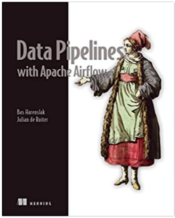
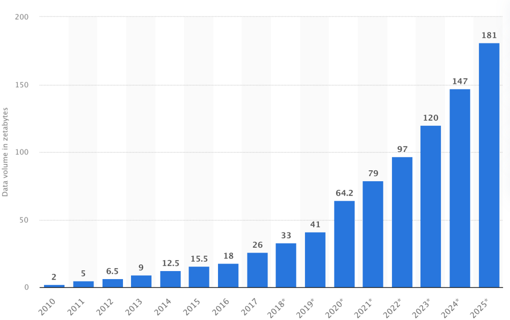
UMD DATA605 - Big Data Systems

8.1: Cluster Architecture

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



Big Data: Storing and Computing

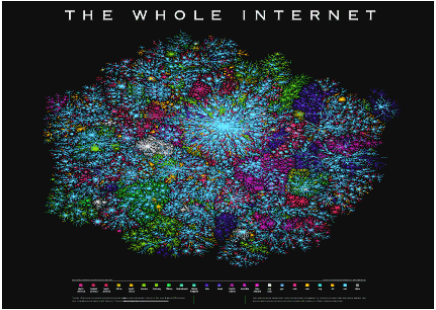


Volume of data in the world

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

Processing the Web: Example

- Web contains (in 2024):
  - 20+ billion pages
  - 5M TBs
- Need 1M 5TB hard drives
  - \$100/HDD -> \$100M in total
- One computer reads 300 MB/sec
  - 4,433 years to read web serially!
- Much more time needed for data processing!



SCIENCE ACADEMY

How to Store Big Data?

- There are many different solutions to storing big data?
  1. **Distributed file systems**
  2. **Sharding across multiple DBs**
  3. **Parallel and distributed DBs**
  4. **Key-value stores**

1) Distributed File Systems

- 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
  - Range partition (e.g., timeseries)
  - Hash partition
- **Pros**
  - Scale beyond 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

- Store and process data on multiple machines (cluster)
- **Pros**
  - From programmer viewpoint
    - Traditional relational DB interface
    - Appears as a single-machine DB
  - Scale to 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

SCIENCE ACADEMY

4) Key-value Stores

- **Problem**
  - Applications store billions of small records
  - RDBMSs lack multi-machine constraints and transactions
- **Solution**
  - Key-value stores / Document / NoSQL systems
  - Store, update, retrieve records by 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

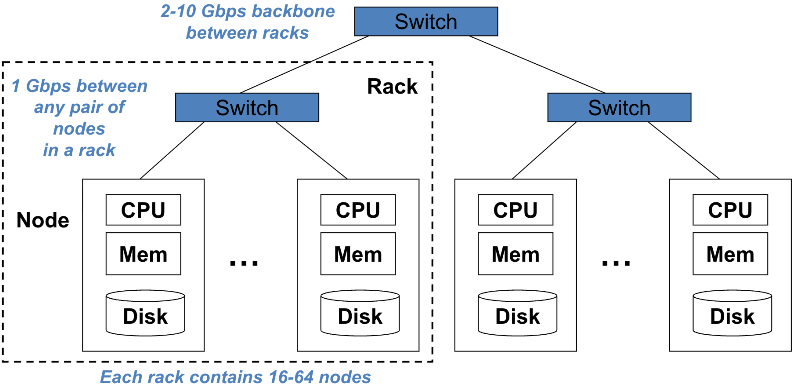
- **Parallel key-value stores**
  - Redis
  - Google BigTable
  - Apache HBase (open source BigTable)
  - AWS Dynamo, S3
  - Cassandra (Facebook)
  - Azure cloud storage (Microsoft)
- **Parallel document stores**
  - MongoDB cluster
  - Couchbase
- **In-memory caching systems**
  - Store relations in-memory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

How to Compute with Big Data?

- **Challenges**
  - Distribute computation
  - Simplify writing distributed programs
    - Distributed/parallel programming is hard
  - Store data in a distributed system
  - Survive failures
    - One server lasts $\sim$3 years (1,000 days)
    - With 1,000 servers, expect 1 failure/day
    - E.g., 1M machines (Google in 2011) $\rightarrow$ 1,000 machines fail daily
- **MapReduce**
  - Solve 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

- Standard architecture for big data computation:
  - Cluster of commodity Linux nodes
  - Commodity network (typically Ethernet) to connect nodes
  - 2011: Google ~1M machines
  - 2025: ~10-15M (?)



-

SCIENCE ACADEMY

Cluster Architecture

SCIENCE ACADEMY

Cluster Architecture: Network Bandwidth

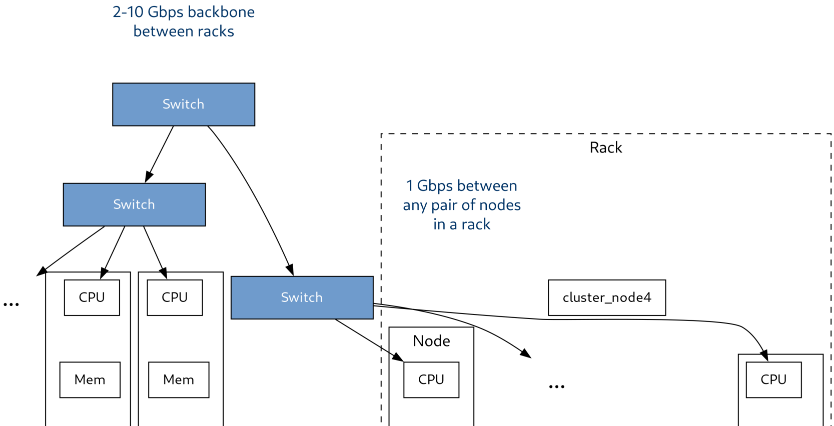
- **Problems**
  - Data hosted on different machines
  - Network data transfer takes time
- **Solutions**
  - Bring computation to data
  - Store files multiple times for reliability/performance
- **MapReduce**
  - Addresses these problems
  - Storage: distributed file system
    - Google GFS, Hadoop 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