|  |  |
| --- | --- |
| **하둡이란?**  하둡은 아파치 프로젝트다. 오픈 라이선스 소스로 모든 것이 구성되어 있다.  하둡은 맵리듀스(MapReduce) 패러다임을 활용하여 파일 분산 처리 시스템을 제공한다.   |  | | --- | | MapReduce란? |   **HDFS?**  하둡의 클러스터는 computation capacity, storage capacity, IO bandwidth를 scale(확대?)한다.  그리고 이 하둡의 파일 시스템을 구성하는 것이 바로 **HDFS**다. |
| **HDFS**  HDFS는 file system metadata와 application data를 분리해서 저장한다.   |  | | --- | | file system metadata?  application data? |   HDFS에서 파일 컨텐츠들은 DataNodes들에게 여러 개로 복제된다.  이는 안정성을 높여줄 뿐만 아니라 bandwidth를 크게 증가시킬 수 있다.   |  | | --- | | Bandwidth? |  |  | | --- | | namespace란?  파일과 디렉토리의 계층 구조 | |
| 아키텍쳐(Architecture) |
| **1. Namenode**  HDFS는 모든 namespace를 RAM에 저장한다.  **네임노드의 기능.**  파일과 디렉토리는 NameNode에 inode로 표시된다. 여기서 inode란 permission, modification, access time, namespace, and disk space quotas와 같은 특성들을 기록하는 것을 말한다.  파일은 여러 개의 큰 블락들(보통 128메가짜리로)로 나뉘게되고 이 블락들은 복제되어서 여러개의 DataNode들에게 복사된다.  Namenode는 journal(the modification log of the image)를 저장한다.  다른서버에 journal과 checkpoint는 여러 개로 복제되어 저장된다.  다시시작될 때 namenode는 namespace를 읽고 journal을 다시 실행시키면서 복원된다.  **데이터를 읽을 때**  먼저 NameNode에게 데이터 블락의 위치를 물어본다. 그리고 client에게 가장 가깝게 있는 Datanode에서 그 블락을 가져와 읽는다.  **데이터를 쓸 때**  NameNode는 3개의 DataNode에게 블락 복제물들의 묶음을 관리하도록 임명한다.  그리고 이를 pipeline 형태로 DataNode에게 입력한다.  현재로써는 하나의 cluster당 하나의 NameNode를 가지고 있도록 하고 있다.   |  | | --- | | image – inode data와 각 파일에 속한 block들의 리스트로 구성된 metadata of the name system  checkpoint – the persistent record of the image stored in the local host's native files system. | |
| **2. DataNodes**  **DataNode의 기능**  각각의 블락 복제물들은 두개의 파일로 나타난다.  하나는 데이터를 가지고있고, 다른 하나는 블락의 메타데이터를 가지고 있다.  그리고 블락은 각각의 DataNode들에게 저장된다.  namespace ID -> 만들어질때 파일시스템에 지정됨.  ->클러스터의 모든 노드들에게 저장된다.  **Handshake**  During startup, 데이터노드는 네임노드와 handsahke라는걸 한다.  이는 namespace ID와 소프트웨어 버전을 검증하는 과정이다.   * storage ID -> an internal identifier of the datanode   **Datanode가 하는일**  **1. block report**  Datanode는 자신에게 있는 블락 복제물들을 block report를 통해 Name node에게 보고한다.  ->block report contains the block id, the generation stamp, and the length for each block replica the server hosts.  **2. Heartbeat**  heartbeat -> carry information about total storage capacity, fraction of storage in use, and the number of data transfers currently in progress.  시스템이 정상적으로 작동될 때 datanode들은 heartbeat을 namenode에게 보낸다. 만약 10분안에 못받게 되면 name node는 datanode가 고장나/ㅆ다고 생각하고 거기에 있었던 block replica를 다른 datanode에게 보낸다.  **Namenode가 Datanode를 다루는 방식**  Namenode들은 Datanode를 직접적으로 부리지 않는다. Datanode가 보낸 heartbeat에 답을 하면서 여러가지 지시사항을 전달하는데, 그 지시사항은 다음의 명령을 포함한다.  1. replicate blocks to other nodes  2. remove local block replicas  3. re-register or to shut down the node  4. send an immediate block report  **이렇게 하면서 namenode는 다른 namenode에게 영향을 주지 않으면서 초마다 수천개의 heartbeat를 처리할 수 있다.** |
| **3. HDFS client**  -> HDFS client, a code library that exports the HDFS file system interface    **클라이언트의 기능**  **When Read**  Client가 네임노드에게 블락의 복제물을 가지고 있는 데이터노드들의 리스트를 받는다. 그리고이는 데이터노드에게 원하는 블락을 달라고 한다.  **When Write**  Client가 무언가 내용을 작성하고자 할때는 네임노드에게 블락을 host할 데이터노드를 고르라고 한다.  **organize Pipeline**  클라이언트는 organizes a pipeline from node-to-node and sends the data.  When the first first block is filled, the client requests new DataNodes to be chosen to host replicas of the next block.  그 다음에 a new pipeline is organized and the client sends the further bytes of the file.   |  | | --- | | pipeline이 머지?? | |
| **Image and Journal**  **Image**  Image is the file system metadata that describes the organization of application data as directories and files  **Checkpoint**  checkpoint is a persistent record of the image written to disk   * the checkpoint file is never changed by the NameNode; it is replaced in its entirety when a new checkpoint is created during restart, when requested by the administrator, or by the CheckpointNode described in the next section.   **Journal**  Journal is a write-ahead commit log for changes to the file system that must be persistent   * for each client-initiated transaction, the change is recorded in the journal   the journal file is flushed and syhcned before the change is committed to the HDFS client  **이들의 중요성**  If either the checkpoint or the journal is missing, or becomes corrupt, the namespace information will be lost partly or entirely. |
| **CheckpointNode**  Namenode can alternatively execute either of two other roles, either a CheckpointNode or a Backupnode. The role is specified at the node startup.  **CheckpointNode란?**  CheckpointNode periodically combines the existing checkpoint and journal to create a new checkpoint and an empty journal   * It downloads the current check-point and journal files from the NameNode, merges them locally, and returns the new checkpoint back to the NameNode.   **왜 하냐고?**  Creating a checkpoint lets the NameNode truncate the tail of the journal when the new checkpoint is uploaded to the NameNode. -> (journal이 너무 길면 다시 킬 때 일주일이 걸릴 수도 있음) |
| **BackupNode**  Backupnode is capable of creating periodic checkpoints but in addition it maintains as in-memory, up-to-date image of the file system namespace that is always synchronized with the state of the NameNode.  **Job:**  the BackupNode accepts the journal stream of namespace transactions from the active NameNode, saves them to its own storage directories, and applies these transactions to its own namespace image in memory.  The NameNode treats the BackupNode as a journal store the same as it treats journal files in its storage directories.  Backupnode can be viewed as a read-only NameNode. It contains all file system metadata information except for block locations. |
| **Snapshopts**  The snapshot mechanism lets administrators persistently save the current state of the file system, so that if the upgrade results in data loss or corruption it is possible to rollback the upgrade and return HDFS to the namespace and storage state as they were at the time of the snapshot. |

|  |
| --- |
| **Block Placement**  For a large cluster, it may not be practical to connect all nodes in a flat topology.  A common practice is to spread the nodes across multiple racks  Nods of a rack share a switch, and rack switches are connected by one or more core switches. Communication between two nodes in different racks has to go through multiple switches    HDFS estimates the network bandwidth between two nodes by their distance. a shorter distance between two nodes means that the greater bandwidth they can utilize to transfer data.  **Default HDFS replica placement policy**  1. No Datanode contains more than on replica of any block  2. No rack contains more than two replicas of the same block, provided there are sufficient racks on the cluster |
| **Replication management**  The NameNode endeavors to ensure that each block always has the intended number of replicas.  When a block becomes over replicated, the NameNode chosses a replica to remove  When a block becomes under-replicated, it is put in the replication priority queue.  The NameNode also makes sure that not all replicas of block are located on one rack. |
| **Balancer**  HDFS block placement strategy does not take into account DataNode disk space utilization. The balancer is a tool that balances disk space usage on an HDFS cluster.   * data vailability. * optimizes the balancing process by minimizing the inter-rack data copying. |
| **Block Scanner**  Each DataNode runs a block scanner that periodically scans its block replicas and verifies that stored checksums match the blockdata. |
| **Decommissioning.**  The cluster administrator specifies which nodes can join the cluster by listing the host addresses of nodes that are permitted to register and the host addresses of nodes that are not permitted to register.  A present member of the cluster that becomes excluded is marked for decommissioning. Once a DataNode is marked as decommissioning, it will not be selected as the target of replica placement, but it will continue to serve read requests. |