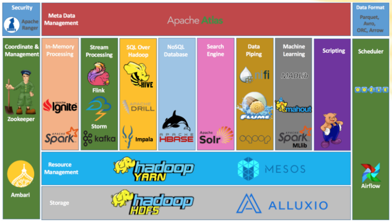
UMD DATA605 - Big Data Systems

8.3: Apache Hadoop

- **Instructor**: Dr. GP Saggese - gsaggese@umd.edu
- **References**
  - Ghemawat et al.: *The Google File System*, 2003
  - Dean et al.: *MapReduce: Simplified Data Processing on Large Clusters*, 2004

Hadoop Ecosystem (aka Hadoop Zoo)

- **Hadoop Map-Reduce**
- **HDFS**
  - Distributed file system
- **Pig**
  - High-level data-flow framework for parallel computation



- **HBase**
  - Scalable, distributed database
  - Structured data storage for large tables (like Google BigTable)
- **Cassandra**
  - Scalable multi-master database with no single points of failure
- **Hive**
  - Data warehouse infrastructure
  - Provide data summarization and ad-hoc querying
- **ZooKeeper**
  - High-performance coordination service for distributed applications
- **YARN, Kafka, Storm, Spark, Solr, ...**

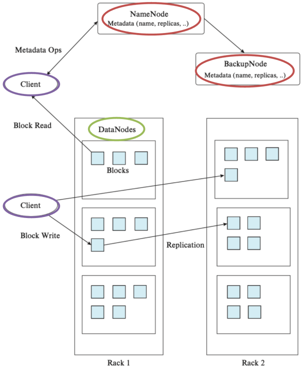
Hadoop Distributed File System (HDFS)

- **HDFS** is a **distributed file system**
  - Designed to store large data sets reliably
  - Part of the Apache Hadoop ecosystem
  - Inspired by the Google File System (GFS)

1. Optimized for **high-throughput access** to large files
   - Suitable for batch processing
   - Not low-latency access
2. Designed for **fault tolerance and scalability**
   - Ensures fault tolerance through replication
     - Blocks are stored on different nodes and racks
     - Provides data availability even if some nodes fail
   - Follows a primary-secondary architecture
   - Replication strategy improves read performance

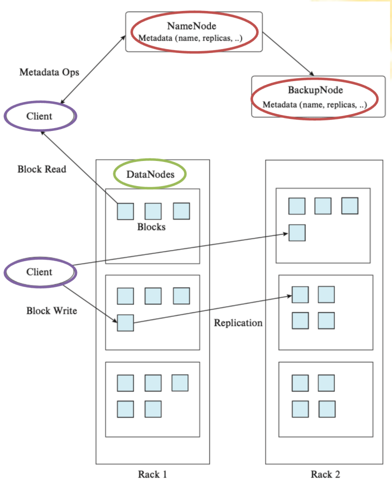
HDFS Architecture

- **NameNode**
  - Store file/dir hierarchy
  - Store file metadata
    - E.g., block location, size, permissions
- **DataNodes**
  - Store actual data blocks
  - Split file into 16-256MB blocks
  - Replicate chunks (2x or 3x) across multiple *DataNodes*
  - Keep replicas in different racks
- **Client**
  - API (e.g., Python, Java) to library
  - Mount HDFS on local filesystem



HDFS: Read / Write Protocols

- **Read**
  - Contact *NameNode* for *DataNode* and block pointer
  - Choose the nearest *DataNode* for each block
  - Connect to *DataNode* for data access
  - Reads blocks in parallel to improve performance
  - Data is reassembled by the client in correct order
- **Write**
  - *NameNode* creates blocks
  - Assign blocks to multiple *DataNodes*
  - Client sends data to *DataNodes*
  - *DataNodes* store data
  - Blocks are pipelined to other replicas
  - Write is considered successful after all replicas acknowledge



SCIENCE ACADEMY

Fault Tolerance and Recovery

- *NameNode* monitors *DataNode* heartbeat signals
  - On failure, blocks are re-replicated to maintain replication factor
- *NameNode* itself is a single point of failure
  - Solved with HDFS High Availability
- Data integrity ensured using checksums

HDFS vs Traditional File Systems

- Best for **storing and processing large-scale files**
  - E.g., logs, media, sensor data
  - Commonly used in data lakes and ETL pipelines
  - Supports very large files and directories
  - Performance degrades with many small files
- Optimized for **write-once, read-many** access pattern
- Lacks low-latency access, but provides **high throughput**
  - Good for analytics (OLAP)
  - Not suitable for transactional systems (OLTP)
    - E.g., bank

MapReduce: Hadoop

- **Hadoop**: open-source MapReduce implementation

- **Functionalities**
  - Partition input data (HDFS)
  - Input adapters
    - E.g., HBase, MongoDB, Cassandra, Amazon Dynamo
  - Schedule program execution across machines
  - Handle machine failures
  - Manage inter-machine communication
  - Perform *GroupByKey* step
  - Output adapters
    - E.g., Avro, ORC, Parquet
  - Schedule multiple *MapReduce* jobs