JOURNAL OF APPLIED METEOROLOGY AND CLIMATOLOGY

Metadata Load Balancing Policies and Key-Value Stores

MICHAEL SEVILLA*

ABSTRACT

Enter the text of your abstract here.

1. Introduction

Migrating resources is a useful tool for optimizing performance for systems that service highly accessed data[1] but deciding how to make the migrations is a risky tradeoff. 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

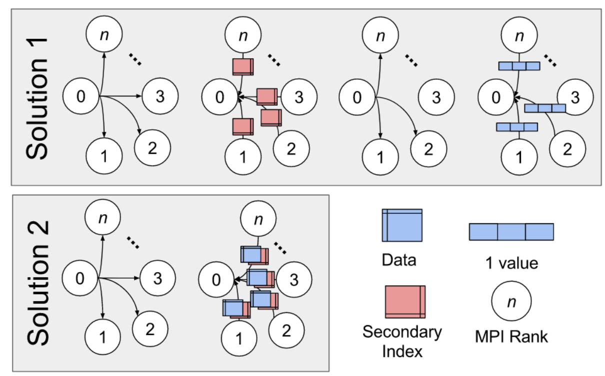

FIG. 1. For a multi-part query with locality, migrating for distribution (solution 1) takes more RPCs while migrating for concentration (solution 2) risks of overloading the client.

To show the efficacy of this approach, we examine multi-part queries that have both a high computational footprint, which suggests distribution to avoid hot spots, and data locality, which alternatively encourages concentration so the system can use its functionality for secondary indices, bulk operations, and key redistribution. The motivating example for this integration is finding the maximum value x of the neighbors in a mesh of another maximum value y. For example, finding the highest temperatures of the neighbors of mesh cells with the highest pressure. Given that the hash is defined by mesh location, finding the highest pressure is one RPC per server. Unfortunately, even with an index based on the maximum pressures, finding the highest temperatures for *neighboring* cells with the highest pressures requires an additional RPC per server.

Specifically, the API helps us explore the space of solutions for these types of queries. As shown in Figure 1, queries with locality have two solutions: (1) pull the index and re-query every server, or (2) pull the index and partial set of results that can be satisfied locally. Solution 1 em-

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

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

phasizes distribution and incurrs extra RPCs while Solution 2 opts to concentrate data at the expense of data transfer, consistency issues, and increased memory usage. Both options have advantages but inserting an API to control the mechanisms helps future programmers quickly evaluate both options and design solutions for their workload.

In this paper, we make the following contributions:

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.

2. Background

This paper takes the API and load balancers designed in Mantle [9], the programmabile file system metadata load balancer for Ceph, and applies them to HXHIM [2], the distributed key-value store designed for HPC.

a. 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 [7], 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 [12, 3, 6, 5, 1]. 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 2. 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 2 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

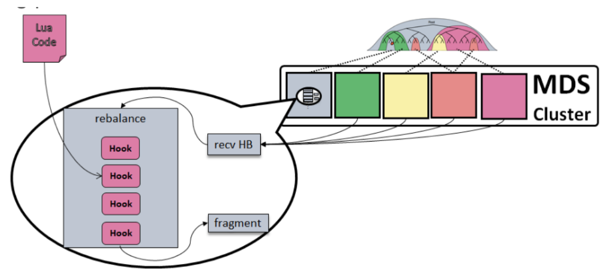

FIG. 2. The Mantle API lets adminstrators control load balancing by changing the poicies for how to distribution or concentrate file system metadata. It was merged into CephFS and inherits many aspects of that architecture. Although it has the load balancing structure and logic from CephFS (gray boxes), 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
```

FIG. 3. The Greedy Spill balancer written in Lua using the Mantle API.

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 3. 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 [11, 10], 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 [8] 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.

b. HXHIM: Key-Value Store for HPC

HXHIM is a key-value store designed for HPC architectures and multi-dimensional data.

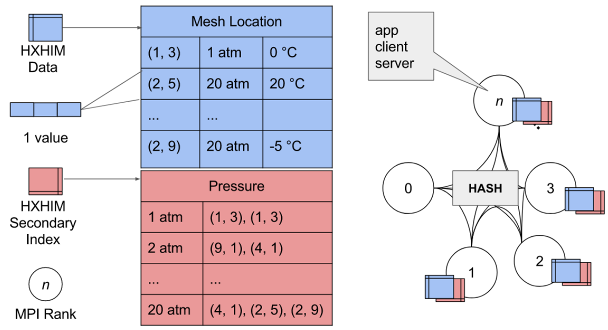

FIG. 4. The HXHIM architecture.

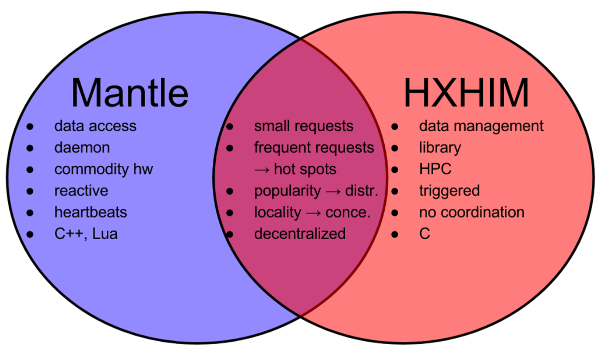

FIG. 5. Comparing the design goals and implementations of Mantle and HXHIM.

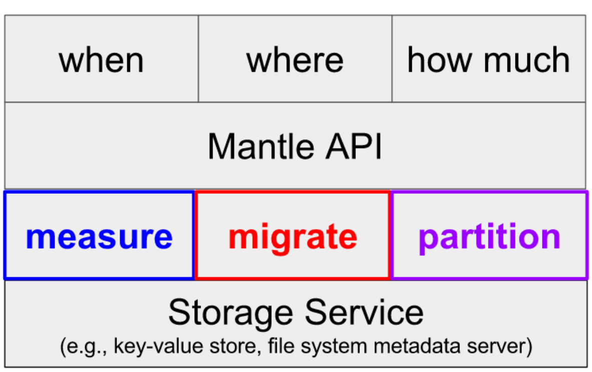

FIG. 6. The storage service must: **measure** resource usage, **migrate** resources, and **partition** resources.

c. Comparing Mantle and HXHIM

3. Methodology: Extracting Mantle Library

a. Pluggable Interfaces

1) MEASURE

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] 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.

[7] 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.

[8] 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`.

[9] M. A. Sevilla, N. Watkins, C. Maltzahn, I. Nassi, S. A. Brandt, S. A. Weil, G. Farnum, and S. Fineberg. Mantle: A Programmable

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

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.

[10] 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.

[11] 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.

[12] 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.