4V's of Big Data

- Volume
- Velocity
- Variety
- Veracity

Volume: amount of data

Velocity: speed of data generation

Variety: number of data types

Veracity: uncertainty of data

5-Step Data Analytical Process

handout 2

(i) Term Frequency: TF

- More frequent terms in a document are more important, i.e., more indicative of the topic

  $tf_{i,j}$ = frequency of term T_i in document d_j

(ii) Inverse Document Frequency: IDF

- Terms that appear in many different documents are less indicative of overall topic in a document

  df_i = number of documents containing term T_i

  idf_i = inverse document frequency of term T_i

  = $\log_2(\frac{N}{df_i})$ (N : total number of documents)

TF-IDF Weighting

- The combined term importance indicator is called TF-IDF weighting:

$$w_{i,j} = tf_{i,j} \cdot idf_i$$

DBSCAN: The Algorithm

1. Set all points as UNPROCESSED.
2. Select an UNPROCESSED point p.
3. Find Eps-Neighborhood of p using parameter Eps.
4. If p is a **core** point (based on $MinPts$), a cluster is formed. p is set as CORE.
   A. Put all UNPROCESSED points in Eps-Neighborhood of p in a queue S.
   B. For each point $q \in S$, check whether it is a **core** point.
      a. if yes, set q as CORE, merge two clusters.
         Note: Points in q's neighborhood will be added to S
      b. if no, set q as BORDER.

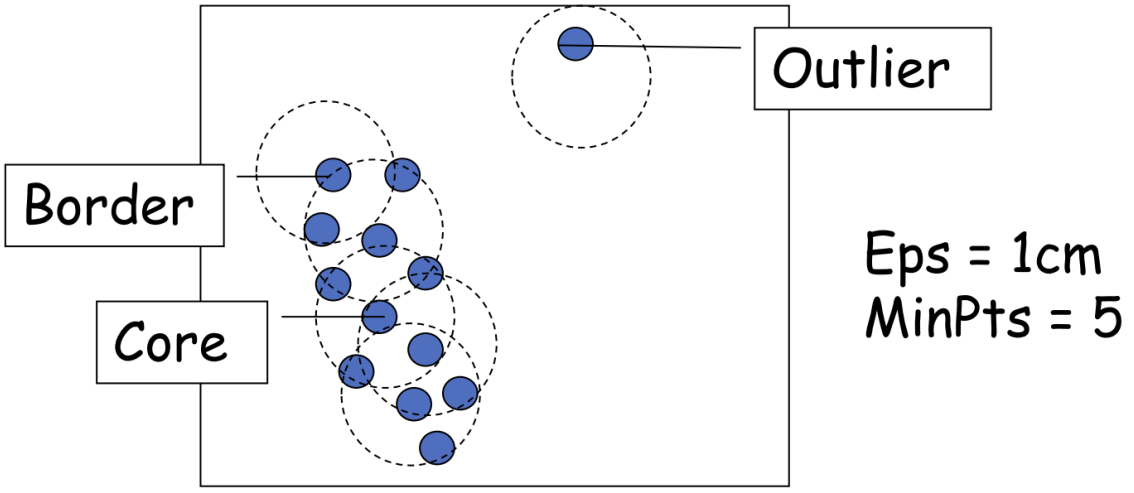
   Repeat until S is empty.
5. If p is NOT a core point, set p as **UNASSIGNED** to a cluster
   A. p may be included to a cluster later if it is found to be in the Eps-Neighborhood of a core point (**BORDER**)

   (If p is not in any clusters when terminates, it is set as **OUTLIER**)
6. If there are other UNPROCESSED points, goto 2.

The status of a point: UNPROCESSED, CORE, BORDER, OUTLIER, UNASSIGNED (intermediate status)

Density-Based Clustering

- A **core** point and its ***Eps*-neighborhood** define a **cluster**.
- If two **core points** p and q **belong to** the Eps-neighborhoods of **each other**, **merge** the corresponding clusters
- Continue the merge until no clusters can be merged



- **Eps-neighborhood** of point p = {all points within distance Eps from p}
  - $N_{Eps}(p) = \{q \mid dist(p,q) <= Eps\}$
  - Eps: Maximum **radius** of the neighborhood
- If an object q is not a core point, but it belongs to the Eps-neighborhood of a core point, then q is a **border** object

Hierarchical Clustering

- Single Link: smallest distance between any points in two clusters
- Complete Link: largest distance between any points in two clusters
- Centroid: distance between the centroids of two clusters
- Average Link: average distance of all pairwise points in clusters
  - Average of the distances of the 4*3 pairs of points in the example

handout 3

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
  - **Rule Evaluation Metrics**
    - **Support (s)**: Fraction of transactions that contain both X and Y
    - **Confidence (c)**: Measures how often items in Y appear in transactions that contain X

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{\text{Milk, Diaper}\} \Rightarrow \{\text{Beer}\}$$

$$s = \frac{\sigma(\text{Milk, Diaper,Beer})}{|\text{T}|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk,Diaper,Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

$$confidence(X \Rightarrow Y) = \frac{support\_count(X \cup Y)}{support\_count(X)}$$

Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

Reducing Number of Candidate Itemsets

- **Apriori** principle:
  - If an itemset is frequent, all its subsets must also be frequent
  - If an itemset is infrequent, any *supersets* containing it must be infrequent

$$\forall X, Y: (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

Entropy$(Y) = -\sum_{i=1}^{m} p_i \log_2(p_i)$, where p_i is the probability of class i

- **Information gain** of splitting on attribute *A*
  = entropy(parent) – weighted average entropy(children)

Naïve Bayes Classifier: Likelihood

- $P(Y = no|x_1, x_2, x_3) \propto \boldsymbol{P(x_1, x_2, x_3|Y = no)}P(Y = no)$
- $P(Y = yes|x_1, x_2, x_3) \propto \boldsymbol{P(x_1, x_2, x_3|Y = yes)}P(Y = yes)$

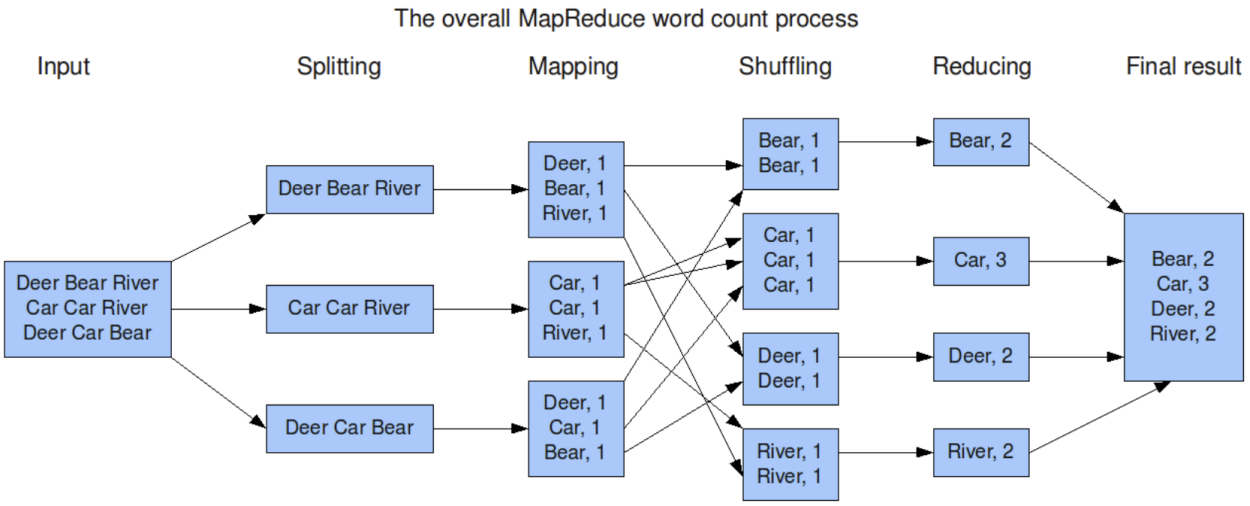
- **Assumption: features are independent**

handout 4

- Worker: a basic instance to undertake the tasks.

MapReduce Workflow Definitions

- **Map**: a mapping that is responsible for dividing the data and transforming the original data into key-value pairs
- **Shuffle and Sort**: the process of further organizing and delivering the Map output to the Reduce
  - the output of the Map must be sorted and segmented
  - then passed to the corresponding Reduce
  - Communication occurs
- **Reduce**: a merge that processes the values with the same key and then outputs to the final result



- **Map(): input data → (key, value) pairs**
  - produces set of intermediate pairs
- **Reduce(): (key, value) pairs → <result>**
  - combines all intermediate values for a particular key
  - produces a set of merged output values
  - Is the data **Write Once Read Many (WORM)**?
  - Note that MapReduce is not designed to be good at solving all problems
- Many examples of algorithms depend crucially on the existence of **shared global state** during processing -> **difficult** to implement in MapReduce

MapReduce Summary

- Very large-scale data: peta, exa-bytes
- WORM data: allows for parallelism without mutexes
- Map and Reduce are the main operations: simple code
- All Map should be completed before Reduce starts
- Number of map tasks and reduce tasks are configurable
- Operations are provisioned near the data
- Commodity hardware and storage
- Runtime takes care of splitting and moving data
- Special distributed file system, e.g., Hadoop Distributed File System and Hadoop Runtime

handout 5

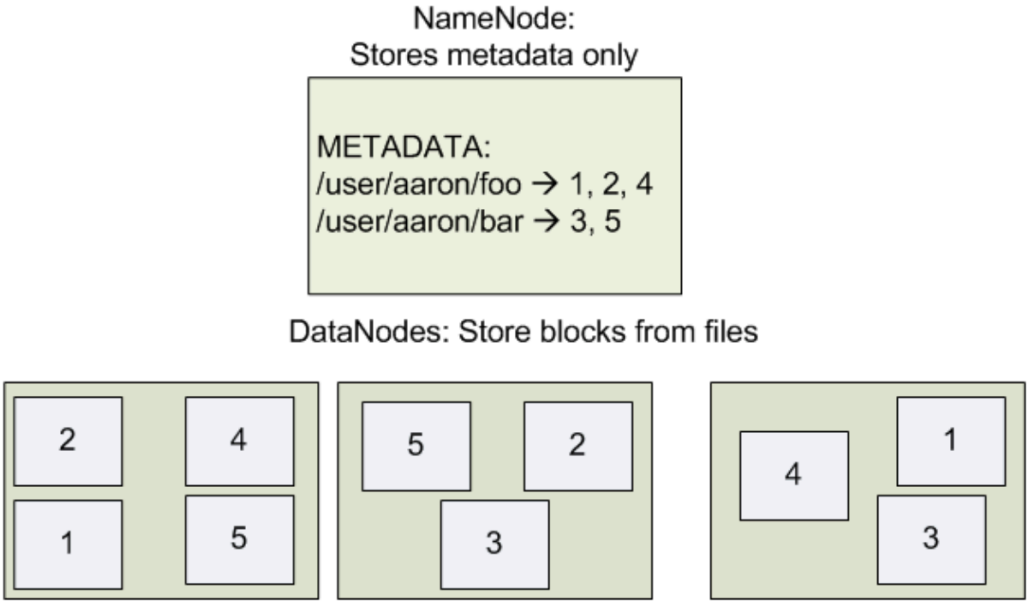
Goals of HDFS (Cont.)

- HDFS is designed to 'just work', to process large data sets with **write-once-read-many (WORM)** semantics, it is **not for low latency** access
- Can be built out of **commodity hardware**

DataNode

- Where HDFS stores the actual data
- A Machine
  - Stores data blocks in the local file system (e.g. ext3)
  - Stores metadata of a block
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode (heartbeat)
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

DataNodes and NameNode Illustration



Read Operation in HDFS

1. Client connects to NN to request to read data
2. NN tells client where (i.e., DN) to find the data blocks
3. Client reads blocks directly from data nodes (without going through NN)
4. In case of node failures, client connects to another node that serves the missing block

Write Operation in HDFS

- Client retrieves a list of DataNodes on which to place replicas of a block from NameNode
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the Pipeline
- When all replicas are written, the client moves on to write the next block in file

---

Job Scheduling

- **JobTracker** & **TaskTracker**
  - A central control node runs **JobTracker (JT)**.
  - Worker Nodes also runs **TaskTracker (TT)**.
  - JobTracker splits up data into smaller tasks("Map") and sends it to the TaskTracker process in each node
  - A TaskTracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a JobTracker

Job Scheduling Workflow

- Client applications submit jobs to the **JobTracker**.
- The **JobTracker** talks to the **NameNode** to determine the location of the data
- The **JobTracker** locates **TaskTracker** nodes with available slots at or near the data
- The **JobTracker** submits the work to the chosen **TaskTracker** nodes.
- The **TaskTracker** nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different **TaskTracker**.
- A **TaskTracker** will notify the **JobTracker** when a task fails. The **JobTracker** decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may even blacklist the **TaskTracker** as unreliable.
- When the work is completed, the **JobTracker** updates its status.
- Client applications can poll the **JobTracker** for information.

Refinement: Redundant Execution

- Slow workers significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
- **Solution: spawn backup copies of tasks**
  - Whichever one finishes first "wins"

Failure Recovery

- Failures are common in commodity hardware
- Worker (TaskTracker) failure
  - Detect failure via periodic heartbeats
  - Re-execute in-progress map/reduce tasks
- Master (JobTracker) failure
  - Single point of failure
  - Resume from Execution Log

Locality Optimization

- Master scheduling policy
  - Asks HDFS for locations of replicas of input file blocks
  - Map tasks scheduled so HDFS input block replica are on same machine or same rack
- Benefits
  - Thousands of machines read input at local disk speed
  - Eliminate network bottleneck

Resilient Distributed Datasets (RDDs)

- **Resilient Distributed Datasets (RDD)** is a fundamental data structure of Spark. It is an immutable distributed collection of objects
  - **Resilient** – capable of rebuilding data on failure
  - **Distributed** – partitions data across various nodes in cluster
  - **Dataset** – collection of partitioned data with values

Spark Transformation

- Spark Transformation is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output
- As applications share the RDD abstraction, you can mix different kind of transformations to create new RDDs
- Fault tolerant

Summary

- Hadoop (HDFS, MapReduce)
  - Provides an easy solution for processing of Big Data
  - Brings a paradigm shift in programming distributed system
- Spark
  - Has extended MapReduce for in memory computations
  - for streaming, interactive, iterative and machine learning tasks

handout 6

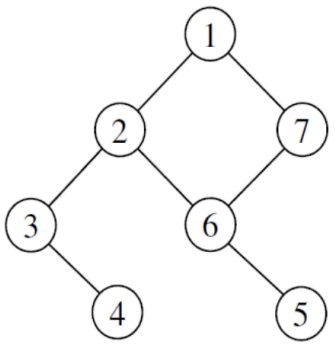
handout 7

Degree Centrality

- Degree centrality
  - The degree of a vertex
  - The higher the degree, the more central the node is.

Closeness Centrality

- The **closeness centrality** of a vertex v is defined as $C(v) = \frac{1}{\sum_u dist(v,u)}$
- $dist(v,u)$ denotes the *shortest path distance* between v and u
- Example: the shortest path distances from vertex 2 to 1, 3, 4, 5, 6, 7 are 1, 1, 2, 2, 1, 2, respectively. So $C(2) = \frac{1}{(1+1+2+2+1+2)} = \frac{1}{9}$



Interpret PageRank by Random Walks

- Start a lot of random walks from all pages in a graph
- PageRank of v:
  - $PR(v) = \Pr[Random\ walks\ arrive\ at\ v]$

Compute PageRank in Iterations

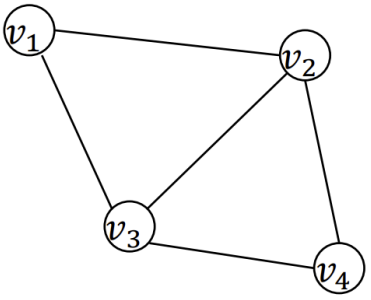
- Given n nodes: $v_1, v_2, v_3, \dots, v_n$
- Initially, set $PR(v) = \frac{1}{n}$ for all nodes

- Updating rule at the t-th iteration:
- $PR^t(v) = \frac{1-d}{n} + d \cdot (\sum_{v' \in N_{in}(v)} \frac{PR^{t-1}(v')}{|N_{out}(v')|})$

In every iteration, the sum of all PR values is 1.
In every iteration, for all nodes, the total $(1-d)$ amount will be evenly distributed to every node, i.e., $\frac{1-d}{n}$.

Recommendation/Link Prediction on Graphs

- Predict if there will be an edge built between v_1 and v_4
  - Or how likely will there be an edge between v_1 and v_4
- What do we need?
  - Similarity/Proximity between vertices



Jaccard Similarity

- Measure the similarity between two nodes in an undirected graph

$$Jaccard(v_1, v_4) = \frac{|N(v_1) \cap N(v_4)|}{|N(v_1) \cup N(v_4)|}$$

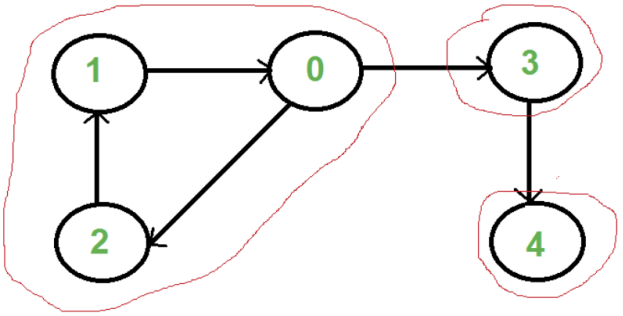
$$N(v_i) = Neighbors(v_i)$$

Connected Components in Undirected Graphs

- A connected component (CC) is a maximal subgraph in which each pair of nodes is connected to each other via paths
  - A node itself can be a CC

Strongly Connected Components (SCC) in Directed Graphs

- A strongly connected component (**SCC**) of a directed graph is a *maximal* strongly connected subgraph, in which, *there is a path between all pairs of vertices in the subgraph*
  - A node itself can be an SCC



Weakly Connected Components (WCCs) in Directed Graphs

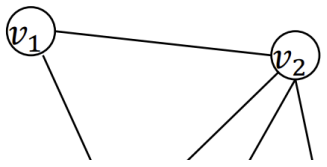
- Regard a directed graph as undirected one (i.e., ignoring edge directions), and the connected components in the undirected graph

Local Clustering Coefficient

- The local clustering coefficient C_v of a vertex v in a graph quantifies how close its neighbors are to being a complete graph
- Given a vertex v in an undirected graph G
- C_v = $$\frac{the\ number\ of\ edges\ between\ any\ pairs\ of\ neighbors\ of\ v}{the\ number\ of\ edges\ in\ the\ complete\ graph\ formed\ by\ v's\ neighbor}$$

$$= \frac{the\ number\ of\ edges\ between\ any\ pairs\ of\ neighbors\ of\ v}{\left(\frac{|N(v)|*(|N(v)|-1)}{2}\right)}$$

- $C_2 = \frac{2*2}{4*3} = \frac{1}{3}$



Local Clustering Coefficient

The local clustering coefficient C_v of a vertex v in a graph quantifies how close its neighbours are to being a complete graph

Given a node v in a directed graph G

C_v= $$\frac{\text{In } G\text{, the number of edges between any nodes } v_i \text{ and } v_j \text{ in } N_{in}(v)\cup N_{out}(v)}{\text{number of edges in the complete directed graph formed by nodes in } N_{in}(v)\cup N_{out}(v)}$$

= $$\frac{\text{In } G\text{, the number of edges between any nodes } v_i \text{ and } v_j \text{ in } N_{in}(v)\cup N_{out}(v)}{|N_{in}(v)\cup N_{out}(v)|*(|N_{in}(v)\cup N_{out}(v)|-1)}$$

- $C_2 = \frac{2}{4*3} = \frac{1}{6}$

Graph Average Clustering Coefficient

- Given a graph with *n* nodes, the average clustering coefficient of the graph is the *average of the local clustering coefficient* of all nodes in the graph.

handout 8

Transactions in RDBMS

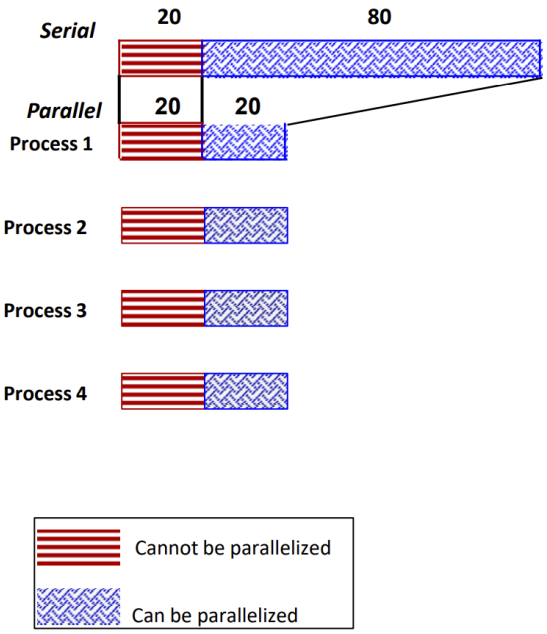
- RDBMS: Relational Database Management System

ACID in RDBMS

- RDBMS provides ACID guarantees to *each* transactions
  - Atomicity: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.
  - Consistency: Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.
  - Isolation: No transaction has access to any other transaction that is in an intermediate or unfinished state. Each transaction is independent.
  - Durability: Ensure that once a transaction commits to the database, it is preserved using backups and transaction logs. Changes to data persist and are not undone, even in the event of a system failure.

Amdahl's Law

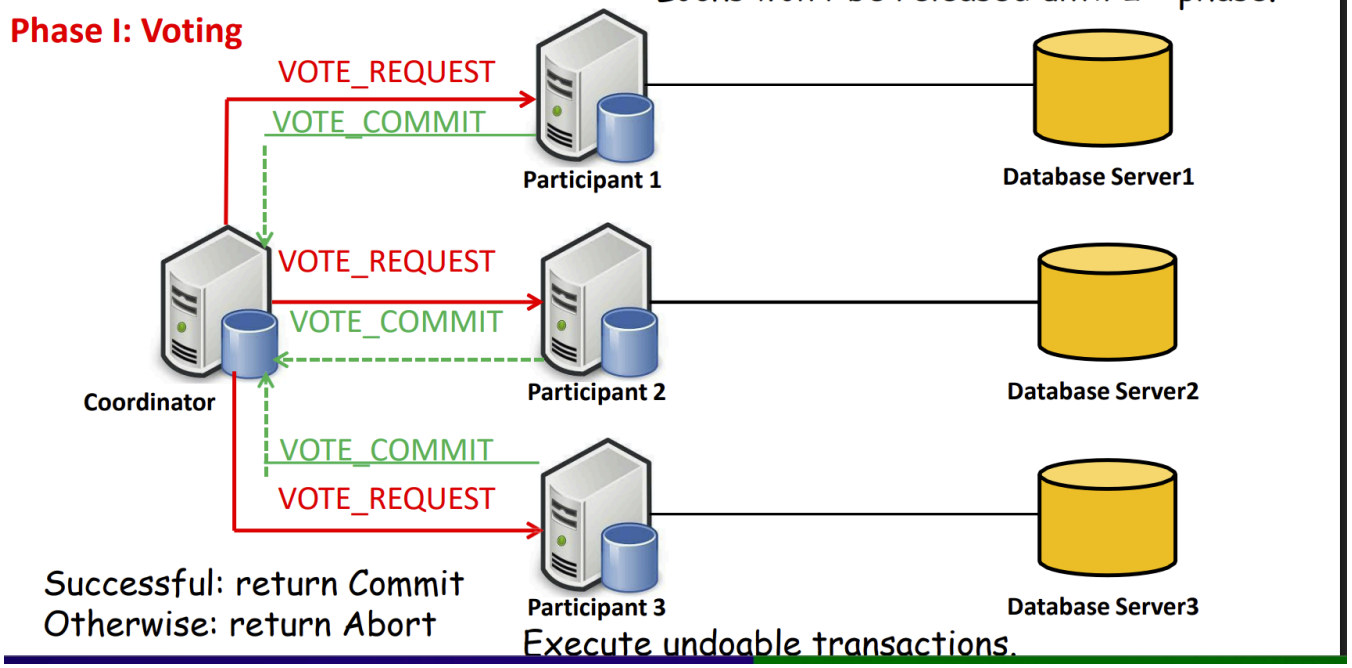
- Suppose that
  - The sequential execution of a program takes T_1 time units
  - The parallel execution on p processors/machines takes T_p time units.
- Suppose that out of the entire execution of the program, s fraction of it is not parallelizable while $1 - s$ fraction is parallelizable.
- Then we parallelize the program to SPEED UP (Amdahl's Formula)
  - $\frac{T_1}{T_p} = \frac{T_1}{\left(T_1 * s + T_1 * \frac{1-s}{p}\right)} = \frac{1}{s + \frac{1-s}{p}}$



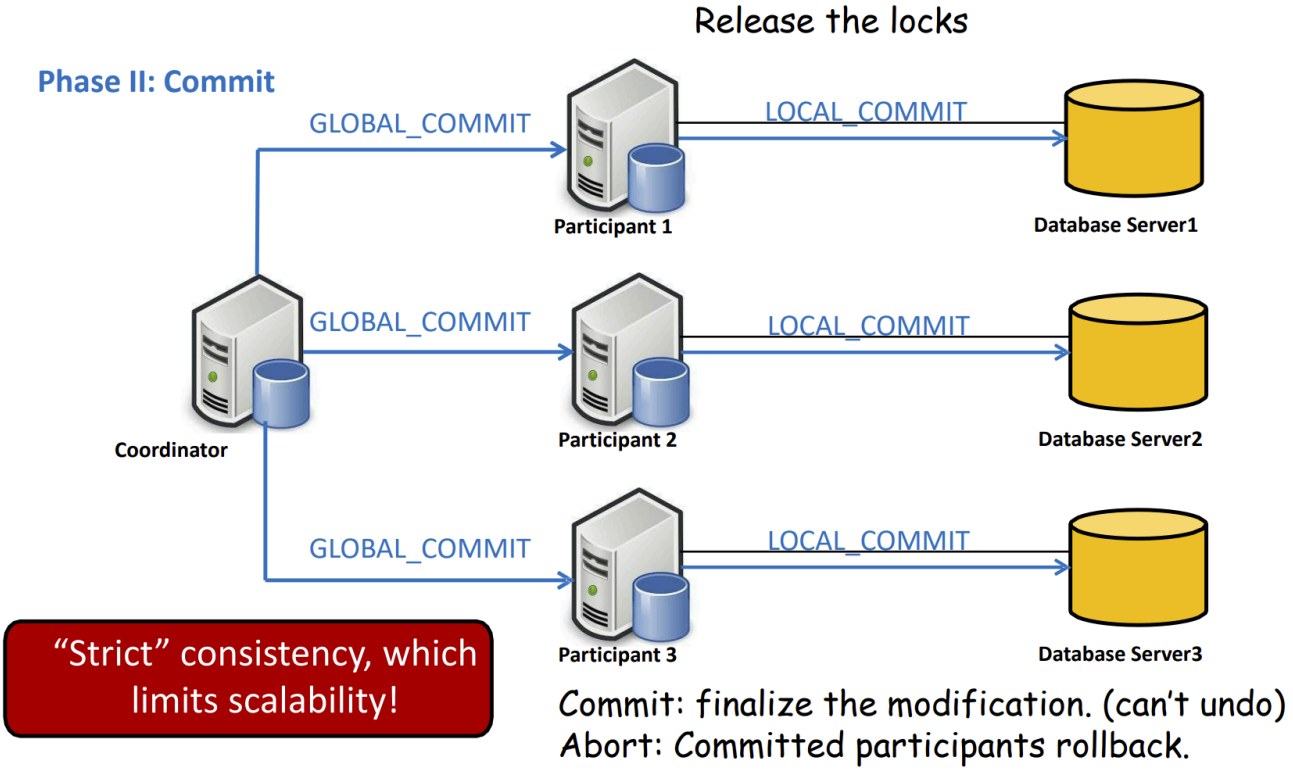
Two-Phase Commit Protocol

- The Two-Phase Commit Protocol (2PC) can be used to ensure atomicity and consistency.

Locks won't be released until 2nd phase.



Two-Phase Commit Protocol



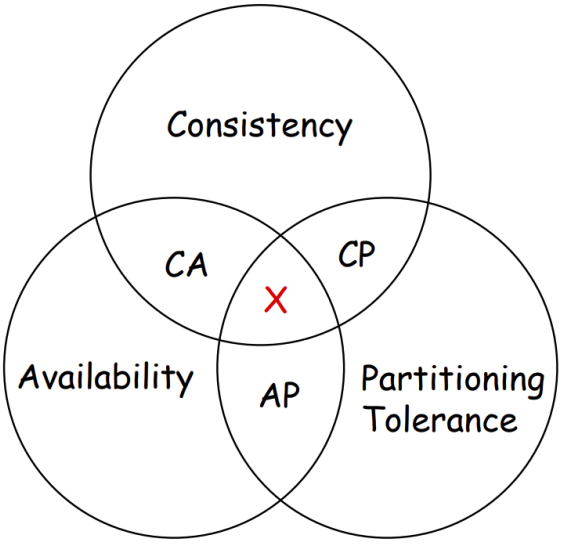
"Strict" consistency, which limits scalability!

Commit: finalize the modification. (can't undo)
Abort: Committed participants rollback.

CAP Theorem

- C: Consistency
  - Every read operation receives the most recently updated data or an error.
  - Different from Consistency in ACID
- A: Availability
  - Every database request receives a successful response, without guaranteeing it contains the most recently updated data.
- P: Partitioning tolerance
  - The cluster must continue to work despite any number of communication breakdowns between nodes in the system.

- Any distributed database with shared data can have at most two of the three desirable properties, C, A, or P.



- When a network fails, it must be decided whether to do ONE of the following:
  - Cancel the operation and thus decrease the availability but ensure consistency. (C+P)
  - Proceed with the operation and thus provide availability but risk consistency. (A+P)

From ACID to BASE Properties

- ACID prioritizes consistency over availability.
- BASE prioritizes availability over consistency.

- BASE is a RELAXED ACID on consistency.

- In particular, such databases apply BASE properties
  - Basically Availability: The database appears to work most of the time.
  - Soft-state: Data can have transient or temporary states that may change over time, even without external triggers or inputs. It describes the record's transitional state when several applications update it simultaneously.
  - Eventually Consistency: The record will achieve consistency when all the concurrent updates have been completed. Although the replicas may be different, they will be identical eventually.

NoSQL Key Features

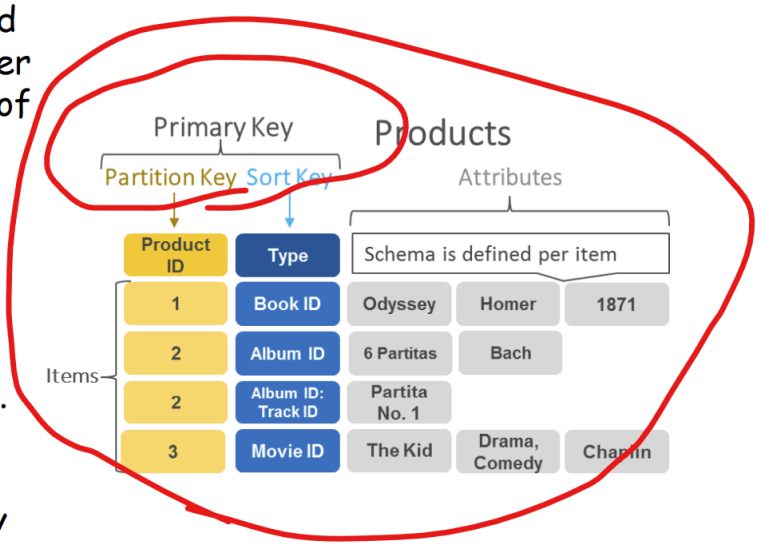
- Non-relational
- Does not require strong schema
- Data is replicated to multiple nodes and can be partitioned:
  - Down nodes can be easily replaced.
  - No single point of failure
- Horizontal scalable
- Cheap, easy to be implemented (open-source)
- Massive write performance
- Fast access

NoSQL Categories

- Key-value
  - Example: DynamoDB, Voldemort, Scalaris
- Column-based
  - Example: BigTable, Cassandra, Hbase
- Document-based
  - Example: MongoDB, CouchDB
- Graph-based
  - Example: Neo4J, InfoGrid

Key-Value

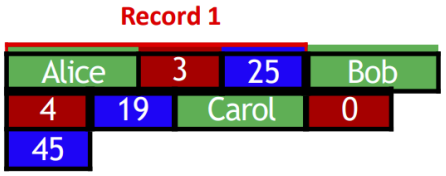
- Scaling to huge amounts of data
- Designed to handle massive load
- Based on Amazon's dynamo paper
- Data model: (global) collection of Key-value pairs
- Dynamo ring partitioning and replication
- Example: (DynamoDB)
  - The primary key is a composite of two keys: Partition Key and Sort Key.
  - The combination of the Partition Key and the Sort Key forms a unique primary key, which maps to a single value in the database.
  - Each item has its own schema.



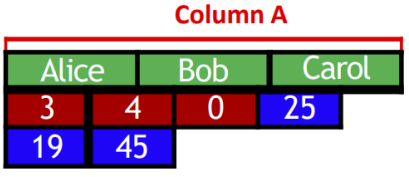
Column-based NoSQL

- Data model
  - Collection of Column Families
  - Column family = (key, value) where value = set of related columns
- Features
  - One column family can have variable numbers of columns
  - Cells within a column family are sorted "physically"

Alice	3	25
Bob	4	19
Carol	0	45



Row-Order



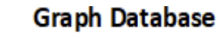
Columnar (or Column-Order)

Document-based

- Can model more complex objects
- Data model: collection of documents
- Document: JSON (JavaScript Object Notation is a data model, key-value pairs, which supports objects, records, structs, lists, array, maps, dates, Boolean with nesting), XML, other semi-structured formats
- Example: (MongoDB) document

Graph-based

- Focus on modeling the structure of data (interconnectivity)
- Scales to the complexity of data
- Inspired by Graph Theory
- Data model:
  - Nodes may have properties (including ID)
  - Edges may have labels or roles
  - Key-value pairs on both
- Interfaces and query languages vary
- Graph databases are powerful for graph-like queries (e.g., find the shortest path between two elements)
- Example: Neo4j, FlockDB, Pregel, InfoGrid …



handout 9

- Simple n-Period Moving Average
  - $F_t = \widehat{Y_t} = \frac{\sum_{i=t-n}^{t-1} Y_t}{n}$
  - $\sum_{i=t-n}^{t-1} Y_t$ is the sum of actual values in the previous n period

Weighted Moving Average Model

- Weighted n-Period Moving Average
  - $F_t = w_1 Y_{t-1} + w_2 Y_{t-2} + \cdots + w_n Y_{t-n}$
- Typically, weights are decreasing: $w_1 > w_2 > \cdots > w_n$
- Sum of the weights: $\sum_{i=1}^{n} w_i = 1$
- Flexible weights reflect relative importance of each previous observation in forecasting.

Simple Exponential Smoothing

- Moving Average technique that requires little record keeping of past data.
- Use a smoothing constant α with a value between 0 and 1.
- Applies α to most recent period and applies $1 - \alpha$ distributed to previous values that are used to predict the value at the most recent period.
  - α: the weight assigned to the latest period (smoothing constant)
  - Forecast = α (Actual value at $t-1$) + $(1-\alpha)$ (Forecast at $t-1$)
  - $F_t = \widehat{Y_t} = \alpha Y_{t-1} + (1-\alpha)\widehat{Y_{t-1}} = \widehat{Y_{t-1}} + \alpha\left(Y_{t-1} - \widehat{Y_{t-1}}\right)$
  - $\widehat{Y_1} = Y_1$
- Can also forecast for period $t-1$ plus α times the difference between the actual value and forecast in period $t-1$

Pattern-based Forecasting: Seasonal

- Deseasonalization
  - $Deseasonalized\ Data = \frac{Actual\ Value}{Seasonal\ Index}$
- Reseasonalization
  - $Reseasonalized\ Forecast = (Deseasonalized\ Forecast) * (Seasonal\ Index)$

How to calculate Seasonal Index

1. Calculate the mean for each cycle (e.g., year).
2. Calculate the proportion of the cycle for each season.
3. Calculate the average proportion for each season.

The result of Step 3 is the Seasonal Index of each season.

Step 1: mean(4 seasons) for each year

Step 2: season value / mean , for each year

Step 3: average proportion , for each season

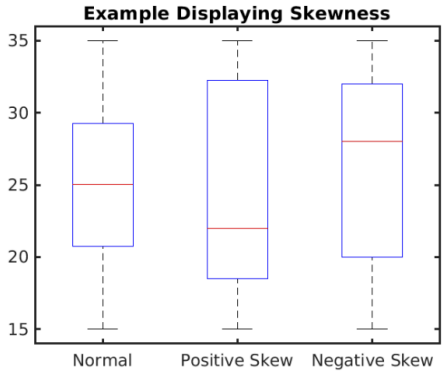
---

handout 10

Boxplot

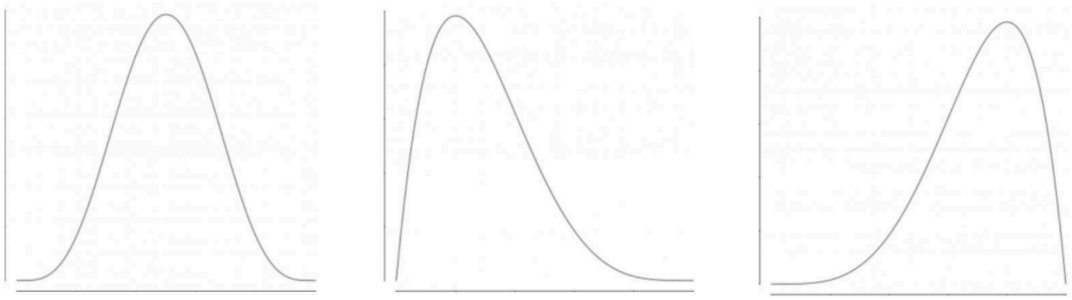
- A boxplot is a standardized way of displaying the dataset based on the five-number summary.
  - Minimum (Q_0 or 0-th percentile): the lowest data point in the dataset excluding any outliers.
  - Maximum (Q_4 or 100-th percentile): the highest data point in the dataset excluding any outliers.
  - Median (Q_2 or 50-th percentile): the middle value of the dataset.
    - $\left(\frac{(n+1)}{4} * 2\right)$-th number
  - First quartile (Q_1 or 25-th percentile): the median of the lower half of the dataset.
    - $\left(\frac{(n+1)}{4} * 1\right)$-th number
  - Third quartile (Q_3 or 75-th percentile): the median of the upper half of the dataset.
    - $\left(\frac{(n+1)}{4} * 3\right)$-th number

Boxplot can illustrate skewness



Bowley Skewness (Quartile skewness):

$$\frac{(Q_3 - Q_2) - (Q_2 - Q_1)}{Q_3 - Q_1}$$



No Skew | Positive Skew | Negative Skew

- $var(X) = \frac{\sum_{i=1}^{m}(x_i - \bar{x})^2}{m-1}$
- $m - 1$ instead of m: Bessel's Correction to remove bias.
- $cov(X, Y) = \frac{\sum_{i=1}^{m,}(x_i - \bar{x})(y_i - \bar{y})}{m-1}$

$$S_X = \begin{bmatrix} cov(X_1, X_1) & cov(X_1, X_2) \\ cov(X_2, X_1) & cov(X_2, X_2) \end{bmatrix}$$

$$S_X = \begin{bmatrix} 51.3 & 50.55 \\ 50.55 & 53.3 \end{bmatrix}$$

Given a mean-subtracted data matrix X, its covariance matrix $S_X = \frac{1}{m-1} X^T X$.