SIG Proceedings Paper in LaTeX Format

Ben Trovato[*]
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
trovato@corporation.com

G.K.M. Tobin[†]
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

ABSTRACT

TODO

ACM Reference format:
Ben Trovato and G.K.M. Tobin. 1997. SIG Proceedings Paper in LaTeX Format. In *Proceedings of ACM Woodstock conference, El Paso, Texas USA, July 1997 (WOODSTOCK'97),* 4 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Key-Value stores scale because they support (1) fine scale annotation and (2) flexible, extensible formats. Science applications are structured, entropy increases over time (e.g., Figure 1 shows key distribution and key popularity changing over time). Our hypothesis is that re-distributing keys requires dynamic load balancing policies, similiar to distributed file systems. A one-size-fits all policy is not sufficient. Our contributions are:

(1) an analysis of the ParSplice keyspace
(2) using a modern distributed key-value store
(3) positive effects of Mantle

2 BACKGROUND

2.1 ParSplice

ParSplice [6] is a molecular dynamics simulation developed at LANL. It has 4 phases, as depicted in Figure 2:

(1) splicer (S) tells producers (P) to compute segments for state s_i
(2) P's pull initial coordinates $\{x_i, y_i\}$ from database
(3) a P inserts completed coordinates for segment s_i into database and S broadcasts next segment(s) s_j
(4) P's pull new segment coordinates $\{x_j, y_j\}$

The database is single node (LevelDB or BerkeleyDB) with caches in front. Our goal is to replace this architecture with a distributed key-value store to solve the immediate sync problems that the ParSplice team is facting. Sliding in something like MDHIM [2] has the added benefit of enabling load balancing. ParSplice has

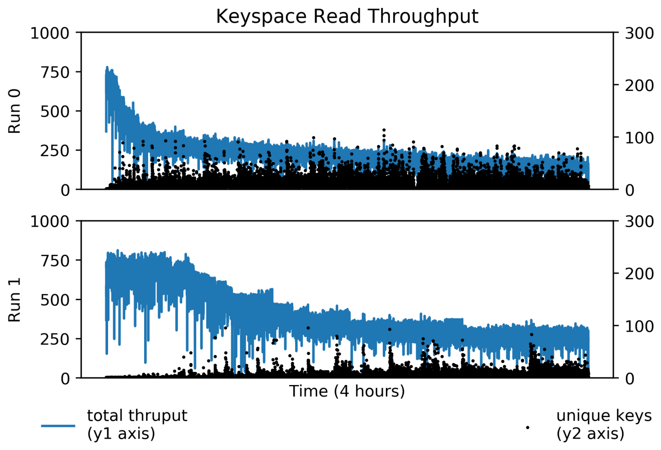

Figure 1: Caching the 100 most recently accessed keys is sufficient for run 1 but more active keyspaces like run 0 need dynamic load balancing policies.

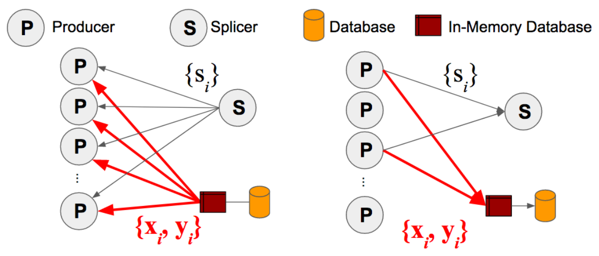

Figure 2: ParSplice is a ready-heavy HPC application where producers use a database for consistency. Replacing the single-node database with HXHIM improves performance with load balancing.

distinct workload phases and a well-known keyspace (Figure 4) so its performance should improve with better load balancing.

2.2 MDHIM: Key-Value Store for HPC

MDHIM is a key-value store designed for HPC architectures and multi-dimensional data. It is based off MDHIM [2], the multi-dimensional indexing middleware. Figure 3 has a crude sketch of the MDHIM architecture. Each MPI rank has an instance of the application, which has the client library linked in. An MPI rank can also have a "range server", which stores the key-value pairs in a local databse (either

[*]Dr. Trovato insisted his name be first.
[†]The secretary disavows any knowledge of this author's actions.

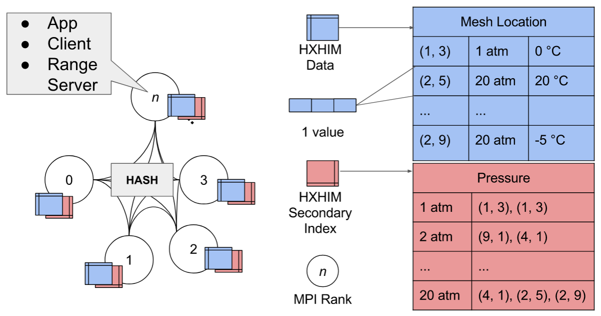

Figure 3: The MDHIM architecture.

LevelDB or MySQL). Data is located with a consistent hash, which is configurable.

The primary and secondary indices shown on the right side of Figure 3 are views of the data that the range server manages. The primary index is the same hash used by the global partitioner. The secondary index or indices are user-defined tables organized in a different way from the primary index. The goal of the secondary indices is to speed up queries that need to aggregate dat (*e.g.* find the maximum values). In the example, the range server and the key in the primary index is located with a hash of the mesh location. The secondary index is organized by pressure, so queries asking for a certain atmosphere can be serviced in O1, consisting of one lookup in the pressure index and one lookup into the primary index.

MDHIM tailors its mechanisms and policies to HPC, showing improved performance over cloud-based key-value stores like Cassandra. It has cursor types for walking the key-value store, bulk operations for exploiting data locality, per-job server spawning, and pluggable backends for its local database and network type (infiniband/RDMA). Its policies are flexible, supporting customized partitioning strategies and user-defined secondary indices. This allows the system to choose whether to send load to the client or server.

2.3 Comparing Mantle and MDHIM

The "Both" column of Table 1 shows how Mantle and MDHIM have similar designs. The workloads are very similar as the the services respond to small and frequent requests, which results in hot spots and flash crowds. As a result, popularity of the data, not the size, drives distribution in both systems. Both workloads also have data locality so the systems have mechanisms for leveraging requests with similar semantic meaning. Finally, the overall design of both systems is decentralized meaning that there is no centralized scheduler and each server has an inconsistent global view.

Despite the similarities, integrating the Mantle API with MDHIM has both design and technical challenges. Mantle is reactive to the workload as opposed to MDHIM migrations, which are triggered based on the request type. As a result, Mantle has functionality for exchanging server utilization (CPU, network, memory) and workload (tracks request types). MDHIM

Results should show, In order from most likely to least likely:

(1) HPC key-value store workloads are structured (because they are mostly workflows and simulations) that their job phases can be learned and exploited using dynamic load balancing policies.
(2) HPC key-value store workloads are so structured that one policy-fits-all
(3) HPC key-value store workloads are not structured enough to be learned
(4) HPC key-value store workload hotspots/flash crowds are too fast to be exploited

3 METHODOLOGY

IT HAS 8K keys!

3.1 Contribution 1: ParSplice Keyspace Analysis

Part 2: As Parsplice simulates, entropy increases resulting in keyspace inbalance (Figure **??** shows how imbalance controlled by "growth rate")

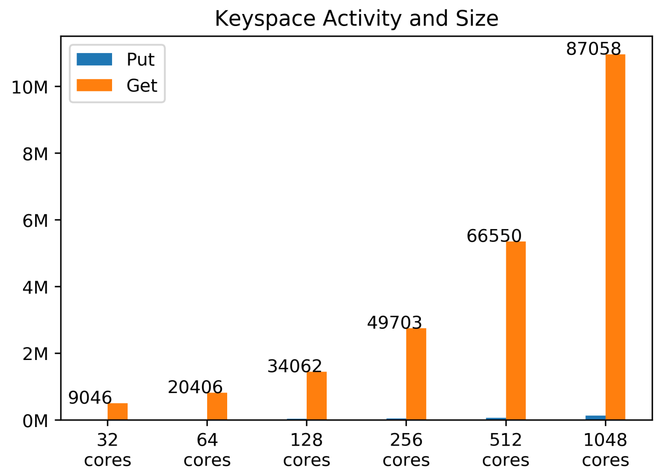

Figure 4: The ParSplice keyspace is small (numbers above bars) but must satisfy many reads as workers calculate new segments. The active keyspace is difficult to predict a priori but the optimal load balancing strategy strikes a good balance between preformance and utiization.

3.2 Contribution 2: Positive Effects of Load Balncing

3.3 Contribution 3: The Need for Dynamic Load Balancing Policies

Why doesn't 1 policy fit all?

- keyspace gets less active
- beginning: levelDB can't handle it
- end: levelDB can handle it

(1) fix DB size to 4GB
(2) trajectory default (1M and 100K)
(3) trajectory with naiive balancer

		Both	Mantle/CephFS	MDHIM
workload	characteristics	small/frequent requests	data access	data management
	write-intensive	partition across cluster	fragment directories*	NOT IMPLEMENTED
	read-intensive	replicate across cluster	copy directories*	NOT IMPLEMENTED
system mechanisms	measure workload	yes	directory temperature	range server counts
	measure utilization	yes	CPU, network, memory	range server buffer size
	migrate resources	almost	`export_dir()`	`mdhimB{Get, Put}()`
	partition resources	yes	subtrees & dirfrags	secondary index, cursor type, bulk operations
migration decisions	interval	configurable	every 10 seconds	every query
	global state	decentralized decisions	heartbeats for metrics	NOT IMPLEMENTED

*Mechanisms implemented in CephFS, not integrated into Mantle

Table 1: Comparing the design goals and implementatons of Mantle and MDHIM.

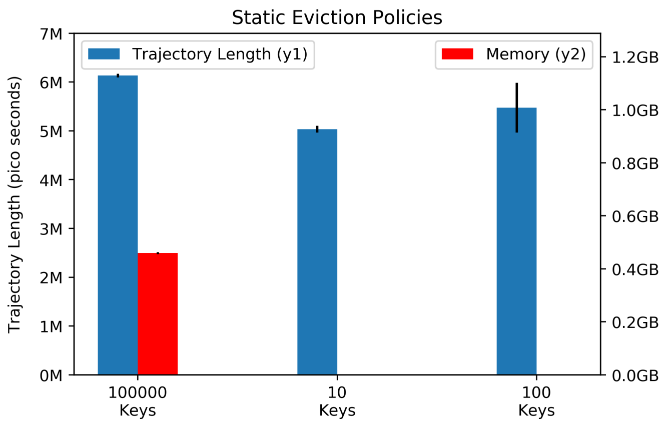

Figure 5: The optimal cache size must strike a balance between performance and resoure utilization. Here we show the trade-off for a static load balancing policy that evicts keys when the cache reaches a certain size. For this configuration, a 100K key cache has the best performance/utilziation, despite the keyspace being 250K keys.

- Mantle (approach/API) to explore dynamic load balancing policies
  - quantifies effect of load balancing
  - formalized effective FS balancers
  - debugging tool
- HXHIM is a good fit because it has migration mechanisms for load balancing
  - bulk operations (`put/get()`)
  - key partitioners
  - secondary indices
- we need a way to learn these regimes
  - Figure **??** shows the same phases as 100K but that the timestamps affected by delay

Mantle is an API that lets admnistrators control file system metadata load balancing. Mantle speeds up file systems by making metadata access faster, leveraging the fact that file system metadata IO imposes small and frequent accesses on the underlying storage system. Since data IO does not scale like metadata IO [8], finding optimal ways to measure, migrate, and partition metadata load is a relatively new field, but has been shown to lead to large performance increases and more scalable file systems [1, 3, 5, 7, 12]. Mantle can use strategies from these papers to control how to distribute or concentrate file system metadata.

It was built on CephFS, the file system above Ceph, so it inherits many of characteristics of the CephFS architecture, like the dedicated metadata cluster and heartbeat mechanisms shown at the top of Figure 6. Each metadata server manages differently sized subtrees of the logical namespace and migration decisions are made synchronously, every 10 seconds. CephFS already had the mechanisms for load balancing, namely the ability to measure the load on a subtree, to migrate subtrees, and to partition subtrees into smaller subtrees, but it had hard-coded, ad-hoc policies for guiding the migrations. Mantle reads user-defined policies written in Lua and returns decisions for how load should be migrated given the state of the cluster and the behavior of the workload. The hooks in Figure 6 show where CephFS calls out to the Mantle library to make decisions. While the decisions were made by Mantle, CephFS used its internal mechanisms to do the load balancing.

The Mantle paper implemented three balancers and tested them under metadata-intensive workloads. The Greedy Spill balancer, which was based on [5], sheds have its load aggressively when there are avaiable servers. Part of the Lua code for implementing this balancer is shown in 7. The Fill and Spill balancer, which was based on [4] sheds a fraction of the load only when the server is overloaded. Finally, the Adaptable balancer, which was based on [10, 11], sheds a fraction of the load frequently.

It was merged[1] and is starting to get users who are frustrated with the hard-coded load balancing policies that are shipped with CephFS. It was re-implemented using the "programmable storage" approach [9] to reduce lines of code for doing things like versioning and distributing balancer version. Although Mantle is heavily integrated the daemons that compose an Ceph cluster, using Ceph's naming conventions and internal libraries like Ceph's version of protocol buffers, there is no reason that it cannot be extracted.

[1]https://github.com/ceph/ceph/pull/5155

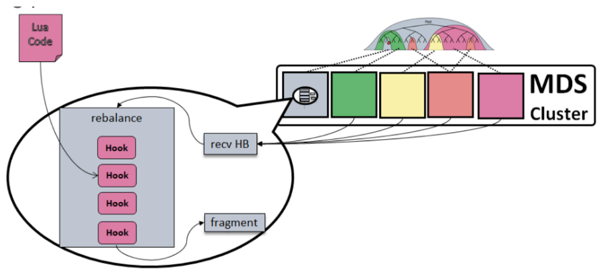

Figure 6: The Mantle API lets adminstrators control load balancing by changing the poicies for how to distribution or concentrate file system metadata. It was merged into CephFS and inherits many aspects of that architecture. Although it has the load balancing structure and logic from CephFS (gray boxes), the actual API is not dependent on that code base.

```
-- Balancer when policy
if MDSs[whoami]["load"]>.01 and
   MDSs[whoami+1]["load"]<.01 then
-- Balancer where policy
targets[whoami+1]=allmetaload/2
```

Figure 7: The Greedy Spill balancer written in Lua using the Mantle API.

3.4 Contribution 4: Machine Learning Keyspace Activity

We can't find the optimal keypsace size for every permuation, finding the key threshold changes with

- number of nodes
- size of delay

4 CONCLUSION

(1) analysis of Parsplice keyspace
(2) using a modern distributed kv store
(3) positive effects of Mantle

REFERENCES

[1] Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Lan Xue. 2003. Efficient Metadata Management in Large Distributed Storage Systems. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST '03)*.

[2] Hugh Greenberg, John Bent, and Gary Grider. 2015. MDHIM: A Parallel Key/Value Framework for HPC. In *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*.

[3] Gary Grider, Dave Montoya, Hsing-bung Chen, Brett Kettering, Jeff Inman, Chris DeJager, Alfred Torrez, Kyle Lamb, Chris Hoffman, David Bonnie, Ronald Croonenberg, Matthew Broomfield, Sean Leffler, Parks Fields, Jeff Kuehn, and John Bent. 2015. MarFS - A Scalable Near-Posix Metadata File System with Cloud Based Object Backend. In *Work-in-Progress at Proceedings of the 10th Workshop on Parallel Data Storage (PDSW'15)*.

[4] Vivek S. Pai, Mohit Aron, Gaurov Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. 1998. Locality-aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*.

[5] Swapnil V. Patil and Garth A. Gibson. 2011. Scale and Concurrency of GIGA+: File System Directories with Millions of Files. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)*.

[6] Danny Perez, Ekin D Cubuk, Amos Waterland, Efthimios Kaxiras, and Arthur F Voter. Long-Time Dynamics Through Parallel Trajectory Splicing. *Journal of chemical theory and computation* (????).

[7] Kai Ren, Qing Zheng, Swapnil Patil, and Garth Gibson. 2014. IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. In *Proceedings of the 20th ACM/IEEE Conference on Supercomputing (SC '14)*.

[8] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson. 2000. A Comparison of File System Workloads. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '00)*. 4–4.

[9] Michael A. Sevilla, Noah Watkins, Ivo Jimenez, Peter Alvaro, Shel Finkelstein, Jeff LeFevre, and Carlos Maltzahn. Malacology: A Programmable Storage System. In *Proceedings of the 12th European Conference on Computer Systems (Eurosys '17)*. Belgrade, Serbia. To Appear, preprint: https://www.soe.ucsc.edu/research/technical-reports/UCSC-SOE-17-04.

[10] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design & Implementation (OSDI'06)*.

[11] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, and Ethan L. Miller. 2004. Dynamic Metadata Management for Petabyte-Scale File Systems. In *Proceedings of the 17th ACM/IEEE Conference on Supercomputing (SC'04)*.

[12] Qing Zheng, Kai Ren, and Garth Gibson. 2014. BatchFS: Scaling the File System Control Plane with Client-funded Metadata Servers. In *Proceedings of the 9th Workshop on Parallel Data Storage (PDSW' 14)*.