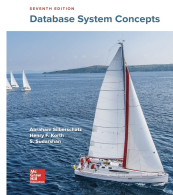
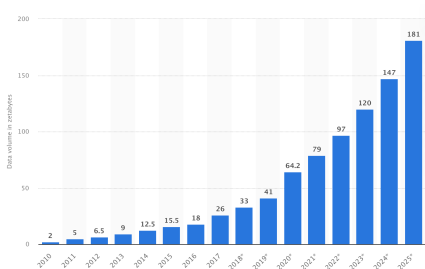


8.1: Cluster Architecture

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



Big Data: Storing and Computing

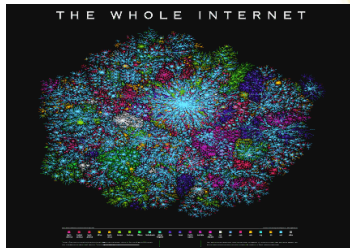


Volume of data in the world

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

Processing the Web: Example

- Web contains (in 2024):
 - 20+ billion pages
 - 5M TBs of content
- **Storing the web**
 - Need 1M 5TB hard drives
 - \$100/HDD -> \$100M in total
 - Not bad
 - Wait, one computer reads 300 MB/sec
 - 4,500 years to read web serially!
- **Processing the web**
 - Much larger time / cost needed!



How to Store Big Data?

- **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 yet single file-system view to clients**
 - E.g.,
 - Google File System (GFS)
 - Hadoop File System (HDFS)
 - 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
 - Approach scales 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

4) (Parallel) Key-value Stores

- **Problem**

- Applications store billions of small records
- Relational DBs lack multi-machine constraints and transactions

- **Solution**

- Key-value stores / Document / NoSQL systems
- E.g.,
 - Redis
 - Apache HBase (open source of Google BigTable)
 - ...
 - AWS Dynamo, S3
 - Azure cloud storage (Microsoft)
 - MongoDB cluster

- **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

How to Compute with Big Data?

- **Challenges**

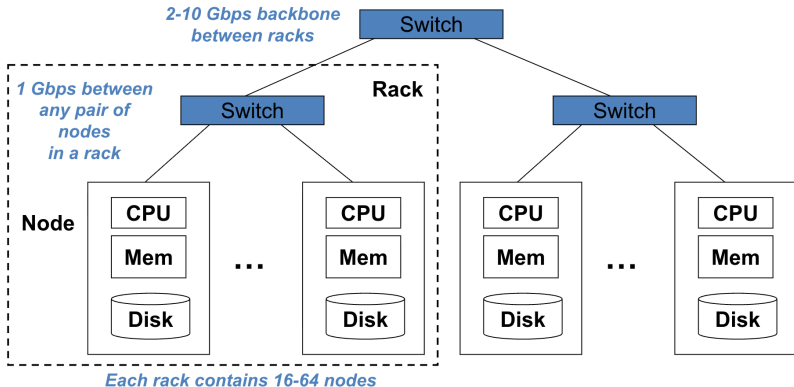
- Distribute computation
- Simplify writing distributed programs
 - Distributed/parallel programming is hard
- Store data in a distributed system
- Survive failures
 - One server lasts ~3 years (1,000 days)
 - With 1,000 servers, expect 1 failure/day
 - E.g., 1M machines (Google in 2011) → 1,000 machines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

- **MapReduce**

- Solve 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 (?)



Cluster Architecture



Cluster Architecture: Network Bandwidth

- **Problems**
 - Data on different machines
 - Network data transfer delays
- **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 and HDFS implement all these ideas