JOURNAL OF APPLIED METEOROLOGY AND CLIMATOLOGY

Metadata Load Balancing Policies and Key-Value Stores

MICHAEL SEVILLA*

ABSTRACT

Enter the text of your abstract here.

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

1. Introduction

Migrating resources is a useful tool for balancing load and improving performance for systems that service highly accessed data (*e.g.*, key-value stores, file system metadata, etc.) but deciding where and how to migrate is complicated by the scale and complexity of today's HPC architectures. One of the most difficult trade-offs to reason about is to keep data[1] distribution vs. concentration.

Deciding between distribution and concentration is difficult because the techniques partition data differently; distribution spreads it across a larger set of nodes while concentration keeps it on a smaller set of nodes. These migration techniques are designed to improve performance in different ways; distributing data can divide the across more nodes, reducing the burden on overloaded servers while data concentrated on a smaller number of nodes benefit from techniques that reduce the number of RPCs, like batching, caching, and network partitioning, As a result, selecting the wrong technique can have catastrophic consequences, *e.g.*, migrating data to an already overloaded server or increasing the network hops by spreading data across an underutilized cluster.

This paper takes an API designed to migrate file system metadata, applies it to an HPC key-value store, and shows the generality of the load balancing API approach. 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 domain, 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

The motivating example for integrating the API into our HPC key-value store is locality of the key-value pairs in the dataset. The key-value store supports client-defined indices and has functions for adjusting the key distribution, so if the hash is defined by mesh location range queries, like finding the temperatures for the cells with the highest pressures, require only one RPC per server. Unfortunately, even with an index based on the maximum pressures, finding the highest temperatures for cells *neighboring* cells with the highest pressures requires extra RPCs.

Specifically, the API helps us explore the space of solutions for this problem that has locality. For example, the problem above has 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. Option 1 emphasizes distribution and incurrs extra RPCs while option 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 policies for their workload.

In this paper, we make the following contributions:

1. protype that controls concentration and distribution using the key redistribution and cursor type mechanisms from [1].

2. quantifies benefits of server/client-side caching, many small messages, and bulk operations.

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

2. Background

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. It uses resource migration to load balance file system metadata across a dedicated metadata cluster and

Mantle [3] is a programmable file system metadata balancer implemented on the file system above Ceph (CephFS).

system metadata IO does not scale like metadata IO [2]

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

TABLE 1. This is a sample table caption and table layout. Enter as many tables as necessary at the end of your manuscript. Table from Lorenz (1963).

N	*X*	*Y*	*Z*
0000	0000	0010	0000
0005	0004	0012	0000
0010	0009	0020	0000
0015	0016	0036	0002
0020	0030	0066	0007
0025	0054	0115	0024

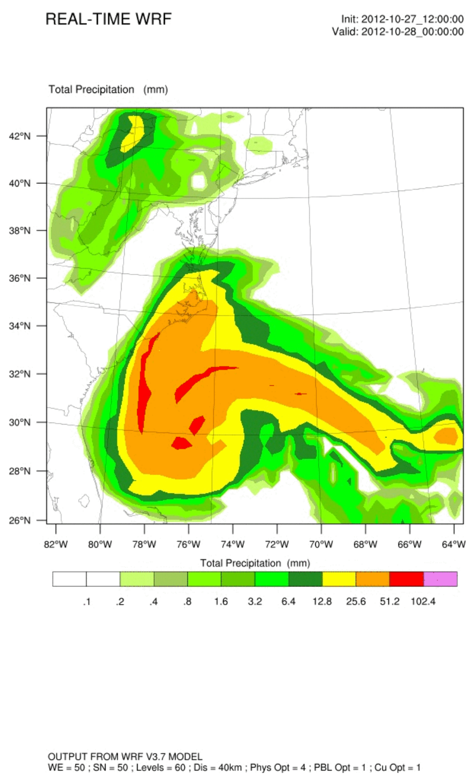
Acknowledgments. Start acknowledgments here.

References

[1] 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.

[2] 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.

[3] 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.



OUTPUT FROM WRF V3.7 MODEL
WE = 50 ; SN = 50 ; Levels = 60 ; Dis = 40km ; Phys Opt = 4 ; PBL Opt = 1 ; Cu Opt = 1

FIG. 1. Enter the caption for your figure here. Repeat as necessary for each of your figures.