CS3231
AY23/24 S1
github.com/SeekSaveServe

Lectures

L1: Introduction

Four Vs of Data Science

- Volume
- Variety
- Velocity
- Veracity - uncertainty of data

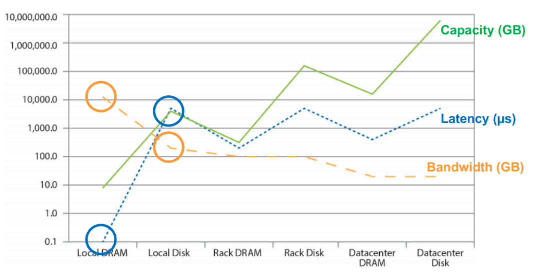
Storage Hierarchy

- Volume: Server le Rack le Cluster
- Speed: Server ge Rack ge Cluster

Bandwidth vs Latency

- **Throughput** Actual rate at which data is transmitted across the network over a period of time
- **Bandwidth** Maximum (capacity) amount of data that can be transmitted per unit time
- **Latency** Time taken for 1 data packet to go from source to destination (or both ways)
- Latency does not matter when transmitting a large amount of data
- Bandwith does not matter when transmitting a small amount of data

Cost of moving data



- Bandwidth drops and latency increases as we move up the data hierarchy
- Disk reads are also much more expensive

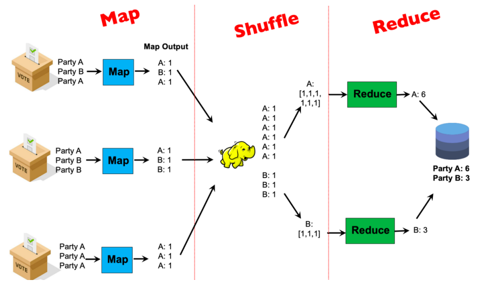
Big ideas of data processing

- Horizontal scaling is cheaper than vertical scaling
- Move data processing to the machine with the data since data clusters have limited bandwidth
- Process data sequentially and avoid random access to reduce total seek time
- Seamless scalability $\rightarrow$ use more machines to reduce time taken to process data

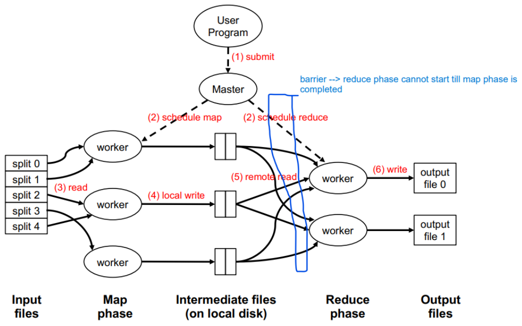
Challenges

- Machine failures
- Synchronisation
- Programming difficulty

L2 Map reduce



- Map: extract something of interest from each. Emits a key value pair
- Shuffle: Shuffle intermediate results by key value pairs
- Reduce: Aggregate intermediate results
- Each of these three processes can occur concurently across different machines



Map Reduce Implementation

1. **Submit:** User submits mapreduce program and configuration (e.g. no. of workers) to Master node
2. **Schedule:** Master schedules resource for map and reduce tasks (master does not handle actual data)
3. **Read:** Input files are separated into splits of 128MB. Each split corresponds to one map task. Each worker executes map tasks 1 ata a time
4. **Map phase:** Each worker iterates over each key,value tuple and applies the map function
5. **Local write:** each worker writes the output of map to intermiediate files on its local disk.These filees are partitioned by key
6. **Remote read:** each reduce worker is responsible for $\geq$ key. For each key, it reads the data it needs from the corresponding partitioon of each mapper's local disk
7. **Write:** output of the reduce function is written (usually to a distributed file system such as HDFS)

Interface

- map(k,v) $\rightarrow$ list(k',v')
- reduce

L3: No SQL Overview

NoSQL

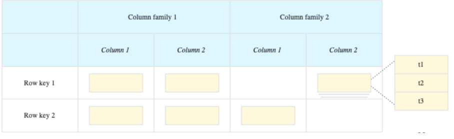
- Not Only SQL: can include sql
- Stores data in a format other than relational DB
- Sql refers to to relational DBMS, not the querying language - NoSQL can have querying lang too
- Used for large volumes of data and data that does not fit in a structured data (e.g. some has image, some don't)

Propertiess

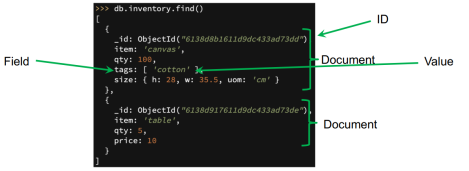
- Horizontal Scalable: easy to partition and distribute across machines
- Replicate and distributed over many servers
- Simple call interface
- Often weaker concurrency model than RDBMS
- Efficient use of distributed indexes and RAM
- Flexible schema

Major NoSQL DB

- Key-stores
  - Stores mapping (associations) bewteen keys and values
  - Keys are usually primitives (int,str,raw bytes etc) that can be easily queried
  - Values can be primitive or complex; usually cannot be easily queried (lists, JSON, HTML, BLOB)
  - **Operations**
    - Get - fetch value with key
    - Put - set value with key
    - Multi-Get, multi-put, range queries (must be comparable, e.g. int, str)
  - **Suitable for**
    - Smal continuous read and writes
    - Storing basic information or no clear schema
    - Complex queries are rarely required
  - E.g. Storing user sesions, caches, user data that is often processed individually
  - **Implementation**
    - Non-persistent: Just a big in memory hash table (E.g. redis, memcached) that needs to be regularly backed up to disk
    - Persistent: data is stored persistently to disk (E.g. RocksDB, Dynamo, Riak)
- Wide-column databases - stored sparsely

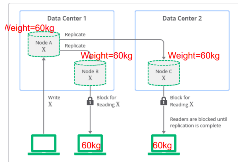


  - Rows describe entities
  - Related groups of columns are grouped as column families (similar to separate tables, except they share the same row)
  - Sparsity: If a cloumn is not used for a row, it doesn't use space (saves space for sparse data)
- Document stores

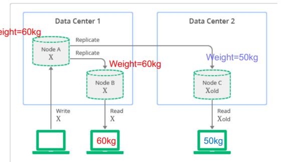


  - no schema (flexible schema)
  - A data base can have multiple collections
  - A collection (tables) can have multiple documents (rows)
  - A document is a JSON-like object with field (columns) and values
  - Different documents can have different field and can be nested
  - **Querying**
    - Unlike key val stores, doc stores allow querying based on the content
    - If the field does not exist on the doc, we just skip it when doing CRUD
- Graph databases
  - Need to store information about the nodes and edges
- Vector Databases
  - Store vectors (each row is a point in d dimensions)
  - Usually dense, numerical, and high-dimensional
  - Allow fast similarity search via locality sensitive hashing (LSH), similar to min-hashing
  - Scalable, real-time updates, replication
  - Good for LLM and vision models as they need to be converted to vectors, and search, recommendation, clustering can be easily added
  - E.g. Milvus, Radis, MongoDB, Atlas, Weaviate

Consistency



All readers read new value (60kg)



Readers may read old value (50kg)

- Strong: Any reads on all observers immediately read the same result after update (uses locks, higher latency)
- Eventual: If the system is working and we wait long enough, eventually all reads will produce the same value (correctness affected)

BASE

- **B**asically **A**vailable - basic writing and reading operations are available most of the time
- **S**oft state: without guarantees, we only have some probability of knowing the state at any time
- **E**ventually consistent
- Eventual consistency offers better availability at the cost of weaker consistency
- NoSQL allows for weaker consistency guarantees, and can be tuned to be stronger (tunable consistency)
- Suitable for statistical queries and social media feed but not suited for financial transactions

Duplication

- Motivation: Support join statments $\rightarrow$ how do we join 2 tables to form 1 new table
- Some optimizations in SQL may not be possible in NoSQL
- **Denormalization:**
  - Storage is cheap! Duplicate data to boost efficiency
  - Tables are designed around potential join queries (pre-create the join tables)
  - Good if the queries types are fixed
  - What if a field is updated? $\rightarrow$ changes need to be propagated to multiple table

Pros

+ **Flexible / dynamic schema**: suitable for less well-structured data

+ **Horizontal scalability**: we will discuss this more next week

+ **High performance and availability**: due to their relaxed consistency model and fast reads / writes

Cons

- **No declarative query language**: query logic (e.g. joins) may have to be handled on the application side, which can add additional programming

- **Weaker consistency guarantees**: application may receive stale data that may need to be handled on the application side

- Depends on:

1. if denormalization is suitable
2. importance of consistency
3. complexity of queries (joins Vs read/write)