I. Introduction and Related Work HDFS is a core component of Apache Hadoop, enabling the partitioning of data and computation across numerous hosts. This allows clusters to scale storage capacity, computation power, and I/O bandwidth by adding more commodity servers. HDFS separates metadata storage (on a dedicated **NameNode**) from application data storage (on **DataNodes**), similar to GFS, PVFS, and Lustre. Unlike systems using RAID, HDFS ensures data durability by replicating file content across multiple DataNodes (typically three), which also enhances data transfer bandwidth and opportunities for computation locality.

II. Architecture

- **A. NameNode:**
    - Manages the file system namespace (a hierarchy of files and directories represented by inodes).
    - Splits files into large blocks (e.g., 128MB, configurable per file) and maps these blocks to DataNodes where replicas are stored.
    - Maintains the entire namespace metadata (the "image") in RAM for fast access.
    - Persists the image as a "checkpoint" on local disk and records all modifications in a write-ahead "journal." For durability, checkpoint and journal can be stored redundantly.
    - On startup, it restores the namespace by loading the latest checkpoint and replaying the journal.
- **B. DataNodes:**
    - Store and retrieve data blocks on the local file systems of the machines they run on, as instructed by clients or the NameNode. Each block replica has a data file and a metadata file (checksums, generation stamp).
    - At startup, DataNodes perform a handshake with the NameNode to verify namespace ID and software version.
    - They register with the NameNode using a unique, persistent storage ID.
    - Periodically (e.g., hourly, and after registration) send **Block Reports** to the NameNode, listing all block replicas they host.
    - Send **Heartbeats** to the NameNode every few seconds to indicate liveness. If heartbeats are missed for a defined period (e.g., 10 minutes), the DataNode is considered dead, and its replicas are re-replicated. Heartbeats also carry storage usage statistics.
    - The NameNode uses heartbeat replies to send instructions (commands) to DataNodes (e.g., replicate/delete blocks, re-register, shut down).
- **C. HDFS Client:**
    - A library used by applications to interact with HDFS (read/write/delete files, create/delete directories).
    - For reads, the client gets block locations from the NameNode and reads data directly from the closest DataNode.
    - For writes, the client asks the NameNode to nominate DataNodes for replicas. The client then writes data to these DataNodes in a **pipeline** fashion.
    - HDFS provides an API exposing block locations, allowing frameworks like MapReduce to schedule tasks near the data.
- **D. Image and Journal:**

    - The NameNode's image (namespace metadata) and journal (transaction log) are critical.
    - Changes are first logged in the journal (flushed and synced) before being acknowledged to the client.
    - Checkpoints are periodically created to replace the old image file and allow journal truncation.
    - To optimize performance, the NameNode batches multiple client transactions into a single flush-and-sync operation for the journal.

- **E. CheckpointNode:**

    - A node (can be the NameNode process in a special role or a separate host) that periodically downloads the current checkpoint and journal from the active NameNode, merges them locally, and uploads the new checkpoint back. This protects metadata and helps manage journal size.

- **F. BackupNode:**

    - Creates periodic checkpoints (like a CheckpointNode) and *also* maintains an in-memory, up-to-date copy of the namespace image, continuously synchronized with the active NameNode by receiving a stream of transactions.
    - Can create checkpoints more efficiently and can serve as a read-only NameNode. Provides an option for the active NameNode to run without its own persistent storage, delegating that to the BackupNode.

- **G. Upgrades, File System Snapshots:**

    - HDFS supports snapshots to protect data during software upgrades. A snapshot saves the current FS namespace and block state.
    - DataNodes create local snapshots using hard links and copy-on-write for modifications, preserving old block replicas.
    - The system can be rolled back to a snapshot. A "layout version" tracks data format changes, with automatic conversion during upgrades (requiring a snapshot).

III. File I/O Operations and Replica Management

- **A. File Read and Write:**

    - HDFS uses a single-writer, multiple-reader model. Files are write-once (or append-only).
    - A client writing to a file is granted an exclusive **lease**, periodically renewed.
    - Data is written in a pipeline to the chosen DataNodes for replicas.
    - The `hflush` operation guarantees data written so far is visible to readers.
    - HDFS uses **checksums** to verify data integrity during reads; clients detect corruption, report it to the NameNode, and fetch a different replica.
    - Optimized for high-throughput sequential reads/writes common in batch processing.

- **B. Block Placement:**

    - HDFS is rack-aware. It estimates network bandwidth by node distance (nodes on same rack are closer than nodes on different racks).
    - The default replica placement policy aims to balance write cost, reliability, availability, and read bandwidth. For 3 replicas:
        1. First replica on the writer's node (if in cluster).

    2. Second replica on a different node in a *different* rack.
    3. Third replica on a different node in the *same rack* as the second. (The paper's description slightly differs: "second and the third replicas on two different nodes in a different rack". However, common HDFS policy is one on local rack, two on remote rack but on different nodes).
  - The general rules are: no DataNode contains more than one replica of a block, and no rack contains more than two replicas (if enough racks exist).

- **C. Replication Management:**
  - The NameNode ensures each block maintains its desired replication factor.
  - If under-replicated, blocks are queued for re-replication with priority.
  - If over-replicated, surplus replicas are removed, prioritizing keeping replicas on different racks and removing from DataNodes with less free space.
  - Ensures replicas of a block are not all on one rack.

- **D. Balancer:**
  - A tool to balance disk space usage across DataNodes, as default block placement doesn't consider DataNode utilization.
  - Iteratively moves replicas from more utilized to less utilized DataNodes, while maintaining replica count and rack diversity.

- **E. Block Scanner:**
  - Each DataNode periodically scans its block replicas and verifies checksums to detect corruption ("bit rot"). Corrupt blocks are reported to the NameNode for re-replication.

- **F. Decommissioning:**
  - Administrators can gracefully remove DataNodes from a cluster. Blocks on decommissioning nodes are replicated to other nodes before the node is fully decommissioned.

- **G. Inter-Cluster Data Copy:**
  - The `DistCp` tool (a MapReduce job) facilitates large-scale parallel data copying between HDFS clusters.