databases Candidates

Distributed Databases Tested

Benchmarks

open/close in a loop

```
$ smbtorture //172.18.111.10/test -U slow%x \
  smb2.bench.path-contention-shared \
  --unclist unclist-test.txt \
  --option=torture:timelimit=10 \
  --option=torture:nprocs=[1-500]
```

Samba Cluster

- 3 nodes: VMs with 4 cores, 12 GB RAM each, SSD
- Clustered `locking.tdb`, but node local `smbXsrv_open_global.tdb`

Database: fdb, Cassandra, Scylla, etcd

- 3 nodes: VMs with 8 cores, 64 GB RAM, SSD

Ceph/RADOS

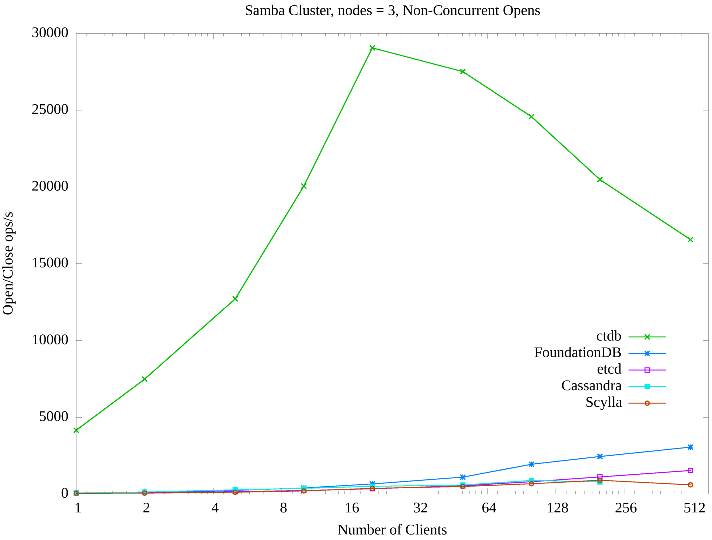
- 3 mons, 3 osds: VMs with 2 cores, 8 GB RAM, SSD

Results

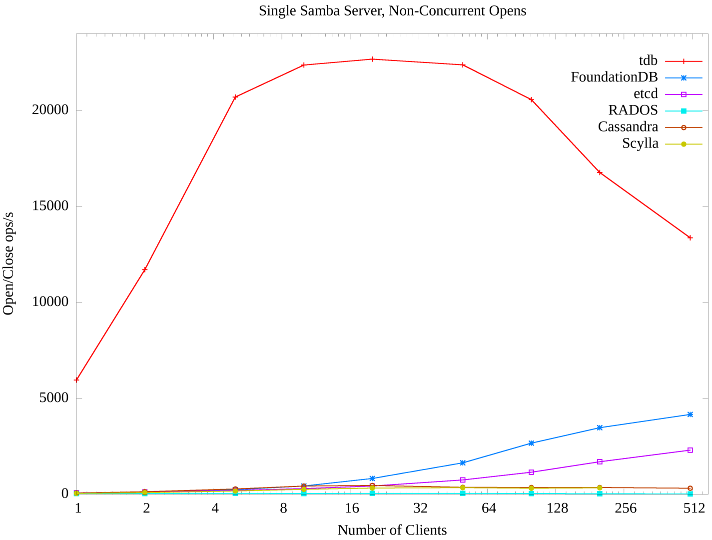
Results

- FoundationDB is the clear winner
  - achieves 10% max throughout compared to `ctdb`
  - has multi-region / multi-datacenter support
- etcd comes next at half the throughput of FoundationDB
- Ceph/RADOS performs surprisingly bad and does not scale at all
- For contended workloads all but FoundationDB run into serious trouble
  - etcd is overloaded and logs `failed to send out heartbeat on time ...`
  - Cassandra and Scylla log LWT errors and cause application failures

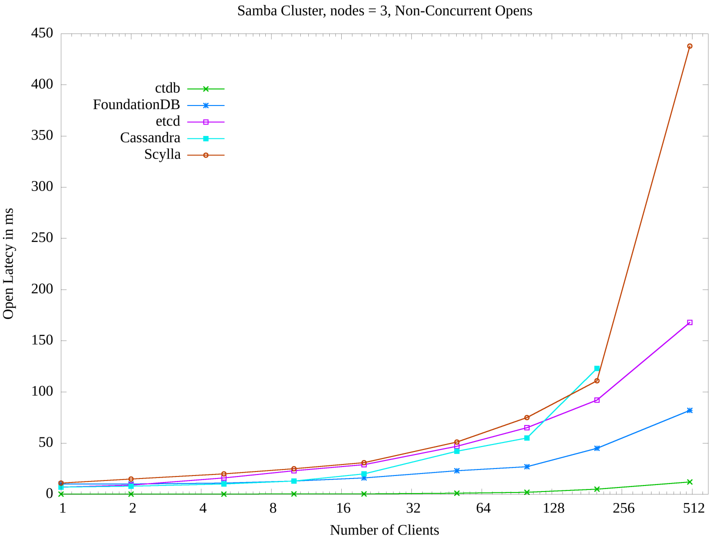
Samba cluster, n=3, non-concurrent opens



Single Samba Server, non-concurrent opens



Samba cluster, 3 nodes, non-concurrent opens, Latency



`dbwrap`

- Samba's pluggable database abstraction `dbwrap`
- Like all of Samba's fileserver code it `dbwrap` is C code
- It's C, so it's verbose, `dbwrap_ctdb.c` is ~2000 lines

`dbwrap_py`

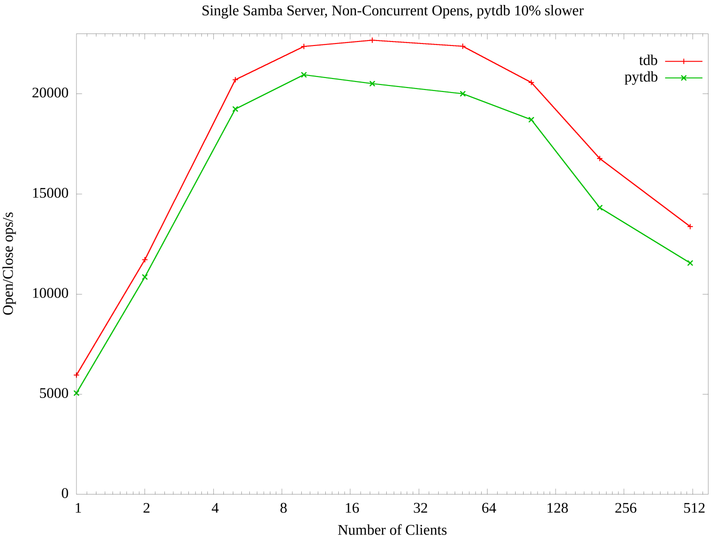
- To simplify new backend development I wrote a new backend in C that uses Python C bindings to call Python scripts that implement the backend
- Roughly 1000 lines of C code (without txn support)
- Being able to use Python for the backend allows rapid prototyping and testing

```
$ wc -l python/samba/samba3/dbwrap_py_*
  338 python/samba/samba3/dbwrap_py_cassandra.py
  414 python/samba/samba3/dbwrap_py_etcd3.py
  303 python/samba/samba3/dbwrap_py_fdb.py
   47 python/samba/samba3/dbwrap_py_tdb.py
```

With a little help from my friend

- Python `etcd` backend written by Jule Anger
- C Ceph/RADOS backend provided by Samuel Cabrero
- Thank you!

Comparing `tdb` and `pytdb`, non-concurrent opens



Conclusions

Conclusions

And the winner is. . .

- FoundationDB for performance and features
- We need more tests on larger clusters

Write our own?

- Writing a scalable distributed database is hard
- Single shard PAXOS and RAFT are simple but do not scale
  - use a consensus group per solves this but:
  - now you need consensus for the shard key ranges
  - changing the ranges when adding or removing nodes becomes a hard problem
  - TiKV does this, so it's doable
    (unfortunately TiKV has neither C nor Python bindings)
- Research for efficient and fast Consensus Protocols is ongoing
- Advanced features like datacenter and region awareness

Outlook

- Highly anticipating the release of Apache Cassandra 5.0
- Cassandra is kind of the Open Source industry standard for BASE databases
- 5.0 ships with strong consistency based on a new consensus protocol ACCORD
- ACCORD is a leaderless consensus protocol allowing better scalability
- ACCORD achieves consensus in one round for non-simultaneous requests

Q&A

Thank you!
Questions?

Ralph Böhme
slow@samba.org
rb@sernet.de

Outtakes: Distributed Databases

Dream of a Distributed Database

The Dream

- Consistent, atomic, isolated
- Efficient, scalable, high throughput, low latency
- Highly available, partition tolerant, failure tolerant

Building Blocks

- Sharding - for scalability and performance
- Replication - for safety and availability

The Challenge

- Ordering of operations in the face of unreliable time sources and network delays
- Reliable and consistent replication

You can't have your cake and eat it

- Strong consistency requires communication
  - Communication takes time
  - Communication requires connectivity
- CAP Theorem: Consistent, Available, Partition Tolerant. Choose two!
- PACELC:
  - Under Network Partition, be Available or Consistent, else
  - Choose between Latency or Consistency
- So what means strongly consistent?
- What would then be weak consistency?
- And what form of consistency does Samba need?

Strong Consistency

So what is strong consistency

- The replicated database behaves like a single copy
  - as if reads and writes are done from/to one place, not many
- All requests are strictly ordered
  - as if done by a single thread
  - ordered according to real time
- The technical term for strong consistency is Linearizability
- This is orthogonal to ACID of SQL databases
  - ACID doesn't deal with replicated databases at all
  - the I in ACID deals with txn isolation when reading and writing multiple objects
  - ACID does NOT require transaction ordering
    - transactions can be executed in any order
    - as long as they are Isolated by some of the configured level
- In Samba tdb is linearizabile, but ctdb is not

And what is weak consistency?

- BASE: Basically Available, Soft State, Eventually Consistent
- Basically Available: prefer availability over consistency
- Soft State: with time, state converges and we only have some probability of knowing the state
- Eventually Consistent: consistent state emerges over time

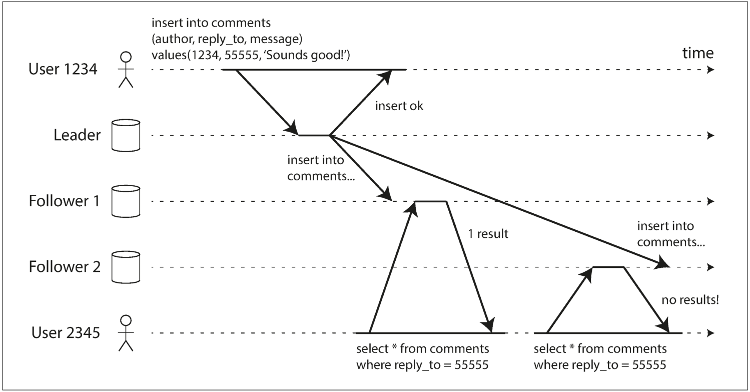


Figure 5-4. A user first reads from a fresh replica, then from a stale replica. Time appears to go backward. To prevent this anomaly, we need monotonic reads.

Figure 1: From: `https://dataintensive.net/`

Weak Consistency, Example 2: Quorum Reads and Writes

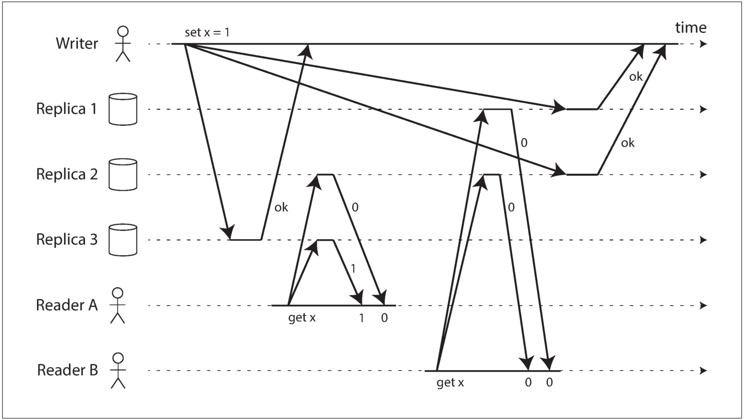


Figure 9-6. A nonlinearizable execution, despite using a strict quorum.

Figure 2: From: `https://dataintensive.net/`

The Consistency Landscape

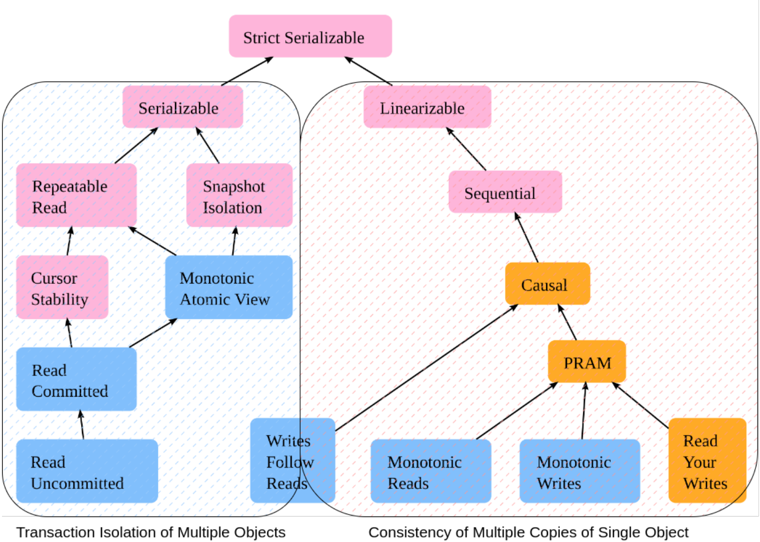


Figure 3: From: `https://jepsen.io/consistency`

Weak Consistency, Implementation and Examples

Examples

- Amazon Dynamo, Apache Cassandra
- Introduced in the late 2000's
- Highly scalable Key-Value Databases (NoSQL) that underpinned webservices like Amazon and Facebook

Implementation

- Clients send read and write requests to one or more nodes at once
- Basically use (configurable) quorum sizes for reads and writes
- Reads can be made linearizable via read repair
- Writes can be made linearizable via previous quorum read
- Atomic compare-and-set can't be implemented as that requires consensus

Peformance

- High throughput, low latency, excellent scalibilty

Examples

- Google Bigtable, Google Spanner, Amazon DynamoDB, Azure CosmosDB, FoundationDB, Fanua, TiKV, Ceph/RADOS

Implementation by Consensus Algorithms

1. Select a leader
2. Leader replicates client operations to followers
3. Rinse and repeat, goto step 1 (dynamic leader) or 2 (strong leader)

The hard part is leader election, typically done via quorum votes and heartbeets for liveliness.

The devil's in the detail and that's where Consensus Algorithms do things differenty:

Consensus Algorithms History

- 1988: Viewstamped Replication by Barbara Liskov and James Cowling
- 1990: Paxos by Leslie Lamport
- 2011: ZAB (Zookeeper Atomic Broadcast) by Flavio P. Junqueira et al.
- 2014: Raft by Diego Ongaro and John Ousterhout

Strong-Leader vs Dynamic-Leader

- Camp strong leader: VR, ZAB, Raft, Multi-Paxos (goto 2)
- Camp dynamic leader: Paxos (goto 1)

Pros and Cons of Leader-based Consensus Algorithms

Advantage of leader-based algorithms

- (Relatively) Simple implementation

Disadvantage of leader-based algorithms

- All operations must be processed by a single thread in the leader
- The leader can become a bottleneck
- WAN deployments further increase latency for clients
  in other regions than the leader

Avoiding the Leader Bottleneck: Consensus Group per Shard

Single shard PAXOS and RAFT are simple but do not scale

- use a consensus group per solves this but:
- now you need consensus for the shard key ranges
- changing the ranges when adding or removing nodes becomes a hard problem
- TiKV and all distributes SQL servers do this

Avoiding the Leader Bottleneck: FoundationDB

Seperate sequencing from replication

1. Agree on a sequencer via an election round using majority quorum (sequencer = Timestamp Oracle)
2. The sequencer assigns a monotically increasing timestamp
3. Client request processing:
   3.1. Request the timestamp from the sequencer
   3.2. Send request to a follower who further coordinates and replicates the request

The sequencer is still a singleton in the cluster but it performs much less work compared to the leader that also does the replication.

Leaderless Consensus Algorithms and other Enhancements

Leaderless, Flexible Quorums

- Fast Paxos (2005): leaderless, 1 RTT for non-simultaneous ops
- Epaxos (2013): another leaderless algorithms
  - explicit dependency tracking, more complex then Fast Paxos without advantages (?)
- Flexible Paxos (2016): flexible quorums for replication and leader election
- Fast Flexible Paxos (2021): combines Fast Paxos and Flexible Paxos
- Accord (2022): leaderless, 1 RTT for non-simultaneous ops
  - based on Fast Flexible Paxos plus Timestamp Reorder Buffer
  - reduces conflicts of simultaneous ops by reodering received messages in a receive buffer based on operation timestamp and node distance

Real world implementations

- Unfortunately no Open Source real world system implementation any of those
- Apache Cassandra 5.0 will ship an implementation of Accord in late 2023

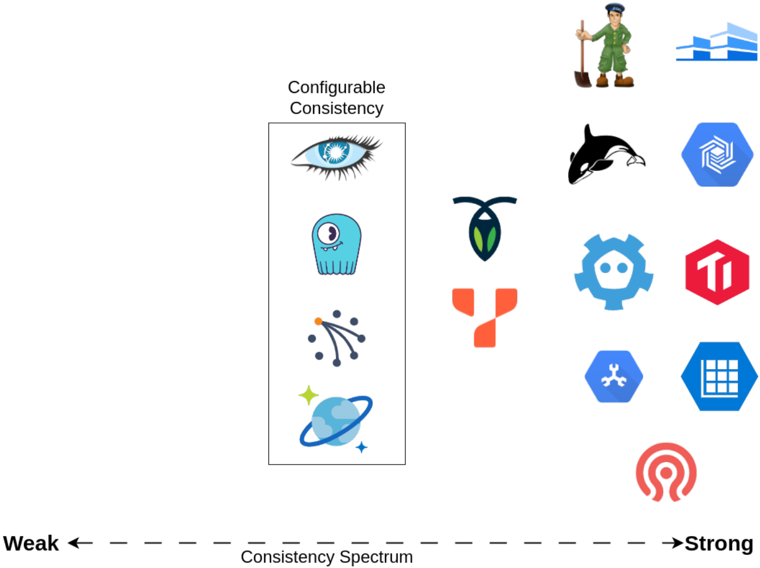
Pros and Cons of Leaderless Consensus Algorithms

Advantage of leaderless algorithms

- Avoid the leader bottleneck

Disadvantage of leaderless algorithms

- Significantly increased implementation complexity


Configurable
Consistency
Weak
Strong
Consistency Spectrum

Zoo of Distributed Databases: SQL vs NoSQL