|  |  |
| --- | --- |
| 1.**CheckSum**: | |
|  | |  |
|  | A checksum is an error-detection method in a the transmitter computes a numerical value according to the number of set or unset bits in | |
|  | a message and sends it along with each message frame. At the receiver end, the same checksum function (formula) is applied to the message | |
|  | frame to retrieve the numerical value. If the received checksum value atches the sent value, the transmission is considered to be successful | |
|  | and error-free. | |
|  |  | |
|  | LocalFileSystem uses ChecksumFileSystem to do its work, and this class makes it easy to add checksumming to other (nonchecksummed) | |
|  | filesystems, as ChecksumFileSystem is just a wrapper around FileSystem. The general idiom is as follows: | |
|  | FileSystem rawFs = ... | |
|  | FileSystem checksummedFs = new ChecksumFileSystem(rawFs); | |
|  |  | |
|  | The underlying filesystem is called the raw filesystem, and may be retrieved using the getRawFileSystem() method on ChecksumFileSystem. | |
|  | ChecksumFileSystem has a few more useful methods for working with checksums, such as getChecksumFile() for getting the path of a checksum file | |
|  | for any file. | |
|  |  | |
|  | How CheckSum Works: | |
|  | 1.) The HDFS client software implements checksum checker. When a client creates an HDFS file, it computes a checksum of each block of the | |
|  | file and stores these checksums in a separate hidden file in the same HDFS namespace. | |
|  | 2.) 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. | |
|  | 3.) If not, then the client can opt to retrieve that block from another DataNode that has a replica of that block. | |
|  | 4.) If checksum of another Data node block matches with checksum of hidden file, system will serve these data blocks. | |
|  |  | |
|  | **2. Anatomy of File Write to HDFS:** | |
|  | The important components involved in a file write; | |
|  | \* Data Queue: When client writes data, DFSOS splits into packets and writes into this internal queue. | |
|  | \* DataStreamer: The data queue is consumed by this component, which also communicates with name node for block allocation. | |
|  | \* Ack Queue: Packets consumed by DataStreamer are temporarily saved in an this internal queue | |
|  |  | |
|  | a.) DistributedFileSystem object do a RPC call to namenode to create a new file in filesystem namespace with no blocks associated to it | |
|  | b.) Namenode process performs various checks like 1) client has required permissions to create a file or not 2) file should not exists earlier. | |
|  | In case of above exceptions it will throw an IOexception to client. | |
|  | c.) Once the file is registered with the namenode then client will get an object i.e. FSDataOutputStream which in turns embed DFSOutputStream | |
|  | object for the client to start writing data to.DFSoutputStream handles communication with the datanodes and namenode. | |
|  | d.) As client writes data DFSOutputStream split it into packets and write it to its internal queue i.e.data queue and also maintains | |
|  | an acknowledgement queue. | |
|  | e.) Data queue is then consumed by a Data Streamer process which is responsible for asking namenode to allocate new blocks by picking | |
|  | a list of suitable datanodes to store the replicas. | |
|  | f.) The list of datanodes forms a pipeline and assuming a replication factor of three, so there will be three nodes in the pipeline. | |
|  | g.) The data streamer streams the packets to the first datanode in the pipeline, which then stores the packet and forward it to second | |
|  | datanode in the pipeline. Similarly the second node stores the packet and forward it to next datanode or last datanode in the pipeline | |
|  | h.) Once each datanode in the pipeline acknowledge the packet the packet is removed from the acknowledgement queue. | |
|  |  | |
|  | **3. Handling of Failures during File Write in HDFS:** | |
|  |  | |
|  | When One of the machines fails during FileWrite i.e. When part of the pipeline which has datanode process running fails, Hadoop has inbuilt | |
|  | functionality to handle this scenario (hdfs is fault tolerant). If a datanode fails while data is being written to it, then the following | |
|  | actions are taken, which are transparent to the client writing the data : | |
|  |  | |
|  | a.) First, the pipeline is closed, and any packets in the ack queue are added to the front of the data queue so that datanode that | |
|  | are downstream from the failed node will not miss any packets. | |
|  | b.) The current block on the good datanode is given a new identity, which is communicated to the namenode, so that the partial | |
|  | block on the failed datanode will be deleted if the failed datanode recovers later on. | |
|  | c.) The failed datanode is removed from the pipeline, and the remainder of the block’s data is written to the two good datanodes in | |
|  | the pipeline. | |
|  | d.) The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node. | |
|  | Subsequent blocks are then treated as normal. | |