Yahoo! Developer Network Blog

« Previous | Main

MAY 5, 2010

Scalability of the Hadoop Distributed File System

In his fictional story **"The Library of Babel"**, **Jorge Luis Borges** describes a vast storage universe composed of all possible manuscripts uniformly formatted as 410-page books. Most are random meaningless sequences of symbols. But the rest excitingly forms a complete and an indestructible knowledge system, which stores any text written in the past or to be written in the future, thus providing solutions to all problems in the world. Just find the right book.

The same characteristic fascinates us in modern storage growth: The aggregation of information directly leads to proportional growth of new knowledge discovered out of it. A skeptic may doubt that further reward in knowledge mining will not justify the effort in information aggregation. What if by building sophisticated storage systems we are chasing the 19th century's **horse manure problem**, when at the dawn of the automobile era the scientific world was preoccupied with the growth of the horse population that threatened to bury the streets of London nine feet deep in manure?

The historical judgment will turn one way or another. In the meantime, we want to know how far we can go with existing systems even if only out of pure curiosity. In my **USENIX ;login:** article **"HDFS Scalability: the Limits to Growth,"** I studied scalability and performance limitations imposed on a distributed file system by the single-node namespace server architecture. The study is based on experience with largest deployments of the **Hadoop Distributed File System (HDFS)** currently in production at Yahoo!.

The first part of the study focuses on how the single name-server architecture limits the number of namespace objects (files and blocks) and how this translates to the limitation of the physical storage capacity growth.

The second part explores the limits for linear performance growth of HDFS clusters bound by natural performance restrictions of a single namespace server. As the cluster grows, the linear increase in the demand for processing resources puts a higher workload on the single namespace server. When the workload exceeds a certain threshold, it saturates the server, turning it into a single-node bottleneck for linear scaling of the entire cluster.

In 2006, the Hadoop group at Yahoo! formulated long-term target **requirements** for HDFS and outlined a list of projects intended to bring the requirements to life. Table 1 summarizes the targets and compares them with the current achievements:

	Target	Deployed
Capacity	10 PB	14 PB
Nodes	10,000	4,000
Clients	100,000	15,000
Files	100,000,000	60,000,000

Table 1: Targets for HDFS vs. actual deployed values as of 2009

The bottom line is that we achieved the target in Petabytes and got close to the target in the number of files. But this is done with a smaller number of nodes and the need to support a workload close to 100,000 clients has not yet materialized. The question now is whether the goals are feasible with the current system architecture.

Namespace Limitations

HDFS is based on an architecture where the namespace is decoupled from the data. The namespace forms the file system metadata, which is maintained by a dedicated server called the *name-node*. The data itself resides on other servers called *data-nodes*.

The namespace consists of files and directories. Files are divided into large (128 MB) blocks. To provide data reliability, HDFS uses block replication. Each block by default is replicated to three data-nodes. Once the block is created its replication is maintained by the system automatically. The block copies are call *replicas*.

The name-node keeps the entire namespace in RAM. This architecture has a natural limiting factor: the memory size; that is, the number of namespace objects (files and blocks) the single namespace server can handle.

Estimates show that the name-node uses less than 200 bytes to store a single metadata object (a file inode or a block). According to statistics on Y! clusters, a file on average consists of 1.5 blocks. Which means that it takes 600 bytes (1 file object + 2 block objects) to store an average file in name-node's RAM

> To store 100 million files (referencing 200 million blocks), a name-node should have at least 60 GB of RAM.

Storage Capacity vs. Namespace Size

With 100 million files, each having an average of 1.5 blocks, we will have 200 million blocks in the file system. If the maximal block size is 128 MB and every block is replicated 3 times, then the total disk space required to store these blocks is about 60 PB.

> 100 million file namespace needs 60 PB of total storage capacity on the cluster

As a rule of thumb, the correlation between the representation of the metadata in RAM and physical storage space required to store data referenced by this namespace is:

> 1 GB metadata ≈ 1 PB physical storage

The rule should not be treated the same as, say, the **Pythagorean Theorem**, because the correlation depends on cluster parameters (the block to file ratio and the block size). But it can be used as a practical estimate for configuring cluster resources.

Cluster Size and Node Configuration

Next we can estimate the number of data-nodes the cluster should have to accommodate the namespace of a certain size. On Yahoo's clusters, data-nodes are usually equipped with four disk drives of size 0.75 – 1 TB, and configured to use 2.5 – 3.5 TB of that space per node. The remaining space is allocated for MapReduce transient data, system logs, and the OS.

If we assume that an average data-node capacity is 3 TB, then we will need on the order of 20,000 nodes to store 60 PB of data. To keep it consistent with the target requirement of 10,000 nodes, each data-node should be configured with eight 1TB hard drives.

> To accommodate data referenced by a 100 million file namespace, a HDFS cluster needs 10,000 nodes equipped with 8TB of total hard drive capacity per node

Note that these estimates are true under the assumption that the block per file ratio of 1.5 and the block size remain the same. If the ratio or the block size increases, a gigabyte of RAM will support more petabytes of physical storage and vice versa. Sadly, based on practical observations, the block to file ratio tends to decrease during the lifetime of a file system, meaning that the object count (and therefore the memory footprint) of a single namespace server grows faster than the physical data storage. That makes the object-count problem — which becomes file-count problem when the average file to block ratio is close to 1 — the real bottleneck for cluster scalability.

The Internal Load

Apart from the hierarchical namespace the name-node's metadata includes a list of registered data-nodes, and a block to data-node mapping, which determines physical block locations.

A data-node identifies block replicas in its possession to the name-node by sending block reports. The first block report is sent at the startup. It reveals the block locations, which otherwise are not known to the name-node at startup time. Subsequently, block reports are sent periodically every 1 hour by default and serve as a sanity check providing that the name-node has an up-to-date view of block replica distribution on the cluster.

During normal operation, data-nodes periodically send heartbeats to the name-node to indicate that the data-node is alive. The default heartbeat interval is 3 seconds. If the name-node does not receive a heartbeat from a data-node in 10 minutes, it pronounces the data-node dead and schedules its blocks for replication on other nodes.

The block reports and heartbeats form the internal load of the cluster. If the internal load is too high, the cluster becomes dysfunctional, able to process only a few, if any, external client operations such as ls, read, or write. The internal load depends on the number of data-nodes. Assuming that the cluster is built of 10,000 data-nodes having 8 hard drives with 6 TB of effective storage capacity each, we estimate that the name-node will need to handle

- 3 block reports per second, each reporting 60,000 replicas
- 10,000 heartbeats per second

Using the standard HDFS benchmark called NNThroughputBenchmark, we measure the actual name-node performance with respect to the two internal operations. Table 2 summarizes the results. Note that the block report throughput is measured in the number of blocks processed by the name-node per second.

	Throughput
Number of blocks processed in block reports per second	639,713
Number of heartbeats per second	300,000

Table 2: Block report and heartbeat throughput

The implication of these results is:

> The internal load for block reports and heartbeat processing on a 10,000 node HDFS cluster with the total storage capacity of 60 PB will consume 30% of the total name-node processing capacity.

The internal load is proportional to the number of nodes in the cluster and the average number of blocks on a node. Thus, if a node had only 30,000 blocks — half of the estimated amount — then the name-node will dedicate only 15% of its processing resources to the internal load. Vice versa, if the average number of blocks per node grows, then the internal load will grow proportionally. The latter means that the decrease in block to file ratio (more small files with the same file system size) increases the internal load and therefore negatively affects the performance of the system.

Reasonable Load Expectations

We have learned by now that the name-node can use 70% of its time to process external client requests. Even with a handful of clients one can saturate the name-node performance by letting the clients send requests to the name-node with very high frequency. The name-node most probably would become unresponsive, potentially sending the whole cluster into a tailspin because internal load requests do not have priority over regular client requests.

In practice, the extreme load bursts are uncommon. Regular Hadoop clusters run MapReduce jobs, and jobs perform conventional file reads or writes. To get or put data from or to HDFS, a client first accesses the name-node and receives block locations, and then directly talks to data-nodes to transfer file data. Thus the frequency of name-node requests is bound by the rate of data transfer from data-nodes.

Typically a map task of a MapReduce jobs reads one block of data. We estimate how much time it takes for a client to retrieve a block replica and, based on that, evaluate the expected read load on the name-node — namely, how many “get block location” requests the name-node should expect per second from 100,000 clients. We apply the same technique to evaluate the write load on the cluster.

Performance Limitations

The read and write throughputs have been **measured** by the DFSIO benchmark and are summarized in Table 3.

	Throughput
Average read throughput	66 MB/s
Average write throughput	40 MB/s

Table 3: HDFS read and write throughput

Another series of throughput results produced by NNThroughputBenchmark (Table 4) measures the number of open (the same as get block location) and create operations processed by the name-node per second:

	Throughput
Get block locations	126,119 ops/s
Create new block	5,600 ops/s

Table 4: Open and create throughput

The read throughput in Table 3 indicates that 100,000 readers are expected to produce 68,750 get-block-location requests to the name-node per second. And Table 4 confirms that the name-node is able to process that many requests even if only 70% of its processing capacity is dedicated to this task.

> A 10,000 node HDFS cluster with the internal load at 30% will be able to handle an expected read-only load produced by 100,000 HDFS clients.

The write performance is not as optimistic. According to Table 3, 100,000 writers are expected to provide an average load of 41,667 create block requests per second on the name-node. This is way above 3,920 creates per second — 70% of possible processing capacity of the name-node (see Table 4).

> A reasonably expected write-only load produced by 100,000 HDFS clients on a 10,000 node HDFS cluster will exceed the throughput capacity of a single name-node.

We have seen that a 10,000-node HDFS cluster with single name-node is expected to handle a workload of 100,000 readers well. However, even 10,000 writers can produce enough workload to saturate the name-node, making it a bottleneck for linear scaling.

Such a large difference in performance is attributed to get-block-locations (read workload) being a memory-only operation, while creates (write workload) requires journaling, which is bounded by the local hard drive performance.

There are ways to improve the single name-node performance. But any solution intended for single namespace server optimization lacks scalability.

Looking into the future — especially taking into account that the ratio of small files tend to grow — the most promising solutions seem to be based on distributing the namespace server itself both for workload balancing and for reducing the single server memory footprint.

I hope you will benefit from the information provided above. Please refer to the **USENIX ;login:** article for more details.

Konstantin V. Shvachko
Principal software engineer, HDFS Team at Yahoo!

Posted at May 5, 2010 12:40 PM