Module - III

BIG DATA

Lecture - 1

Introduction

BIG DATA - Introduction

- Outline
  - Motivation
  - Querying Big Data
  - Big Data Storage Systems
    - Distributed file systems
    - Shardring across multiple databases
    - Key-value storage systems
    - Parallel and distributed databases
  - Replication and Consistency

Motivation

- Very large volumes of data being collected
  - Driven by growth of web, social media, and more recently internet-of-things
  - Web logs were an early source of data
    - Analytics on web logs has great value for advertisements, web site structuring, what posts to show to a user, etc
- Big Data: differentiated from data handled by earlier generation databases
  - **Volume**: much larger amounts of data stored
  - **Velocity**: much higher rates of insertions
  - **Variety**: many types of data, beyond relational data

Querying Big Data

- Transaction processing systems that need very high scalability
  - Many applications willing to sacrifice ACID (Atomicity, Consistency, Isolation and Durability) properties and other database features, if they can get very high scalability
- Query processing systems that
  - Need very high scalability, and
  - Need to support non-relation data

Big Data Storage Systems

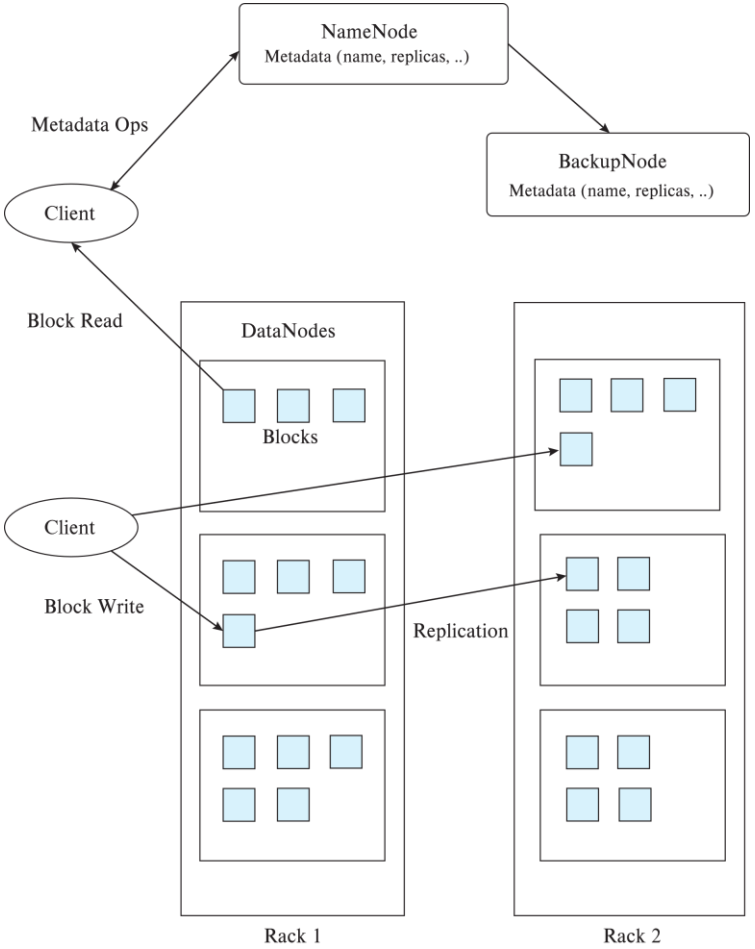
- Distributed file systems
- Sharding across multiple databases (Sharding is a method for distributing a single dataset across multiple databases, which can then be stored on multiple machines)
- Key-value storage systems
- Parallel and distributed databases

Distributed File Systems

- A distributed file system stores data across a large collection of machines, but provides single file-system view
- Highly scalable distributed file system for large data-intensive applications.
  - E.g., 10K nodes, 100 million files, 10 PB (Petabyte)
- Provides redundant storage of massive amounts of data on cheap and unreliable computers
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them
- Examples:
  - Google File System (GFS)
  - Hadoop File System (HDFS)

Hadoop File System Architecture

- Single Namespace for entire cluster
- Files are broken up into blocks
  - Typically 64 MB block size
  - Each block replicated on multiple DataNodes
- Client
  - Finds location of blocks from NameNode
  - Accesses data directly from DataNode



Hadoop Distributed File System (HDFS)

- **NameNode**
  - Maps a filename to list of Block IDs
  - Maps each Block ID to DataNodes containing a replica of the block
- **DataNode**: Maps a Block ID to a physical location on disk
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Distributed file systems good for millions of large files
  - But have very high overheads and poor performance with billions of smaller tuples

Sharding

- **Sharding**: partition data across multiple databases
- Partitioning usually done on some ***partitioning attributes*** (also known as ***partitioning keys*** or ***shard keys*** e.g. user ID
  - E.g., records with key values from 1 to 100,000 on database 1, records with key values from 100,001 to 200,000 on database 2, etc.
- Application must track which records are on which database and send queries/updates to that database
- Positives: scales well, easy to implement

Sharding

- Drawbacks:
  - Not transparent: application has to deal with routing of queries, queries that span multiple databases
  - When a database is overloaded, moving part of its load out is not easy
  - Chance of failure more with more databases
    - need to keep replicas to ensure availability, which is more work for application

Parallel Databases and Data Stores

- Supporting scalable data access
  - Approach 1: memcache or other caching mechanisms at application servers, to reduce database access
    - Limited in scalability
  - Approach 2: Partition ("shard") data across multiple separate database servers
  - Approach 3: Use existing parallel databases
    - Historically: parallel databases that can scale to large number of machines were designed for decision support not OLTP
  - Approach 4: Massively Parallel Key-Value Data Store
    - Partitioning, high availability etc. completely transparent to application

Key Value Storage Systems

- Key-value storage systems store large numbers (billions or even more) of small (KB-MB) sized records
- Records are **partitioned** across multiple machines and
- Queries are routed by the system to appropriate machine
- Records are also **replicated** across multiple machines, to ensure availability even if a machine fails
    - Key-value stores ensure that updates are applied to all replicas, to ensure that their values are **consistent**

Key Value Storage Systems

- Key-value stores may store
  - **uninterpreted bytes**, with an associated key
    - E.g., Amazon S3, Amazon Dynamo
  - **Wide-table** (can have arbitrarily many attribute names) with associated key
    - Google BigTable, Apache Cassandra, Apache Hbase, Amazon DynamoDB
    - Allows some operations (e.g., filtering) to execute on storage node
  - JSON (JavaScript Object Notation) - lightweight data interchange format
    - MongoDB, CouchDB (document model)

Key Value Storage Systems

- **Document stores** store semi-structured data, typically JSON
- Some key-value stores support multiple versions of data, with timestamps/version numbers

Data Representation

An example of a JSON object is:

```
{
  "ID": "22222",
  "name": {
           "firstname: "Albert",
           "lastname: "Einstein"
  },
  "deptname": "Physics",
  "children": [
       { "firstname": "Hans", "lastname":
"Einstein" },
       { "firstname": "Eduard", "lastname":
"Einstein" }
   ]
}
```

Key Value Storage Systems

- Key-value stores support
  - ***put***`(key, value)`: used to store values with an associated key,
  - ***get***`(key)`: which retrieves the stored value associated with the specified key
  - ***delete***`(key)` -- Remove the key and its associated value
- Some systems also support *range queries* on key values
- Document stores also support queries on non-key attributes
  - Ex., MongoDB queries

Key Value Storage Systems

- Not supporting above features makes it easier to build scalable data storage systems
  - Also called NoSQL systems

Parallel and Distributed Databases

- Parallel databases run multiple machines (cluster)
  - Developed in 1980s, well before Big Data
- Parallel databases were designed for smaller scale (10s to 100s of machines)
  - Did not provide easy scalability
- **Replication** used to ensure data availability despite machine failure
  - But typically restart query in event of failure
    - Restarts may be frequent at very large scale
    - Map-reduce systems can continue query execution, working around failures

Replication and Consistency

- **Availability** (system can run even if parts have failed) is essential for parallel/distributed databases
  - Via replication, so even if a node has failed, another copy is available
- **Consistency** is important for replicated data
  - All live replicas have same value, and each read sees latest version
  - Often implemented using majority protocols
    - E.g., have 3 replicas, reads/writes must access 2 replicas
- **Network partitions** (network can break into two or more parts, each with active systems that can't talk to other parts)
- In presence of partitions, cannot guarantee both availability and consistency
  - Brewer's CAP "Theorem"

Replication and Consistency

- Very large systems will partition at some point
  - Choose one of consistency or availability
- Traditional database choose consistency
- Most Web applications choose availability
  - Except for specific parts such as order processing