A Few things happened on the way to LMDB

Presented by Andrew Bartlett
Samba Team - Catalyst // June 2018

catalyst
open source technologists

SAMBA TEAM

Samba is a member project of the
Software Freedom Conservancy

software freedom
conservancy

Andrew Bartlett and Catalyst's Samba Team

- Samba Developer since 2001
- Based in Wellington, New Zealand
- Team lead for the Catalyst Samba Team, including:
  - **Garming Sam**
  - **Douglas Bagnall**
  - **Gary Lockyer**
  - Tim Beale
  - Joe Guo
  - Aarron Haslett
  - Jamie McClymont

Not really a story about LMDB

- LMDB pretty much did what it said on the tin
- Instead LMDB taught us about Samba and LDB
- Numerous locking issues found and fixed
- A new key-value layer added
- And so, so many tests

Customer request: 64-bit DB

- Concerned that the 4GB DB could be filled too quickly
  - Wanting to store > 100,000 users in a single domain!
- Main concern is the hard limit of TDB
- LMDB chosen as a modern key-value store
  - Used in OpenLDAP

Timeline

- LMDB was prototyped in Jan 2017
  - Garming Sam
  - Inspired by, but rewritten from Jakob Hrozek's earlier prototype
- GUID-index LDB implemented in July/August 2017
  - LMDB requires a maximum 511 byte key length
- Primary development Jan → May 2018
  - Gary Lockyer

A new approach: Key/Value layer

- Jakob's approach was to copy ldb_tdb and modify it
- Garming and I decided to instead add a key-value layer
  - Avoid code duplication
  - Allow more than just LMDB (perhaps LMDBx, LevelDB)
  - Share performance and correctness improvements with ldb_tdb
  - Like dbwrap in concept but specific to LDB needs

Key-value API

- int (*__store__)(struct ltdb_private *ltdb, struct ldb_val key, struct ldb_val data, int flags)
- int (*__delete__)(struct ltdb_private *ltdb, struct ldb_val key);
- int (*__iterate__)(struct ltdb_private *ltdb, ldb_kv_traverse_fn fn, void *ctx);
- int (*__update_in_iterate__)(struct ltdb_private *ltdb, struct ldb_val key, struct ldb_val key2, struct ldb_val data, void *ctx);
- int (*__fetch_and_parse__)(struct ltdb_private *ltdb, struct ldb_val key, int (*parser)(struct ldb_val key, struct ldb_val data, void *private_data), void *ctx);

Locking API

- Read Locks
  - int (*__lock_read__)(struct ldb_module *);
  - int (*__unlock_read__)(struct ldb_module *);
- Transactions
  - int (*__begin_write__)(struct ltdb_private *);
  - int (*__prepare_write__)(struct ltdb_private *);
  - int (*__abort_write__)(struct ltdb_private *);
  - int (*__finish_write__)(struct ltdb_private *);

Meta API

- int (*__error__)(struct ltdb_private *ltdb);
- const char * (*__errorstr__)(struct ltdb_private *ltdb);
- const char * (*__name__)(struct ltdb_private *ltdb);
- bool (*__has_changed__)(struct ltdb_private *ltdb);
- bool (*__transaction_active__)(struct ltdb_private *ltdb);

First Hurdle: Locking

- Even the prototype found issues!
  - Demonstrated the lack of whole-DB locking
  - Fixed for Samba 4.7 last year
- Probably behind many of our replication issues

Second Hurdle: More Locking!

- It just wouldn't pass make test!
  - More strange failures in replication
- Unlock ordering issues in replication
  - highestCommitedUSN visible before the data
  - Fixes proposed for backport to Samba 4.7 and 4.8
- Modification without locks (at startup) in Samba 4.8
  - DB-init time only, but not good
  - Added checks to key-value layer to prevent re-occurrence

Third hurdle: Maximum key size

- TDB has an unlimited key size
- LMDB is limited to 511 bytes
- LDB traditionally used the DN as the key
  - Addressed by the new GUID key system

But what about indexes?

- Index created by putting index key and value in the TDB key
  - @INDEX:SAMACCOUNTNAME:abartlet
- Original plan was to keep the index in TDB
  - But the more we understood the locking the less we wanted to mix TDB and LMDB lock ordering
- Addressed by truncating the index and coping with multiple matches
  - Ironically found and fixed the 4.8 upgrade bug
  - But didn't realise the importance before everyone noticed

And what about performance?

- Three performance tools measured so far:
  - Make perftest on our Hardware test server
    - Old AMD Athalon!
  - Traffic replay tool in the cloud
  - Adding users and users into groups of my workstation

Make perftest: a small dissapointment

- 30% performance loss!
  - LMDB uses write(), and a read-only mmap()
  - socket_wrapper intercepts write()
- Workaround:
  - Use Linux userspace namespaces instead of socket_wrapper
  - Patches to upstream this still pending
- End result is no major change, perhaps 10% slower

Traffic replay

- This is a tool to replay an amplified anonymous traffic capture
- Similar numbers to TDB
- Need to re-try with a larger DB
  - We think LMDB will show most strength at large sizes

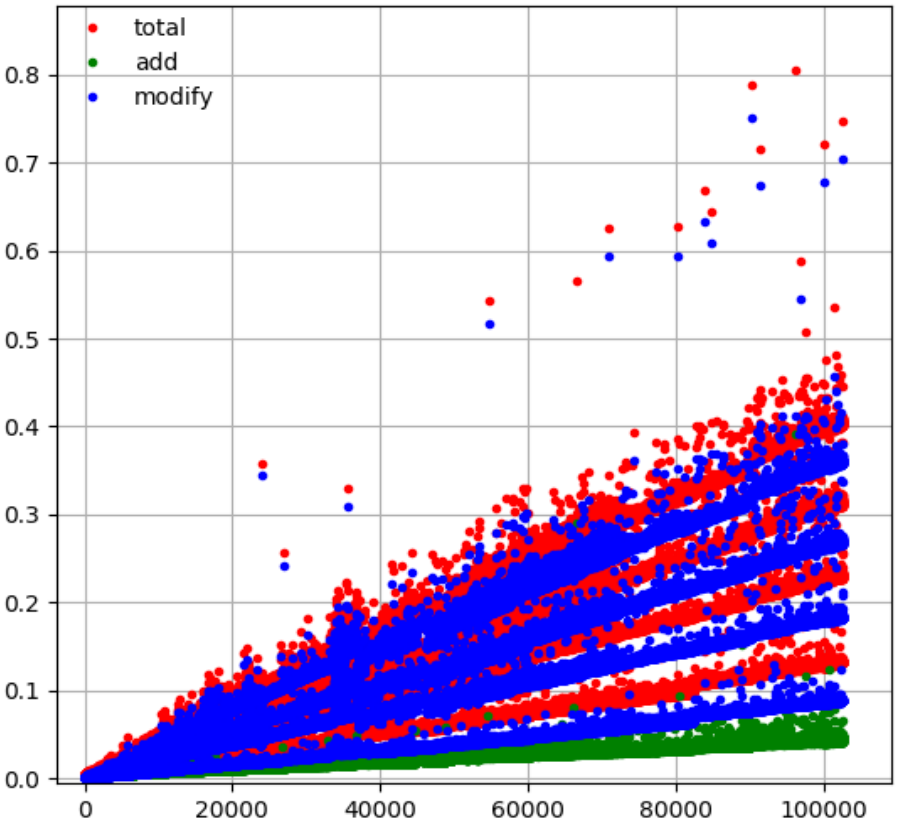
Adding users and users into groups of my workstation

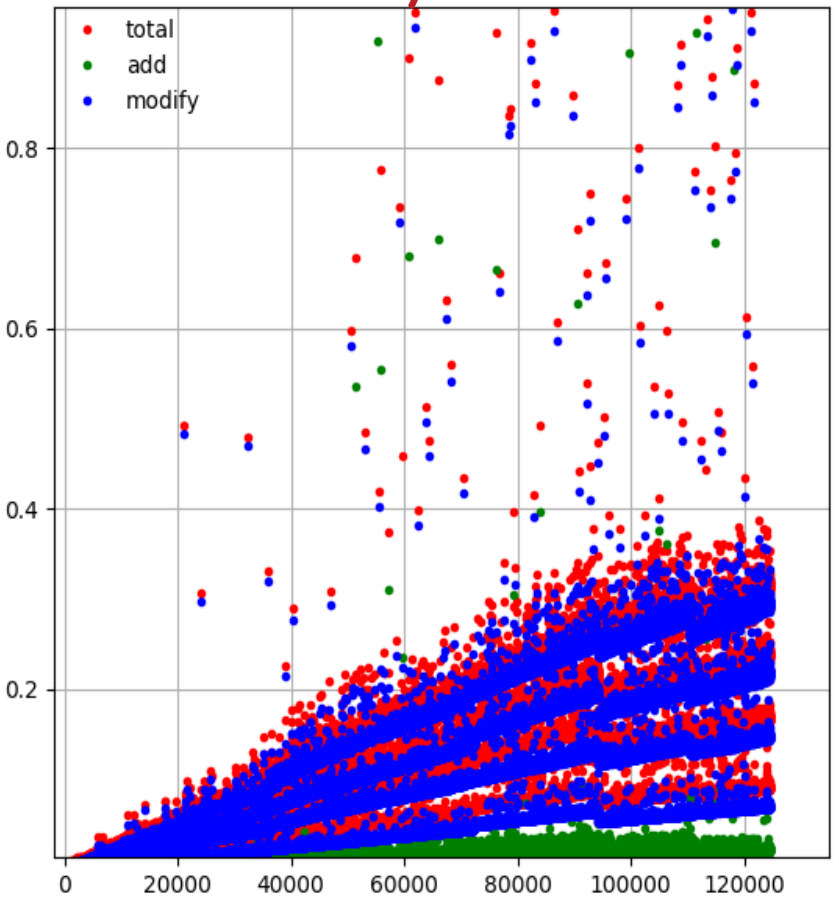
- In a four-hour benchmark adding users and adding to one to four groups (in rotation):
  - Samba 4.4: 26,000 users
  - Samba 4.5: 48,000 users
  - Samba 4.6: 55,000 users
  - Samba 4.7: 85,000 users
  - Samba 4.8: 100,000 users
  - Samba 4.9: 100,000 users (TDB)
  - Samba 4.9: 45,000 users (LMDB)

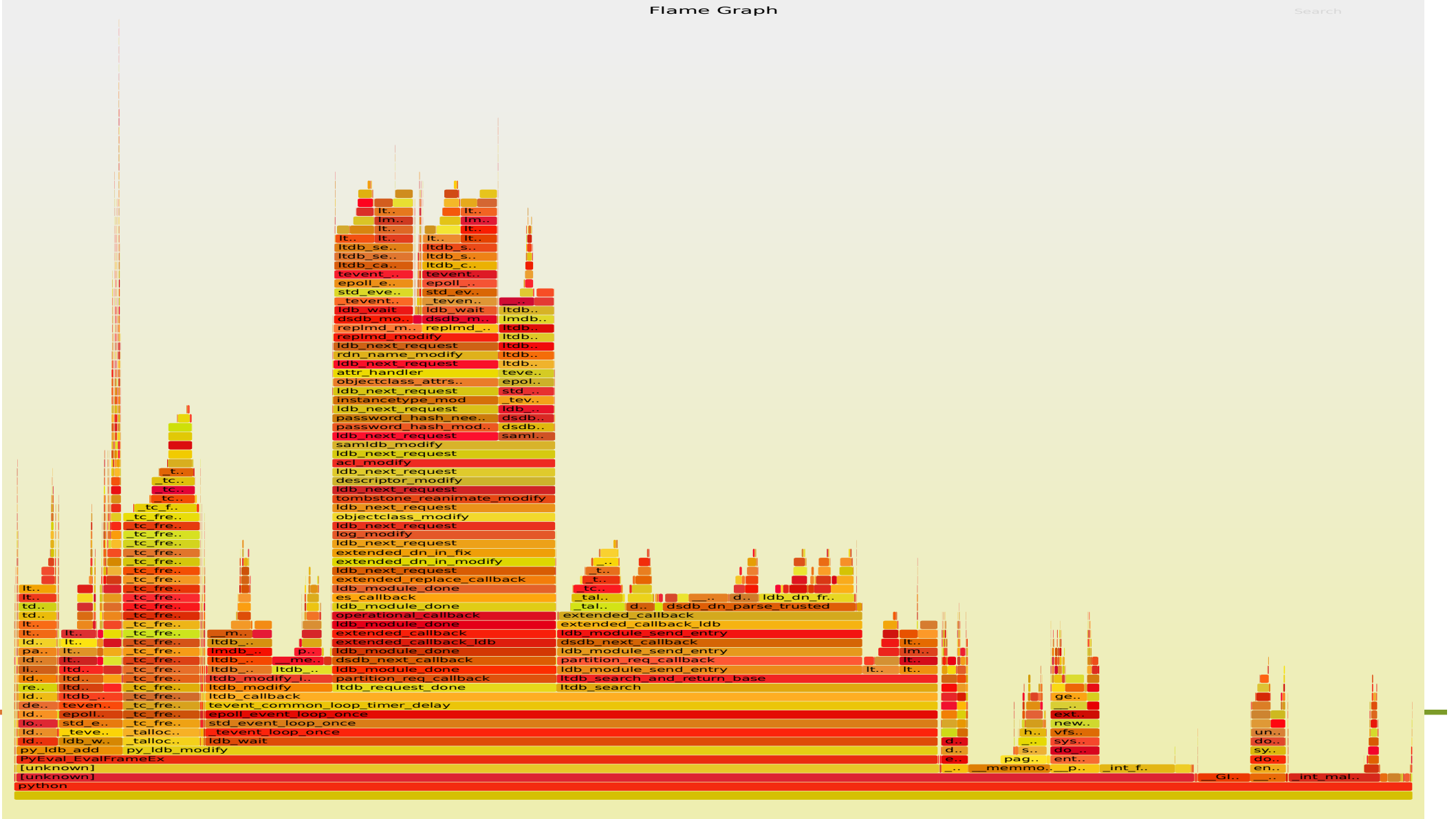
Ouch. What went wrong!

- fsync()/fdatasync()/msync() still called
- Patches quickly written
- New numbers:
  - Samba 4.9: 100,000 users (TDB)
  - Samba 4.9: 124,000 users (LMDB, no fsync())
- Lesson:
  - Samba's module stack is still the slowest factor

TDB vs LMDB (latency vs number of users added)






Flame Graph
Search
python
[unknown]
PyEval_EvalFrameEx
py_ldb_add
py_ldb_modify
ldb_wait
tevent_loop_once
epoll_event_loop_once
tevent_common_loop_timer_delay
ltdb_callback
ltdb_modify
ltdb_request_done
ltdb_search
ldb_module_send_entry
extended_callback
operational_callback
partition_req_callback
ldb_module_done
extended_callback_ldb
dsdb_next_callback
ldb_search_and_return_base
dsdb_dn_parse_trusted
ldb_dn_fr..
__memmo..
__p..
_int_f..
_GI..
_int_mal..

OK, so not so bad

- We addressed the customer's desire for scale
  - Currently limited to 6GB but that is compile-time constant only
- Opens up new opportunities
- Current Status:
  - Still accessed behind sam.ldb (a TDB)
  - Still one-subtree per DB file

LMDB: The future

- Use the LMDB b-tree properties in for the index
  - Allow prefix matching
  - Allow <= etc
- Use sub-databases:
  - Perhaps one per index?
  - Perhaps one per sub-tree?
- Use nested transactions to make the index safer

LMDB: Sharp Edges

- Different locking behaviour (no exclusive access)
- Files are sparse by default
  - DB operations can fill the file and partition without going via a specific resize
- Files are not extended automatically
  - The inverse to the above, when a file is full unlike TDB there is no auto-resize
  - Requires that the admin or Samba know the size up-front
    - LDB / Samba has not required this kind of planning in the past
- Need real-world experience

Beyond LMDB: Supporting more connections on each DC

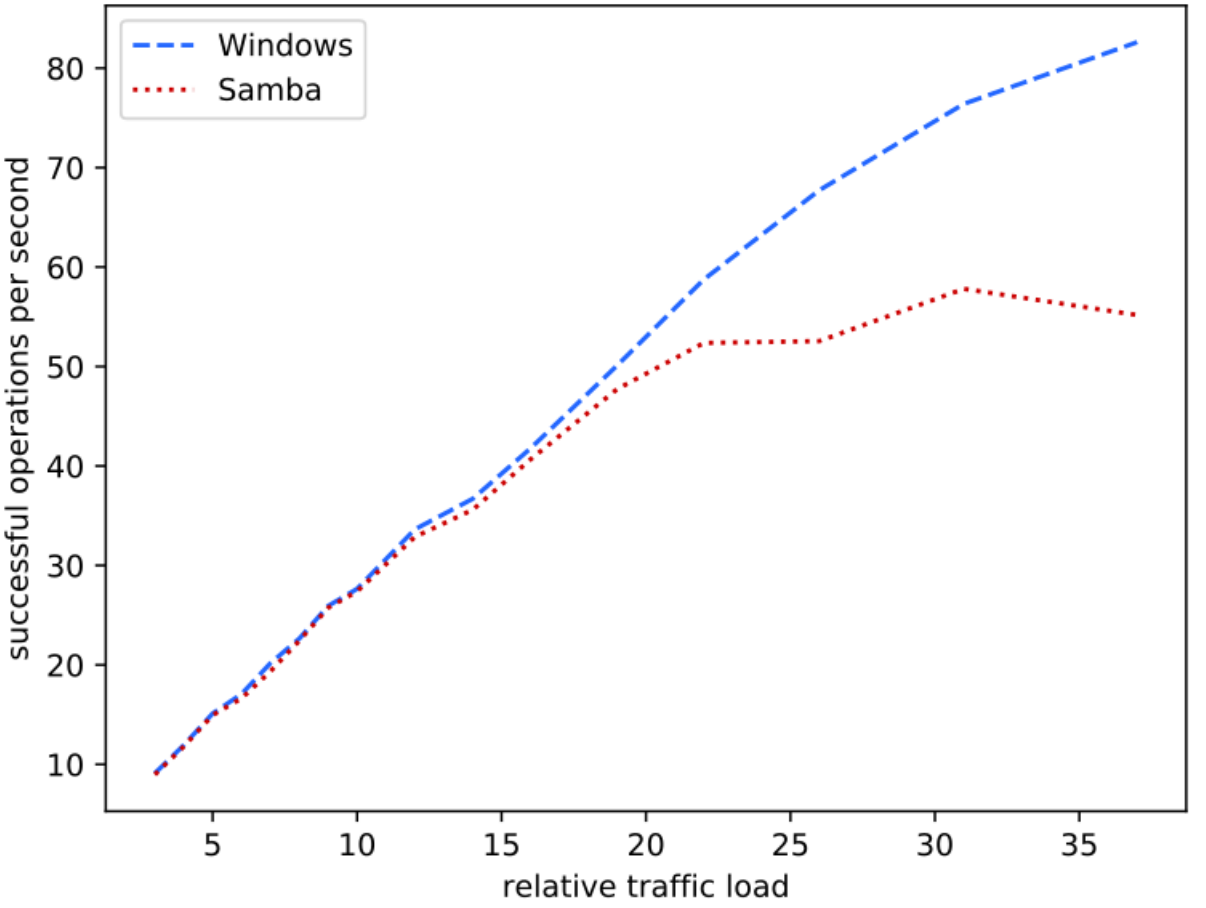
- Samba 4.6 removes single-process restrictions on NETLOGON
  - Really important for 802.1x backed authentication
- Samba 4.7 supports a multi-process LDAP server
  - Actually reduces number of connections you can fit in memory (oops)
- Samba 4.8 adds a prefork mode for LDAP
  - Great for a big AD DC with many, many clients
- Samba 4.9: should we make prefork the default?
  - However NETLOGON would be single-process in that mode (ouch)

New traffic_replay Performance tool

- We can now record and re-play traffic
  - Recreate a real-world load
  - Amplify the traffic
- Comparative testing now possible between Windows and Samba
- Samba is now within about 50% of Windows performance
  - Against a single-CPU target system
  - Allows us to slow both down enough for the traffic_replay to saturate it

Performance against Samba and Windows

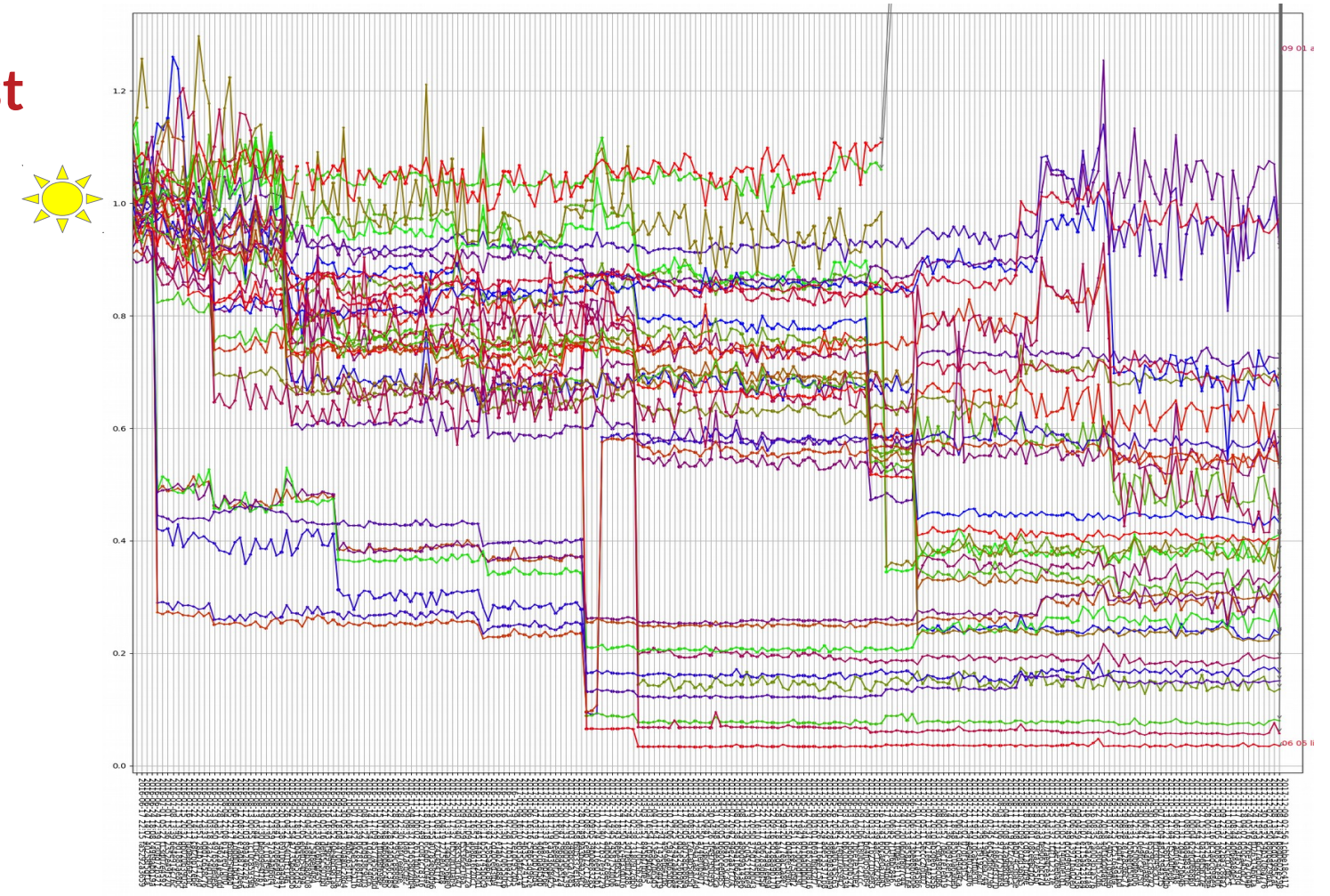
- 1 vCPU
- Catalyst Cloud
- After 35x speed the tool exhausts itself
- So this is not the upper bound



Wanted: More network captures

- We need more sample of network traffic
  - Anonymised with the traffic_summary.pl script
- Ideally with permission to publish (eg the Samba wiki)
- Diverse real world loads will avoid skewed perf testing

make perftest graphs - April 2016 to Dec 2017



Catalyst development beyond performance

- Encrypted secrets (4.8)
  - Use a local file key to encrypt DB secrets
  - (could then be network-deployed)
- Unix-compatible passwords (4.7)
  - Store and retrieve sha256/sha512 crypt() passwords to sync with other systems
- Audit Logging (4.7 and 4.9)
  - Output audit logs into JSON
- RODC support (4.7)
  - This was experimental until now
- DNS Zone scavenging (for 4.9)
- Demote cleans up own DNS records (4.9)
- Fine Grained Password Policy (4.9)
- Domain Backup (for 4.9)
- Domain Rename (for 4.9)

Catalyst's Open Source Technologies – Questions?

Want to work with my team at Catalyst to make your Samba scale? - talk to me in the hallway track!