10-2009

Understanding and Maturing the Data-Intensive Scalable Computing Storage Substrate

Garth Gibson
Carnegie Mellon University

Bin Fan
Carnegie Mellon University

Swapnil Patil
Carnegie Mellon University

Milo Polte
Carnegie Mellon University

Wittawat Tantisiriroj
Carnegie Mellon University

See next page for additional authors

Understanding and Maturing the Data-Intensive Scalable Computing Storage Substrate

Garth Gibson, Bin Fan, Swapnil Patil, Milo Polte, Wittawat Tantisiriroj, Lin Xiao
Carnegie Mellon University

1 eScience is Data Intensive

Modern science has available to it, and is more productively pursued with, massive amounts of data, typically either gathered from sensors or output from some simulation or processing. The table below shows a sampling of data sets that a few scientists at Carnegie Mellon University have available to them or intend to construct soon. Data Intensive Scalable Computing (DISC) couples computational resources with the data storage and access capabilities to handle massive data science quickly and efficiently. Our topic in this extended abstract is the effectiveness of the data intensive file systems embedded in a DISC system. We are interested in understanding the differences between data intensive file system implementations and high performance computing (HPC) parallel file system implementations. Both are used at comparable scale and speed. Beyond feature inclusions, which we expect to evolve as data intensive file systems see wider use, we find that performance does not need to be vastly different. A big source of difference is seen in their approaches to data failure tolerance: replication in DISC file systems versus RAID in HPC parallel file systems. We address the inclusion of RAID in a DISC file system to dramatically increase the effective capacity available to users.

This work is part of a larger effort to mature and optimize DISC infrastructure services.

2 Parallel versus Data Intensive File Systems

Most DISC applications are characterized by parallel processing of massive datasets stored in the underlying shared storage system. Such distributed programming abstractions are provided by purpose-built frameworks like MapReduce [1], Hadoop [2] and Dryad [3]. These frameworks divide a large computation into many tasks that are assigned to run on nodes that store the desired input data, and avoiding a potential bottleneck resulting from shipping around terabytes of input data. Hadoop, and its data intensive file system HDFS [4], are open-source implementations of Google's MapReduce and GoogleFS [5]. In this work we focus on Hadoop's use of the HDFS data intensive file system.

At a high level HDFS's architecture resembles an HPC parallel file system. HDFS stores file data and metadata on two different types of servers. All files are divided into chunks that are stored on different data servers. The file system metadata, including the per-file chunk layout, is stored on the metadata server(s). For single writer workloads, HDFS differs from HPC parallel file systems primarily in its layout and fault tolerance schemes.

HDFS assigns chunks to compute nodes at random, while HPC file systems use a round robin layout over dedicated storage servers, and HDFS exposes a file's layout information to Hadoop. This exposed layout allows the Hadoop's job scheduler to allocate tasks to nodes in a manner that (1) co-locates compute with data where possible, and (2) load

Contact	Field	Comments
J Lopez, CSD	Astrophysics	SDSS digital sky survey including spectroscopy, 50TB
T DiMatteo, Phys	Astrophysics	Bigben BHCosmo hydrodynamics (1B particles simulated), 30TB
F Gilman, Phys	Astrophysics	Large Synoptic Survey Telescope, LSST (2012) digital sky survey, 15TB/day
C Langmead, CSD	Biology	Xray, NMR, CryoEM images; simulated molecular dynamics trajectories
J Bielak, CE	Earth sciences	USGS sensor images; simulated 4D earthquake wavefields >10TB/run
D Brumley, ECE	Cyber security	Worldwide Malware Archive; 2TB and doubling each year
O Mutlu, ECE	Genomics	50GB per compressed genome sequencing; expands to TBs to process
B Yu, ECE	Neuroscience	Neural recordings (electrodes, optical) for prosthetics; 10-100GB each
J Callan, LTI	Info Retrieval	ClueWeb09, 25TB, 1B high rank web pages, 10 languages
T Mitchell, MLD	Machine Learning	English sentences of ClueWeb for continuous automated reading (5TB)
M Herbert, RI	Image Understanding	Flickr archive (>4TB); broadcast TV archive and street video archives
Y Sheikh, RI	Virtual Reality	Terascale VR sensor, 1000 camera+ 200 microphone, up to 5TB/sec
C Guestrin, CSD	Machine Learning	Blog update archives, 2TB now + 2.7TB/yr (about 500K blogs/day)
C Faloutsos, CSD	Data Mining	Wikipedia change archive (1TB), Fly embryo images (1.5TB), links from web
S Vogel, LTI	Machine Translation	Pre-filtered N-gram language model, on statistics on word alignment, 100 TB
J Baker, LTI	Machine Translation	Spoken language recording archive, many languages and sources, up to 1PB
B Becker, RI	Computer Vision	Social network image/video archive for training computer vision, 1-5TB

	HDFS	vanilla PVFS	PVFS shim
grep performance (over a 64GB data-set on 32 nodes)			
Read throughput (MB/s)	579.4	244.9	597.1
Avg CPU utilization	43%	27%	43%
Completion time (m:s)	1:45	4:08	1:46

Figure 1: By exposing the file layout mapping through a non-intrusive shim layer, a production parallel file system (PVFS) can match the performance of HDFS on a widely used Hadoop-style workload.

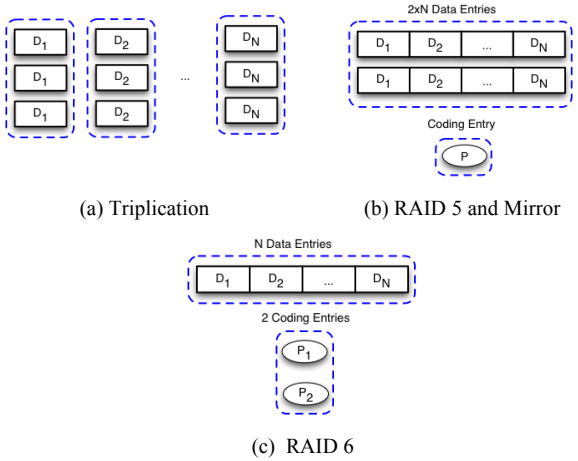
balances the work of accessing and processing data across all the nodes. Thus, the scheduler can mask sub-optimal file layout resulting from HDFS's random chunk placement policy with lots of work at each node [6]. The second big difference between HDFS and HPC file systems is its fault tolerance scheme: it uses triplication instead of RAID. We address this difference in the next section.

Given the growing importance of the Hadoop MapReduce compute model, we ask "Could we use a mature HPC parallel file system in-place of a custom-built DISC file system like HDFS?" While most HPC file systems use separate compute and storage systems for flexibility and manageability, most HPC parallel file systems can also be run with data servers on each compute node.

We built a non-intrusive shim layer to plug a real-world parallel file system (the Parallel Virtual File System, PVFS [7]), into the Hadoop framework storing data on compute nodes [6]. This shim layer queries file layout information from the underlying parallel file system and exposes it to the Hadoop layer. The shim also emulates HDFS-style triplication by writing, on behalf of the client, to three data servers with every application write.

Figure 1 shows that for a typical Hadoop application (grep running on 32 nodes), the performance of shim-enabled Hadoop-on-PVFS is comparable to that of Hadoop-on-HDFS. By simply exposing a file's layout information, PVFS enables the Hadoop application to run twice as fast as it would without exposing the file's layout.

Most parallel large-scale file systems, like PVFS, already expose the file layout information to client modules but do not make it available to client applications. For example, the new version 4.1 of NFS (pNFS) delegates file layout to client modules to allow the client OS to make direct access to striped files [8]. If these layout delegations were exposed to client applications to use in work scheduling decisions, as done in Hadoop or MapReduce, HPC and pNFS file systems could be significantly more effective in DISC system usage.



(a) Triplication (b) RAID 5 and Mirror

(c) RAID 6

Figure 2: Codewords protecting against double node failure

3 Replication versus RAID

To tolerate frequent failures, each data block in a data intensive file system is typically triplicated and therefore capable of recovering from two simultaneous node failures. Though simple, a triplication policy comes with a high overhead cost in terms of disk space: 200%. Traditional RAID systems typically exhibit capacity overheads between 10% and 25% -- about 10 times smaller! We are designing and building *DiskReduce*, an application of RAID in HDFS to save storage capacity.

In HDFS, files are divided into blocks, typically 64 MB, each stored on a data node. Each data node manages all file data stored on its persistent storage. It handles read and write requests from clients and performs "make replica" requests from the metadata node. There is a background process in HDFS that periodically checks a missing blocks and, if found, assigns a data node to replicate the block having too few copies.

DiskReduce exploits HDFS's background re-replication to replace copies with lower overhead RAID encoding. In a manner reminiscent of early compressing file systems [9], all blocks are initially triplicated; that is, uncompressed. Where the background process looks for insufficient number of copies in HDFS, DiskReduce instead looks for blocks not encoded, and replaces copies with encodings. Because it is inherently asynchronous, DiskReduce can further delay encoding when space allows, to allow accesses temporally local to the creation of the data choice among multiple copies for readback.

We have prototyped DiskReduce as a modification to Hadoop Distributed File System (HDFS) version 0.20.0. Currently, the DiskReduce prototype supports only two encoding schemes [10]: "RAID 6" and "RAID 5 and Mirror", in which a RAID5 encoding is augmented with a second complete copy of the data.

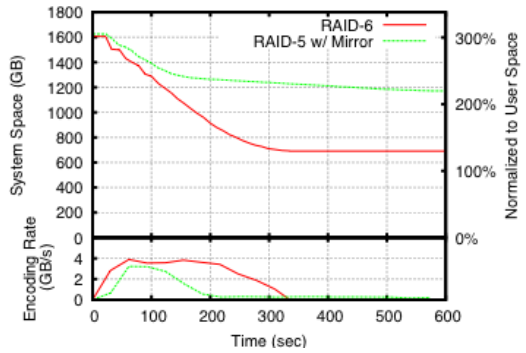


Figure 3: Storage capacity utilized and rate capacity is recovered by Disk Reduce background encoding.

Our prototype runs in a cluster of 63 nodes, each containing two quad-core 2.83GHz Xeon processor, 16 GB of memory, and four 7200 rpm SATA 1 TB Seagate Barracuda ES.2 disks with 32MB buffer. Nodes are interconnected by 10 Gigabit Ethernet. All nodes run the Linux 2.6.28.10 kernel and use the ext3 file system for storing HDFS blocks.

While our prototype is not complete (online reconstruction by a client is still a work-in-progress), it functions enough for a preliminary test. To get a feel for its basic encoding function, we set up a 32 nodes partition and had each node write a 16 GB file into a DiskReduce-modified HDFS spread over the same 32 nodes using RAID groups of eight data blocks each.

Figure 3 shows the storage usage and encoding bandwidth consumed for the encoding of this 512GB of data. While this experiment is simple, it shows the encoding process removing 400GB and 900GB for the RAID 5 and mirror and RAID 6 schemes, respectively, bringing overhead down from 200% to 113% and 25%, respectively.

4 Future Work

Based on a talk about our previous DiskReduce work, a userspace RAID 5 and mirror encoding scheme has been implemented on top of HDFS by HDFS developers [11]. We are working closely with Hadoop and HDFS developers to further explore RAID 6 encodings, delaying of encoding to enable reading soon after write the full benefit of multiple copies, and to quantify this benefit, and delaying of deletion to trade capacity against the encoding cleanup of a partial RAID set delete. Similarly HPC parallel file systems have begun to implement RAID over nodes, instead of just RAID in hardware, led by a spin off of our prior work in scalable file systems [12]. In our long-term vision for data intensive storage systems we see a convergence of the semantic power of HPC parallel file systems with the high degrees of node failure tolerance in data intensive file systems.

Acknowledgements:

The work in this paper is based on research supported in part by the Department of Energy, under award number DE-FC02-06ER25767, by the Los Alamos National Laboratory, under contract number 54515-001-07, by the National Science Foundation under awards OCI-0852543, CNS-0546551 and SCI-0430781, and by a Google research award. We also thank the member companies of the PDL Consortium (including APC, DataDomain, EMC, Facebook, Google, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Microsoft, NEC, NetApp, Oracle, Seagate, Sun, Symantec, and VMware) for their interest, insights, feedback, and support.

References

[1] Dean, J., S. Ghemawat, "Simplified Data Processing on Large Clusters." In 6th Symposium on Operating Systems Design and Implementation (OSDI'04).

[2] Hadoop. Apache Hadoop Project. HTTP://HADOOP.APACHE.ORG/

[3] Isard, M., M. Budiu, Y. Yu, A. Birrell, D. Fetterly. "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks." In 2007 Eurosys Conference.

[4] Borthakur, D., "The hadoop distributed file system: Architecture and design, 2009."

HTTP://HADOOP.APACHE.ORG/COMMON/DOCS/CURRENT/HDFS_DESIGN.HTML

[5] Ghemawat, S., H. Gobioff, S.-T. Lueng, "Google File System." In 19th ACM Symposium on Operating Systems Principles (SOSP'03).

[6] Tantisiriroj, W., S. V. Patil, G. Gibson. "Data intensive file systems for internet services: A rose by any other name…" Tech. Report CMU-PDL-08-114, Carnegie Mellon University, Oct. 2008.

[7] PVFS2. Parallel Virtual File System, Version 2. HTTP://WWW.PVFS2.ORG/

[8] IETF. NFS v4.1 specifications. HTTP://TOOLS.IETF.ORG/WG/NFSV4/

[9] Cate, V., T. Gross. "Combining the concepts of compression and caching for a two-level file system." In ASPLOS-IV, April 1991.

[10] Plank, J.S., J. Luo, C.D. Schuman, L. Xu, Z. Wilcox-O'Hearn. "A performance evaluation and examination of open-source erasure coding libraries for storage." In USENIX Conference on File and Storage Technologies (FAST'09), 2009.

[11] Borthakur, D. "HDFS and erasure codes." Aug 2009,

HTTP://HADOOPBLOG.BLOGSPOT.COM/2009/08/HDFS-AND-ERASURE-CODES-HDFS-RAID.HTML

[12] Welch, B., M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, B. Zhou. "Scalable Performance of the Panasas Parallel File System. In USENIX Conference on File and Storage Technologies (FAST'08), 2008.