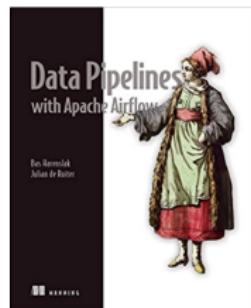




UMD DATA605 - Big Data Systems

8.1: Cluster Architecture

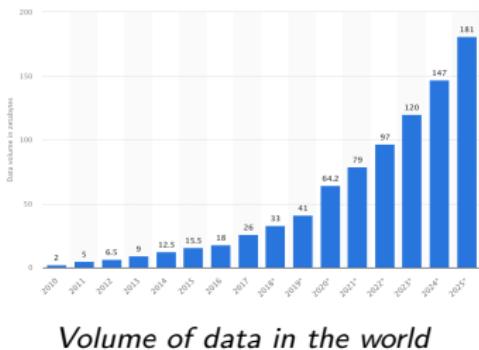
- **Instructor:** Dr. GP Saggese - gsaggese@umd.edu
- Resources
 - Silberschatz: Chap 10



Big Data: Sources and Applications

- **Growth of World Wide Web in 1990s and 2000s**

- Store and query data larger than enterprise data
- Valuable data for advertisements and marketing
- Web server logs, web links
- Social media
- Mobile phone app data
- Transaction data
- Sensor/Internet of Things data
- Communication metadata



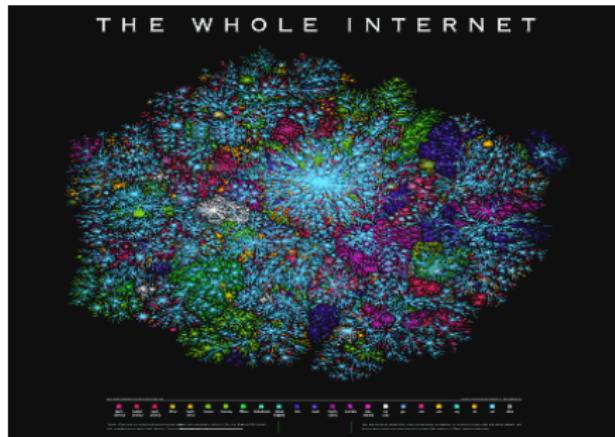
Volume of data in the world

Big Data: Storing and Computing

- Big data needs 10k-100k machines
- **Two problems**
 - Storing big data
 - Processing big data
- **Solve together efficiently**
 - One slow phase slows entire system

Processing the Web: Example

- Web contains:
 - 20+ billion pages
 - 5M TBs = 5 ZB
 - 1M 5TB hard drives needed
 - \$100/HDD -> \$100M total
- One computer reads 300 MB/sec
 - 4,433 years to read web serially
- More time needed for data processing



Big Data: Storage Systems

- Store big data
- **Distributed file systems**
 - Store large files like log files
- **Sharding across multiple DBs**
 - Partition records based on shard key
- **Parallel and distributed DBs**
 - Store data / perform queries across machines
 - Use relational DB interface
- **Key-value stores**
 - Store/retrieve data based on a key
 - Limitations on semantics, consistency, querying
 - E.g., NoSQL DB, Mongo, Redis

1 Distributed File Systems

- **Distributed file system**

- Files stored across machines, single file-system view to clients
 - E.g., Google File System (GFS)
 - E.g., Hadoop File System (HDFS) based on GFS
 - E.g., AWS S3
- Files are:
 - Broken into blocks
 - Blocks partitioned across machines
 - Blocks often replicated
- **Goals:**
 - Store data not fitting on one machine
 - Increase performance
 - Increase reliability/availability/fault tolerance

2 Sharding Across Multiple DBs

- **Sharding:** Partition records across multiple DBs or machines
- Shard keys
 - Aka partitioning keys / partition attributes
- Attributes to partition data
 - Range partition (e.g., timeseries)
 - Hash partition
- **Pros**
 - Scale beyond a centralized DB for more users, storage, processing speed
- **Cons**
 - Replication needed for failures
 - Ensuring consistency is challenging
 - Relational DBs struggle with constraints (e.g., foreign key) and transactions on multiple machines

3 Parallel and Distributed DBs

- **Parallel and distributed DBs:** store and process data on multiple machines (cluster)
 - E.g., mongo
- **Pros**
 - Programmer viewpoint
 - Traditional relational DB interface
 - Appears as a single-machine DB
 - Operates on 10s-100s of machines
 - Data replication enhances performance and reliability
 - Frequent failures with 100s of machines
 - Queries can restart on different machines
- **Cons**
 - Incremental query execution is complex
 - Scalability limits

4 Key-value Stores

- **Problem**

- Applications store billions of small records
- File systems can't handle so many files
- RDBMSs lack multi-machine constraints and transactions

- **Solution**

- Key-value stores / Document / NoSQL systems
- Store, update, retrieve records by key
- Operations: `put(key, value)`, `get(key)`

- **Pros**

- Partition data across machines
- Support replication and consistency
- Balance workload, add machines

- **Cons**

- Sacrifice features for scalability
 - Declarative querying
 - Transactions
 - Non-key attribute retrieval

4 Parallel Key-value Stores

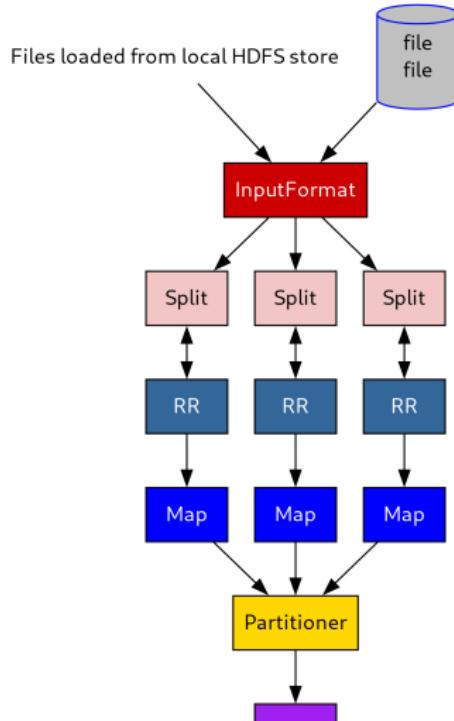
- **Parallel key-value stores**
 - BigTable (Google)
 - Apache HBase (open source BigTable)
 - Dynamo, S3 (AWS)
 - Cassandra (Facebook)
 - Azure cloud storage (Microsoft)
 - Redis
- **Parallel document stores**
 - MongoDB cluster
 - Couchbase
- **In-memory caching systems**
 - Store relations in-memory
 - Replicated or partitioned across machines
 - E.g., memcached or Redis

Big Data: Computing Systems

- How to process Big Data?
- Challenges
 - Distribute computation
 - Simplify writing distributed programs
 - Distributed/parallel programming is hard
 - Store data in a distributed system
 - Survive failures
 - One server may last 3 years (1,000 days)
 - With 1,000 servers, expect 1 failure/day
 - E.g., 1M machines (Google in 2011) → 1,000 machines fail daily
- MapReduce
 - Solve these problems for specific computations
 - Elegant way to work with big data
 - Originated as Google's data manipulation model
 - Not an entirely new idea

Cluster Architecture

- Today, a standard architecture for big data computation has emerged:
 - Cluster of commodity Linux nodes
 - Commodity network (typically Ethernet) to connect them
 - In 2011 it was guesstimated that Google had 1M machines, in 2025



Cluster Architecture



Cluster Architecture: Network Bandwidth

- **Problems**

- Data hosted on different machines
- Network data transfer takes time

- **Solutions**

- Bring computation to data
- Store files multiple times for reliability/performance

- **MapReduce**

- Addresses these problems
- Storage: distributed file system
 - Google GFS, Hadoop HDFS
- Programming model: MapReduce

Storage Infrastructure

- **Problem**
 - Store data persistently and efficiently despite node failures
- **Typical data usage pattern**
 - Huge files (100s of GB to 1TB)
 - Common operations: reads and appends
 - Rare in-place updates
- **Solution**
 - Distributed file system
 - Store files across multiple machines
 - Files are:
 - Broken into blocks
 - Partitioned across machines
 - Replicated across machines
 - Provide a single file-system view to clients

Distributed File System

- Reliable distributed file system
 - Data in “chunks” across machines
 - Each chunk replicated on different machines
 - Seamless recovery from disk or machine failure
- Bring computation directly to the data
 - “chunk servers” also serve as “compute servers”

Hadoop Distributed File System

- **NameNode**

- Store file/dir hierarchy
- Store file metadata (location, size, permissions)

- **DataNodes**

- Store data blocks
- Split file into 16-64MB blocks
- Replicate chunks (2x or 3x)
- Keep replicas in different racks

Hadoop Distributed File System

- **Library for file access**
 - Read:
 - Contact *NameNode* for *DataNode* and block pointer
 - Connect to *DataNode* for data access
 - Write:
 - *NameNode* creates blocks
 - Assign blocks to multiple *DataNodes*
 - Client sends data to *DataNodes*
 - *DataNodes* store data
- **Client**
 - API (e.g., Python, Java) to library
 - Mount HDFS on local filesystem