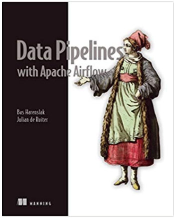
UMD DATA605 - Big Data Systems

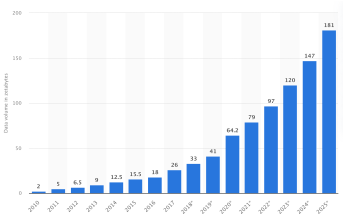
8.1: Cluster Architecture

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



Big Data: Sources and Applications

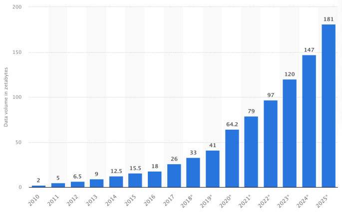
- **Growth of World Wide Web in 1990s and 2000s**
- Storing and querying data much larger than enterprise data
- Extremely valuable data to target advertisements and marketing
- Web server logs, web links
- Social media
- Data from mobile phone apps
- Transaction data
- Data from sensors / Internet of Things
- Metadata from communication data



Volume of data in the world

Big Data: Sources and Applications

- **Big Data characteristics**
- Volume:
  - Amount of data to store and process is much larger than traditional DBs
  - Too big even for parallel DB systems with 10-100 machines
- Velocity
  - Store data at very high rate, due to rate of arrival
  - Data might be processed in real-time (e.g., streaming systems)
- Variety
  - Not all data is relational
  - E.g., semi-structured, textual, graphical data
- **Solution**: process data with 10,000-100,000 machines



Volume of data in the world

Big Data: Sources and Applications

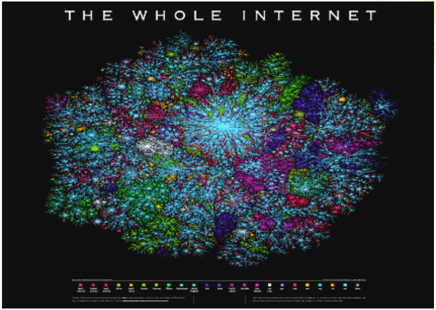
- **Web server logs**
- Record user interactions with web servers
- Billions of users click on thousands links per day $\rightarrow$ TB of data / day
- Decide what information (e.g., posts, news) to present to users to keep them "engaged"
  - E.g., what user has viewed, what other similar users have viewed
- Understand visit patterns to optimize for users to find information
- Determine user preferences and trends to inform business decisions
- Decide what advertisements to show to which users
  - Maximize benefit to the advertiser
  - Websites are paid for click-through or conversion
- **Click-through rate**
- A user clicks on an advertisement to get more information
- It is a measure of success in getting user attention/engagement
- **Conversion rate**
  - When a user actually purchases the advertised product or service

Big Data: Storing and Computing

- Big data needs 10k-100k machines
- **Two problems**
    - Storing big data
    - Processing big data
- **Need to be solved together and *efficiently***
    - If one phase is slow $\rightarrow$ the entire system is slow

Processing the Web: Example

- The web has:
  - 20+ billion web pages
  - Total 5M TBs = 5 ZB
  - 1M 5TB hard drives to store the web
  - 100/HDD -> 100M to store the web
  - Not too bad!
- One computer reads 300 MB/sec from disk
  - 5e6 * 1024 * 1024 * 8 / 300 / 3600 / 24 / 365 = 4,433 years to read the web serially from disk
- It takes even more to do something useful with the data!



SCIENCE ACADEMY

Big Data: Storage Systems

- How can we store big data?
- **Distributed file systems**
  - E.g., store large files like log files
- **Sharding across multiple DBs**
  - Partition records based on shard key across multiple systems
- **Parallel and distributed DBs**
  - Store data / perform queries across multiple machines
  - Use traditional relational DB interface
- **Key-value stores**
  - Data stored and retrieved based on a key
  - Limitations on semantics, consistency, querying with respect to relational DBs
  - E.g., NoSQL DB, Mongo, Redis

1 Distributed File Systems

- **Distributed file system**
  - Files stored across a number of machines, giving a single file-system view to clients
    - E.g., Google File System (GFS)
    - E.g., Hadoop File System (HDFS) based on GFS architecture
    - E.g., AWS S3
  - Files are:
    - Broken into multiple blocks
    - Blocks are partitioned across multiple machines
    - Blocks are often replicated across machines
  - **Goals**:
    - Store data that doesn't fit on one machine
    - Increase performance
    - Increase reliability/availability/fault tolerance

2 Sharding Across Multiple DBs

- **Sharding** = process of partitioning records across multiple DBs or machines
- Shard keys
  - Aka partitioning keys / partition attributes
- One or more attributes to partition the data
  - Range partition (e.g., timeseries)
  - Hash partition
- **Pros**
  - Scale beyond a centralized DB to handle more users, storage, processing speed
- **Cons**
  - Replication is often needed to deal with failures
  - Ensuring consistency is challenging
  - Relational DBs are not good at supporting constraints (e.g., foreign key) and transactions on multiple machines

3 Parallel and Distributed DBs

- **Parallel and distributed DBs**: store and process data running on multiple machines (aka "cluster")
  - E.g., Mongo
- **Pros**
  - Programmer viewpoint
    - Traditional relational DB interface
    - Looks like a DB running on a single machine
  - Can run on 10s-100s of machines
  - Data is replicated for performance and reliability
    - Failures are "frequent" with 100s of machines
    - A query can be just restarted using a different machine
- **Cons**
  - Run a query incrementally requires a lot of complexity
  - Limit to the scalability

4 Key-value Stores

- **Problem**
- Many applications (e.g., web) store a very large number (billions or more) small records (few KBs to few MBs)
- File systems can't store such a large number of files
- RDBMSs don't support constraints and transactions on multiple machines
- **Solution**
- Key-value stores / Document / NoSQL systems
- Records are stored, updated, and retrieved based on a key
- Operations are: **put(key, value)** to store, **get(key)** to retrieve data
- **Pros**
- Partition data across multiple machines easily
- Support replication and consistency (no referential integrity)
- Balance workload and add more machines
- **Cons**
- Features are sacrificed to achieve scalability on large clusters
  - Declarative querying
  - Transactions
  - Retrieval based on non-key attributes

4 Parallel Key-value Stores

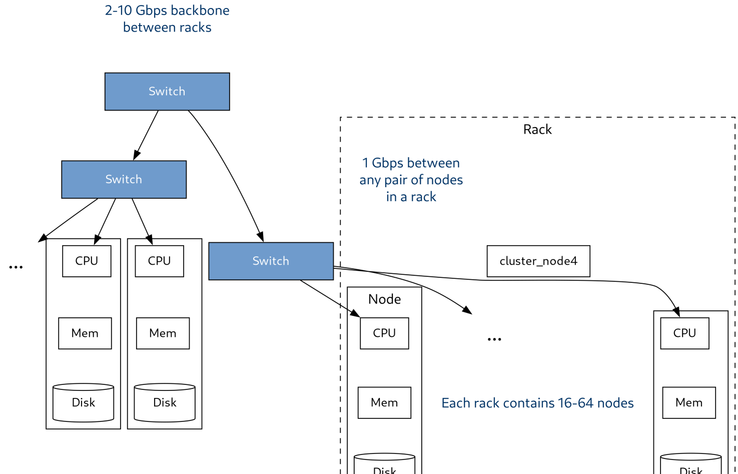
- **Parallel key-value stores**
- BigTable (Google)
- Apache HBase (open source version of BigTable)
- Dynamo, S3 (AWS)
- Cassandra (Facebook)
- Azure cloud storage (Microsoft)
- Redis
- **Parallel document stores**
- MongoDB cluster
- Couchbase
- **In-memory caching systems**
- Store some relations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

Big Data: Computing Systems

- **How to process Big Data?**
- **Challenges**
  - How to distribute computation?
  - How can we make it easy to write distributed programs?
    - Distributed / parallel programming is hard
  - How to store data in a distributed system?
  - How to survive failures?
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to lose 1 / day
    - E.g., 1M machines (Google in 2011) $\rightarrow$ 1,000 machines fail every day!
- **MapReduce**
  - Solve these problems for certain kinds of computations
  - An elegant way to work with big data
  - Started as Google's data manipulation model
    - (But it wasn't 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 ~10-15M (?)



SCIENCE ACADEMY

Cluster Architecture

Cluster Architecture: Network Bandwidth

- **Problems**
  - Data is hosted on different machines in a cluster
  - Copying data over a network takes time
- **Solutions**
  - Bring computation close to the data
  - Store files multiple times for reliability/performance
- **MapReduce**
  - Addresses both these problems
  - Storage infrastructure: distributed file system
    - Google GFS, Hadoop HDFS
  - Programming model: MapReduce

Storage Infrastructure

- **Problem**
  - How to store data *persistently* and *efficiently* when nodes can fail?
- **Typical data usage pattern**
  - Huge files (100s of GB to 1TB)
  - Reads and appends are common operations
  - Data is rarely updated in place
- **Solution**
  - Distributed file system
  - Allow files to be stored across a number of machines
  - Files are:
    - Broken into multiple blocks
    - Partitioned across multiple machines
    - Typically with replication across machines
  - Give a single file-system view to clients

Distributed File System

- Reliable distributed file system
  - Data kept in "**chunks**" spread 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 metadata about files (e.g., where are stored, size, permissions)
- **DataNodes**
  - Store data blocks
  - File is split into contiguous 16-64MB blocks
  - Each chunk is replicated (usually 2x or 3x)
  - Keep replicas in different server racks

Hadoop Distributed File System

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