Data Availability and Durability with the Hadoop Distributed File System

ROBERT J. CHANSLER

Robert J. Chansler is Senior Manager of Hadoop Infrastructure at LinkedIn. This work draws on Rob's experience as manager of the HDFS development team at Yahoo!. A Caltech graduate, Rob earned a PhD in computer science at Carnegie Mellon University investigating distributed systems. After a detour through compilers, printing systems, electronic commerce, and network management, Rob returned to distributed systems, where many problems were still familiar but all the numbers had two or three more zeros.
rchansler@yahoo.com

The Hadoop Distributed File System at Yahoo! stores 40 petabytes of application data across 30,000 nodes. The most conventional strategy for data protection—just make a copy somewhere else—is not practical for such large data sets. To be a good custodian of this much data, HDFS must continuously manage the number of replicas for each block, test the integrity of blocks, balance the usage of resources as the hardware infrastructure changes, report status to administrators, and be on guard for the unexpected. Furthermore, the system administrators must ensure that thousands of hosts are operational, have network connectivity, and are executing the HDFS Name Node and Data Node applications. How well has this worked in practice? HDFS is (almost) always available and (almost) never loses data.

A Survey of HDFS Availability

The Grid Operations team tracks each cluster loss-of-service incident. Many of these incidents are "caused" by facilities other than the file system. But for some incidents, the immediate problem is that HDFS is unavailable or not performing well. Sometimes the initial incident report originates with a user frustrated that HDFS is not performing as well as expected. Automated tools are better at monitoring whether individual host machines and network switches are operational. Yahoo! uses Nagios for alerting. An internal tool called Simon is used for time series collection and presentation from Nagios; Ganglia is a popular tool used elsewhere for this purpose. HDFS itself provides several statistics streams available via JMX and collected by Simon. An introduction to using Nagios and Ganglia for cluster monitoring is available from IBM developerWorks (see Resources). The logs from cluster hosts—both the operating system logs and the HDFS application logs—can be used for fault diagnosis.

Table 1 summarizes all of the "grid down" events recorded during the 500 days preceding 5 April 2011, where loss of HDFS service is the initial or primary complaint. The table includes some data about the aggregate size of the clusters for perspective. This somewhat arbitrary interval was chosen to be long enough to collect an interesting data set, but not so long as to include much data from when HDFS was—to be frank—not so good. Roughly, this corresponds to the deployment Hadoop 0.20 versions.

The HDFS Name Node is a Java application. As such, there is always concern that garbage collection might interrupt service. Indeed, garbage collection (GC) represents 44% of the reported loss-of-service incidents. In a GC event, HDFS is

unavailable for a few (less than 10) minutes. The system resumes normal operation without intervention. The nature of the Java Virtual Machine is such that there can be no promise that GC will not interrupt service. Most GC events are close to the threshold of observability, about five minutes. During mid-2010, the biggest grids (about 4000 nodes) had short interruptions once or twice a week, although only one report was generated each month. Careful provisioning and some thoughtful software changes have reduced the frequency of interruptions, so that today big clusters have fewer than one event every other month. GC issues were nagging problems from time to time, but among HDFS developers the consensus is that HDFS has benefited from the choice to implement the system in Java. Development efficiency was higher, and inconvenient service interruptions that did occur were not the kind of faults that might threaten the integrity of the file system.

Table 2 summarizes each of the "Other HDFS Down Incidents"; HDFS was unavailable for an hour or two while the fault was repaired and the cluster restarted. HDFS ought to be excused from 16 of the events. There is not much the file system can do if LDAP, NFS, power, or a switch fails. Two of the events are attributed to software faults where code did the wrong thing. Nine events occurred when system storage resources were exhausted. Either the data nodes ran out of available space or the name system could not create additional names or blocks. User application behavior was implicated in five cases, where either the rate of requests was extraordinary or the required response to a request was exceptionally large. There is still no good protection from the former, but the latter problem is eliminated. There is no convincing explanation for the remaining four cases. The longest unexcused absence was 3.2 hours.

Clusters 19 April 2011	**GC**	**Other HDFS Down Incidents**	**Nodes**	**Blocks ($\times 10^6$)**	**New Files Per Day ($\times 10^6$)**
Research clusters	15	22	10,404	299.2	16.1
Production clusters	9	14	19,720	357.1	30.6
Totals	24	36	30,124	656.3	46.7

Table 1: Loss of HDFS service incidents for 500 days preceding 5 April 2011. Scheduled maintenance events are not included. Multiple Hadoop clusters have been aggregated into two classes: "research clusters," generally available to the engineering community, and "production clusters," restricted to approved jobs and serving more time-critical processes.

Four continuing initiatives have helped to reduce the number of incidents and the duration of incidents that do occur.

1. Operations now have better guidance on what the practical limits are and can better manage the clusters to avoid encountering the hard limits.
2. Better-provisioned servers have been tested with higher practical limits.
3. The time necessary to restart the name system has been reduced. This involved reducing the minimum necessary time to restart the name system, and better tools to validate the integrity of the system before resuming service.
4. The garbage collection parameters for the Java VM have been more carefully tuned.

Is HDFS getting better? There were seven unexcused absences in the last 100 days of this survey, but only two in the earliest 100 days. But HDFS has over twice as many nodes and clusters under management at the end of the survey as at the beginning, and the introduction of the Security feature means HDFS is more dependent on infrastructure like LDAP and Kerberos servers. Certainly garbage collection is much better, as explained previously. And if resource usage is carefully managed, the likelihood of a surprise is reduced by half.

HDFS can be satisfyingly resistant to multiple insults. In one recent incident, a journal storage volume was lost when the volume filled up under extraordinary client demand and a rack switch died. HDFS continued service and re-created the missing block replicas, although there was a user complaint that the file system was slow.

Down Incidents	Number	Details
Garbage collection	24	JVM "promotion failures" interrupted service.
Excused absence	16	Loss of power, hardware failure, misconfiguration, operation error, network faults, and the like; HDFS restarted.
Bugs	2	Software did wrong; system was restarted without software change.
Space/objects exhausted	9	Insufficient resources for users; sometimes system continued when resources were freed.
User applications	5	HDFS was unable to service demand effectively, but system recovered without intervention when excess demand was removed.
Unknown	4	No convincing explanation; HDFS restarted.

Table 2: Loss-of-service incidents categorized. The apparent cause of the incident was determined by inspection of logs and reports from the Operations team.

An initiative is underway to bring conventional "high availability" to the HDFS name system. The general idea is that a second host is provisioned and able to execute the name system application with only a modest delay (a few seconds) and without interrupting user applications. This would not remedy all of the incidents that have been described here. The loss of a critical infrastructure component would not be remediated, and some operational problems of the name system (resource exhaustion, say) would just be transferred to the other host with no benefit. Leaving aside the question of garbage collection events, 7 of the 16 excused absences would be remediated as would 1 of the 4 unattributed events. Whether to failover to the second host in case of a garbage collection event is a difficult policy question. The decision rests on whether the transfer really is a low-risk activity, and to what extent the transfer really is transparent to user applications. But if transfer in the case of garbage collection is a Good Thing, the number of incidents will not be reduced, but the duration of the interruption will be bounded by the transfer time.

The larger context for this discussion considers not just adventitious interruptions of the file system, but also the many other causes of service interruptions for a cluster. With our problem child as an example (a large cluster with an especially diverse user community), there were 13 reports of file system interruptions, but a total of 86 cluster service interruptions when scheduled maintenance activities are included along with surprises attributed to the job system and other facilities. Twelve of the maintenance windows were for deployment of Hadoop. If Hadoop—including HDFS—were engineered for continuous service during software updates, many of these interruptions could be eliminated. Also, some of the maintenance windows were scheduled to perform administrative functions (e.g., change VM size) that could have been performed with less interruption if a high availability solution had already been at hand.

A Survey of HDFS Data Durability

Data stored in HDFS may become unavailable either because it is not possible to access the bytes on disk that are the data, or else the bytes have inadvertently been changed so that the bytes accessed are not the data originally stored. Since data storage in HDFS is organized as files composed of a sequence of large blocks (256 megabytes is typical), missing data is manifest as "lost" blocks. As the typical file has one or two blocks, the loss of a block is the same as the loss of a file for present purposes.

The failure of a data server does not result in the loss of blocks, as each block has replicas at other nodes. The usual case is that there are two other replicas. But if the application program requested exactly one replica, then loss of the node hosting the single replica must necessarily result in the loss of the block. The failure of a rack switch does not result in the loss of blocks, as each block has a replica at some other rack. The core switches connecting racks are provisioned so that component failures do not isolate a slice of nodes across racks or among multiple racks. Since each node hosts about 50,000 block replicas, it's near certainty that the loss of a slice of the cluster—or multiple racks—will cause data to be unavailable. Fortunately, in the case of switch failures, the switch will be repaired and the lost block replicas will be available again.

When a node fails, new replicas are created for blocks where a replica was hosted on the failed node. All is well if the re-replication process keeps up with the occasional failure of a node. (One percent of nodes fail each month.) Of course, it might just happen that a number of nodes fail during a short interval. How likely is the loss of blocks due to *uncorrelated* failures of multiple nodes? A statistical model (Table 3, next page) suggests that it is improbable that any cluster has observed the loss of a block with three replicas due to uncorrelated failure of nodes. Nor is there any record of this having happened. There can be *correlated* failures of nodes. Experience suggests that several nodes of a cluster will not survive a power-on restart. In that case a few (~10) blocks with three replicas will be lost, consistent with the statistical model for simultaneous failures.

Probability of losing a block on a large (4000 nodes) cluster	
In the next 24 hours	5.7×10^{-7}
In the next 365 days	2.1×10^{-4}

Table 3: Probability model for predicting data loss due to uncorrelated failure of nodes. The details of the model are in the HDFS issue tracking system; a link can be found in the Resources section of this article.

The remaining opportunity for losing blocks is when HDFS just does the wrong thing. HDFS has certainly matured in the past couple of years, from when one bad circumstance might cause a block to be lost to where at least a couple of bad circumstances must occur before the loss of a block is threatened. Two aspects of the HDFS architecture are especially pertinent. First, the strategy of replicating blocks is critically dependent on the correctness of the replication monitor. Second, if a client application begins writing a file but later abandons it, the writer's lease for the file must be recovered. In the former case, the number of different circumstances that might be the proximate cause for needing a new replica is large, and so making the correct diagnosis as to which replica ought to be replaced is difficult logic. In the latter case, the intervening hour presents a long interval when various misfortunes can happen, again complicating the logic.

One type of catastrophic failure requires special mention. If the file system metadata were ever lost, all data in the entire cluster would be compromised. Briefly, the metadata is actively managed by two hosts, the data is written to multiple volumes on different hosts, the integrity is periodically tested, and snapshots are retained for long periods. There is no record of having lost any piece of system metadata in the past two years. If the name system server fails and cannot be immediately restarted, a new host can begin service using the saved metadata. The metadata is managed by a write-ahead log, and so name system server failure will not cause loss of even the most recent metadata records.

A Review of 2009

Table 4 summarizes the record of lost blocks for the clusters managed by the Grid Operations team during 2009. This data has the weakness that reporting is certainly incomplete. With that in mind, the total number of lost blocks in 2009 is likely to be less than 1.5 times the number reported here. To provide some perspective, the table indicates an estimate of the number of blocks managed by the clusters and the number of new files created each day (recall a typical file is one or two blocks).

Clusters October 2009	Blocks Reported Lost	Nodes	Blocks ($\times10^6$)	New Files Per day ($\times10^6$)
Research clusters	631	11,890	222	10.9
Production clusters	10	5830	107	5.7
Totals	641	17,720	329	16.6

Table 4: Reported loss-of-data events for 2009. In one curious case, an unexplained fault changed a host directory into a regular file. The greater number of nodes in Table 1 is due to the increased usage of Hadoop resources during 2010 and early 2011.

In one of the research clusters, for a short period (a few days), client applications were abandoning an extraordinary number of files. An error in recovering the lease for a small fraction (10^{-4}) resulted in losing the block being written. This fault accounted for 533 of the total number of lost blocks. On another cluster, a single user experimented with setting the replication factor for many files to one. He lost his gamble—and 91 blocks—when a node failed. On other clusters, two blocks were lost when a data node reported replicas as too big; four blocks were lost when nodes failed during a cluster cold start; four blocks were lost due to a faulty write pipeline recovery; an unreported (but tiny) number of blocks were lost when nodes returned to service with corrupt replicas. For 2009, maybe two dozen blocks were lost due to all other software faults.

A Review of 2010

It was not possible to repeat this analysis for 2010—well, not impossible, but considerably less convenient, because the apparent loss of blocks became dominated by a new operating procedure. There is never too much space, and when space is short, the third replica of a block looks to be an excess of caution. Blocks with two replicas are, in fact, pretty durable. Furthermore, in our practice it is not uncommon for a file in one cluster to already have a copy on another cluster. If each of those files has only two replicas at each of two clusters, space is saved while durability is enhanced. Therefore we moved to a practice whereby many files have replication set to two. Any two nodes are likely to each have a replica of some block. When many blocks have only two replicas, if any two nodes fail to start, some blocks (in proportion to the fraction of blocks with replication two) will be lost. In this context, it is important to consider two sources of *correlated* failures of data nodes.

A power-on restart of a big cluster begins by switching on each of the cluster hosts, and if the hosts successfully start the operating system, the administrator will issue commands to begin execution of the HDFS Data Node and Name Node applications. In practice, a few dozen nodes will not immediately restart. Also, consider a circumstance where a host can continue service but cannot restart the operating system or Data Node application. This might occur if the system disk volume has failed, for instance. In this case even a warm restart of the cluster (without power interruption) will cause the un-restartable nodes to fail together.

When many blocks have only two replicas, if several nodes fail to join the cluster after a cluster start the number of lost blocks will be in proportion to the number of pairs of failed nodes. The result may be thousands of missing blocks. It is probable that those blocks can be recovered by copying from another cluster or by persuading the failed nodes to join the cluster, but that effort requires a day—or more—of work by an administrator. We are investigating automated recovery! This process dominates reports of lost blocks; no one pays attention to other possible causes anymore. For over a year, no one has done a block-by-block analysis of missing blocks as was done in 2009. The new disk fail-in-place feature of HDFS helps to alleviate the problem of un-restartable nodes.

But what of those problems that were previously known causes of missing blocks? All have been fixed in Hadoop 0.20.204. Only one new cause of missing blocks has been discovered in 2010. A data node that is unable to reliably read data from other nodes might attempt to read each replica of a block and report each replica as corrupt. In this case, the block looks to be missing even though all existing replicas are

good. It is possible for an administrator to recover from such false reports. HDFS has adopted the policy that a report of a corrupt block is only accepted from a node that has read a good replica.

Know Your Neighbors

Can users do anything to improve the robustness and availability of their own data? There is a wide variance among clusters of incidence of service or data unavailability. The clusters that host a more experimental user community experience more untoward events. The "production" clusters with restricted access are more robust than the "research" clusters available to all engineers. Especially with the introduction of permissions two years ago, HDFS users have substantial protection from the carelessness of others. The most immediate threat to data availability comes from an application job that exhausts some cluster resource. The quota features of HDFS are partial protections. How to yet better manage system resources is an area of active investigation. That said, HDFS operated by skilled administrators is a reliable custodian of Yahoo!'s petabytes.

Resources

K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010), May 2010.

Konstantin V. Shvachko, "HDFS Scalability: The Limits to Growth," *;login:*, vol. 35, no. 2, April 2010: http://www.usenix.org/publications/login/2010-04/openpdfs/shvachko.pdf.

Konstantin V. Shvachko, "Apache Hadoop: The Scalability Update," *;login:*, vol. 36, no. 3, June 2011: http://www.usenix.org/publications/login/2011-06/openpdfs/Shvachko.pdf.

Using Ganglia and Nagios: http://www.ibm.com/developerworks/linux/library/l-ganglia-nagios-1/.

HDFS tracking system: "HDFS-2535: A Model for Data Durability": https://issues.apache.org/jira/browse/HDFS-2535.