Data Integrity & Caching

Let's explore how HDFS ensures data integrity and implements caching.

We'll cover the following

Data integrity#

Data Integrity refers to ensuring the correctness of the data. When a client retrieves a block from a DataNode, the data may arrive corrupted. This corruption can occur because of faults in the storage device, network, or the software itself. HDFS client uses checksum to verify the file contents. When a client stores a file in HDFS, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace. When a client retrieves file contents, it verifies that the data it received from each DataNode matches the checksum stored in the associated checksum file. If not, then the client can opt to retrieve that block from another replica.

Block scanner#

A block scanner process periodically runs on each DataNode to scan blocks stored on that DataNode and verify that the stored checksums match the block data. Additionally, when a client reads a complete block match the block data. Additionally, when a client reads a complete block and checksum verification succeeds, it informs the DataNode. The

DataNode treats it as a verification of the replica. Whenever a client or a block scanner detects a corrupt block, it notifies the NameNode. The NameNode marks the replica as corrupt and initiates the process to create a new good replica of the block.

Caching#

Normally, blocks are read from the disk, but for frequently accessed files, blocks may be explicitly cached in the DataNode's memory, in an off-heap block cache. HDFS offers a Centralized Cache Management scheme to allow its users to specify paths to be cached. Clients can tell the NameNode which files to cache. NameNode communicates with the DataNodes that have the desired blocks on disk and instructs them to cache the blocks in off-heap caches.

Centralized cache management in HDFS has many significant advantages:

1. Explicitly specifying blocks for caching prevents the eviction of frequently accessed data from memory. This is particularly important as most of the HDFS workloads are bigger than the main memory of the DataNode.
2. Because the NameNode manages DataNode caches, applications can query the set of cached block locations when making MapReduce task placement decisions. Co-locating a task with a cached block replica improves read performance.
3. When a DataNode has cached a block, clients can use a new, more efficient, zero-copy read API. As the block is already in memory and its checksum verification has already been done by the DataNode, clients can incur essentially zero overhead when using this new API.
4. Centralized caching can improve overall cluster memory utilization. When relying on the OS buffer cache at each DataNode, repeated reads of a block will result in all 'n' replicas of the block being pulled

into the buffer cache. With centralized cache management, a user

can explicitly specify only ‘m’ of the ‘n’ replicas, saving ‘n-m’ memory.

← Back

Anatomy of a Write Operation

Next →

Fault Tolerance

✓ Mark as Completed

Report an Issue