Metadata Load Balancing Policies and Key-Value Stores

MICHAEL SEVILLA*

ABSTRACT

Enter the text of your abstract here.

1. Outline

a. Formulation of a Question: Do HPC kv applications need dynamic load balancing policies or is there a one-size-fits-all policy?

- "Load balancing is hard" – Garth

b. Hypothesis: The Mantle approach is an effective mechanism for switching load balancing policies in HPC key-value applications

- Mantle is a load balancing approach and API.
  - quantifies effect of load balancing
  - formalized effective FS balancers
  - debugging tool
- HXHIM has migration mechanisms for load balancing
  - bulk operations (`put/get()`)
  - key partitioners
  - secondary indices

c. Prediction

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

d. Testing: Combine Mantle and HXHIM to explore dynamic load balancing policies for ParSplice, an HPC application with distinct workload phases

1) PARSPLICE IS STRUCTURED

Figure 1.

2) PARSPLICE BEHAVIOR CAN CHANGE

Figure 2.

e. Analsyis

* *Corresponding author address:* Los Alamos National Laboratory
E-mail: msevilla@ucsc.edu

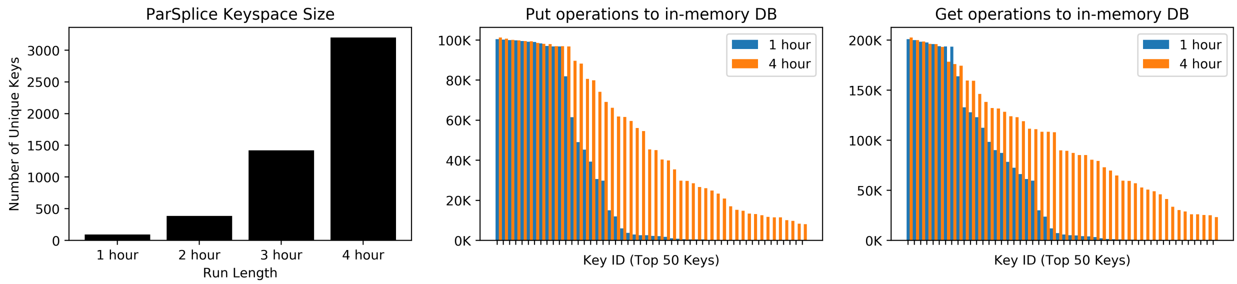

FIG. 1. We can predict how fast the keyspace grows and which parts of the namespace are popular.

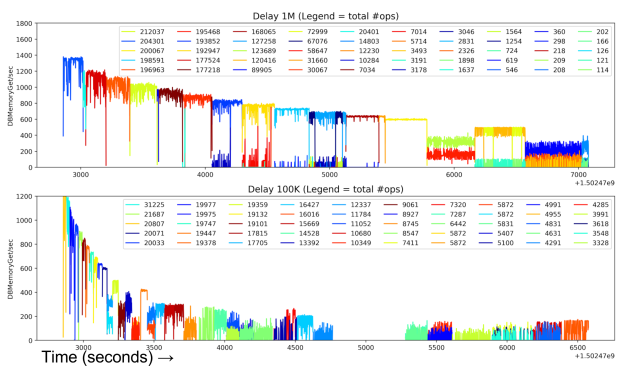

FIG. 2. With a simple parameter tweak, the workload regimes changes (timescale) so one size does not fit all.

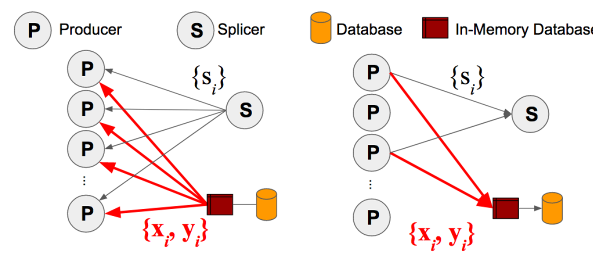

FIG. 3. ParSplice is a ready-heavy HPC application where producers use a database for consistency. Replacing the single-node database with HXHIM improves performance with load balancing.

2. Introduction

Load balancing is a useful tool for optimizing performance in systems that service highly accessed data[1] but deciding how to make the migrations is a risky trade-off. In this paper, we show that a one-size-fits-all data load balancing policy is not sufficient for even the simplest of HPC applications and argue for a dynamic load balancing policy.

Resource migration is the key mechanism for load balancing. In storage, data can be distributed to alleviate overloaded servers or it can be concentrated to exploit locality. These techniques are at odds and selecting the wrong technique can have catastrophic consequences. For example, migrating data to an already overloaded server or increasing the network hops by spreading data across an underutilized cluster will impact performance negatively.

Unfortunately, deciding which optimization to use is difficult to reason about, especially with the scale and complexity of today's HPC architectures. While the mechanisms are usually built into the systems, the policies often times less refined and much more sensitive to the workload. So a system may have the ability exploit locality using techniques like bulk operations, multiple partition strategies, secondary indexes, and caching but deciding when, where, and how to use them is workload dependent and difficult to figure out.

This paper takes an API designed to migrate file system metadata and applies it to an HPC key-value store. The API helps control distribution and concentration by letting the administrator define how to migrate load, where to migrate load, and how much load to migrate. While designed for a different domains, this API encompasses many of the same properties we need for an HPC key-value store, namely:

- services small/frequent requests
- popularity drives distribution
- locality drives concentration

To show the efficacy of this approach, we examine the ParSplice molecular dynamics simulation application shown in Figure 4. ParSplice uses a single-node database for consistency, where producers, P, push and pull coordinates, $\{x_i, y_i\}$, based on the segments, $\{s_i\}$, assigned by the splicer, S. In this paper, we replace the database with

[1] In this paper, we use the term "data" to refer to the partitioned key-value pairs AND file system metadata.

a distributed key-value store designed for HPC enjoy performance optimizations for:

- `put()` because of the distributed sync and load balancing based on:
  - lazy synchronization with tombstones and RPCs
  - strong synchronization with consensus and blocking
- `get()` because of the load balancing

It has 4 phases:

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 $\{x_j, y_j\}$

, which has both a high computational footprint and data locality. The former suggests distribution to avoid hot spots while the latter encourages concentration to leverage the database's secondary indeices, bulk operations, and key redistribution functionality.

To show this approach at scale we study ParSplice [6], an HPC dynamics simulator that has both a high computation footprint, which suggests distribution to avoid hot spots, and data locality, which alternatively encourages concentration so the key-value store can use its functionality for secondary indices, bulk operations, and key redistribution. ParSplice uses both molecular dynamic (MD) and accelerated molecular dynamic methods (AMD) for simulations with long periods of inactivity and short periods of "interesting" events. Molecules in periods with many events are simulated with MD methods, which are exact but can only be run for a fixed, short period of time because the cumulative error grows so large. Alternatively, longer trajectories are simulated with AMD methods, which use statistics and parallelization to show the less precise state-to-tate trajectories. ParSplice tackles "low-barrier problems", where the types of energy barriers separating states of the system are non-uniform (i.e. some require less energy than others). It chops long trajectories into parallelizable units called segments, where the segments can also be spliced together to form longer trajectories; this approach allows ParSplice to trade off accuracy for speed in a configurable way.

ParSplice stores segments in a database while it runs. The splicer pastes segments generated by n producers.

In this paper, we make the following contributions:

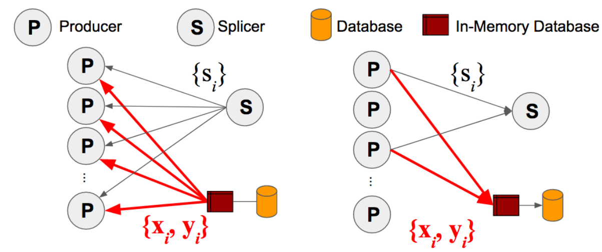

FIG. 4. ParSplice is a ready-heavy HPC application where producers use a database for consistency. Replacing the single-node database with HXHIM improves performance with load balancing.

1. protype that controls concentration and distribution using the bulk operations, secondary indicies, and cursor types mechanisms from [2].
2. quantifies benefits of server/client-side caching, many small messages, and bulk operations.

3. Background

This paper takes the API and load balancers designed in Mantle [10], the programmabile file system metadata load balancer for Ceph, and applies them to MDHIM [2], the distributed key-value store designed for HPC.

a. ParSplice

It has 4 phases:

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 $\{x_j, y_j\}$

b. Mantle: File System Metadata Load Balancer

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 [13, 3, 7, 5, 1]. Mantle can use strategies from these papers to control how to distribute or concentrate file system metadata.

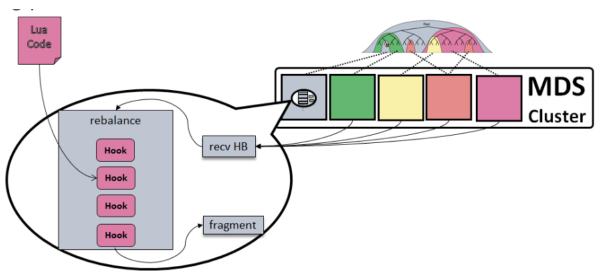

FIG. 5. The Mantle API lets adminstrators control load balancing by changing the poicies for how to distribution or concentrate file system metadata. It was merged into CephFS and inherits many aspects of that architecture. Although it has the load balancing structure and logic from CephFS (gray boxes), the actual API is not dependent on that code base.

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 5. 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 5 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 6. 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 [12, 11], sheds a fraction of the load frequently.

It was merged[2] 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.

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

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

FIG. 6. The Greedy Spill balancer written in Lua using the Mantle API.

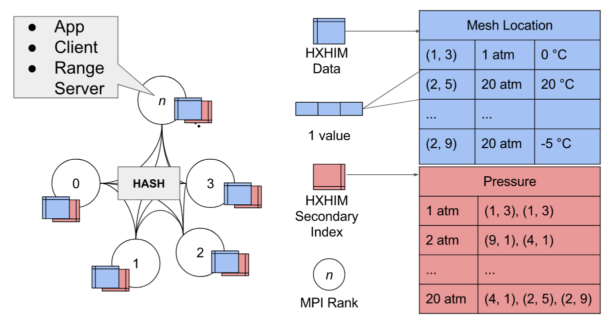

FIG. 7. The MDHIM architecture.

c. 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 7 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 database (either 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 7 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 cus-

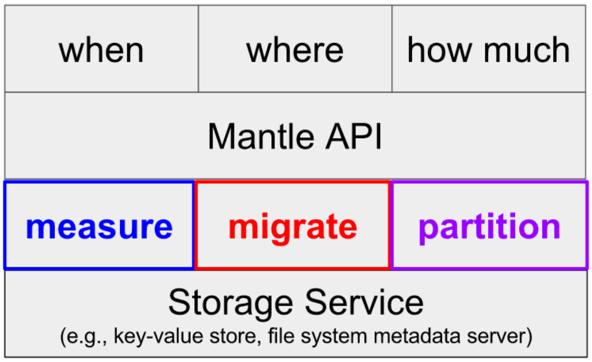

FIG. 8. The storage service must: **measure** resource usage, **migrate** resources, and **partition** resources.

tomized partitioning strategies and user-defined secondary indices. This allows the system to choose whether to send load to the client or server.

d. 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

4. Methodology: Extracting Mantle Library

- C bindings for Mantle

a. Pluggable Interfaces

1) MEASURE

The metrics measured should help the system decide "when" to migrate server load. They should:

- tell us about the state of the server or cluster
- provide some value of load, so we can partition/send it

In Ceph: global and local metrics (*e.g.*, CPU utilization, file system operation counts)

In HXHIM: ???

2) MIGRATE

In Ceph: `export_dir()`
In HXHIM: `mdhimBPut()`, `mdhimBGet()`, "adjusting ... keys" ???

3) PARTITION

In Ceph: subtrees and directory fragments
In HXHIM: secondary indices, cursor types, bul operations

Acknowledgments. Start acknowledgments here.

References

[1] S. A. Brandt, E. L. Miller, D. D. E. Long, and L. Xue. 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, 2003.

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

[3] G. Grider, D. Montoya, H.-b. Chen, B. Kettering, J. Inman, C. DeJager, A. Torrez, K. Lamb, C. Hoffman, D. Bonnie, R. Croonenberg, M. Broomfield, S. Leffler, P. Fields, J. Kuehn, and J. Bent. 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, November 2015.

[4] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. 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, 1998.

[5] S. V. Patil and G. A. Gibson. 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, 2011.

[6] D. Perez, E. D. Cubuk, A. Waterland, E. Kaxiras, and A. F. Voter. Long-Time Dynamics Through Parallel Trajectory Splicing. *Journal of chemical theory and computation*.

[7] K. Ren, Q. Zheng, S. Patil, and G. Gibson. IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. In *Proceedings of the 20th ACM/IEEE Conference on Supercomputing*, SC '14, 2014.

[8] D. Roselli, J. R. Lorch, and T. E. Anderson. A Comparison of File System Workloads. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '00, pages 4–4, 2000.

		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.

[9] M. A. Sevilla, N. Watkins, I. Jimenez, P. Alvaro, S. Finkelstein, J. LeFevre, and C. 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] M. A. Sevilla, N. Watkins, C. Maltzahn, I. Nassi, S. A. Brandt, S. A. Weil, G. Farnum, and S. Fineberg. Mantle: A Programmable Metadata Load Balancer for the Ceph File System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, 2015.

[11] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design & Implementation*, OSDI'06, 2006.

[12] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. In *Proceedings of the 17th ACM/IEEE Conference on Supercomputing*, SC'04, 2004.

[13] Q. Zheng, K. Ren, and G. Gibson. BatchFS: Scaling the File System Control Plane with Client-funded Metadata Servers. In *Proceedings of the 9th Workshop on Parallel Data Storage*, PDSW' 14, 2014.