Improving overall Robinhood performance for use on large-scale deployments

Colin Faber <colin.faber@seagate.com>

SEAGATE

WHAT IS ROBINHOOD?

Robinhood is a versatile policy engine and tool kit used manage large-scale file systems

- Maintains on going replication of target FS metadata
- Utilizes external relational database for metadata management
- Utilizes Lustre Changelog feature
- Utilizes Lustre HSM management features
- Versatile and can be used in non-HSM, and non-Lustre environments
- Provides tool clones which utilize relational database rather than FS directly

Robinhood was originally developed by *CEA the French Alternative Energies and Atomic Energy Commission*, and is actively utilized by many HPC sites, as well as by many HPC vendors for use in their products. *Seagate, Cray, DDN, DKRZ, Kaust, NCSA, Oracle*, as well as many others.

HOW SEAGATE USES ROBINHOOD

- Utilized directly in various Lustre offerings
  - ClusterStor A200 tiered active archive object store
  - ClusterStor L300N / 9000 / 1500

- Currently supports large scale file systems which utilize Robinhood for various tasks
  - 100’s of millions of files and directories
  - Huge amounts of file system activity
  - Stale file detection and removal
  - HSM workloads

Seagate is committed to supporting and improving Robinhood as the current, best Lustre changelog consumer and policy engine available.

CHALLENGES WITH ROBINHOOD

- Relies on Lustre changelogs which can be difficult to manage, requiring occasional rescans of the target file system
  - Seagate has invested considerable development resources to harden the Changelog feature

- Current relational database model leaves room for optimization
  - Backing database choice for Robinhood (MySQL with InnoDB engine) has scaling issues
  - Relational database transactions are inherently inefficient
  - Environments with high file counts and high change rates result in processing backlogs
  - Alternative database engines (Percona TokuDB / PostgreSQL, and various others) may be a better fit for large-scale deployments

MITIGATING CHALLENGES WITH ROBINHOOD

To mitigate challenges encountered in our Robinhood deployments, our development work has focused on hardening and improving Robinhood

- Code improvements for manageability
  - Break up large routines and logic blocks
  - Introduce a development package to build new features against

- Database tuning and experiments
- Lustre client tuning and experiments
- Developed several value-add components through a plug-in architecture we’re developing
- Determined an appropriate sizing guide and techniques for policy engine systems

SIZING A POLICY ENGINE SYSTEM APPROPRIATELY

- A Robinhood-based policy engine system is mostly a large relational database
- Each inode on your file system equals 4 or more records in the database
- Running reports, triggering policies, and utilizing external tools which ship with Robinhood all generate complex queries to which the database has to respond
- A low-powered, low-core count, low CPU frequency machine will result in a low performance Robinhood policy engine
- Large-scale file systems need large-scale database systems for Robinhood to operate against

DATABASE TUNING

Database tuning is critical. InnoDB tuning using Percona’s automatic tuning tool works great!

https://tools.percona.com/wizard

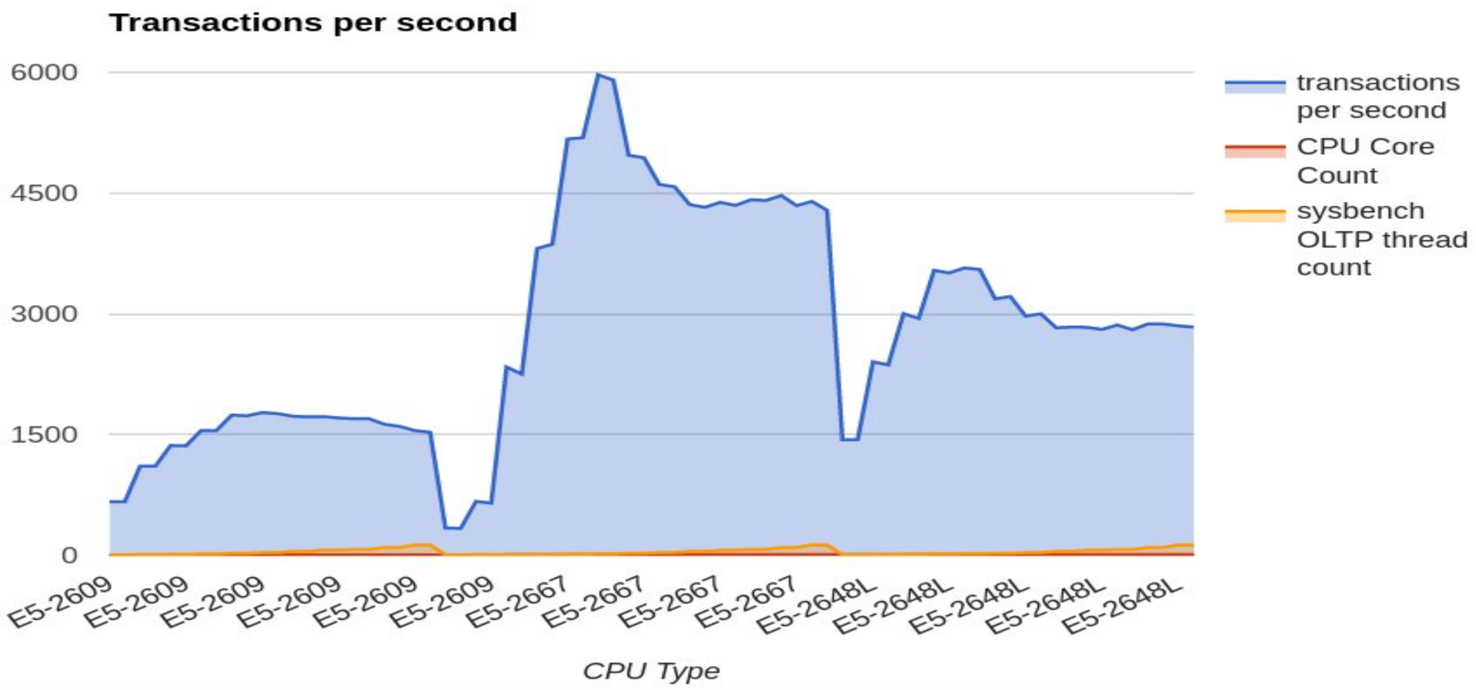
After many experiments we found that our custom tuning was only moderately better than the Percona tuning wizard.

Sysbench version 0.5 with complex OLTP workload is a great way to exercise your database before getting Robinhood involved

- Repeatable / industry standard database benchmark suite
- Various tuning strategies can be employed and tested
- Tuning itself can be scripted, allowing for hands off performance discovery

DATABASE SYSTEM SIZING (CPU)

Surprise surprise! A faster CPU makes for a faster database server



DATABASE SYSTEM SIZING (continued)

- Storage matters, but not as much as CPU and memory
- CPU core count and clock speed are the biggest factors in performance
- More memory means more relational data in cache
- Storage should be appropriate for the file system size
  - 1+ KB per inode + InnoDB log file size == database size on disk
  - The higher the IOP rate, the better
    - Many small random block reads and writes

  - Solid State / NVMe block device preferred
  - For us, EXT4 is the file system of choice
    - Various things can be tuned, with slight performance gains

DATABASE SYSTEM SIZING (continued)

A software RAID solution can work well for database use

- Needs extra CPU cycles for block processing
- Larger chunk size help
- GPT aligned partitions help
- Fast Seagate drives help :)

ROBINHOOD TUNING

Simple tuning makes for vast performance improvements

- Disabling accounting helps significantly
- Robinhood database threads should be based on sysbench benchmarking findings
- Lustre client should be tuned for Robinhood
  - Disable various levels of caching
  - Increase RPCs in flight
  - Tune statahead
  - Flush inode cache regularly

SEAGATE ROBINHOOD CONTRIBUTIONS

- Code refactor
  - Seagate has completed large amounts of code refactor work changing the internal framework to allow for easier changes in the core application
- Plug-in architecture
  - New framework provides development packages and allows for plug-in style development within Robinhood record flow systems
- Performance optimizations
  - New plug-ins developed by Seagate contain enhancements to record flow management, improving overall operational performance
- Plug-in framework allows for further incremental changes improving performance

CHANGELOG UPLOAD ENHANCEMENT (CUE)

- CUE sorts, categorizes and reduces total changelog record processing requirements without losing information
  - Reducing changelog record flow reduces DB and FS load
  - Sorting and categorizing allows for additional enhancements
    - Various opportunistic bulk queries (WHERE IN())
    - Allows for alternative database strategies (NoSQL sharding, etc)

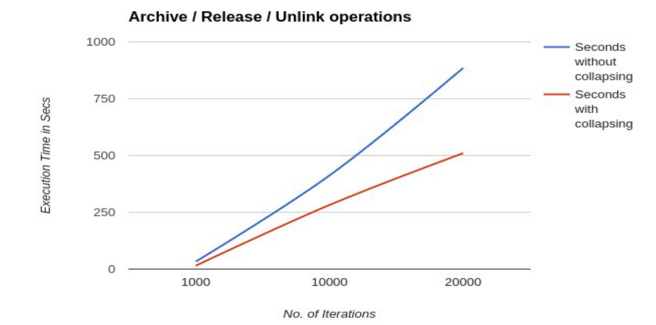
- Improves overall Robinhood performance and reduces hardware requirements

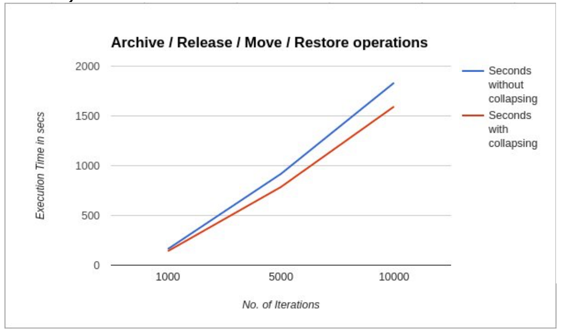
- Provides a path forward to scale further

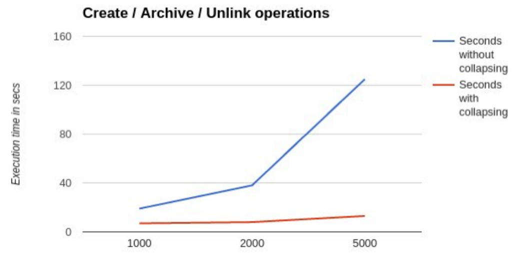
CUE RESULTS SO FAR

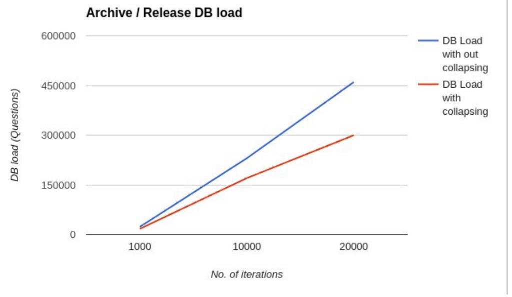
Average performance improvements of a ***400+ %*** reduction in record processing time and a ***40 %*** reduction in database query load for generalized workloads

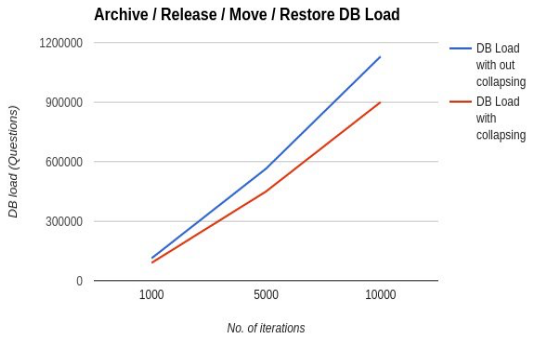
(with feature, without feature lower is better)

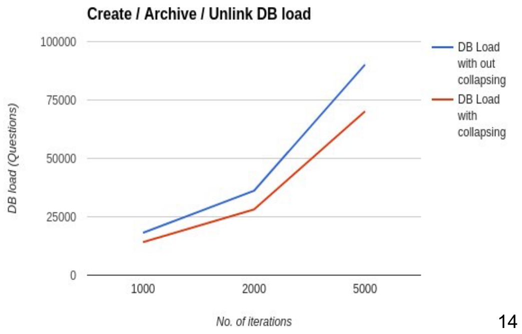












SQL OPTIMIZATION AND SIMPLIFICATION

- Current Robinhood SQL implementation
  - Optimized for single transaction operations
  - Uses expensive / complex operations
    - LEFT JOIN type operations (requires rescan of table space)
    - Disallows SHARDing type horizontal scaling strategies
  - Currently only works MySQL / SQLite RDBMS
- Simplification targets:
  - Optimize for single transaction bulk operations
  - Reduce SQL processing time in common workloads (DELETE / UPDATE)
  - Provide pathway toward alternative relational database engines
  - Allow for per-operation / per-function based testing and profiling

NEXT STEPS

- Test utilizing multiple concurrent client mounts to the same file system, on the same client to attempt to avoid Lustre bottlenecks
  (Single MDC Semaphore when performing FID lookups fixed in recent lustre versions)

- Consolidate SQL query sets into batch transactions where possible
- Further refinements to the CUE plugin
- Investigation and porting work into alternative database systems which are designed for the types of workloads commonly seen within Robinhood deployments
  - Percona's TokuDB engine
  - PostgreSQL
  - Various NoSQL implementations

Questions?

Thank you!

LUSTRE CLIENT TUNING CHEAT SHEET

Sample tuning utilized for scale testing

Overall system setup:

vm.vfs_cache_pressure to **150**

Lustre setup:

llite.*.xattr_cache to **0**
ldlm.namespaces.*osc*.lru_max_age to **1200**
ldlm.namespaces.*osc*.lru_size to **100**
osc/*/max_rpcs_in_flight and mdc/*/max_rpcs_in_flight to **256**
llite.*.statahead_max to **4**

Crontab:

@daily echo 2 > /proc/sys/vm/drop_caches