1-1-2009

Using Hadoop as a Grid Storage Element

Brian Bockelman
University of Nebraska - Lincoln, bbockelman2@unl.edu

Using Hadoop as a Grid Storage Element

Brian Bockelman

Postdoc Researcher, Holland Computing Center, University of Nebraska, Lincoln, NE 68588-0150, USA

E-mail: `bbockelm@cse.unl.edu`

Abstract. Hadoop is an open-source data processing framework that includes a scalable, fault-tolerant distributed file system, HDFS. Although HDFS was designed to work in conjunction with Hadoop's job scheduler, we have re-purposed it to serve as a grid storage element by adding GridFTP and SRM servers. We have tested the system thoroughly in order to understand its scalability and fault tolerance. The turn-on of the Large Hadron Collider (LHC) in 2009 poses a significant data management and storage challenge; we have been working to introduce HDFS as a solution for data storage for one LHC experiment, the Compact Muon Solenoid (CMS).

1. Introduction

The Large Hadron Collider in Geneva, Switzerland is not only a unique experiment for investigations in particle physics, but also presents an exceptional computing challenge for those tasked with recording and processing huge amounts of data. When originally tasked with the problem, the four detector experiments on the LHC decided to take a decentralized processing approach through using grid computing. The grid formed to handle this task is the Worldwide LHC Computing Grid (WLCG).

At the Holland Computing Center in Nebraska, we have established a Tier-2 site for the CMS experiment. A Tier-2 site is primarily an analysis and simulation site for physicists. The site currently has about 200TB of storage and will grow to 400TB during 2009. The heaviest usage of the storage system comes from transferring data from a Tier-1 site (such as FNAL in the US) and from running I/O-intensive analysis jobs. The downloads are expected to sustain 5 Gbps and burst at 9 Gbps. The analysis jobs read at a rate of a few MB/s, but currently have a huge number of random read operations. The compute cluster is approximately 1,000 cores; as certain analysis jobs are CPU-bound, the cluster needs a few million I/O operations per minute.

Since the LHC experiments formed the WLCG, several Internet-based companies have emerged as having far larger data needs. Google grew from processing 100 TB of data a day in 2004 [1] to processing 20 PB a day [2]. As a comparison, the CMS estimated 2009 needs for total tape space are 25PB [3]. Google, Yahoo, Facebook, and others have approached the large data processing challenge with a processing model called MapReduce. If a user casts their problem into the MapReduce paradigm (loosely borrowed from functional programming), they will be able to scale the problem from a laptop to a cluster of 2,000 machines. A MapReduce implementation needs both an intelligent job scheduler and a very scalable, fault-tolerant distributed file system.

While Google's MapReduce implementation is proprietary, other large companies use an open-source implementation called Hadoop [4]. Hadoop is a product of the Apache Software

Foundation; the primary contributor is Yahoo. The largest Hadoop cluster is approximately 14PB of online disk in 4,000 machines. Hadoop is meant to work on thousands of commodity machines and therefore the Hadoop distributed file system (HDFS) has been designed to treat hardware failure as a normal occurrence that should be worked around. The reputation for scalability and fault-tolerence led us to evaluate and integrate HDFS at a large scale at our site.

The architecture of HDFS (and the GoogleFS, which HDFS is based on) is covered in [5, 6]. It is composed of many datanodes containing storage and a single namenode, which handles the namespace and its serialization. Any metadata operations goes to the namenode; all data movement goes through one of the datanodes. The I/O operations scale based on the datanode hardware capabilities. The files are broken up into 64MB chunks called "chunks" and distributed amongst the datanodes based on a given policy. The namenode is a clever design which keeps all metadata in memory, allowing it to scale well in terms of operations per second, but has the drawback that the number of files is limited by the machine's RAM.

2. Adopting Hadoop for the WLCG

Hadoop and HDFS are designed for processing and storing large volumes of data for web companies. Each Hadoop instance is usually run by the same company using it; transfers between Hadoop installs primarily happen within the same datacenter (or at least the same administrative domain). On the other hand, grid computing emphasizes the ability to transfer data between sites using common protocols; each site is under a different administrative domain and possibly using a completely different underlying storage system.

On the WLCG, the protocol used for WAN data transfers is GridFTP [7] and the protocol used for metadata operations is SRM [8]; both are necessary for interoperability between site storage and the grid. In addition, most grid users expect an efficient way for jobs to access data within the site's storage system. These three needs were addressed in the following way:

- For local access, HDFS has a binding for Linux called FUSE [9]. FUSE allows one to write a mountable file system in userspace (greatly simplifying the development of a file system and removing the need for custom kernels). Hence, worker nodes can mount HDFS and users access it in a similar manner to a NFS server. This transforms HDFS into more "normal looking" file system for users, as opposed to a more specialized data storage system. In the process of testing and deploying FUSE at a large scale, we have contributed back several bug reports and improvements. To our knowledge, we are the largest and heaviest user of FUSE, as none of the physics applications use the native Hadoop API.
- For SRM access, we have utilized the Berkeley Storage Manager (BeStMan) SRM server [10]. BeStMan's gateway mode implements all of the SRM calls utilized by CMS and ATLAS (but not all the SRM protocol functionalities). This stripped-down implementation has an advantage in that it can be used on any POSIX-like file system; other implementations of the SRM protocol are often tied to a specific storage system. The FUSE mounts which provide data access to jobs on worker nodes also provide metadata access to the SRM server. Hence, we were able to use BeStMan to provide SRM access to HDFS without any initial modifications.
- For GridFTP, we selected the Globus GridFTP server. The Globus server is in wide use elsewhere, and allows for pluggable modules to be implemented to interact with a storage system. Unlike SRM, GridFTP actually writes data in and out of HDFS. Because HDFS only supports appends, and not random writes, we had to implement in-memory data re-ordering and write an interface module between GridFTP and HDFS's C bindings.

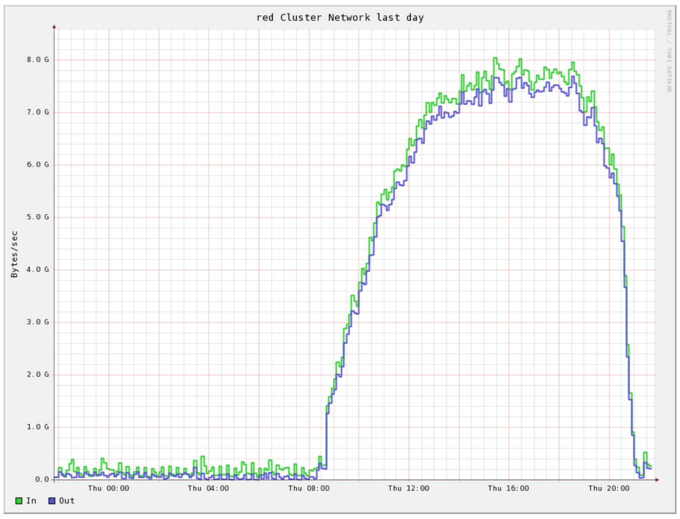

Figure 1. Network graph from a CMS user application; this graph is an aggregate for the HDFS cluster. The read rate peaks at around 8GB/s. This graph was recorded while the system was in production. The user's application had a very large readahead (10MB; the currently used one is 64KB), which causes an impressive network rate but doesn't happen for normal configurations. The readahead value is the minimum network transfer size used by the filesystem; it is an effective way to reduce latency in sequential reads, but the buffer thrashes constantly for random read workloads.

3. Performance Aspects of Hadoop

This section deals with the performance aspects of the Hadoop install at Nebraska. We currently have a dedicated Hadoop namenode (a machine with similar hardware to our worker nodes), a SRM server virtualized in a Xen machine, and 5 dedicated GridFTP servers. There are three datanode types: those having a single 80GB hard drive, those having two 1.5TB hard drives, and two Sun X4500 "Thumpers" with 48TB each. Except for the thumpers, each datanode has a 1Gbps network connection (non-blocking); the Thumpers have 4Gbps each.

Figure 1 shows the network rate (peaking at 8GB/s) from a user application reading from Hadoop. Figure 2 shows the namenode metadata operation rate we have observed in production; we have measured about 450 metadata operations / second through tests for "open" operations, and much higher for mixed workflows. Figure 3 shows the aggregate number of I/O operations per minute. This translates into 60 IOPS per hard drive (our cluster has approximately 330 hard drives), which is roughly the limit for the consumer-class harddrives that are contained in the majority of the cluster. All three measurements we show have been observed in production, but for unusual workflows. We have significant headroom with respect to performance for normal day-to-day operations, and believe we can scale further with more drives.

4. Hadoop Reliability and Manageability

As a CMS T2, Nebraska needs to provide more than just a highly-scalable Grid storage element. A storage element provides essential data to jobs; storage that can provide gigabytes a second under test condition is useless if it is not highly available under production. The SE must prove itself in terms of:

- Reliability. Even when reliable hardware is used (unless very *expensive* hardware is used), the system as a whole would be unreliable if it depended on every piece of hardware being available at all times. HDFS provides reliability by having the namenode track every block's

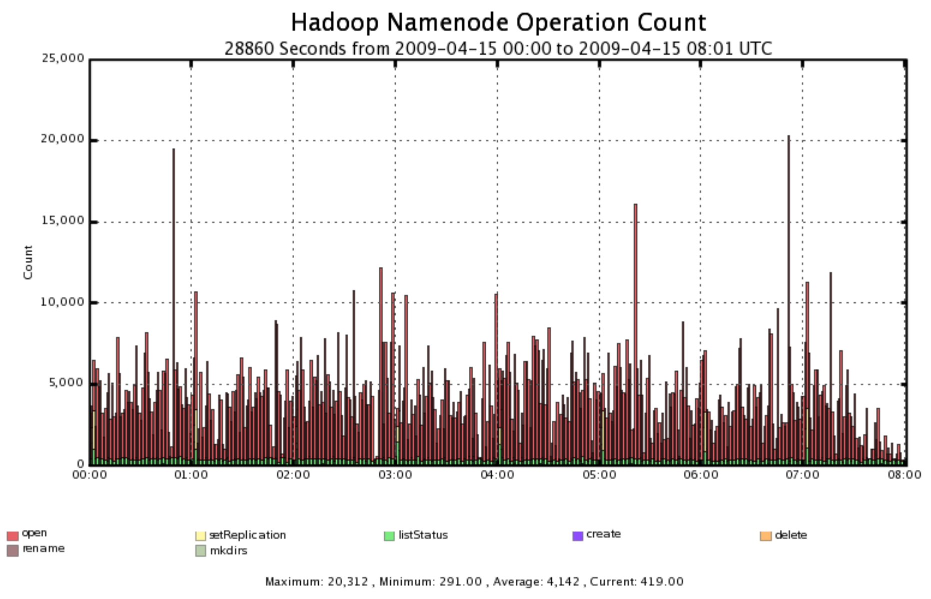

Figure 2. A graph derived from the logs of the Hadoop namenode. Each bar shows the number of times a certain operation is performed in a minute. The peak rate is 338 operations / second. During the time period shown, the load was primarily generated by a grad student working with many small files (about 64KB); it was not artificially generated. The CMS workflows use 2GB files, and do not have such a load on the metadata server.

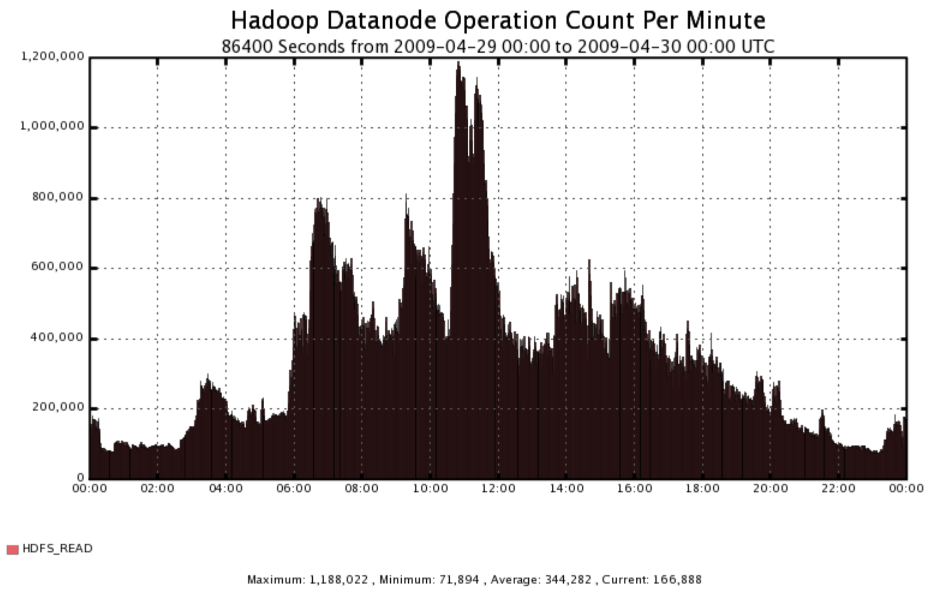

Figure 3. The number of data operations per minute delivered by the cluster. This graph shows the busiest day we've recorded during production. 1.2M I/O operations per minute is 20,000 IOPS; this is about 60 IOPS per hard drive. This is approximately the hardware limit for the class of hard drives we utilize.

location and replicate blocks as soon as they are lost. HDFS keeps a checksum of each block and performs background checksums on blocks on each datanode. The namenode is not currently highly available, but this can be mitigated by using more reliable hardware and RAIDs (this is an actively worked-on area).

- Manageability. Common management tasks must either be automated or simple. HDFS has a built-in tool to distribute a pool's data to others in order to allow it to be taken offline. It has a command-line tool that tells the admin the system is corrupt if a file block has no physical replicas. Hadoop has hooks for site monitoring tools such as Ganglia and JMX.
- Support. Other storage solutions have a close working relationship with CMS. With HDFS, CMS has no ability to influence development priorities of core developers or steer the file system's development. We feel this is mitigated by a few facts. HDFS has a large active development team which addresses bugs quickly when they are discovered; if a bug critically affects the T2, it would affect larger deploys which do employ developers. We find that HDFS covers CMS's core needs - no feature development is critically needed; we have been running the current install at the desired scale for months. Finally, if an unmet need does appear, there are companies such as Cloudera which provide paid support for Hadoop.

5. Conclusions and Future Directions

We have successfully adopted and used Hadoop's Distributed File System at our CMS site. We believe this shows HDFS is a viable alternative for a grid SE, especially in terms of scalability, reliability, and managability. While it has scaled well in our testing, there are improvements we would like to pursue. MapReduce provides capabilities CMS currently lacks; even without using the MapReduce framework, job scheduling ideas could be reused. CMS splits tasks on the submit side, and is unaware of the optimal splits on the remote cluster. Hadoop's scheduler tries to collocate the jobs and data; a MapReduce job might be able to schedule 70% or greater of its jobs so inputs are on local disk. Currently, this only happens in about 2% of the jobs. By using MapReduce - or a data-aware batch scheduler - CMS could drastically reduce its networking footprint.

There are desired improvements in HDFS and its grid adaptation which would have appreciable improvements at our site:

- Decreasing the memory footprint of GridFTP servers. As we re-order data from multiple streams in memory, we pay a significant penalty in memory per GridFTP process. We wish to explore other transport layers, such as UDT instead of TCP, which scale better and do not require multiple streams to achieve high throughput.
- Providing a strong security system. Currently, the security model is similar to NFS: it depends on anyone already inside the cluster's private network being trusted. There is no encryption or strong authentication. This is something that Hadoop hopes to address (perhaps by integration with Kerberos) in the 6-8 month timeframe.
- Implementing datacenter-aware policies. The Holland Compute Center has datacenters in two cities. A Hadoop instance is not currently optimized to understand the network topology characteristic of having two datacenters; we would like to see it optimized for data placement and access policies that understands having a segmented cluster.

Acknowledgments

Acknowledgements. This work was jointly funded by the Open Science Grid, funded jointly by DOE and NSF. This work was completed utilizing the Research Computing Facility with the associated USCMS Tier-2 site at the University of Nebraska - Lincoln. The requirements for a grid SE was provided by USCMS, and other USCMS Tier-2 sites provided valuable testing for the Hadoop installation packages.

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Proceedings of USENIX OSDI'04*, pages 137–150, 2004.
[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
[3] D Espriu. Computing resources scrutiny group. Technical Report CERN-RRB-2008-106, CERN, Geneva, Oct 2008.
[4] Apache hadoop, 2009. http://hadoop.apache.org/.
[5] Dhruba Borthakur. Hdfs architecture, 2009. http://hadoop.apache.org/core/docs/r0.20.0/hdfs_design.html.
[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *19th ACM Symposium on Operating Systems Principles*, October 2003.
[7] I Mandrichenko, W Allcock, and T Perelmutov. Gridftp v2 protocol description, 2005. http://www.ggf.org/documents/GFD.47.pdf.
[8] A Sim and A Shoshani et al. The storage resource manager interface specification version 2.2, 2008. http://www.ggf.org/documents/GFD.129.pdf.
[9] Filesystem in userspace, 2009. http://fuse.sourceforge.net.
[10] A Shoshani et al. Storage resource managers: Recent international experience on requirements and multiple co-operating implementations. In *24th IEEE Conference on Mass Storage Systems and Technologies*. IEEE Computer Society, September 2007.