Apache Hadoop
The Scalability Update

KONSTANTIN V. SHVACHKO

Konstantin V. Shvachko is a veteran Hadoop developer. He is a principal Hadoop architect at eBay. Konstantin specializes in efficient data structures and algorithms for large-scale distributed storage systems. He discovered a new type of balanced trees, S-trees, for optimal indexing of unstructured data, and he was a primary developer of an S-tree-based Linux file system, treeFS, a prototype of reiserFS. Konstantin holds a PhD in computer science from Moscow State University, Russia. He is also a member of the Project Management Committee for Apache Hadoop.
kshvachko@ebay.com

Scalability is one of the primary forces driving popularity and adoption of the Apache Hadoop project. A typical use case for Hadoop is an emerging Web site starting to run a five-node Hadoop cluster and then gradually increasing it to hundreds of nodes as business grows.

Last year *;login:* published my article [12] summarizing one aspect of Hadoop scalability, namely, the limits of scalability of the Hadoop Distributed File System [13]. There are many other dimensions to Hadoop scalability. Here I would like to address some of them.

Source of Innovation

This has been an eventful year for Apache Hadoop [1]. The project has emerged as a data mining platform, becoming an industry standard for Big Data. Apache Hadoop is successfully used in science and a variety of industries. Scientific applications include mathematics, high energy physics, astronomy, genetics, and oceanography. The platform adoption has scaled far beyond information technology—its original target area—into most industries, excluding only hunting and fishing, but probably not for long.

Started as a computational platform for search engines, Apache Hadoop is now used for data warehousing, behavioral analysis, recommendation engines, cryptanalysis, meteorology, fraud and spam detection, natural language processing, genomic analysis, image processing, semantic text analysis, etc.

Apache Hadoop was used to compute the record two quadrillionth (10^{15}) digit of π [15], which turned out to be 0, and helped IBM's Watson to win on *Jeopardy* in the "Man versus Machine race," as media presented it. Recognized for its influence on technological innovation, the Apache Hadoop project has won the 2011 MediaGuardian Innovation Award over nominees such as iPad and WikiLeaks.

While Hadoop was the driving force of technological innovation, its internal innovation has been on a rather slow rise. Not to imply that it's inert. On the contrary, a lot of development is going on in the field. Hard to underestimate the value of implementing security for Hadoop, or building analytical tools with Hadoop, or stabilizing internal company releases, which are essential for businesses running Hadoop. However, due to the lack of dominating gravitational force, these highly dedicated activities did not materialize in common productionready releases, uniformly supported by different developer groups, which would have consolidated the project.

Size Matters

Incremental improvements in HDFS performance led to gradual growth of Hadoop clusters over the year. The largest Hadoop clusters are run by Yahoo and Facebook, with eBay catching up in a hurry.

- Yahoo reportedly ran numerous clusters having 4000+ nodes with four 1 TB drives per node, 15 PB of total storage capacity, 70 million files, and 80 million blocks using 50 GB NameNode heap.
- Facebook's 2000-node warehouse cluster [2] is provisioned for 21 PB of total storage capacity. Extrapolating the announced growth rate, its namespace should have close to 200 million objects (files + blocks) by now, but an immense 108 GB heap should allow room for close to 400 million objects.
- eBay runs a 700-node cluster. Each node has 24 TB of local disk storage, 72 GB of RAM, and a 12-core CPU. Total cluster size is 16 PB. It is configured to run 26,000 MapReduce tasks simultaneously.

As observed in the past, the average size of HDFS files is decreasing. This trend is sustainable as the cluster grows and becomes available for a larger variety of applications. The phenomenon is characterized by the decreasing block-to-file ratio, which has dropped from 2 in 2008 to 1.5 in 2009 and to 1.1 today.

DataNode's local storage capacity is increasing as cheaper 2 TB drives fall under the category of commodity hardware. Combined with the growing number of cores per processor and larger RAM sizes, this leads to more compact but powerful clusters.

While the clusters become more compact, the network bandwidth becomes the limiting factor of the cluster performance. Typical network bandwidth between nodes on the same rack is 1 Gbps, which converts into a 119 MB/s data transfer rate. The read rate for a single disk drive usually exceeds 60 MB/s. This means that if one runs a read-intensive job, such as DFSIO-read, then in order to saturate the network capacity of a single node, it is enough to have only two tasks reading from two different drives on that node. In practice the I/O rate of a combination of random reads and writes is lower, and only a fraction of those I/Os results in actual data transfers. Based on observations on busy Hadoop clusters, the average data transfer rate per client is 10 MB/s. In this case 12 clients accessing data from 12 different drives of a single node will saturate the node's network capacity. Therefore, adding more drives will not increase the aggregate throughput of the cluster, which means that dense local storage is beneficial only for the cluster that stores a vast amount of rarely accessed data, as in data warehouses.

Decreasing file sizes and compaction of the clusters: both of these trends drive the demand for higher Hadoop scalability.

MapReduce: Scaling the Framework

The simplicity of the MapReduce [5] computational model combined with its power to incorporate and utilize distributed resources of commodity hardware is the second driving force of Hadoop's popularity.

A MapReduce job is a two-stage computation, with each stage defined by a plain Java (or C++, or Python) program—a task. The input data for each stage is distributed across the nodes of the cluster, and the same task is run against different blocks of the input data on the node containing the data block. The job produces distributed output. This type of computation minimizes data transfer by moving computation to data and not vice versa.

The Apache Hadoop MapReduce framework has reportedly reached its scalability limit at 40,000 clients simultaneously running on the cluster. This corresponds to a 4,000-node cluster with 10 MapReduce clients—slots, in Hadoop terminology—per node.

The implementation of the Hadoop MapReduce framework follows the same single master architecture as HDFS. The single master is called JobTracker. It shepherds the distributed herd of slaves called TaskTrackers. The JobTracker serves two primary functions:

1. Job scheduling and resource allocation
2. Job monitoring and job life-cycle coordination

The first function is fundamentally centralized, but the second one is not. Coordination of many jobs running on thousands of TaskTrackers makes the single JobTracker a constraining resource for the entire cluster.

There are ongoing efforts ([10], [7]) to improve scalability of the MapReduce engine by delegating the coordinating function to different cluster nodes for different jobs. That way, even if the JobTracker fails the jobs will continue to run, as their lifecycles are controlled by other nodes.

This also intends to address another weak point of today's implementation—the static partitioning of cluster resources. In current MapReduce architecture the cluster is divided into a fixed number of map and reduce slots, which are uniformly configured per node. Each slot can be used for tasks of the assigned type (map or reduce) only and therefore cannot be reallocated to another type if the demand for the latter increases. Static cluster configuration also does not take into account the amount of resources—RAM, CPU, disk space, network bandwidth—required for different tasks, which may lead to underutilized resource usage for some tasks and starvation for others.

HDFS: Static Partitioning

Hadoop deployments have reached the architectural limit. With HDFS clusters running at capacity, the only direction for growth is a horizontal increase in the number of clusters. The demand to support smaller files, and the evolutionary growth of storage devices and computational power of servers, which allows aggregating more resources per cubic foot, are the two major factors contributing to the demand for higher scalability in distributed storage systems.

Current HDFS architecture by design assumes that a single server, the NameNode, dedicated to maintaining the file system metadata consisting of files, directories, and blocks, controls the work of other cluster nodes, DataNodes, handling the actual data blocks of files. The system is designed to scale linearly on the number of DataNodes, as they can process data transfers independently of each other. However, the NameNode is a single source of metadata information.

The HDFS scalability is limited solely by NameNode resources [12]. In order to process metadata requests from thousands of clients efficiently, NameNode keeps the entire namespace in memory. *The amount of RAM allocated for the NameNode limits the size of the cluster.*

The number of active clients is proportional to the size of the cluster. As the cluster grows, the increasing number of clients provides higher load on the NameNode. Being a single point of entry, *the NameNode's efficiency limits the aggregate cluster performance.*

It is clear that further scaling of HDFS requires a scalable architecture for its namespace.

One approach, called Federation [11], is based on the idea that multiple independent namespaces can share a common pool of DataNodes as a block storage layer. The namespaces representing isolated file systems, called volumes, are maintained by dedicated NameNodes—one per volume—and evolve independently of each other. Each DataNode maintains blocks of multiple volumes and reports those blocks to corresponding NameNodes. The cluster then is defined as a family of volumes sharing the same pool of DataNodes.

In order to conceive the isolation of the file systems from clients, the volumes are "federated" under client-side mount tables. The client-side mount table is a virtual file system, called ViewFS, which provides a common view of the cluster for a group of clients unified by common cause. ViewFS in general is a sequence of symbolic links—fully qualified HDFS paths—that can be passed over via a job configuration file to all tasks of that job in order to supply them with a common view of the world.

A federated cluster can store more data and handle more clients, because it has multiple NameNodes. However, each individual NameNode is subject to the same limits and shortcomings, such as lack of High Availability (HA), as a non-federated one.

The federated approach provides a static partitioning of the federated namespace. If one volume grows faster than the other and the corresponding NameNode reaches the limit, its resources cannot be dynamically repartitioned among other NameNodes except by manually copying files between file systems.

The Distributed Namespace Challenge

On the next evolutionary step, HDFS should become a distributed highly available file system without a single point of failure, which can:

- Store hundreds of billions of objects
- Support millions of concurrent clients
- Maintain an exabyte (10^{18}) of total storage capacity

The main motivation for building such a system is the ability to *grow the namespace*. The current namespace limit is 100 million files. Static partitioning will scale the federated namespace to billions of files. Estimates show that implementation of a dynamically partitioned namespace will be able to support 100 billion objects.

Service continuation and availability is another strong motivation for the system. A big HDFS installation with a NameNode operating in a large JVM is vulnerable to frequent full garbage collections, which may take the NameNode out of service for several minutes. "Bad" clients, producing a high number of metadata operations, can saturate the NameNode, effectively making it unavailable for other tasks. And, finally, a failure of the NameNode makes the file system inaccessible for up to an hour—the time it takes to restart the NameNode.

Building a large system compared to maintaining a number of smaller ones simplifies its operability. Currently, the effort of operating clusters of 400 nodes is roughly the same as for clusters of 4000 nodes. The cost of maintaining different clusters is proportional to the number of clusters.

Building such a system from scratch can take years, as this is how long creating viable file systems takes. A simpler approach is to construct the system from existing reliable components. For HDFS, one needs to find a component able to maintain a distributed namespace. It turns out that the Hadoop family has just the one.

Apache HBase [14] organizes data into big, sparse, loosely structured tables. The elements of a table are rows, having unique row keys. An HBase table can have an arbitrary number of columns, grouped into a small predefined number of column families. The columns can be created dynamically, but not the column families. Tables are partitioned into regions—horizontally across rows and vertically across column families. Regions are stored as (compressed) files in HDFS. HBase runs on a cluster of machines called region servers. Each region server caches a number of regions and serves this data to HBase clients. The goal is to provide a structured yet flexible presentation of data for random, near real-time read and write access to very big tables, consisting of billions of rows and millions of columns. Apache HBase is an implementation of Google's BigTable [3].

HBase can be used to maintain the file system namespace, making it a scalable replacement for the HDFS's NameNode. With this approach, files and directories become rows of a very large HBase table representing the entire file system. The file blocks will still be stored on and served from DataNodes. This will preserve the decoupling of data and metadata—one of the key principles of HDFS architecture.

Google used this approach to build its next-generation GFS Colossus [4]. Prototypes such as VoldFS and CassFS [6] have been built based on the same principles but using a competing database, with HBase metadata stores Voldemort and Cassandra, respectively. Pomegranate [9] implements it own tabular storage utilizing the same idea.

Of the many design challenges facing such systems, probably the main two are:

- Namespace partitioning
- Atomic rename

Namespace partitioning is the problem of mapping the hierarchical file system tree to the flat table structure. A naïve approach is to simply hash file paths, with the hash value defining the partition the file belongs to. This approach lacks the important principle of locality, as a simple listing of a directory requires accessing multiple partitions, which may be located on different nodes. Another approach is a Ceph-like [16] partitioning into full subtrees. This provides good *locality of reference*, even somewhat too good, assuming that the degree of locality of files in a tree is proportional to the number of their common ancestors. Somewhere in between is an approach which partitions the tree into small fixed-height tiles. For example, for height 2 and for a given directory D the tile contains D itself, all its children, and all grandchildren. The tiles can then be arbitrarily combined into partitions. This is similar to the way reiserFS partitions the namespace into fixed-size blocks.

Renaming is tightly related to the partitioning problem, because when the row (file) keys are based on file paths, even a simple case of rename—that is, changing a file's local name—may move the file into a different partition. And a directory rename in that case can lead to a massive cross-partition relocation of files in the subtree. Therefore, row keys should be based on unique file IDs (inode numbers) rather than paths. This still leaves unsolved a more general case of rename, required by POSIX semantics: an atomic move of a file from one directory to another. The problem is hard, as it requires a consistent update of multiple partitions potentially distributed across the cluster. A solution involves use of PAXOS-like [8] consensus-building algorithms. A "lazy" approach is to sacrifice this functionality, relaxing the semantics of rename to support only the simple case (in-place rename) in favor of higher scalability. Applications relying on the atomicity of cross directory moves will have to implement it internally. In many cases this is easier than building a generic solution.

The Final Dimension

An open source platform like Apache Hadoop,usually provides a generic tool to do things. Given the tool, companies and organizations initially benefiting from the ability to quickly adopt the system for their business use cases then tend to continue investing in testing, performance tuning, and refining the tool for their production needs. Ideally, this drives innovation, and the platform evolves into a highly tunable and adaptive system with various controls to make it fit many practical use cases.

Flexibility of the system adds another axis to the Hadoop scalability universe. I would like to thank my numerous colleagues in the Apache Hadoop community for contributing to this multidimensional endeavor.

References

[1] Apache Hadoop. http://hadoop.apache.org/.

[2] D. Borthakur, "Facebook Has the World's Largest Hadoop Cluster!": http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html.

[3] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *Proceedings of the 7th Symposium on Operating System Design and Implementation*, November 2006.

[4] J. Dean, "Large-Scale Distributed Systems at Google: Current Systems and Future Directions," Keynote at Large-Scale Distributed Systems and Middleware, October 2009.

[5] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, December 2004.

[6] J. Darcy, VoldFS and CassFS: https://github.com/jdarcy/VoldFS, https://github.com/jdarcy/CassFS.

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," Proceedings of NSDI '11: 8th USENIX Symposium on Networked Systems Design and Implementation, March 2011.

[8] L. Lamport, "The Part-Time Parliament," *ACM Transactions on Computer Systems,* vol. 16, no. 2, May 1998, pp. 133–169.

[9] C. Ma, Pomegranate: https://github.com/macan/Pomegranate/wiki.

[10] A.C. Murthy, "The Next Generation of Apache Hadoop MapReduce," Yahoo! Developer Network Blog, February 14, 2011: http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen/.

[11] S. Radia, S. Srinivas, "Scaling HDFS Cluster Using Namenode Federation," HDFS-1052, August 2010: https://issues.apache.org/jira/secure/attachment/12453067/high-level-design.pdf.

[12] K.V. Shvachko, "HDFS Scalability: The Limits to Growth," *;login:*, vol. 35, no. 2, April 2010, pp. 6–16.

[13] K.V. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *Proceedings of Symposium on Mass Storage Systems and Technologies*, May 2010.

[14] Apache, *The Apache HBase Book*, October 2010: http://hbase.apache.org/book.html.

[15] Tsz-Wo Sze, "The Two Quadrillionth Bit of Pi Is 0! Distributed Computation of Pi with Apache Hadoop," arXiv:1008.3171v2 [cs.DC], August 2010.

[16] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, November 2006.