Week 8 – Lecture 1
Web-Scale Link Analysis
- Link Spam

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- Midterm
  - We will go over the questions
  - Simple correction will be done in the class if needed
  - Please return your answer sheet
  - You can browse during the GTAs' office hours
  - Your final exam is not a comprehensive test
- Extra help hours will be available this week
- PA3 will be posted: Team assignment (work with your term project team)

1

1

Topics discussed in the previous class

- Web-Scale Link Analysis – Dead end and Spider Trap
- Overview of Midterm

Topics to be covered

- Topic 1: Link Spam
- Midterm review (Last 30 minutes)

Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

Link spam

How can I obtain the first search result place?

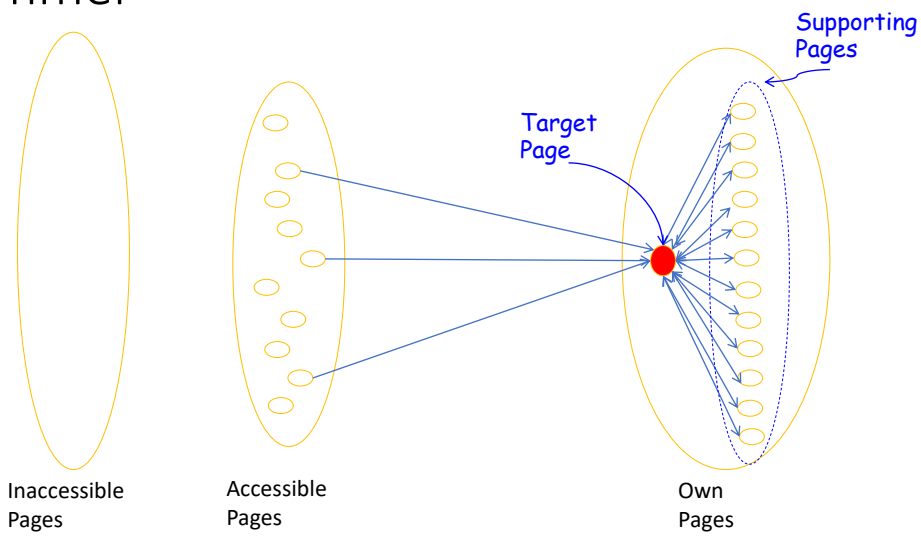
- The holiday season
- "Samsonite carry on luggage"
  - J.C. Penny was first on the list
    - Ahead of Samsonite.com
    - For months
- "black hat" optimization?
  - It is NOT illegal!
- "casual dresses", "evening dresses", "little black dress", or "cocktail dress"
  - 2,105 pages
  - Most of them were hosted by unrelated sites
  - Bounces back to J.C. Penny

https://www.nytimes.com/2011/02/13/business/13search.html

Architecture of a Spam Farm

- ***Spam Farm***
  - A collection of pages whose purpose is to increase the PageRank of a certain page or pages
- From the point of view of the **spammer**, the Web is divided into two parts
  - **Inaccessible pages**
    - The pages that the spammer cannot affect
    - Most of the Web
  - **Accessible pages**
    - Those pages that, while they are not controlled by the spammer, can be affected by the spammer

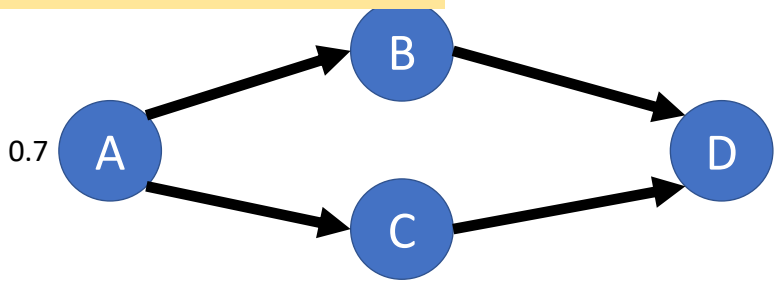
The Web from the point of view of the link spammer



Calculating the PageRank value

What is the Pagerank value of B after the current iteration?

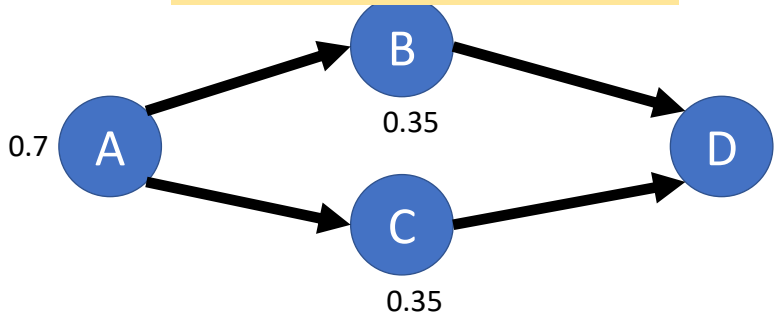
a. 0.7
b. 1
c. 0.35
d. 1.4



Calculating the PageRank value

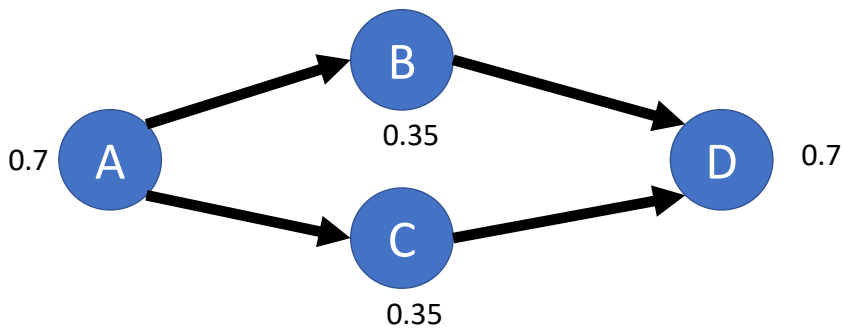
What is the pagerank value of D after the current iteration?

a. 0.7
b. 1
c. 0.35
d. 1.4



Understanding Spam Farm (1/2)

- Setting the links to the target page
  - Without link from outside, the spam farm is not useful
  - e.g. Blogs or news papers
    - Comments like "I agree. Please see my article at www.mySpamFarm.com"



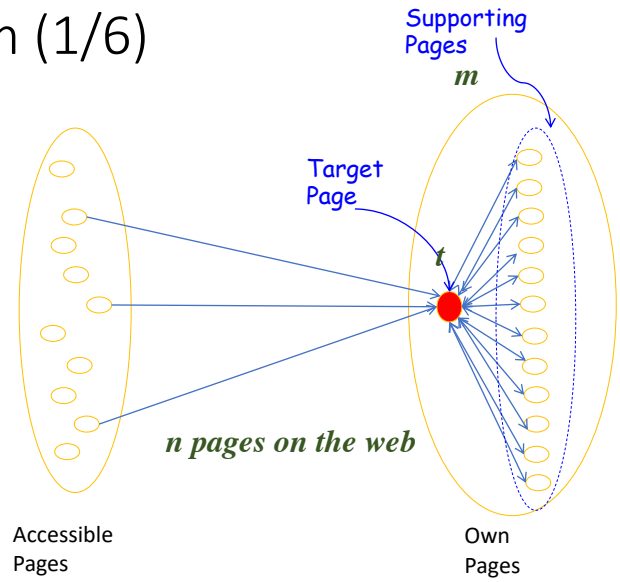
Understanding Spam Farm (2/2)

- There is **one page t**, the **target page**
  - Spammer attempts to place **as much PageRank as possible**

- There are a large number of **m supporting pages**
  - **Accumulate the portion of the PageRank** that is distributed equally to all pages
  - The fraction $1\text{-}\beta$ of the PageRank that represents surfers going to a random page
  - **Prevent the PageRank of t from being lost**
    - Note that all of the supporting pages links only to t

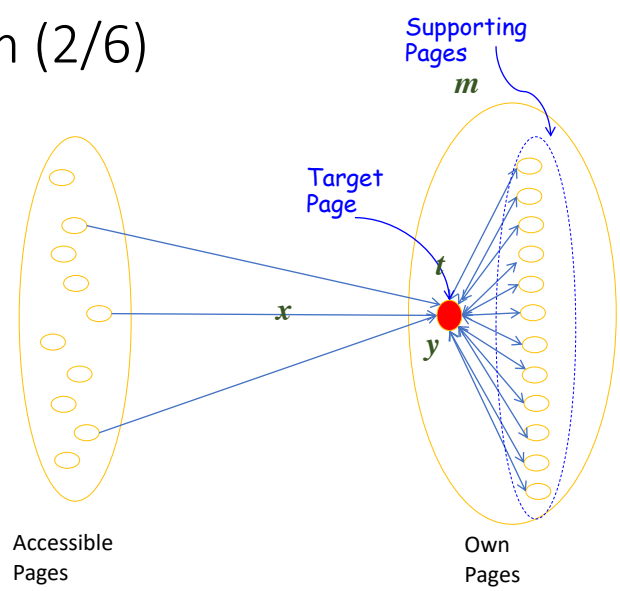
Analysis of a Spam Farm (1/6)

- A taxation parameter β
  - The fraction of a page's PageRank that gets distributed to its successors at the next round
- Let there be,
  - ***n* pages on the Web** in total
  - A target page ***t***
  - ***m* supporting pages**



Analysis of a Spam Farm (2/6)

- Let ***x*** be **the total amount of PageRank contributed by the <u>accessible pages</u>**
  - x is the sum over all accessible page p with a link to t, of the PageRank of p times β divided by the number of successors of p
- Finally, let ***y*** be the unknown PageRank of ***t***



Analysis of a Spam Farm (3/6)

- The PageRank of each supporting page
  - $\beta y/m+(1-\beta)/n$
- First term represents the **contribution from t**
  - βy is distributed to t's successors
- Second term is the supporting page's share of **the fraction $1-\beta$ of the PageRank that is divided equally among all pages on the Web**

Analysis of a Spam Farm (4/6)

- PageRank value of the target page t
  - y= β(*sum of the contributions from neighboring vertices*)+$(1-\beta)e/n$
- ***sum of the contributions from neighboring vertices (1+2)***

1. Contribution x from outside
   - x is the sum over all accessible page p with a link to t, of the PageRank of p times β divided by the number of successors of p
   - Assume that β is already multiplied
2. Contribution from the supporting pages
   - $m(\beta y/m+(1-\beta)e/n)$

- $(1-\beta)e/n$
  - The share of the fraction $1-\beta$ of the PageRank that belongs to t

  This amount is negligible

Analysis of a Spam Farm (5/6)

- From (1) and (2),

$$y = x + \beta m\left(\frac{\beta y}{m} + \frac{1-\beta}{n}\right) = x + \beta^2 y + \beta(1-\beta)\frac{m}{n}$$

$$y = \frac{x}{1-\beta^2} + c\frac{m}{n}$$

Where

$$c = \beta(1-\beta)/(1-\beta^2) = \beta/(1+\beta)$$

Analysis of a Spam Farm (6/6)

- If we choose $\beta=0.85$, then $1/(1-\beta^2)=3.6$
- $c= \beta/(1+\beta)=0.46$
- The structure has amplified the external PageRank contribution by 360%
- Also, it obtained an amount of PageRank that is 46% of the fraction of the Web, m/n, that is in the spam farm

Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

Combatting Link Spam

Combatting Link Spam

- **Detecting and eliminating link spam** have been critical for search engines
  - Just as it was critical to eliminate term spam in the previous decade
- Detecting particular structures
  - Spam farm
    - **One page links to a very large number of pages**
    - **Each of which links back to it**

Combatting Link Spam

- Modifying PageRank to lower the rank of link-spam pages automatically
  - **TrustRank**
  - **Measuring Spam mass**

TrustRank

- TrustRank is a **topic-sensitive PageRank**
  - **"topic" is a set of pages believed to be trustworthy (not spam)**
- Develop a suitable **teleport set of trustworthy pages**
  - Let humans examine a set of pages and decide which of them are trustworthy
- **Pick a domain** whose membership is controlled
  - University pages
  - .mil, or .gov

Calculating TrustRank (1/2)

- Then the topic-sensitive **PageRank for S** is the limit of the iteration,

$$v' = \beta M v + (1-\beta) e_S / |S|$$

- M is the transition matrix of the Web, and $|S|$ is the size of set S

Calculating TrustRank (2/2)

- Suppose we use $\beta=0.8$, and our trust rank is represented by the teleport set (trustworthy pages) ***S={B,D}***

$$\beta M = \begin{bmatrix} 0 & 2/5 & 5/4 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix}$$

$$v' = \begin{bmatrix} 0 & 2/5 & 5/4 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 1/10 \\ 0 \\ 1/10 \end{bmatrix}$$

$$\begin{bmatrix} 0/2 \\ 1/2 \\ 0/2 \\ 1/2 \end{bmatrix} \begin{bmatrix} 2/10 \\ 3/10 \\ 2/10 \\ 3/10 \end{bmatrix} \begin{bmatrix} 42/150 \\ 41/150 \\ 26/150 \\ 41/150 \end{bmatrix} \begin{bmatrix} 62/250 \\ 71/250 \\ 46/250 \\ 71/250 \end{bmatrix} \ldots \begin{bmatrix} 54/210 \\ 59/210 \\ 38/210 \\ 59/210 \end{bmatrix}$$

B and D get a higher PageRank than before

Spam Mass

- Measures the **fraction of its PageRank that comes from spam(non-trusted pages)** for each page
- For an arbitrary page p,
  - PageRank r
    - Computing ordinary PageRank
  - TrustRank t
    - Computing the TrustRank based on some teleport set of trustworthy pages
  - The spam mass
    - $(r - t)/r$

- A **negative or small positive spam mass**
  - p is probably not a spam page
- **Page with high spam mass score**
  - Should be eliminated

Example

- Suppose that both the **PageRank** and **TrustRank** were computed
- Teleport set was page **B** and **D**
- **Question 1**
  - Page B is not a spam page (True/False)
- **Question 2**
  - Page A has a higher chance to be a link spam compared to the page C. (True/False)

Web Page	PageRank	TrustRank	SpamMass
A	3/9	54/210	0.229
B	2/9	59/210	-0.264
C	2/9	38/210	0.186
D	2/9	59/210	-0.264

Example

- Suppose that both the **PageRank** and **TrustRank** were computed
- Teleport set was page **B** and **D**
  - Which nodes are not the link spams?
    - B and D
    - C has lower chance to be the link spam compared to A

Web Page	PageRank	TrustRank	SpamMass
A	3/9	54/210	0.229
B	2/9	59/210	-0.264
C	2/9	38/210	0.186
D	2/9	59/210	-0.264

Week 8 – Lecture 2
Scalable Clustering with MapReduce

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- PA2
  - Submission deadline extended
  - Extra help hours will be available

1

Topics discussed in the previous class

- Topic 1: Link Spam
- Midterm review (Last 30 minutes)

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 8-B Fall 2021 Computer Science

Topics to be covered

- Clustering and scalable solutions

Part 1. Large Scale Data Analytics

2. Clustering

Scalable *K*-Means Clustering

Clustering: Core concept

- Set of N-dimensional vectors
  - Can be in the order of millions
- **Group (or cluster) them** based on their **proximity** (or similarity) to each other in an N-dimensional space
  - Vectors or objects in a cluster (or group) are more similar to each other than in any other group

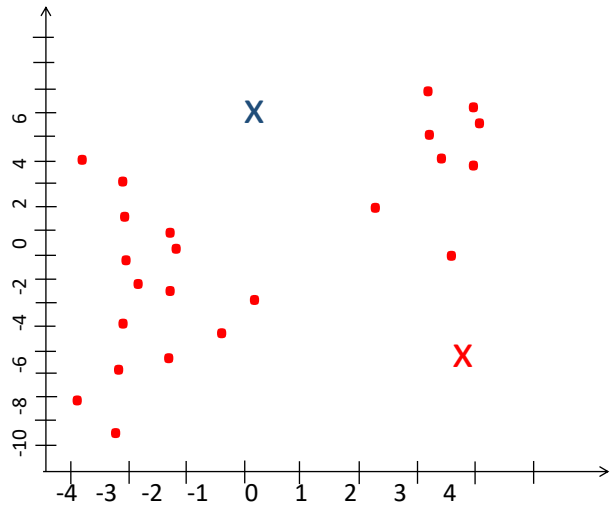
Clustering: Applications

- **Anomaly detection**
  - Fraud detection
- Recommendation systems
- **Medical imaging**
- Market research
- **Human genetic clustering**

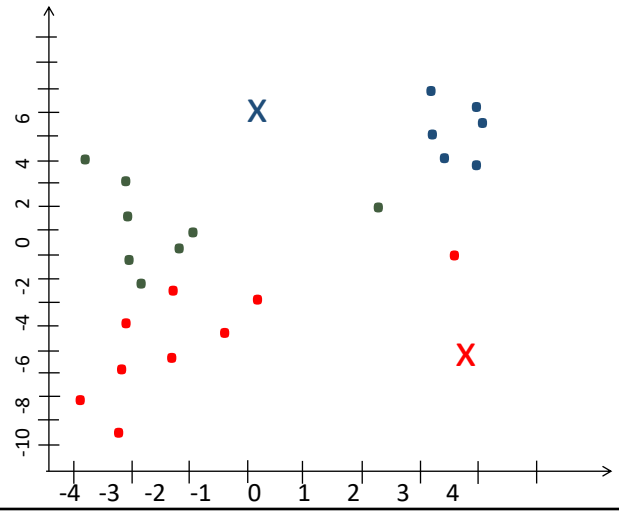
k-Means Clustering

- **Unlabeled** dataset
- Aims to partition m **observations** into k **clusters**
  - Each observation belongs to the cluster with the nearest mean

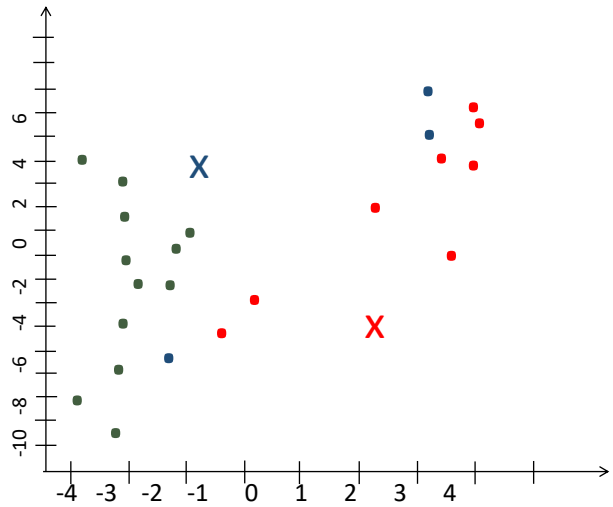
Concept: k-Means Clustering (1/4)



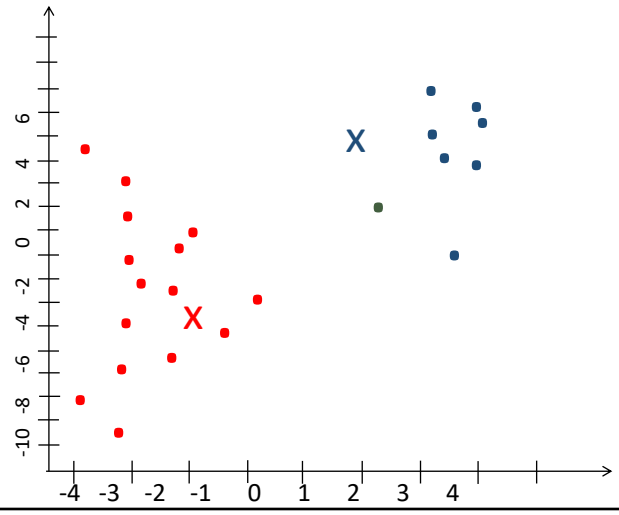
Concept: k-Means Clustering (2/4)



Concept: k-Means Clustering (3/4)



Concept: k-Means Clustering (4/4)



k-Means algorithm - Lloyd's Algorithm (1/2)

- Input
  - k (number of clusters)
  - Training set $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots x^{(m)}\}$

    $x^{(i)} \in R^n$

    (drop x_0= 1 convention)

k-Means algorithm - Lloyd's Algorithm (2/2)

- Randomly initialize K cluster centroids

$$\mu_1, \mu_1, \ldots \mu_k \in R^n$$

```
repeat{
  for i = 1 to m
    c(i):=index (from i to K) of cluster centroid
                                        closest to x(i)
  for k = 1 to K
     μk:= average (mean) of points assigned to cluster k

}
```

Cost function

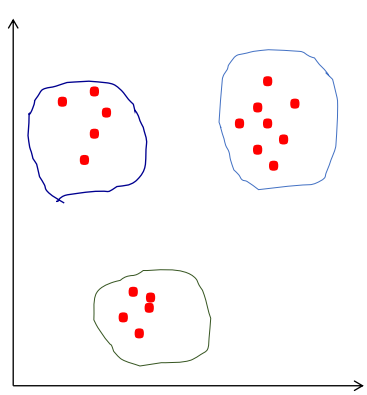
- The **objective is to find**:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} (\| x - \mu_i \|)^2$$

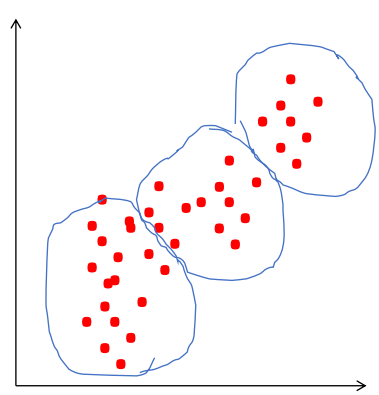
- Where μ_i is the mean of points in S_i

k-Means for non-separated clusters

Separated clusters

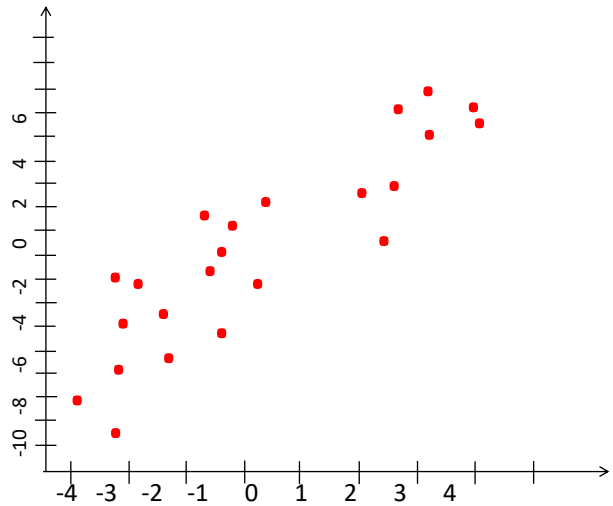


Non-Separated clusters

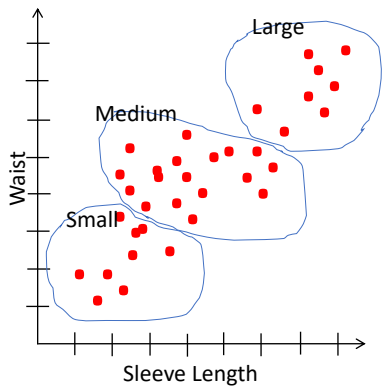


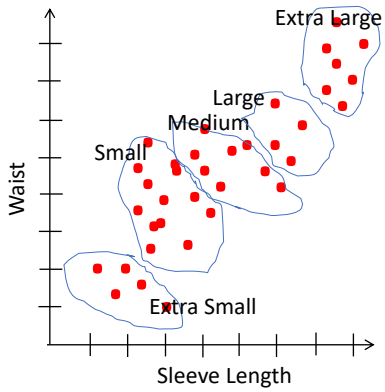
How to choose the number of clusters

- Value k in the algorithm



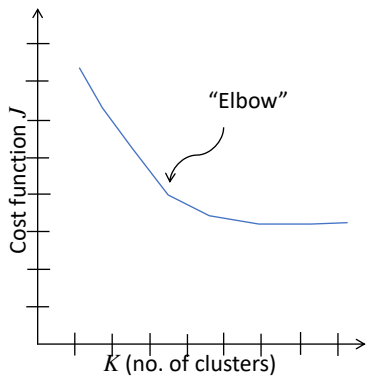
Choosing the value K (2/2)

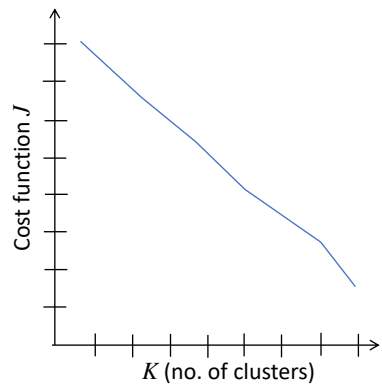




Choosing the value K (1/2)

Elbow Method





Evaluation of your model

- Are your clusters good?

- **Extrinsic Measures**
  - Require ground truth labels
  - Adjusted Rand index, Fowlkes-Mallows scores, Mutual information based scores, Homogeneity, Completeness and V-measure
- **Intrinsic Measures** *(more popular cases)*
  - Does not require ground truth labels
  - Silhouette Coefficient, Calinski-Harabasz Index, Davies-Bouldin Index

Distance Measures

- Euclidean Distance
- Manhattan Distance
- Cosine Distance
- Hamming Distance
- Jaccard Dissimilarity
- Edit Distance
- Smith Waterman Similarity
- Image Distance
- Etc.

Part 1. Large Scale Data Analytics
2. Clustering

Implementing Scalable k-Means using MapReduce
(a) Canopy clustering algorithm

Scalable k-Means using MapReduce

- Computing the Euclidean distance between the sample vectors and the centroids can be **parallelized**
  - By splitting the data into individual subgroups and clustering samples in each subgroup separately
    - **By the mapper**
- Recalculating new centroid vectors
  - Divide the sample vectors into subgroups
  - Compute the sum of vectors in each subgroup in parallel
  - Reducer will add up the partial sums and compute the new centroids
- **Question**
  - **How much of data should be transferred for this steps?**
  - **How do we effectively parallelize this process?**

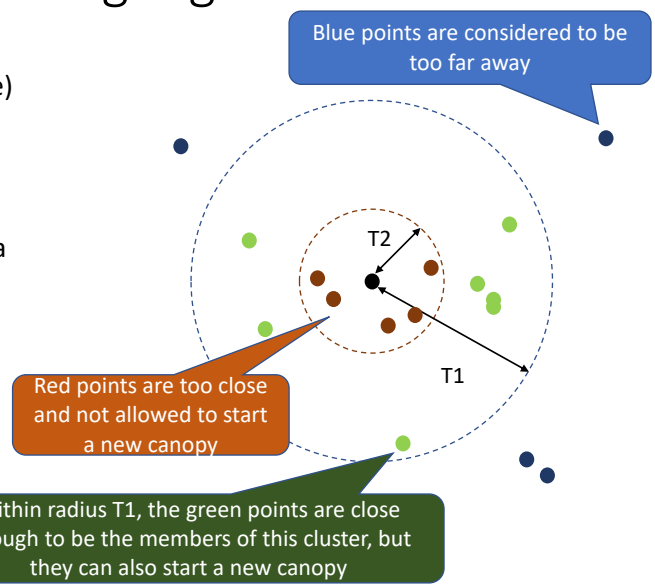
Canopy clustering algorithm

- **Unsupervised pre-clustering algorithm**
  - Defines **the proximity regions**
    - **Instead of starting with random points**
  - Often used as **preprocessing step** for k-Means
- Major goal of this algorithm is to **speed up** clustering operations on large datasets

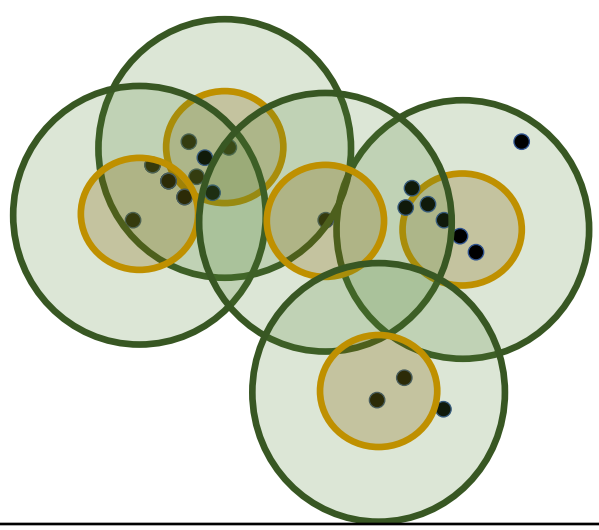
General Canopy Clustering Algorithm

- Using two thresholds T_1 (the loose distance) and T_2 (the tight distance), where $T_1 > T_2$

1. Begin with the set of data points to be clustered
2. **Remove** a point from the set, beginning a new "canopy"
3. For each point left in the set, assign it to the new canopy **if the distance is less than the loose distance T_1**
   - If the distance of the point is **less than** the tight distance T_2, **remove it** from the original set
4. Repeat steps 2-3-4, **until there are no more data points** in the set to cluster



Example of canopies



Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 1. Canopy generation phase
  - **Mapper**
    - Each mapper processes **a subset of the total points** and applies the chosen distance measure and thresholds
    - Input: a subset of the total points
    - Functionality: run the canopy algorithm over a subset
    - Output: <"a_constant_key", canopy_centroid>
      - we will use only one reducer
  - **Reducer**
    - A single reducer
    - Input: <"a_constant_key", [a list of canopy_centroids]>
    - Functionality: run the canopy algorithm over the local canopy centroids
    - Output: Final set of canopy centroids

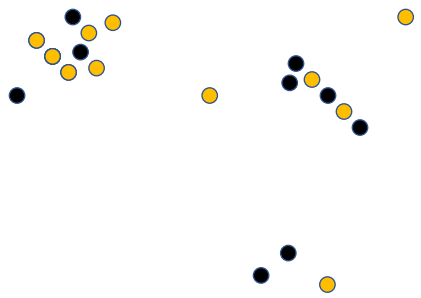
COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 8-B Fall 2021 Computer Science

Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 2. Clustering phase (k-Means)
  - **Mapper**
    - Each mapper reads the **Canopies produced by the first phase (using a Distributed Cache)**
    - Input: a subset of the total points
    - Functionality: find the closest centroid to this point
    - Output: <canopy_centroid_ID, a_point_info>
  - **Reducer**
    - Input: <canopy_centroid_ID, [a list of points in this cluster]>
    - Functionality: calculate new centroid for this cluster
    - Output: a new centroid
- **Repeat Step 2 with the new set of centroids** (from reducer)

Orange points: Mapper A

Black points: Mapper B



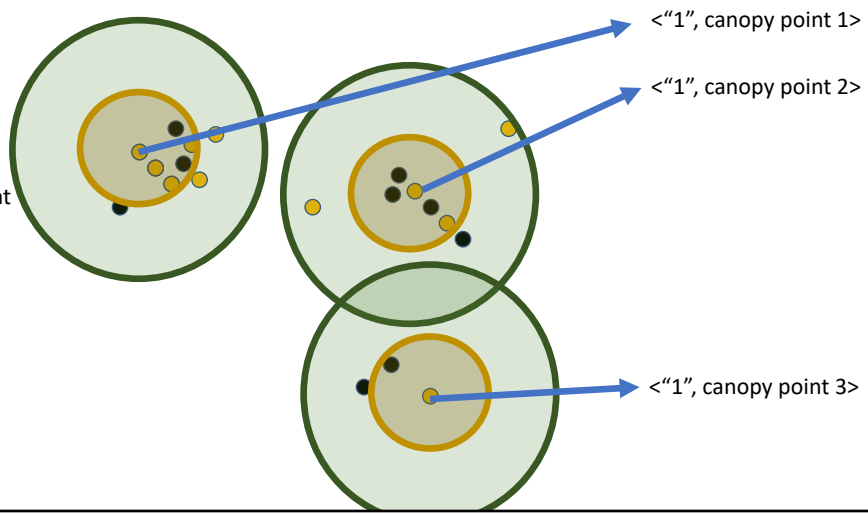
28

Phase 1: Mapper A

Orange points: Mapper A

Black points: Mapper B

Mapper A will select the canopy point 1, 2, and 3 and emits them with a constant key



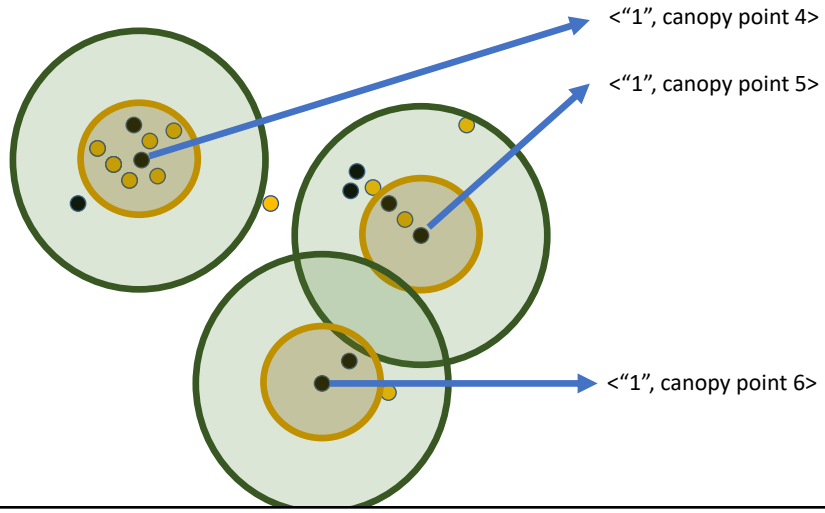
29

Phase 1: Mapper B

Orange points: Mapper A

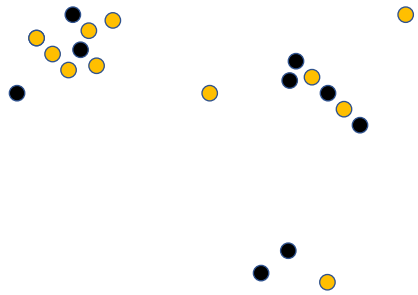
Black points: Mapper B

Mapper B will select the canopy point 1, 2, and 3 and emits them with a constant key



Phase 1: Single Reducer

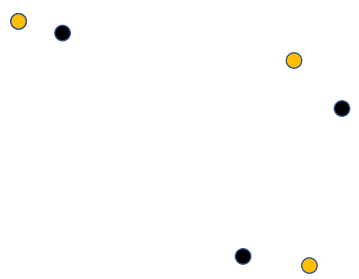
<“1”, [canopy point 1, canopy point 2, canopy point 3, canopy point 4, canopy point 5, canopy point 6] >



Phase 1: Single Reducer

Canopy points only

<"1", [canopy point 1, canopy point 2, canopy point 3, canopy point 4, canopy point 5, canopy point 6] >

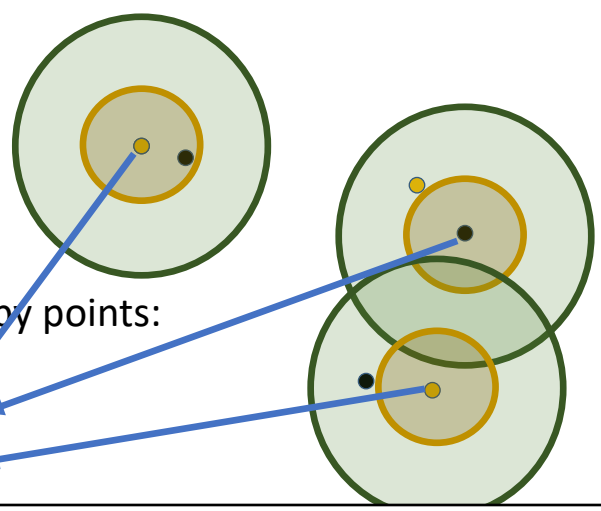


Now run the Canopy Algorithm here

Phase 1: Single Reducer

Canopy points only

<"1", [canopy point 1, canopy point 2, canopy point 3, canopy point 4, canopy point 5, canopy point 6] >



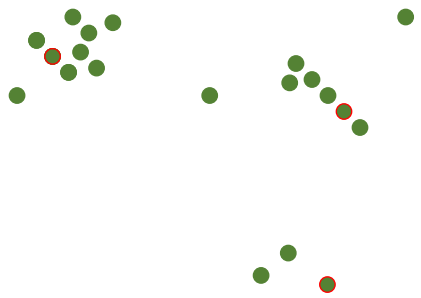
Final set of Canopy points:
Canopy point a
Canopy point b
Canopy point c

Phase 2: Mapper

Red points: canopy centroids

Green points: non-centroid points

Go over all of the points assigned to the current mapper
For the point k, calculate the distance between k to all of the canopy centroids and select the closest centroid
Emit the pair <closest centroid, k>

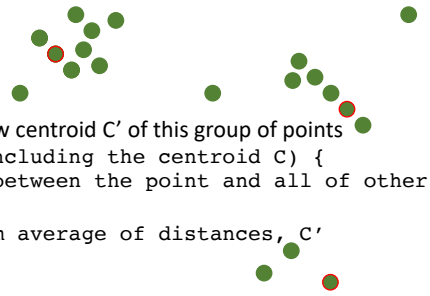


Phase 2: Reducer

Red points: canopy centroids

Green points: non-centroid points

All the points with the same closest centroid will be transferred to the same reducer
Input to a reducer will be: <centroid C, [a list of points sharing c as the closest centroid]>

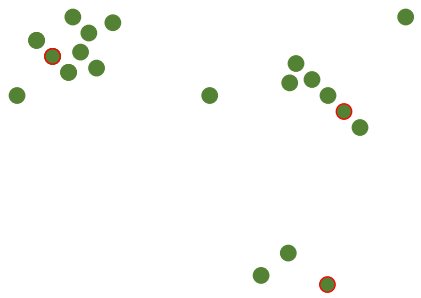


Functionality of this reducer: calculating a new centroid C' of this group of points

```
for all the points in this list (including the centroid C) {
   calculate average of distances between the point and all of other points in this list
}
select a point that has the minimum average of distances, C'
emit C'
```

Phase 2 will be iterated

Phase 2 will be started with the set of C' as new centroids
Repeat map and reduce until the set of centroids converges



REPEAT Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 1. Canopy generation phase
  - **Mapper**
    - Each mapper processes **a subset of the total points** and applies the chosen distance measure and thresholds
    - Input: a subset of the total points
    - Functionality: run the canopy algorithm over a subset
    - Output: <"a_constant_key", canopy_centroid>
      - we will use only one reducer
  - **Reducer**
    - A single reducer
    - Input: <"a_constant_key", [a list of canopy_centroids]>
    - Functionality: run the canopy algorithm over the local canopy centroids
    - Output: Final set of canopy centroids

REPEAT

Using Canopy Clustering Algorithm to Perform k-means using MapReduce

- Step 2. Clustering phase (k-Means)
  - **Mapper**
    - Each mapper reads the **Canopies produced by the first phase (using a Distributed Cache)**
    - Input: a subset of the total points
    - Functionality: find the closest centroid to this point
    - Output: <canopy_centroid_ID, a_point_info>
  - **Reducer**
    - Input: <canopy_centroid_ID, [a list of points in this cluster]>
    - Functionality: calculate new centroid for this cluster
    - Output: a new centroid
- **Repeat Step 2 with the new set of centroids** (from reducer)

Part 1. Large Scale Data Analytics

2. Clustering

Scalable k-Means algorithm
(b) k-Means++

This material is built based on,

- Arthur, D.; Vassilvitskii, S. (2007). "*k*-means++: the advantages of careful seeding" (PDF). *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035
- Bahmani, B., Moseley, B., Vattani, A., Kumar, R. and Vassilvitskii, S., 2012. Scalable k-means++. *arXiv preprint arXiv:1203.6402*.
- Apache Spark Mllib: Clustering
  - https://spark.apache.org/docs/latest/ml-clustering.html

k-Means algorithm- Lloyd's Algorithm: Strengths

- Embarrassingly parallel
  - little or no effort is needed to separate the problem into a number of parallel tasks
- Converges to a local minima
- $O(nkdi)$ runtime
  - n: number of data points
  - k: number of clusters
  - d: dimensionality of a data point
  - i: number of iterations

k-Means algorithm- Lloyd's Algorithm: Weaknesses

- $O(nkdi)$ runtime
- Large number of local minima
- Many local minima are poor
- k is unknown

The K-Means++ Algorithm

- Avoiding cold-start improves results
  - Reducing the number of total iterations
- Initialize cluster centers sequentially
- Only the first center is randomly selected
  - Each further center is selected probabilistically to be far from existing centers
- Result of this is an $O(\log k)$ approximation to the global optima

The K-Means++ Algorithm: Step-by-Step description

Step 1. **Choose one center uniformly at random** from among the data points

Step 2. For each data point *x*, **compute D(*x*)**, the shortest distance between *x* and the nearest center that has already been chosen

Step 3. Choose one new data point at random as a new center, using a weighted probability distribution where a point *x* is chosen with **probability proportional to D(*x*)^2.** ***(D^2 weighting)***

- ***D^2 weighting***: Take a new center, choosing x with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$

Step 4. Repeat Steps 2 and 3 until *k* centers have been chosen

Step 5. Now that the initial centers have been chosen, **proceed using standard k-Means**

More info: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

The K-Means++ Algorithm: How can we improve this?

- GOAL 1: Can we reduce the number of iterations for initializing centroids,
  - Can we select multiple centroids at a time?
- GOAL 2: While considering un-uniformly distributed dataset,
  - Non-uniform selection?
- GOAL 3: With a reasonable approximation guarantees?

Scalable K-Means++(A.K.A. K-Means||) Algorithm

- Parallel version for initializing the centers
  - Oversampling factor $l = \Omega(k)$
- Select an initial center (uniformly at random)
- Computes initial cost of the clustering after this selection, Ψ
- Proceeds in log Ψ iterations
  - Samples each x with probability $ld^2(x, C)/\phi_X(C)$

Scalable K-Means++(A.K.A. K-Means||) Algorithm

- Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of points in the d-dimensional Euclidean space
- Let k be a positive integer specifying the number of clusters
- Let $\| a - b \|$ denote the Euclidean distance between a and b
- For a point x and a subset $Y \subseteq X$ of points, the distance is defined as

$$d(x, Y) = min_{y \in Y} \| x - y \|$$

- For a subset $Y \subseteq X$, its centroid be given by
  - $centroid(Y) = \frac{1}{|Y|} \sum_{y \in Y} y$
- Let C=$\{c_1, c_2, \dots c_k\}$ be a set of points and let $Y \subseteq X$
- The cost of Y with respect to C as
- $\phi_Y(C) = \sum_{y \in Y} d^2(y, C) = \sum_{y \in Y} \min_{i=1,\dots,k} \| y - c_i \|^2$

Scalable K-Means++(A.K.A. K-Means||) Algorithm

1: C ← sample a point uniformly at random from X
2: ψ←ϕ_X(C)
3: for O(log ψ) times do
4: C← sample each point x ∈ X independently with probability of $ld^2(x,C)/\phi_X(C)$
5: C ← C ∪ $C_{\phi X(C)}$
6: end for
7: For x∈C, set w to be the number of points in X closer to x than any other point in C
8: Recluster the weighted points in C into k clusters

- l: Number of points chosen in each iteration
- Total number of points in C is $llog\psi$ $(> \mathrm{k})$
- To reduce the number of centers
  - Step 7 assigns weights to the points in C
  - Step 8 reclusters these weighted points to obtain k centers

48

48

K-Means|| Algorithm: A Parallel Implementation

1: C ← sample a point uniformly at random from X
2: ψ←ϕ_X(C)
3: for O(log ψ) times do
4: C← sample each point x ∈ X independently with probability of $ld^2(x,C)/\phi_X(C)$
5: C ← C ∪ $C_{\phi X(C)}$
6: end for
7: For x∈C, set w to be the number of points in X closer to x than any other point in C
8: Recluster the weighted points in C into k clusters

- Line 2: calculate the cost function
- Line 4: each mapper can sample independently
- Line 7: identical to the line 2
  - Each mapper calculate the local

49

49

K-Means|| Algorithm: A Parallel Implementation

1: C ← sample a point uniformly at random from X
2: ψ←ϕ_X(C)
3: for O(log ψ) times do
4: C ← sample each point x ∈ X independently with probability of $ld^2(x, C)/\phi_X(C)$
5: C ← C ∪ $C'_{\phi X(C)}$
6: end for
7: For x∈C, set w to be the number of points in X closer to x than any other point in C
8: Recluster the weighted points in C into k clusters

- Line 2: calculate the cost function
  - Mapper: Go over the inputsplit and find the closest centroid and aggregate the shortest distances
  - Reducer aggregate the global cost
- Line 4: each mapper (RDD) can sample independently
  - Random number generation per points, r. if r>=probability, sample it otherwise drop it
- Line 7: identical to the line 2

K-Means|| Algorithm: Properties

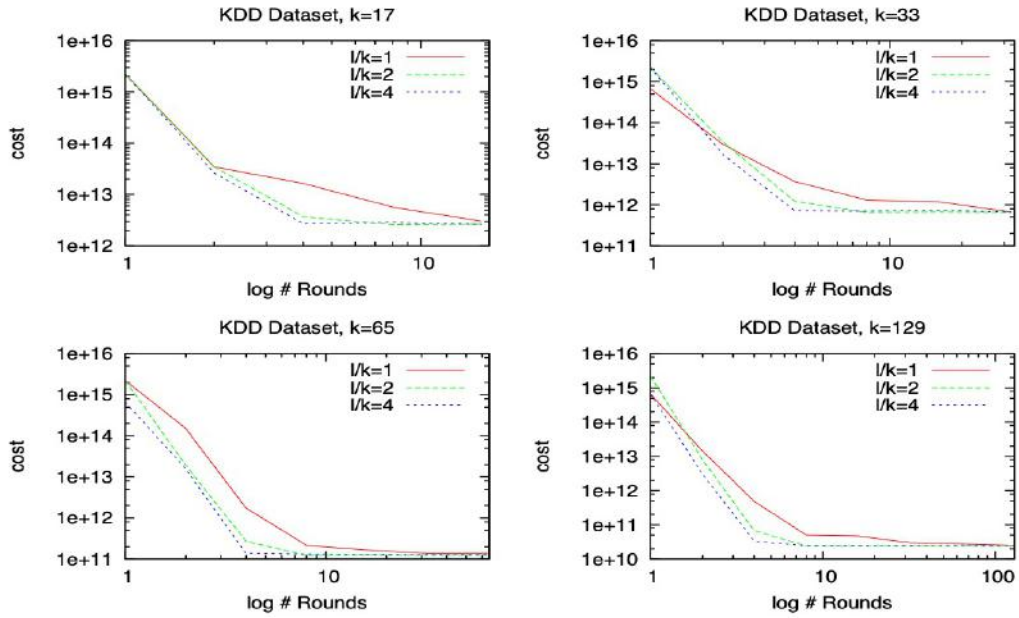
- Requires $O(log\psi)$iterations
- A constant number of iterations is usually enough
- Creates an $O(\log k)$ approximation to the global optima
- Uses results from probability and sequence theory
- Each iteration reduces error by a constant amount and a term proportional to the global error
- Expected value of the cost function approaches a multiple of the global optima
- Performance (with KDDCup 1999 data)
- Paper: Bahmani, B., Moseley, B., Vattani, A., Kumar, R. and Vassilvitskii, S., 2012.

Scalable k-means++. *arXiv preprint arXiv:1203.6402*

- Data: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

	$k = 20$	$k = 50$	$k = 100$
Random	176.4	166.8	60.4
k-means++	38.3	42.2	36.6
k-means‖ $\ell = 0.5k, r = 5$	36.9	30.8	30.2
k-means‖ $\ell = 2k, r = 5$	23.3	28.1	29.7

Note: all values scaled down by 10^{10}


KDD Dataset, k=17
KDD Dataset, k=33
KDD Dataset, k=65
KDD Dataset, k=129
l/k=1
l/k=2
l/k=4
cost
log # Rounds
1e+16
1e+15
1e+14
1e+13
1e+12
1e+11
1e+10
1
10
100

Questions?

Week 9 – Lecture 1
Recommendation Systems

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- PA2
  - Submission deadline extended
  - Extra help hours will be available
- Term project proposal
  - Check if your team members are still available
  - If your team has less than 3 members, please let me know ASAP!

1

Topics discussed in the previous class

- Large scale clustering algorithms
  - Canopy algorithm
  - K-means++ and K-means||
  - Implementing with MapReduce

Topics to be covered

- Term project proposal
- Introduction to Recommendation Systems

Term project: Proposal

- Submit via Canvas
  - Team submission
  - 1,000 ~ 1,200 words
  - Document only
- Components of proposal
  - Title
  - Problem formulation
  - Your strategy to solve the problem
  - Your dataset
  - Timeline
  - Bibliography

Dataset example - Web data

- Web data commons
  - http://webdatacommons.org
  - Hyperlink Graph
    - 3.5 billion web pages and 128 billion hyperlinks
  - Product data corpus
    - 5.6 million product records
  - What are the example project?
  - Web analysis
    - Where is your web page?
    - Using In-degree, out-degree, harmonic centrality, PageRank…
  - https://www.wim.uni-mannheim.de/fileadmin/lehrstuehle/ki/pub/Meusel-etal-GraphStructureOfTheWeb.pdf

Dataset example: Enron Email data

- Enron Email dataset
  - 1,227,255 emails with 493,384 attachments
  - https://www.cs.cmu.edu/~./enron/

  - Possible project
    - Spam filter

Dataset example: Network data

- SNAP
  - Stanford Large Network Dataset Collection
  - https://snap.stanford.edu/data/index.html

  - Amazon network
  - Twitter messages
  - Memetracker messages
  - Online community (Reddit, and Flicker)
  - And more

Dataset example: Kaggle

- https://www.kaggle.com/
- Read Kaggle's questions
- Pros: Many sets of interesting data
- Cons: Size of the most of datasets is compact.
- Please check the data size and make sure it is a "Big Data" problem

Dataset: etc.

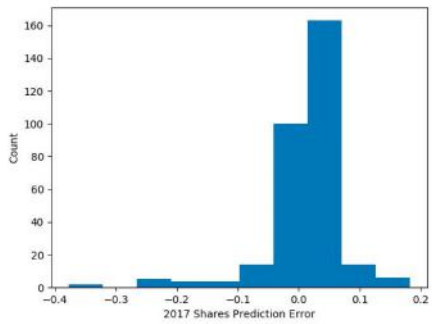
- AWS public data collection
- Reddit comments
  - https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/
  - ~1.7 billion comments (250GB compressed)
- Wikipedia dump
  - https://meta.wikimedia.org/wiki/Data_dumps

Example project

- Predicting Weekly Domestic Box Office Share with Convolutional Neural Networks
  - CS435 2018: Alper Karslioglu, Ben Goodwin, Calvin Le, David Im
- Problem formulation
  - Why is "the movie recommendation" important?
- Methodology
  - Predicting Box Office share using Neural Networks
  - IMDB and Google Trends API

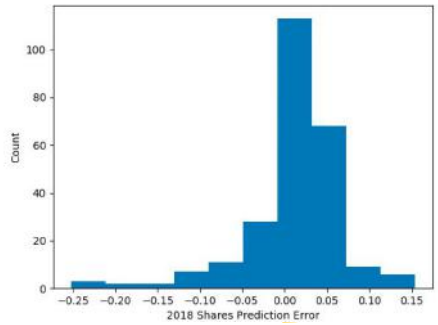
10

Example project



312 movies in 2017

246 movies in 2018



Overall Mean Absolute Error: 0.0459

11

Example project from Fall 2020

- The 2020 Election According to Twitter: Sentiment Analysis vs. the Polls and Election Results
- Predicting the Stock Market using Spark and Machine Learning
- Creating and Testing Stock Trading Algorithms Using Spark
- News and Elon Musk Tweets Effect on Tesla Stock Price
- Exploration of Movie Recommender System Implementations
- Basic Airline Delay Prediction in the United States
- Collaborative Filtering Recommendation Algorithm on Yelp Dataset
- Data Analysis of Wikipedia Network of Top Categories
- Application of Generative Adversrial Networks to Colorize Greyscale Images

Example projects from Fall 2019

- Guess Who's Coming to Dinner
- Incident Analysis and Prediction in San Francisco
- Bitcoin Transaction Patterns
- Finding Helpful Yelp Reviews Using Textual Analysis
- Identifying, Predicting, & Blocking Clickbait Headlines
- Optimal Time to reach YouTube Trending list
- Does initial article tone affect publicity for world events?
- Composition ratio drift during a painters life

Example projects from Fall 2018

- The Six Degrees of Separation in the Movie Industry
- Weather or Not, There is Crime - Relationships between crime and weather patterns in Chicago
- Recognizing ASL Hand Signs Using Spark Deep Learning
- Using Apache Spark to Predict Ozone, Nitrogen Dioxide, and Sulfur Dioxide Effect on Temperature
- Analyzing Song Lyrics to Predict Song Popularity Across 53 Countries
- Shrimp and Whales- An in-depth look into the purchasing traits and group dynamics of Steam accounts
- Top “Invisible” Reasons Businesses Get Their Reviews

Example projects from Spring 2017

- Product rating and classification using NLP
- Amazon DVD Sales Analysis Based on Reviews and Genre
- Aiming for business success using Yelp and Big Data
- Finding Global Health Outcome Correlations
- Analysis of Bike Sharing Service in the Bay Area
- NBA Team Success and Relation to Hometown
- Using Big Data Analytics to Track the Cost/Benefit Ration of Higher Education
- Too Many Games, Not Enough Time
- Recommendation System using Amazon Reviews
- Analysis Prediction and Visualization of Chicago Crime Rate by Area in Real-time
- Hashtag/tweet location correlation analysis
- Effectiveness of intrusion detection using old datasets

Example projects from Spring 2016

- Combined Fuel Economy of Medium Duty Hybrid Trucks
- Where to Live
- Analysis of Reddit Comments for Word Popularity and Trends Using Hadoop
- Music Recommendation Based on Genre-using Million song data
- Trend Analysis to Improve Donations
- Movie recommendation system based on movie attributes and human-movie interaction information
- Song Genre generation
- Predicting Wind Speeds Based on Atmospheric Data
- Darknet Market Survival and Similarity Analysis
- Trends in Baby Names and Predicting Popular Names
- Predicting PM2.5
- Predicting Stock Similarity Using K-Means
- Analyzing Trends on Twitter

Example projects from Spring 2015

- Similarity and Clustering on The Million Song DatasetWhere to Live
- Analysis on Methods Used to Analyze Eclipse and Mozilla Bug Data
- Movie Recommendation using Collaborative Filtering
- A Retrospective Study Of Change In The English Language Via Textual Analysis
- 1929 - 2009 Climate Visualization and Predictive Analysis
- Wikipedia Link analysis
- Google book N gram analysis
- Analysis of stock market trends
- Hazardous Materials Source Mapping
- Wikipedia Page Traffic Statistics Analysis
- Million Song Dataset Geographical Analysis
- Trending Topics and Page Count Prediction on Wikipedia Traffic Log Data

What is a "good" proposal?

- Your proposal must demonstrate
  - **Importance**
  - **Challenging**
  - **Feasibility**
  - Also, your topic should be appropriate to the academic environment
- Final grading (12/20) will be based on
  - **Relevance**
  - **Completeness**
  - **Challenging**
- Your approach
  - Well-structured
  - Your approach should lead you to "good results"
  - Then, what is the "good results"?
    - Interesting knowledge
    - Demonstrating validity of your approach
    - Providing estimable accuracy/performance

Part 1. Large Scale Data Analytics
3. Recommendation Systems
Introduction

This material is built based on

- Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining* (ICDM '08). IEEE Computer Society, Washington, DC, USA, 263-272. DOI=http://dx.doi.org/10.1109/ICDM.2008.22
- Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills, Advanced Analytics with Spark, O'Reilly, 2015

"What percentage of the top 10,000 titles in any online media store (Netflix, iTunes, Amazon, or any other) will rent or sell at least once a month?"

a) 25%
b) 50%
c) 75%
d) More than any of above answers

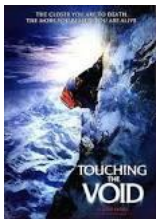
The long tail phenomenon [1/2]

- Distribution of numbers with a portion that has a large number of occurrences far from the "head" or central part of the distribution
  - The vertical axis represents popularity
  - The items are ordered on the horizontal axis according to their popularity
  - The long-tail phenomenon forces online institutions to recommend items to individual users

Erik Brynjolfsson, Yu (Jeffrey) Hu, and Duncan Simester. 2011. Goodbye Pareto Principle, Hello Long Tail: The Effect of Search Costs on the Concentration of Product Sales. *Manage. Sci.* 57, 8 (August 2011), 1373-1386. DOI=http://dx.doi.org/10.1287/mnsc.1110.1371

The long tail phenomenon [2/2]

- "Touching the Void", Joe Simpson, 1988



- "Into Thin Air: A Personal Account of the Mt. Everest Disaster" , Jon Krakauer, 1997

Recommendation systems

- Seek to predict the "rating" or "preference" that a user would give to an item

Applications of Recommendation Systems

- **Product recommendations**
  - Amazon or similar online vendors
- **Movie recommendations**
  - Netflix offers its customers recommendations of movies they might like
- **News articles**
  - News services have attempted to identify articles of interest to readers based on the articles that they have read in the past
  - Blogs, YouTube

Netflix Prize

- The Netflix Prize challenge concerned recommender systems for movies (October, 2006)
- Netflix released a training set consisting of data from almost 500,000 customers and their ratings on 18,000 movies.
  - More than 100 million ratings
- The task was to use these data to build a model to predict ratings for a hold-out set of 3 million ratings

Large Scale Data Analytics

3. Recommendation Systems

Background: Data Similarity

Jaccard Coefficient (a. without description)

- Compare two sets P and Q with the following formula:

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|}$$

- **Measures the fraction of the data that is shared between P and Q**
- Compared to all data available in the union of these two sets.
- What are P and Q?
  - Set of tokens from Strings
  - Complete description about data (candidates)

Example

- ***StringJaccard***

$S(c_1)$ = {Thomas, Sean, Connery}
$S(c_2)$ = {Sir, Sean, Connery}
What is the string Jaccard coefficient between c1 and c2?

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|} = 2/4 = 0.5$$

Jaccard Coefficient (b. with description)

- Given two candidates, the Jaccard coefficient of two candidates *c1* and *c2* is given by,

$$DescriptionJaccard(c1,c2) = \frac{|OD(c1) \bigcap OD(c2)|}{|OD(c1) \bigcup OD(c2)|}$$

Example

- Now, specify the parts of a person's name as
  - **title, firstname, middlename, and lastname**

$OD(c_1) = \{(\textbf{firstname}, Thomas), (\textbf{middlename}, Sean), (\textbf{lastname}, Connery)\}$
$OD(c_2) = \{(\textbf{title}, Sir), (\textbf{middlename}, Sean), (\textbf{lastname}, Connery)\}$

$DescriptionJaccard(c_1, c_2) = 2/4$

Question

- *DescriptionJaccard* and *StringJaccard* have the same value. Is this always true?

a) Yes

b) No

Example

- ***What if "Sean" would have been put in the firstname/middlename attribute?***

$OD(c_1) = \{(middlename, Thomas), (\mathbf{firstname, Sean}), (lastname, Connery)\}$
$OD(c_2) = \{(title, Sir), (\mathbf{middlename, Sean}), (lastname, Connery)\}$

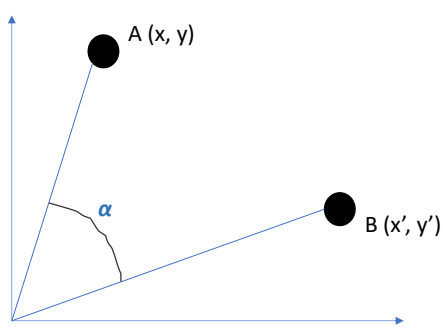
$DescriptionJaccard(c_1, c_2) = 1/5$

Deficiencies of the Jaccard Similarity

- Some attribute is more descriptive
  - **`Title` is less descriptive** than `firstname` and `lastname`
- Very sensitive to **typographical errors** in single tokens
  - ***Shean Conery*** and ***Sean Connery*** have a similarity of zero.

Cosine similarity

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors***



Cosine similarity - Continued

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors*** as

$$CosineSimilarity(V,W) = \cos(\alpha) = \frac{V \bullet W}{\|V\| \times \|W\|}$$

Where $\|V\|$ is the length of the vector $V = [a,b,c..]$ computed as

$$\sqrt{a^2 + b^2 + c^2 + \ldots}$$

Cosine similarity - Continued

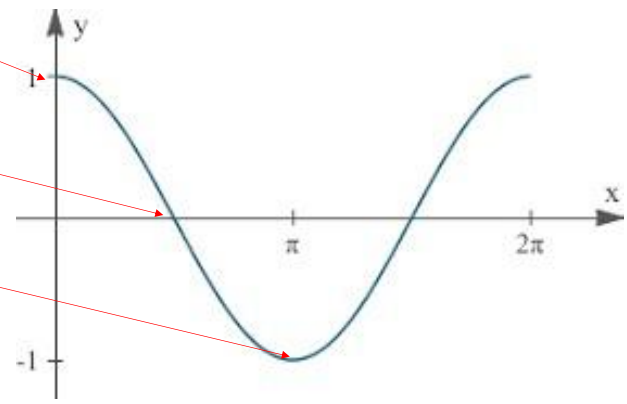
- The vectors V and W
  - Tokens in a string
  - Descriptions of a candidate
- The ***d* dimensions** of these vectors correspond to all ***d* distinct tokens** in a set of strings.
  - Denoted as D
- For a large database, d may be large
  - V and W have high dimensionality d

Cosine similarity - Continued

If the degree between the vectors is 0
→Cosine similarity: 1

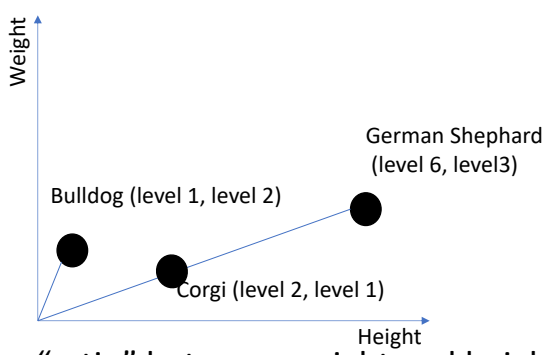
If the vectors are orthogonal
→ Cosine similarity: 0

If the vectors are opposite
→ Cosine similarity : -1



Cosine similarity - Continued

- Which pair of breeds has the highest cosine similarity?



Corgi

Bulldog

German Shephard

- Preserving the "ratio" between weight and height

Wait! They don't look similar enough..

Weight of Token

- Vector contains a **weight** for each of the d distinct tokens
- How to measure the weight?
  - **Measuring frequency**
  - **Term frequency – inverse document frequency** (tf-idf)

Term frequency

- **Number of times** that term t occurs **in the document d**
  - Raw term frequency: $f_{t,d}$
  - Boolean "frequencies" $tf(t,d)$ is 1 if t occurs in d and 0 otherwise
  - e.g. Logarithmically scaled frequency:

$$tf(t,d) = 1 + \log f_{t,d}$$

    - 0 if $f_{t,d}$ is zero

Example: What is the raw $tf_{"American", C3}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

Example: What is the raw $tf_{"American", C3}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

$tf_{"American", C3} = 1$

Inverse document frequency

- **Assigns higher weights** to tokens that **occurs less frequently in the scope of all candidate descriptions**

$$idf(t,D) = \log_{10} \frac{N}{|\{d \in D : t \in d\}|}$$

- Where, N is the total number of documents in the corpus
- Number of documents where the term t appears (i.e. $tf(t,d) \neq 0$)

Example: What is the $idf_{"American", D}$? (D: this corpus)

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

Example: What is the $idf_{"American", D}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

$idf_{"American", D} = log(10/3)$

tf-idf weighting

- The product of its *tf* weight and its *idf* weight

$$W_{t,d} = (1 + \log_{10} tf_{t,d}) \times \log_{10}(N / df_t)$$

For the total number of documents, N

- If the numbers of terms are different in the documents, you should normalize $tf_{t,d}$ to $(tf_{t,d} / (\text{number of words in } d))$
- Best known weighting scheme in information retrieval
  - Note: the "-" in *tf-idf* is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Example: What is the $tf\text{-}idf_{"American", C3,D}$ and $tf\text{-}idf_{"American", C8,D}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John
Hancock
Liberty
…

Example: What is the $tf\text{-}idf_{"American", C3,D}$ and $tf\text{-}idf_{"American", C8,D}$?

CID	Name
C1	Allstate
C2	American Automobile Association
C3	American National Insurance Company
C4	Farmers Insurance
C5	GEICO
C6	John Hancock Insurance
C7	Liberty Insurance
C8	Mutual Insurance of American Life Insurance
C9	Safeway Insurance Group
C10	Westfield

Q
Allstate
American
Automobile
Association
National
Insurance
Company
Farmers
GEICO
John

$tf\text{-}idf_{"American", C3,D} = (1+\log(1/4)) \times \log(10/3)$

$tf\text{-}idf_{"American", C8,D} = (1+\log(1/6)) \times \log(10/3)$

Example continued

- Compute the similarity between the two strings s1="Farmers Insurance", s2 = "Liberty Insurance"
- Term vector = (Allstate, American, Automobile, Association, National, Insurance, Farmers, Liberty, ….)
- *s1="Farmers Insurance" = (0, 0, 0, 0, 0, 1, 1, 0 …)*
- *s2="Liberty Insurance" = (0, 0, 0, 0, 0, 1, 0, 1, …)*
- Next step: find the weights

Example continued

CID	Name
c1	Allstate
c2	American Automobile Association
c3	American National Insurance Company
c4	Farmers Insurance
c5	GEICO
c6	John Hancock Insurance
c7	Liberty Insurance
c8	Mutual Insurance of America Life Insurance
c9	Safeway Insurance Group
c10	Westfield

Q	V	W
Allstate	0	0
America	0	0
Automobile	0	0
Association	0	0
National	0	0
Insurance		
Company	0	0
Farmers		0
GEICO	0	0
John	0	0
Hancock	0	0
Liberty	0	
…		

Example continued

- Compute the similarity between the two strings *s1="Farmers Insurance", s2 =" Liberty Insurance"*

- Six of the candidates contain the token "***Insurance***".
- $idf_{\text{"Insurance"}, D} = \log_{10}(10/6) = 0.78$
- $idf_{\text{"Farmers"}, D} = \log_{10}(10/1) = 1$
- $idf_{\text{"Liberty"}, D} = \log_{10}(10/1) = 1$

- tf-$idf_{\text{"Farmers"}, c4} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/1) = 0.7$
- tf-$idf_{\text{"Insurance"}, c4} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/6) \approx 0.55$
- tf-$idf_{\text{"Liberty"}, c7} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/1) = 0.7$
- tf-$idf_{\text{"Insurance"}, c7} = (1+\log_{10} 1/2) \text{ x } \log_{10}(10/6) \approx 0.55$

Example continued

CID	Name
c1	Allstate
c2	American Automobile Association
c3	American National Insurance Company
c4	Farmers Insurance
c5	GEICO
c6	John Hancock Insurance
c7	Liberty Insurance
c8	Mutual Insurance of America Life Insurance
c9	Safeway Insurance Group
c10	Westfield

Q	V	W
Allstate	0	0
America	0	0
Automobil e	0	0
Association	0	0
National	0	0
Insurance	0.55	0.55
Company	0	0
Farmers	0.7	0
GEICO	0	0
John	0	0
Hancock	0	0
Liberty	0	0.7
…		

Example continued

- Compute the similarity between the two strings **s1=Farmers Insurance**, **s2 = Liberty Insurance**

$$CosSimilarity(V,W) = \cos(\alpha) = \frac{V \bullet W}{\| V \| \times \| W \|}$$

$$= \frac{0.55 \times 0.55}{\sqrt{0.55^2 + 0.7^2}^2} \cong 0.38$$

- What is the ***Jaccard similarity*** for the same case?

Questions?

Week 9 – Lecture 2
Recommendation Systems

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- Term project proposal
  - Check if your team members are still available
  - If your team has less than 3 members, please let me know ASAP!

1

Topics discussed in the previous class

- Term project proposal
- Introduction to Recommendation Systems

Topics to be covered

- Recommendation Systems: Collaborative Filtering
- Evaluation Methods

Large Scale Data Analytics

3. Recommendation Systems

Collaborative Filtering

REVIEW

Jaccard Coefficient (a. without description)

- Compare two sets P and Q with the following formula:

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|}$$

- **Measures the fraction of the data that is shared between P and Q**
- Compared to all data available in the union of these two sets.
- What are P and Q?
  - Set of tokens from Strings
  - Complete description about data (candidates)

REVIEW

Example

- ***StringJaccard***

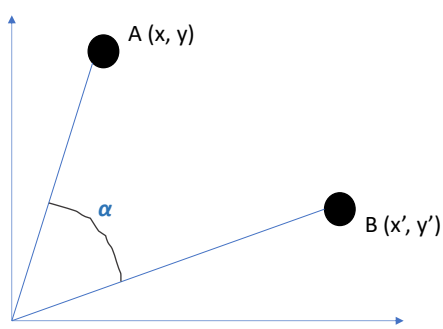
$S(c_1)$ = {Thomas, Sean, Connery}
$S(c_2)$ = {Sir, Sean, Connery}
What is the string Jaccard coefficient between c1 and c2?

$$StringJaccard(P,Q) = \frac{|P \cap Q|}{|P \cup Q|} = 2/4 = 0.5$$

REVIEW

Cosine similarity

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors***



REVIEW

Cosine similarity - Continued

- Given two n-dimensional vectors V and W, the ***cosine similarity*** computes ***the cosine of the angle α between these two vectors*** as

$$CosineSimilarity(V,W) = \cos(\alpha) = \frac{V \bullet W}{\|V\| \times \|W\|}$$

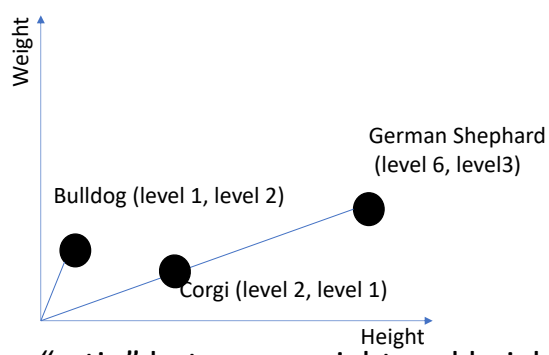
Where $\|V\|$ is the length of the vector $V = [a,b,c..]$ computed as

$$\sqrt{a^2 + b^2 + c^2 + \ldots}$$

REVIEW

Cosine similarity - Continued

- Which pair of breeds has the highest cosine similarity?



Corgi

Bulldog

German Shephard

- Preserving the "ratio" between weight and height

REVIEW

Weight of Token

- Vector contains a **weight** for each of the d distinct tokens
- How to measure the weight?
  - **Measuring frequency**
  - **Term frequency – inverse document frequency** (tf-idf)

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 9-B Fall 2021 Computer Science

Methodologies for the Recommendation Systems

- Content-based filtering
  - Based on a description of the item and a profile of the user's preferences
  - Learning a classifier for the user's preference based on an item's features
  - User's preferences (e.g. previous ratings)
  - Example: Since Alice liked an oversized tennis racket before, she will like "an oversized badminton racket"
- Collaborative filtering
  - "*People who agreed in the past will agree in the future*"
  - The system generates recommendations using only information about rating profiles for different users or items
  - Example: "Since Alice and Bob showed a similar ranking patterns for movies, Alice will like the movie that Bob liked yesterday."

Collaborative filtering

- Focus on the **similarity** of the **different customers' user ratings** for items
- Users are **similar if their rating vectors are close** according to some distance measure
  - E.g. Jaccard or cosine distance
- Collaborative filtering
  - The process of identifying similar users and recommending what similar users like

Measuring similarity between users

- How to measure similarity of users or items from their rows or columns in the utility matrix?
  - Jaccard Similarity for A and B: 1/5 (without considering the weight)
  - Jaccard Similarity for A and C: 2/4 (without considering the weight)
- For user A, user C **might have similar opinion** than user B
- Can user C provide a prediction for A?

	AV	EG	ET	TW	FF1	FF2	FF3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Cosine similarity (1/2)

- We can treat blanks as a 0 values
  - A = (4,0,0,5,1,0,0), B = (5,5,4,0,0,0,0)
- The **cosine of the angle** between A and B is

$$\frac{4 \times 5}{\sqrt{4^2+5^2+1^2}\sqrt{5^2+5^2+4^2}} = 0.380$$

	AV	EG	ET	TW	FF1	FF2	FF3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Cosine similarity (2/2)

- We can treat blanks as 0 values
  - A = (4,0,0,5,1,0,0), C = (0,0,0,2,4,5,0)
- The **cosine of the angle** between A and C is

$$\frac{5 \times 2 + 1 \times 4}{\sqrt{4^2+5^2+1^2}\sqrt{2^2+4^2+5^2}} = 0.322$$

- A is slightly closer to B (0.380) than to C (0.322)

Then, what will you recommend to the user A?

	AV	EG	ET	TW	FF1	FF2	FF3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Normalizing ratings (1/3)

- Generous vs. Harsh Ratings
- What if we **normalize** ratings by subtracting from each rating the average rating of that user?
- Some rating (very low) will turn into negative numbers
- If we take the cosine distance, the opposite views of the movies will have vectors in **almost opposite directions**
  - It can be as far apart as possible

Normalizing ratings (2/3)

- What if we **normalize** ratings by subtracting from each rating the average rating of that user?

	AV	EG	ET	TW	FF1	FF2	FF3	Average
A	4 (12/10-10/3 = 2/3)			5	1			10/3
B	5 (15/3 – 14/3 = 1/3)	5 (15/3 – 14/3 = 1/3)	4					14/3
C				2	4	5		11/3
D		3 (6/2 – 6/2 = 0)					3	6/2

Normalizing ratings (3/3)

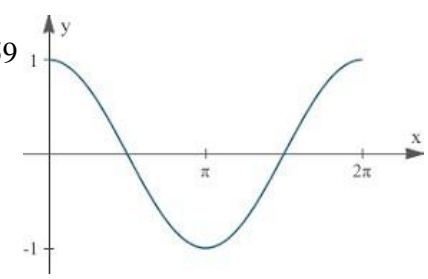
- The cosine of the angle between A and B

$$\frac{(2/3)\times(1/3)}{\sqrt{(2/3)^2+(5/3)^2+(-7/3)^2}\sqrt{(1/3)^2+(1/3)^2+(-2/3)^2}}=0.092$$

	AV	EG	ET	TW	FF1	FF2	FF3
A	2/3			5/3	-7/3		
B	1/3	1/3	-2/3				
C				-5/3	1/3	4/3	
D		0					0

- The cosine of the angle between A and C

$$\frac{(5/3)\times(-5/3)+(-7/3)\times(1/3)}{\sqrt{(2/3)^2+(5/3)^2+(-7/3)^2}\sqrt{(-5/3)^2+(1/3)^2+(4/3)^2}}=-0.559$$



- A and C are much further apart than A and B.
- Neither pair is very close
- **A and C disagree on the two movies** they rated in common, while **A and B give similar scores** to the one movie they rated in common

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3			5/3	-7/3		
B	1/3	1/3	-2/3				
C				-5/3	1/3	4/3	
D		0					0

Computational complexity

- Worst case
  - ***O(MN)*** where M is the number of customers and N is the number of product catalog items
  - It examines ***M* customers** and up to ***N* items** for each customer

Computational complexity (1/3)

- The average customer vector is **extremely sparse!**
  - The algorithm's performance tends to be closer to ***O(M+N)***
  - Scanning every customer
    - ***O(M)* not *O(MN)***
    - Almost every customer has very small N
  - Few customers who have purchased or rated a significant percentage of items
    - Requires $O(N)$
  - **10 million customers and 1 million items?**

Computational complexity (2/3)

- We can reduce M by:
  - **Randomly sampling** the customers
  - Discarding customers with few purchases
- We can reduce N by:
  - Discarding **very popular or unpopular items**
  - **Partitioning the item space** based on the product category or subject classification

Computational complexity (3/3)

- Dimensionality reduction techniques can reduce M or N by a large factor
  - **Clustering**
  - Principal component analysis

Disadvantage of space reduction

- Reduced recommendation quality
  - Sampled customer
    - More similar customers will be dropped
  - Item-space partitioning
    - It will restrict recommendations to a specific product or subject area
  - Discarding most popular or unpopular items
    - They will never appear as recommendations

Large Scale Data Analytics

3. Recommendation Systems

Amazon.com : Item-to-item collaborative filtering

This material is built based on,

- Greg Linden, Brent Smith, and Jeremy York, "Amazon.com Recommendations, Item-to-Item Collaborative Filtering" IEEE Internet Computing, 2003

- **Amazon.com** uses recommendations as a targeted marketing tool
  - Email campaigns
  - Most of their web pages

Item-to-item collaborative filtering

- It does **NOT** match the **user to similar customers**
- Item-to-item collaborative filtering
  - **Matches each of the user's purchased and rated items to similar items**
  - Combines those similar items into a recommendation list

Determining the most-similar match

- The algorithm builds a **similar-items table**
  - By finding items that **customers tend to purchase** **together** in the customer's history
- How about building a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair?
- Many product pairs have no common customer
  - If you already bought a TV today, will you buy another TV again today?

Co-occurrence in a product-to-product matrix

- Suppose that there are 4 users A, B, C, D, and E with 5 products P1, P2, P3, P4, and P5
- A purchased P1, P2, P3
- B purchased P2, P3, P5
- C purchased P3, P4
- D purchased P2, P4

P1

P2

P3

P4

P5

Co-occurrence in a product-to-product matrix

- Suppose that there are 4 users A, B, C, D, and E with 5 products p1, p2, p3, p4, and p5
  - A purchased p1, p2, p3
    - (P1,P2) (P2,P1): recorded as 1
    - (P2,P3)(P3,P2): recorded as 1
    - (P1,P3)(P3,P1): recorded as 1
  - B purchased p2, p3, p5
  - C purchased p3, p4
  - D purchased p2, p4

	p1	**p2**	**p3**	**p4**	**p5**
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Computing similarity

- Using **cosine measure**
  - Measuring the similarity between two items
  - M dimensions correspond to customers who have purchased that item

Building a Similarity product-to-product matrix

- Similarity between products
  - P1: T-shirt
  - P2: video cable
  - P3: projector
  - P4: tv
  - P5: smart tv
- P1 = (0,1,1,0,0)
- P2 = (1,0,1,1,1)
- P3 = (1,1,0,1,0)
- P4 = (0,1,1,0,0)
- P5 = (0,1,0,0,0)

P1 P2 P3 P4 P5

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Co-occurrence matrix

Building a Similarity product-to-product matrix

- Similarity between products
- P1 = (0,1,1,0,0)
- P2 = (1,0,1,1,1)
  - CosSim(P1, P2) $=\frac{1}{\sqrt{2}\sqrt{4}}=\frac{1}{2\sqrt{2}}$ = 0.35
- P1 = (0,1,1,0,0)
- P3 = (1,1,0,1,0)
  - CosSim(P1, P3) = $\frac{1}{\sqrt{2}\sqrt{3}}=\frac{1}{\sqrt{6}}$=0.41

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Building a Similarity product-to-product matrix

$\sqrt{2} = 1.41$
$\sqrt{3} = 1.73$
$\sqrt{6} = 2.45$

- Similarity between products
- P2 = ?
- P3 = ?
  - CosSim(P2, P3) =?
- P2 = ?
- P4 = ?
  - CosSim(P2, P4) = ?
- P2 = ?
- P5 = ?
  - CosSim(P1, P3) =?

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Building a Similarity product-to-product matrix

- Similarity between products
- P2 = (1,0,1,1,1)
- P3 = (1,1,0,1,0)
  - CosSim(P2, P3) $= \frac{2}{\sqrt{4}\sqrt{3}} = \frac{2}{2\sqrt{3}} = 0.58$
- P2 = (1,0,1,1,1)
- P4 = (0,1,1,0,0)
  - CosSim(P2, P4) $= \frac{1}{\sqrt{4}\sqrt{2}} = \frac{1}{2\sqrt{2}} = 0.35$
- P2 = (1,0,1,1,1)
- P5 = (0,1,0,0,0)
  - CosSim(P2, P5) = 0

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Building a Similarity product-to-product matrix

- Similarity between products
- P3 = (1,1,0,1,0)
- P4 = (0,1,1,0,0)
  - CosSim(P3, P4) $= \frac{1}{\sqrt{3}\sqrt{2}} = \frac{1}{\sqrt{6}} = 0.41$
- Now generate the similarity matrix

	p1	p2	p3	p4	p5
p1	0	1	1	0	0
p2	1	0	1	1	1
p3	1	1	0	1	0
p4	0	1	1	0	0
p5	0	1	0	0	0

Building a Similarity product-to-product matrix

- Similarity between products
  - CosSim(P1, P2) = 0.35
  - CosSim(P1, P3) = 0.41
  - CosSim(P2, P3) = 0.58
  - CosSim(P2, P4) = 0.35
  - CosSim(P2, P5) = 0

  - CosSim(P3, P4) = 0.41

P1 P2 P3 P4 P5

	p1	p2	p3	p4	p5
p1	0	1 0.35	1 0.41	0	0
p2	1 0.35	0	1 0.58	1 0.35	1 0
p3	1 0.41	1 0.58	0	1 0.41	0
p4	0	1 0.35	1 0.41	0	0
p5	0	1 0	0	0	0

Scalability (1/2)

- Amazon.com has around **300 million** customers (200 million Amazon Prime members) and more than **12 million** cataloged items (including market place: more than 350 million items)
- Traditional collaborative filtering does little or no offline computation
- Online computation scales with the number of customers and catalog items.

Scalability (2/2)

- Cluster models can perform much of the computation offline
  - Recommendation **quality is relatively poor**

- REVISIT: Content-based model?
  - It cannot provide recommendations with interesting, targeted titles
  - Not scalable for customers with numerous purchases and ratings

Algorithm

- Calculating the similarity between a single product and all related products (without going through all (#_of_users x #_of_products) combinations)

```
For each item in product catalog, P-a
     For each customer C who purchased P-a
       For each item P-b purchased by customer C
         Record that a customer purchased P-a and P-b

     For each item P-b
         Compute the similarity between P-a and P-b
```

Creating a similarity product-to-product table

- Similar-items table is extremely computing intensive
  - **Offline computation**
  - $O(N^2M)$ in the worst case
    - Where N is the number of items and M is the number of users
  - Average case is closer to $O(NM)$
    - Most customers have very few purchases
  - Sampling customers who purchase best-selling titles reduces runtime even more
    - With little reduction in quality

42

42

Key scalability strategy for amazon recommendations

- Creating the **expensive similar-items table offline**
- Online component
  - Looking up similar items for the user's purchases and ratings
  - Scales independently of the catalog size or the total number of customers
- It is dependent only on how many titles the user has purchased or rated

43

43

Computing with MapReduce

Part 1

```
For each item in product catalog, P-a
   For each customer C who purchased P-a
    For each item P-b purchased by customer C
      Record that a customer purchased P-a and P-b

   For each item P-b
      Compute the similarity between P-a and P-b
```

- A Mapper reads a customer's purchase history
  - Extracts product id(s) and generates co-purchased pairs
  - Emits the combination of the product IDs (sorted) → ((P1, P2), "1")
- A Reducer adds the occurrences of the co-purchased products
  - Emits the (a-pair-of-co-purchased-products, total count)

Computing with MapReduce: Two MapReduce jobs

```
For each item in product catalog, P-a
  For each customer C who purchased P-a
    For each item P-b purchased by customer C
      Record that a customer purchased P-a and P-b

For each item P-b
      Compute the similarity between P-a and P-b
```

Part 2

- **Originally, part 1 and part 2 were combined in a single for-loop. We will separate part 2.**
- Cartesian product design pattern (see the join pattern)
- A pair of product vectors will be provided to a map function
  - E.g. P1 (0, 1, 0, 0, 0, 1, 0), P2(0, 1, 1, 0, 0, 1, 1)
- Mapper will calculate the cosine similarity
  - Emits Cosine_Similarity ((0, 1, 0, 0, 0, 1, 0), (0, 1, 1, 0, 0, 1, 1))
- No reducer required

Recommendation quality

- The algorithm recommends highly correlated similar items
  - Recommendation quality is excellent
  - Algorithm performs well with limited user data

Large Scale Data Analytics

4. Evaluation and Validation

Why evaluation/validation? (1/2)

- Process for model selection and performance estimation
- Model selection (fitting the model)
  - Most of the models have one or more **free parameters**

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

You should find these values to generate (fit) your model.

- How do we select the **"optimal" parameter(s)** or model for a given classification problem/predictive analytics?

Why evaluation/validation? (2/2)

- Performance estimation
  - Once we have chosen a model, how do we **estimate its performance (accuracy)**?
- Performance is typically measured by the **true error rate**
  - e.g. the classifier's error rate on the entire population

Challenges (1/2)

- If we had access to **an unlimited number** of examples these questions have a straightforward answer
  - Choose the model that provides the **lowest error rate** on the entire population
    - Of course, that error rate is the true error rate
- In real applications we only have access to **a subset of examples**, usually smaller than we wanted
  - What if we use the **entire available data to fit our model** and estimate the error rate?
    - The final model will normally overfit the training data
    - We already used the test dataset to train the data

Challenges (2/2)

- This problem is more pronounced with models that have a **large number of parameters**
  - The error rate estimate will be overly optimistic (lower than the true error rate)
    - In fact, it is not uncommon to have 100% correct classification on training data
- A much better approach is to **split the training data** into disjoint subsets: the holdout method

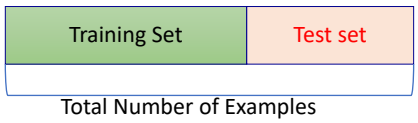
Large Scale Data Analytics

4. Evaluation and Validation

1. The Holdout method

The Holdout Method

- Split dataset into two groups
  - Training set
    - Used to train the model
  - Test set
    - Used to estimate the error rate of the trained model



  - Simplest cross validation method

Drawbacks of the holdout method

- Drawbacks
  - For **a sparse dataset**, we may **not be able to set aside a portion** of the dataset for testing
  - Based on the **where "split" happens**, the **estimate of error can be misleading**
    - Sample might not be representative
- The limitations of the holdout can be overcome with a family of resampling methods
  - More computational expense
  - Stratified sampling
  - Cross Validation
    - Random subsampling
    - K-Fold cross validation
    - Leave-one-out Cross-Validation

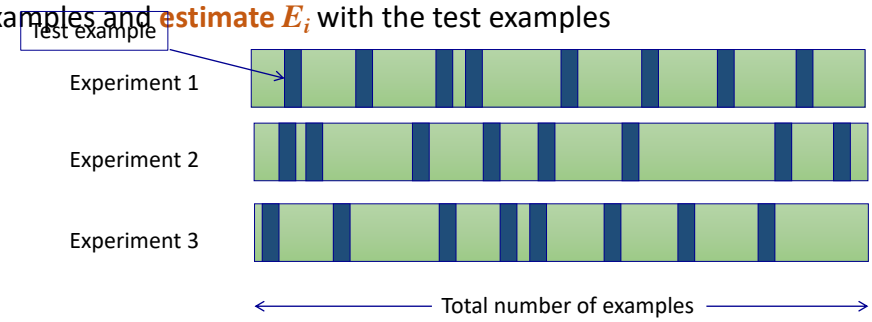
Large Scale Data Analytics

4. Evaluation and Validation

2. Random Subsampling

Random Subsampling

- K data splits of the dataset
  - Each split **randomly selects a (fixed) number of examples** without replacement
  - For each data split, retain the classifier from scratch with the training examples and **estimate E_i** with the test examples



True Error Estimate

- The true error estimate is obtained as the **average of the separate estimates E_i**

- This estimate **is significantly better than the holdout estimate**

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

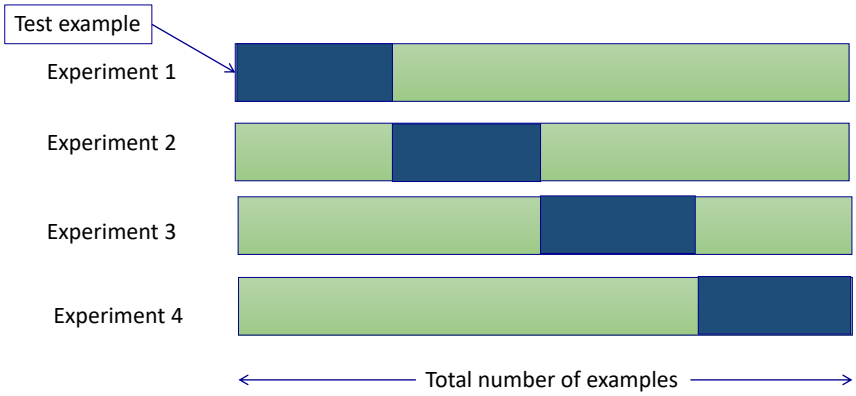
Large Scale Data Analytics

4. Evaluation and Validation

3. k-Fold Cross-validation

k-Fold Cross-validation

- Create a ***k*-fold partition** of the dataset
  - For each of the k experiments use ***K-1* folds for training**
    - The remaining one for testing



True error estimate

- k-fold cross validation is similar to random subsampling
- The advantage of k-Fold Cross validation
  - All the examples in the dataset are **eventually used for both training and testing**
- The true error is estimated as the average error rate

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

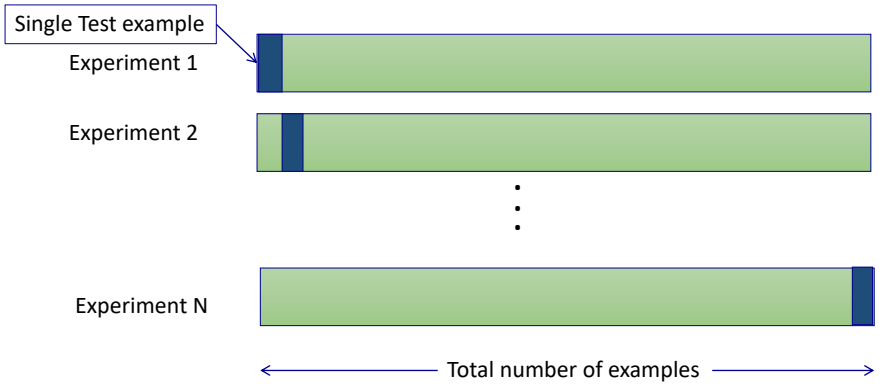
Large Scale Data Analytics

4. Evaluation and Validation

4. Leave-one-out Cross-validation

Leave-one-out Cross Validation

- Leave-one-out is the degenerate case of k-Fold Cross validation
  - k is chosen as the total number of examples
- For a dataset with N examples, perform N experiments
  - Use N-1 examples for training, the remaining example for testing



How many folds are needed?

- The choice of the number of folds depends on the size of the dataset
  - For large datasets, even 3-Fold Cross Validation will be quite accurate
  - For very sparse datasets, you may have to consider leave-one-out
    - To get maximum number of experiments
- A common choice for **k-fold Cross Validation is k=10**

Three-way data splits

- If model selection and true error estimates are computed **simultaneously**
  - The data needs to be divided into three disjoint sets
  - Training set
    - E.g. to find the optimal weights
  - Validation set
    - A set of examples used to tune the parameters of a model
    - To find the "optimal" number of hidden units or determine a stopping point for the back propagation algorithm
  - Test set
    - Used only to assess the performance of a fully-trained model
- **After assessing the final model with the test set, you must not further tune the model**

Large Scale Data Analytics

4. Evaluation and Validation

Classifiers

Classification

Mammals

Fishes

Birds

Insects

Plain Accuracy

- Classifier accuracy
  - General measure of classifier performance

Accuracy = (Number of correct decisions made) / (Total number of decision made)

- 1 – error rate

- Pros
  - **Very easy to measure**
- Cons
  - **Cannot consider realistic cases**

The Confusion Matrix

- A type of **contingency table**
- n classes
  - n x n matrix
    - The columns labeled with actual classes
    - The rows with predicted classes
- Separates out the decisions made by the classifier
  - How one class is being confused for another
  - Different sorts of errors may be dealt with separately

	p (positive)	**n (negative)**
Y (predicted)	True positive	False positive
N (predicted)	False negative	True negative

Problems with Unbalanced Classes

- Consider a classification problem where one class is **rare**
  - Sifting through a large population of normal entities to find a relatively small number of unusual ones
  - Looking for **defrauded customers**, or **defective parts**
  - The class distribution is **unbalanced or skewed**

Confusion Matrix of A: evaluated with 1,000 datapoints

	positive	**negative**
True	250	450
False	250	50

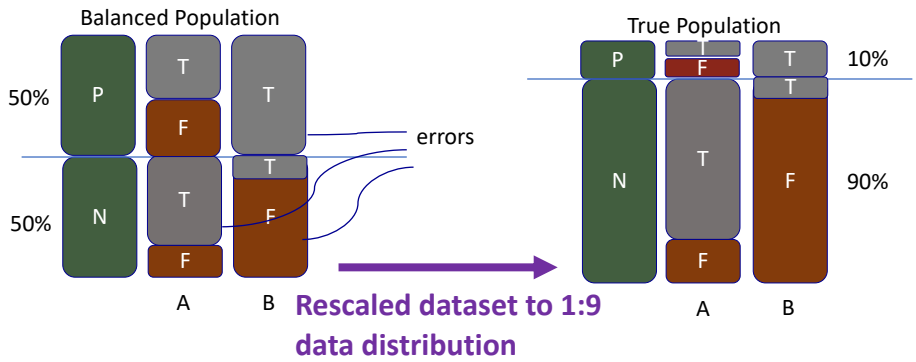
Confusion Matrix of B: evaluated with 1,000 datapoints

	positive	**negative**
True	500	50
False	0	450

- **Which model is better?**

Accuracy can mislead our understanding

Which model is better?



	positive	negative
True	250	450
False	250	50

Confusion Matrix of A

	positive	negative
True	500	50
False	0	450

Confusion Matrix of B

F-measure ($F1$ score)

- Summarizes confusion matrix
- True positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN)

	p	n
Y	TP	FN
N	FP	TN

- True positive rate (sensitivity) = $TP/(TP+FN)$
- False negative rate (miss rate) = $FN/(TP+FN)$
- F-measure = $2(precision \times recall)/(precision + recall)$
  - precision = $TP / (TP+FP)$
  - recall = $TP / (TP+FN)$
- Accuracy = $(TP + TN) / (P + N)$

Questions?

Questions?

CS435 Introduction to Big Data

Week 10 – Lecture 1 In-Memory Cluster Computing -Apache Spark

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- PA 2
  - Please schedule your demonstration
- PA 3
  - Group assignment (Work with your project team)
  - Using RDD should be enough
  - Read the description carefully
- Term project proposal

1

1

Topics discussed in the previous class

- Recommendation Systems: Collaborative Filtering
- Evaluation Methods

Topics to be covered

- PA 3
- Evaluation Methods – continued
- In-memory cluster computing
  - Apache Spark

PA 3: Estimating PageRank Values of Wikipedia Articles using Apache Spark

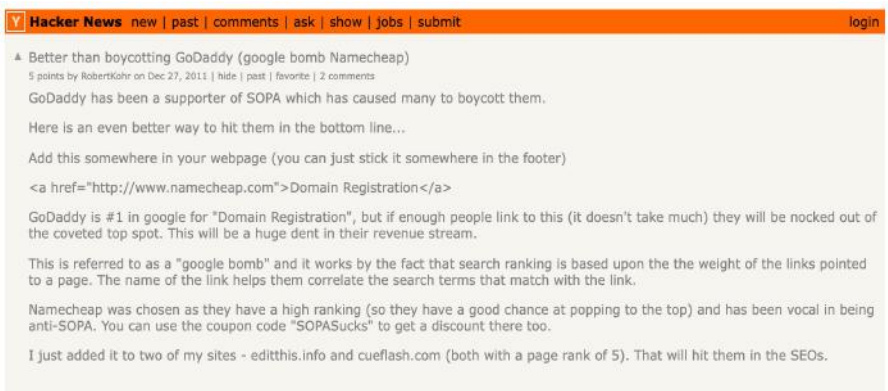
- Use the Taxation method

$$v' = \beta M v + (1-\beta) e / n$$

- Use the random walk style implementation
  - You do NOT need to use matrix-vector multiplication
  - There are many examples available
- Make your Wiki Bomb
  - Google Bomb (A.K.A. Google washing)
    - Making web site ranking highly in web search engine results for irrelevant, unrelated or off-topic search terms by linking heavily
  - Make a Wiki Bomb for the ”Rocky Mountain National Park” page
  - Modify the “link” file
  - Which pages should be modified?
  - This should be done “AFTER” you calculate the page rank values of all the pages

Google Bomb used to boycott GoDaddy.com

- SOPA (Stop Online Piracy Act), October 2011



Hacker News new | past | comments | ask | show | jobs | submit login

Better than boycotting GoDaddy (google bomb Namecheap)

5 points by RobertKohr on Dec 27, 2011 | hide | past | favorite | 2 comments

GoDaddy has been a supporter of SOPA which has caused many to boycott them.

Here is an even better way to hit them in the bottom line...

Add this somewhere in your webpage (you can just stick it somewhere in the footer)

<a href="http://www.namecheap.com">Domain Registration</a>

GoDaddy is #1 in google for "Domain Registration", but if enough people link to this (it doesn't take much) they will be nocked out of the coveted top spot. This will be a huge dent in their revenue stream.

This is referred to as a "google bomb" and it works by the fact that search ranking is based upon the the weight of the links pointed to a page. The name of the link helps them correlate the search terms that match with the link.

Namecheap was chosen as they have a high ranking (so they have a good chance at popping to the top) and has been vocal in being anti-SOPA. You can use the coupon code "SOPASucks" to get a discount there too.

I just added it to two of my sites - editthis.info and cueflash.com (both with a page rank of 5). That will hit them in the SEOs.

Workflow for PA 3

- Calculate PageRank values for Wikipedia pages
- Create your Wiki Bomb for the "Rocky Mountain National Park"
- Simple searching mechanism for the search key "surfing"
  - You may create a command line environment
    - E.g. find "surfing"
    - Returns list of pages sorted on their pagerank values
  - Or, any search capability from text editor will also work

REVIEW

Large Scale Data Analytics

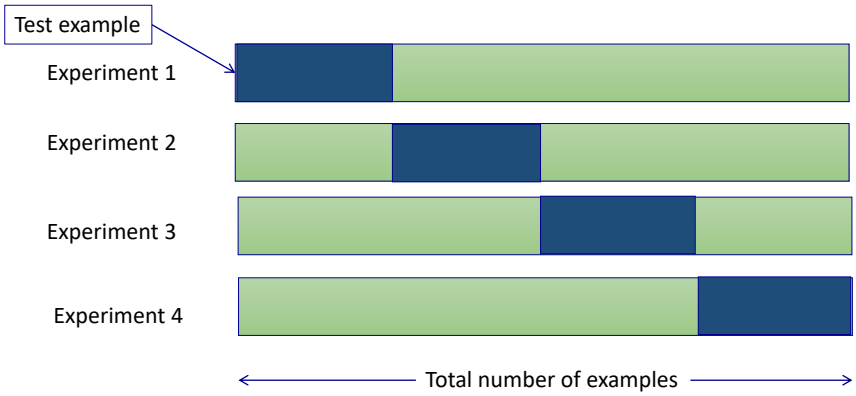
4. Evaluation and Validation

3. k-Fold Cross-validation

REVIEW

k-Fold Cross-validation

- Create a ***k*-fold partition** of the dataset
  - For each of the k experiments use ***K-1* folds for training**
    - The remaining one for testing



REVIEW

True error estimate

- k-fold cross validation is similar to random subsampling
- The advantage of k-Fold Cross validation
  - All the examples in the dataset are **eventually used for both training and testing**
- The true error is estimated as the average error rate

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

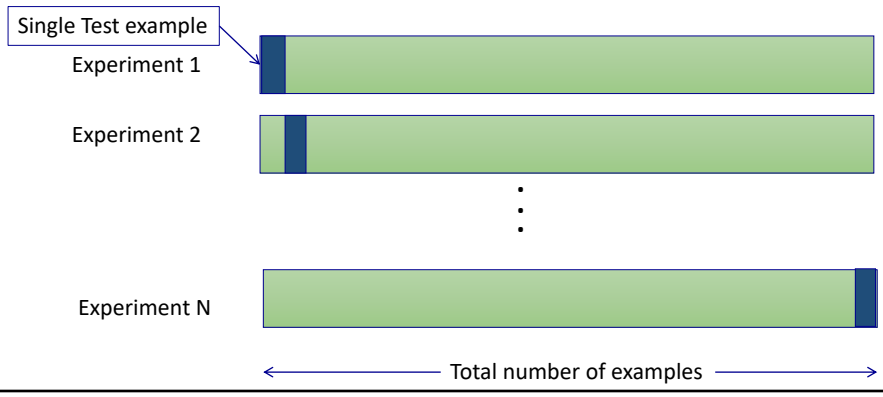
Large Scale Data Analytics

4. Evaluation and Validation

4. Leave-one-out Cross-validation

Leave-one-out Cross Validation

- Leave-one-out is the degenerate case of k-Fold Cross validation
  - k is chosen as the total number of examples
- For a dataset with N examples, perform N experiments
  - Use N-1 examples for training, the remaining example for testing



How many folds are needed?

- The choice of the number of folds depends on the size of the dataset
  - For large datasets, even 3-Fold Cross Validation will be quite accurate
  - For very sparse datasets, you may have to consider leave-one-out
    - To get maximum number of experiments
- A common choice for **k-fold Cross Validation is k=10**

Three-way data splits

- If model selection and true error estimates are computed **simultaneously**
  - The data needs to be divided into three disjoint sets
  - Training set
    - E.g. to find the optimal weights
  - Validation set
    - A set of examples used to tune the parameters of a model
    - To find the "optimal" number of hidden units or determine a stopping point for the back propagation algorithm
  - Test set
    - Used only to assess the performance of a fully-trained model
- **After assessing the final model with the test set, you must not further tune the model**

Accuracy measurements

- Mean Absolute Error (MAE)
  - Avg(Abs(Actual-Forecast))
  - $\frac{1}{n}\sum_{n}^{t=1}|A_t - F_t|$
- Root Mean Square Error (RMSE)
  - *Sqrt(Avg(Power(Actual -Forecast)))*
  - $\sqrt{\frac{1}{n}\sum_{n}^{t=1}|A_t - F_t|^2}$
  - Punishing larger errors (compared to MAE)

Large Scale Data Analytics

4. Evaluation and Validation

Classifiers

Classification

Mammals

Fishes

Birds

Insects

Plain Accuracy

- Classifier accuracy
  - General measure of classifier performance

Accuracy = (Number of correct decisions made) / (Total number of decision made)

  - 1 – error rate

- Pros
  - **Very easy to measure**
- Cons
  - **Cannot consider realistic cases**

The Confusion Matrix

- A type of **contingency table**
- n classes
  - n x n matrix
    - The columns labeled with actual classes
    - The rows with predicted classes
- Separates out the decisions made by the classifier
  - How one class is being confused for another
  - Different sorts of errors may be dealt with separately

	p (positive)	**n (negative)**
Correct	True positive	True negative
Incorrect	False negative	False negative

Problems with Unbalanced Classes

- Consider a classification problem where one class is **rare**
  - Sifting through a large population of normal entities to find a relatively small number of unusual ones
  - Looking for **defrauded customers**, or **defective parts**
  - The class distribution is **unbalanced or skewed**

Confusion Matrix of A: evaluated with 1,000 datapoints

	positive	**negative**
Correct	250	450
Incorrect	250	50

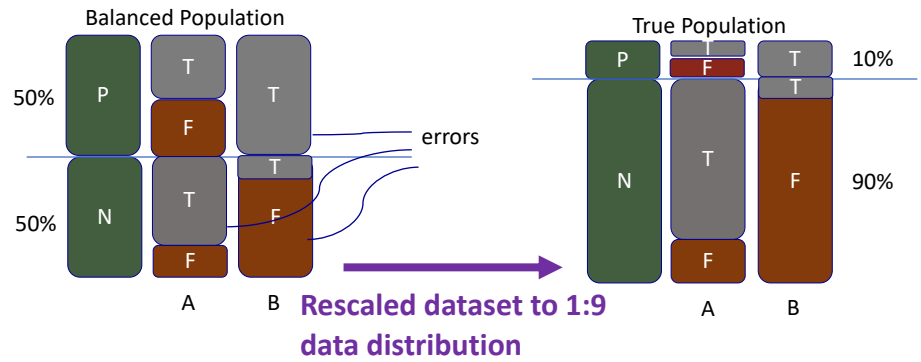
Confusion Matrix of B: evaluated with 1,000 datapoints

	positive	**negative**
Correct	500	50
Incorrect	0	450

- **Which model is better?**

Accuracy can mislead our understanding

Which model is better?



	positive	negative
True	250	450
False	250	50

Confusion Matrix of A

	positive	negative
True	500	50
False	0	450

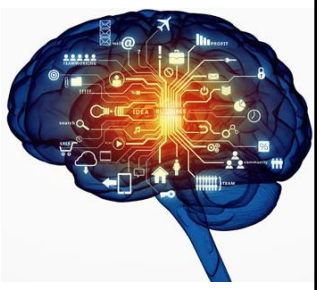
Confusion Matrix of B

F-measure ($F1$ score)

- Summarizes confusion matrix
- True positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN)

	p	n
Y	TP	FN
N	FP	TN

- True positive rate (sensitivity) = $TP/(TP+FN)$
- False negative rate (miss rate) = $FN/(TP+FN)$
- F-measure = $2(precision \times recall)/(precision + recall)$
  - precision = $TP / (TP+FP)$
  - recall = $TP / (TP+FN)$
- Accuracy = $(TP + TN) / (P + N)$



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Large Scale Data Analytics

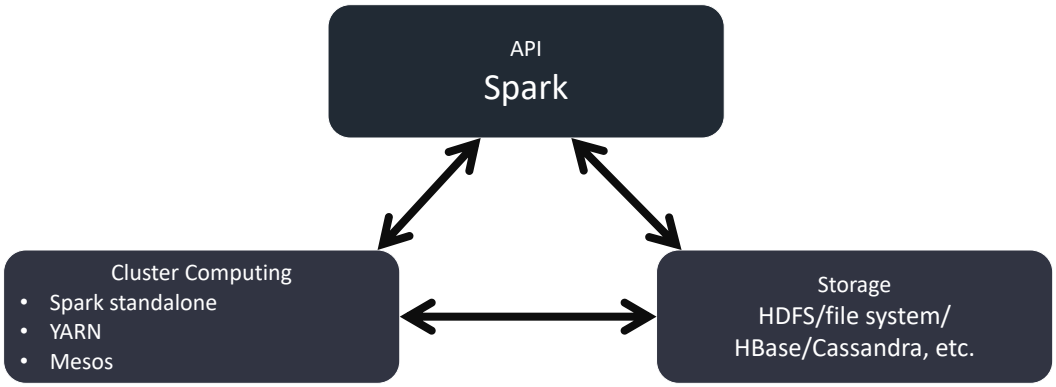
In-Memory Cluster Computing: Apache Spark

Introduction

This material is built based on

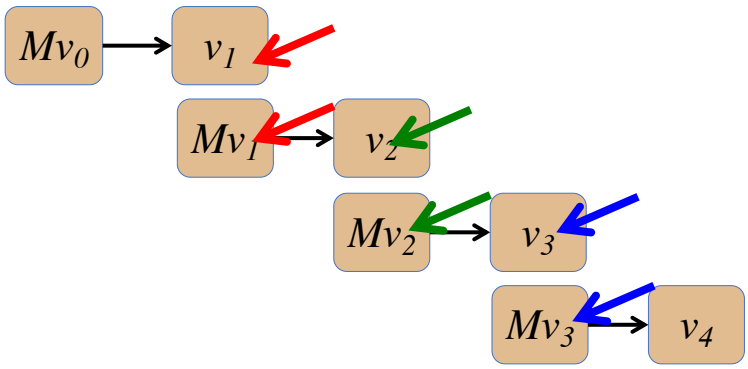
- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)
- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015
- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

Distributed processing with the Spark framework



Inefficiencies for emerging applications: (1) Data reuse

- Data reuse is common in many iterative machine learning and graph algorithms
  - e.g. PageRank, and K-means clustering



Inefficiencies for emerging applications: (2) Interactive data analytics

- User runs multiple ad-hoc queries on the **same subset** of the data

Existing/Previous approaches

- Hadoop
  - Writing output to an external stable storage system
    - e.g. HDFS
    - Substantial overheads due to data replication, disk I/O, and serialization
- Pregel
  - Iterative graph computations
- HaLoop
  - Iterative MapReduce interface
- Pregel/HaLoop support specific computation patterns
  - e.g. looping a series of MapReduce steps

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

RDD (Resilient Distributed Dataset)

RDD (Resilient Distributed Dataset)

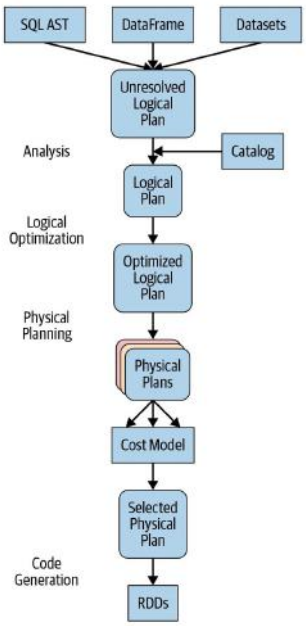
- **Read-only, memory resident partitioned collection of records**
  - A fault-tolerant collection of elements that can be operated on in parallel
- RDDs are the core unit of data in Spark
  - Most Spark programming involves performing operations on RDDs
- DataFrame API
  - Adding concepts with named columns and schemas
    - Table
    - Column has a specific data types: String, integer, array, map, real, date, timestamp..
- SparkSQL API
  - High-level structured functionality
  - Interoperable with other storage systems (e.g. Hbase)

Spark SQL

- Query support
  - Expressive query support
  - Easy to integrate with analytics

<Example>

```
SELECT State, Color, Count, sum(Count) AS
Total
FROM MNM_TABLE_NAME
GROUP BY State, Color, Count
ORDER BY Total DESC
```



RDD?

- Very fundamental data structure of Apache Spark
- Distribution, Partition, Fault tolerance, and scheduling computation
  - All automated by Apache Spark
  - With some user's configuration

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Simple Example with Spark RDD

- *Map vs. Filter vs. FlatMap*

Loading data from a file to an RDD

First RDD with data loaded from a text file

Your first RDD has been used here

Location of the raw data file

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts
        = textFile.flatMap(s -> Arrays.asList(s.split(" "))
                                .iterator())
          Add more operations here…
counts.saveAsTextFile("hdfs://...");
```


Example: Word Count

https://raw.githubusercontent.com/apache/spark/master/examples/src/main/java/org/apache/spark/examples/JavaWordCount.java

```
package org.apache.spark.examples;
import scala.Tuple2;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.sql.SparkSession;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;
```

A compiled representation of a regular expression

```
public final class JavaWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");
  public static void main(String[] args) throws Exception {
    if (args.length < 1) {
      System.err.println("Usage: JavaWordCount <file>");
      System.exit(1);
    }
    SparkSession spark = SparkSession
        .builder()
        .appName("JavaWordCount")
        .getOrCreate();
```

Provide the app name

FlatMap: Each item can be mapped to one or more output items

Generating an RDD from the file

```
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();

JavaRDD<String> words = lines.flatMap(s ->
    Arrays.asList(SPACE.split(s)).iterator());
    JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s,
1));
    JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
    List<Tuple2<String, Integer>> output = counts.collect();
    for (Tuple2<?,?> tuple : output) {
      System.out.println(tuple._1() + ": " + tuple._2());
    }
    spark.stop();
  }
}
```

Tokenizing a string

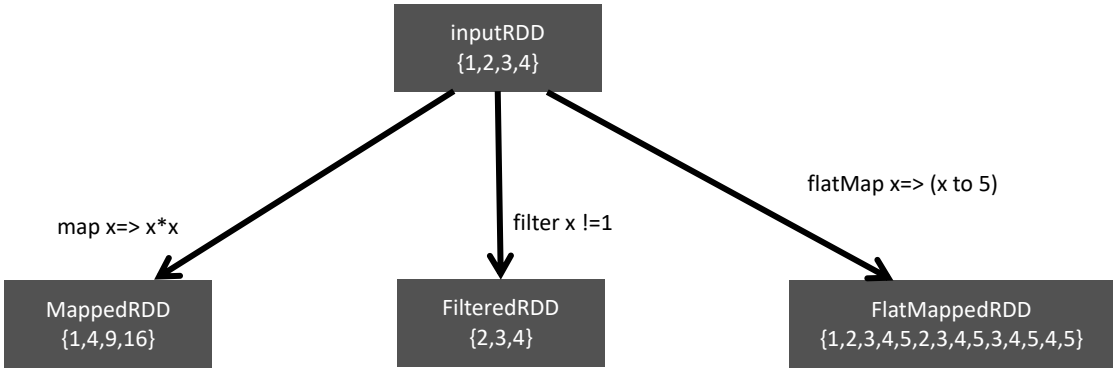
Bring them back to the driver program

Driver?
Program that creates the `SparkContext`, connecting to a given Spark Master.

Simple RDD operations
`map()` vs. `filter()` vs. `flatMap()` [1/3]

- The **`map()`** operation takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD
- The **`filter()`** operation takes in a function and returns an RDD that only has elements that pass the filter() function
- The **`flatMap()`** is similar to map, but each input item can be mapped to 0 or more output items (so *func*should return a Seq rather than a single item).

`map()` vs. `filter()` vs. `flatMap()` [2/3]



`map()` vs. `filter()` vs. `flatMap()` [3/3]

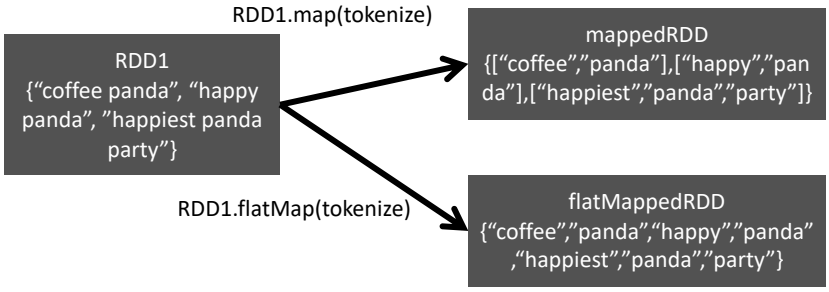
- `map()` that squares all of the numbers in an RDD

```
JavaRDD <Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4));
JavaRDD <Integer> result = rdd.map(new Function < Integer, Integer >() {
   public Integer call(Integer x) {
        return x*x;
   }
});


System.out.println(StringUtils.join(result.collect(),","));
```


`map()` vs. `flatMap()` with String [1/2]

- As results of **`flatMap()`**, we have an RDD of the elements
  - Instead of RDD of lists of elements



`map()` vs. `flatMap()` with String [2/2]

- Using `flatMap()` that splits lines to multiple words

```
JavaRDD < String > lines = sc.parallelize( Arrays.asList("hello world",
"hi"));

JavaRDD < String > words =
  lines.flatMap(new FlatMapFunction < String, String >() {
     public Iterable < String > call( String line) {
       return Arrays.asList(line.split(" "));
     }
  });

words.first();
// returns "hello"
```


Delivering output to the driver node `collect()` vs. `take(n)`

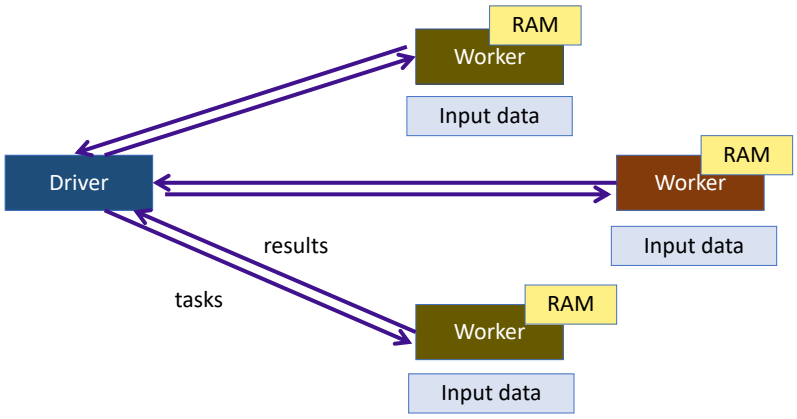
- collect()
  - Retrieves all the elements of the RDD/DataFrame/Dataset (from all nodes) to the driver node
- take()
  - returns `n` elements from the RDD and attempts to minimize the number of partitions it accesses
  - It may represent a biased collection
  - It does not return the elements in the order you might expect
  - Useful for unit testing
- What if a spark program does not use collect or take or an equivalent operation?
  - Local outputs will stay in the local nodes

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Understanding Spark RDD

RDDs in Spark: The Runtime



User's driver program launches multiple workers,
which read data blocks from a distributed file system and can persist computed RDD partitions in memory

Representing RDDs

- A set of partitions
  - Atomic pieces of the dataset
- **A set of dependencies on parent RDDs**
- **A function for computing** the dataset based on its parents
- Metadata about its partitioning scheme
- **Data placement**

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 10-A Fall 2021 Computer Science

Overview of RDD

- **Partitioning**
  - Users can specify the partitioning method across machines based on a key in each record
- **Lineage**
  - How it was derived from other dataset to compute its partitions from data in stable storage?
    - RDDs do not need to be materialized at all times
- **Persistence**
  - Users can indicate which RDDs they will reuse and the storage strategy

Spark Programming Interface to RDD: Type 1. Transformation [1/3]

- **"transformations"**
  - Operations that **create** RDDs
    - Return pointers to new RDDs
    - e.g. `map, filter, and join`
  - RDDs can only be created through **deterministic operations** on either
    - **Data in stable storage**
    - **Other RDDs**

Spark Programming Interface to RDD: Type 2: Action [2/3]

- **"actions"**
  - Operations that **return a value to the application** or **export data to a storage system**
    - e.g. `count`: returns the number of elements in the dataset
    - e.g. `collect`: returns the elements themselves
    - e.g. `save`: outputs the dataset to a storage system

Spark Programming Interface to RDD: Type 3: Persist [3/3]

- **"persist"**
  - Indicates which RDDs they want to **reuse in future operations**
  - Spark keeps persistent RDDs **in memory** by default
  - If there is not enough RAM
    - It can spill them to disk
  - Users are allowed to,
    - store the RDD only on disk
    - replicate the RDD across machines
    - specify a persistence priority on each RDD

Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an **action**
  - Spark internally records metadata to indicate that this operation has been requested
- **Loading data from files into an RDD** is lazily evaluated
- Reduces the number of passes it has to take over our data by grouping operations together

Example: Console Log Mining [1/3]

- Suppose that a web service is experiencing errors and an operator wants to search terabytes of logs in the Hadoop file system (HDFS) to find the cause
- The user load **the error messages** from the logs into the RAM across a set of nodes and query them interactively

```
lines = spark.textFile("hdfs://…")
errors=lines.filter(_.startsWith("ERROR"))
errors.persist()
```

No work has been performed
User can use the RDD in actions

52

52

Example: Console Log Mining [2/3]

- Users can perform further transformations and actions on the RDD

```
//To count number of error messages
errors.count()

//Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

//Return the time fields of errors mentioning
//HDFS as an array (assuming time is field
//number 3 in a tab-separated format
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

After the first **action** involving `errors` runs, Spark will store the partitions of errors in memory.

53

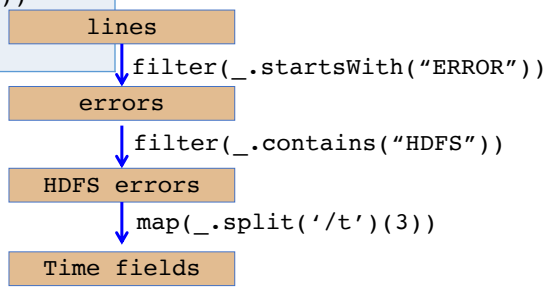
53

Example: Console Log Mining [3/3]

Spark code

```
lines = spark.textFile("hdfs://…")
errors=lines.filter(_.startsWith("
ERROR"))
errors.persist()
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

Lineage graph



If a partition of `errors` is lost Spark rebuilds it by applying a filter on only the corresponding partition of `lines`

Benefits of RDDs as a distributed memory abstraction [1/7]

- RDD vs. Distributed Shared Memory (DSM)?
  - How does RDD work differently compared to DSM?
  - Write/Consistency/Fault-Recovery mechanism/Straggler mitigation
- RDDs can only be created ("written") through **coarse-grained transformations**
  - Coarse-grained transformations are applied over an entire dataset
  - Reads on RDDs can still be fine-grained
    - A large read-only lookup table
  - Applications perform bulk writes
- **More efficient fault tolerance**
  - Lineage based bulk recovery

Benefits of RDDs as a distributed memory abstraction [2/7]

RDD vs. DSM- Read operation

- RDD
  - Either coarse grained or fine grained
- DSM
  - The read operation in Distributed shared memory is fine-grained

Benefits of RDDs as a distributed memory abstraction [3/7]

RDD vs. DSM- Write operation

- RDD
  - The write operation in RDD is coarse grained
- DSM
  - The write operation in Distributed shared memory is fine-grained

Benefits of RDDs as a distributed memory abstraction [4/7]

RDD vs. DSM- Consistency

- RDD
  - RDD is immutable in nature
    - Any changes on RDD is permanent
  - The level of consistency is high
- DSM
  - If the programmer follows the rules, the memory will be consistent and the results of memory operations will be predictable

Benefits of RDDs as a distributed memory abstraction [5/7]

RDD vs. DSM- Fault-Recovery Mechanism

- RDD
  - The lost data can be easily recovered in Spark RDD using lineage graph at any moment
  - For each transformation, new RDD is formed
  - RDDs are immutable
- DSM
  - Fault tolerance is achieved by a checkpointing technique which allows applications to roll back to a recent checkpoint rather than restarting

Benefits of RDDs as a distributed memory abstraction [6/7]

RDD vs. DSM- Straggler Mitigation

- Stragglers
  - Nodes taking more time to complete than their peers
  - Due to load imbalance, I/O blocks, garbage collections, etc.
- RDD
  - **Creates backup copies of slow tasks**
    - without accessing the same memory
- DSM
  - It is quite difficult to achieve straggler mitigation

Benefits of RDDs as a distributed memory abstraction [7/7]

- Runtime can schedule tasks **based on data locality**
  - To improve performance
- RDDs degrade gracefully when there is **insufficient memory**
  - **Partitions that do not fit in the RAM are stored on disk**

Applications not suitable for RDDs

- RDDs are best suited for **batch applications** that apply the same operations to **all elements of a dataset**
  - Steps are managed by **lineage graph** efficiently
  - Recovery is managed effectively
- RDDs would not be suitable for applications
  - Making asynchronous fine-grained updates to shared state
  - e.g. a **storage system for a web application** or an incremental web crawler

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

RDD in Spark

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

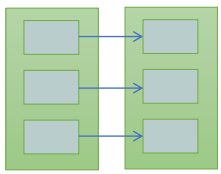
RDD Dependency in Spark

Dependency between RDDs [1/4]

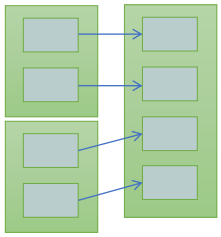
- **Narrow** dependency
- **Wide** dependency

Dependency between RDDs [2/4]

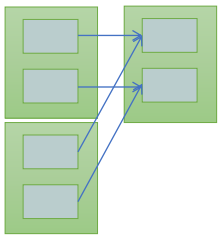
- **Narrow** dependency
  - Each partition of the parent RDD is used by **at most one partition** of the child RDD



map, filter



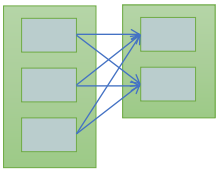
union



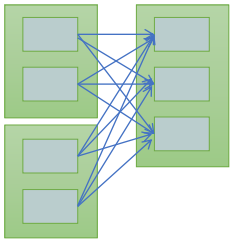
Join with inputs co-partitioned

Dependency between RDDs [3/4]

- **Wide** dependency
  - **Multiple child partitions** may depend on a single partition of parent RDD



groupByKey



Join with inputs not co-partitioned

Dependency between RDDs [4/4]

- Narrow dependency
  - Pipelined execution on one cluster node
  - e.g. a map followed by a filter
  - Failure recovery is more straightforward
- Wide dependency
  - Requires data from all parent partitions to be available and to be shuffled across the nodes
  - Failure recovery could involve a large number of RDDs
    - Complete re-execution may be required

Questions?

CS435 Introduction to Big Data

Week 10 – Lecture 2 In-Memory Cluster Computing -Apache Spark

Sangmi Lee Pallickara

Computer Science, Colorado State University

FAQs

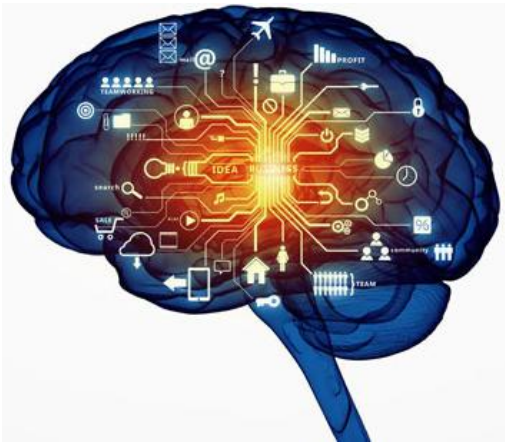
- PA 3
  - Group assignment (Work with your project team)
  - Using RDD should be enough
  - Read the description carefully

Topics discussed in the previous class

- PA 3
- Evaluation Methods – continued
- In-memory cluster computing
  - Apache Spark

Topics to be covered

- In-memory cluster computing - Continued
  - Apache Spark



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

This material is built based on

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)

- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015

- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

REVIEW

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Simple Example with Spark RDD

- *Map vs. Filter vs. FlatMap*

REVIEW

Loading data from a file to an RDD

First RDD with data loaded from a text file

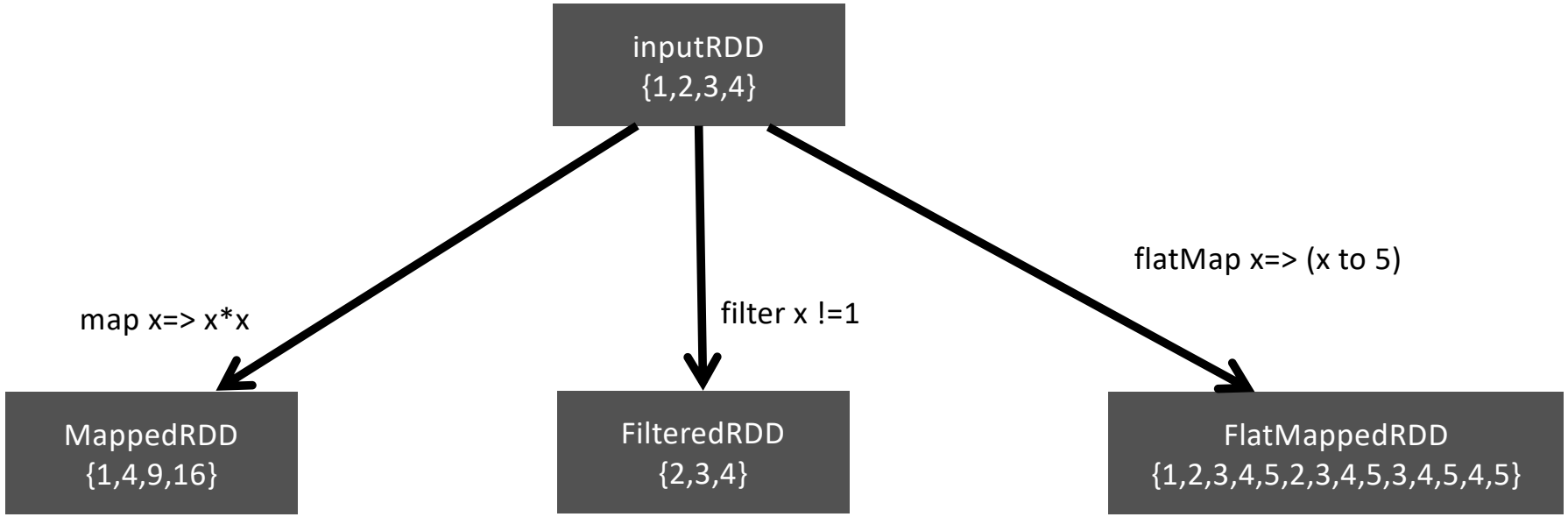
Your first RDD has been used here

Location of the raw data file

```
JavaRDD<Str g> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts
        = textFile.flatMap(s -> Arrays.asList(s.split(" "))
                                .iterator())
          Add more operations here…
counts.saveAsTextFile("hdfs://...");
```

`map()` vs. `filter()` vs. `flatMap()` [2/3]

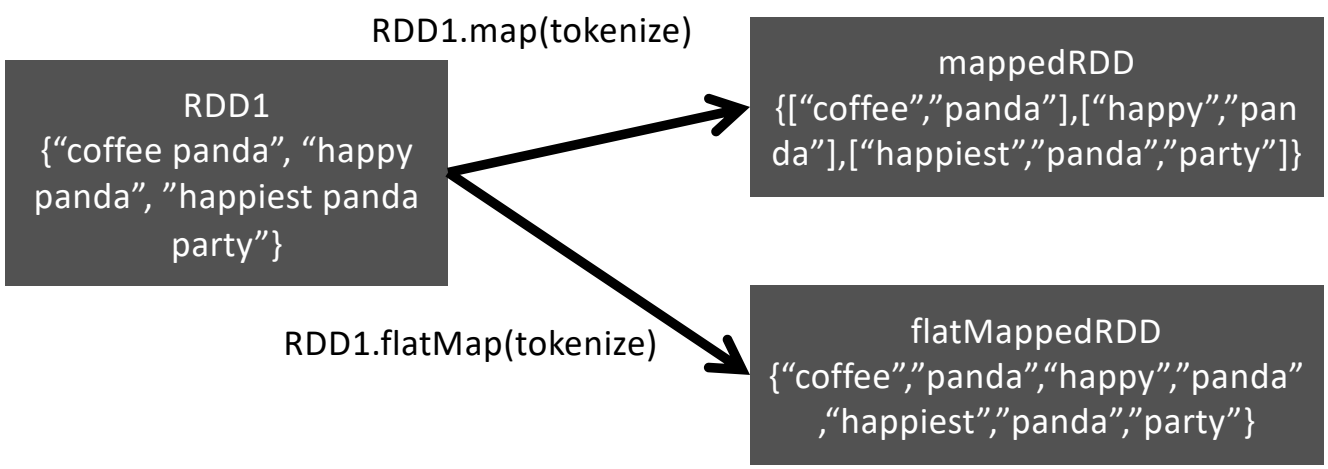
REVIEW



REVIEW

`map()` vs. `flatMap()` with String [1/2]

- As results of **`flatMap()`**, we have an RDD of the elements
  - Instead of RDD of lists of elements



REVIEW

Delivering output to the driver node `collect() vs. take(n)`

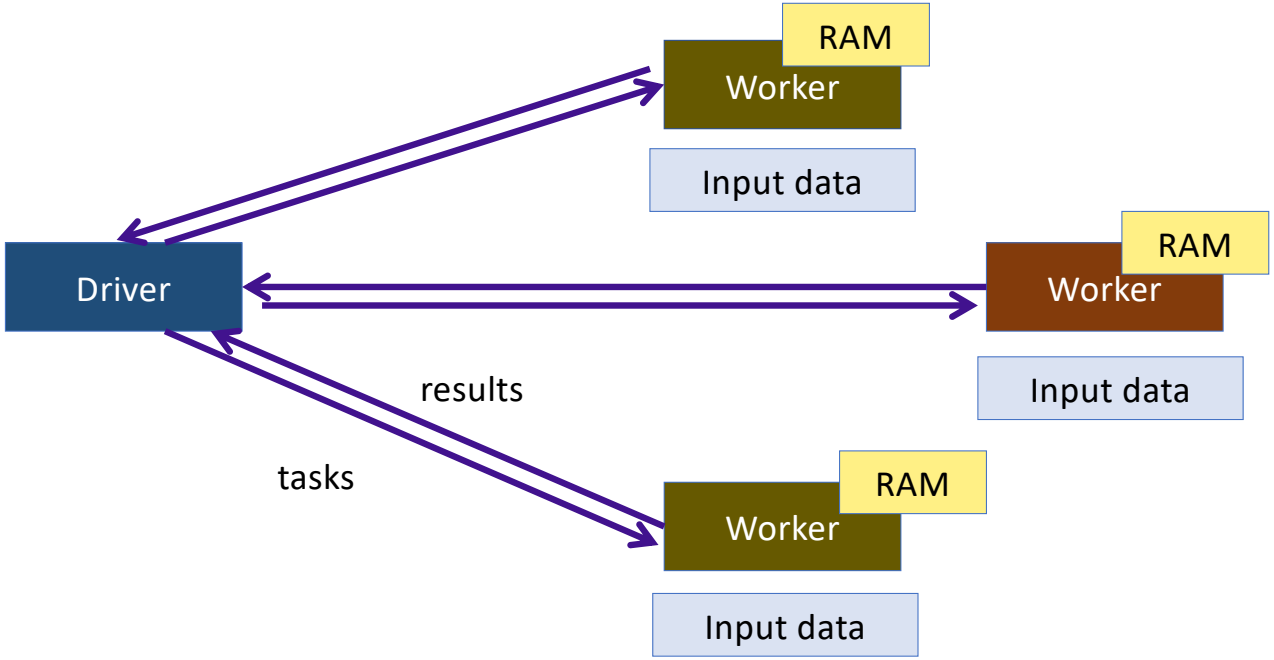
- collect()
  - Retrieves all the elements of the RDD/DataFrame/Dataset (from all nodes) to the driver node
- take()
  - returns `n` elements from the RDD and attempts to minimize the number of partitions it accesses
  - It may represent a biased collection
  - It does not return the elements in the order you might expect
  - Useful for unit testing
- What if a spark program does not use collect or take or an equivalent operation?
  - Local outputs will stay in the local nodes

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Understanding Spark RDD

RDDs in Spark: The Runtime



User's driver program launches multiple workers,
which read data blocks from a distributed file system and can persist computed RDD partitions in memory

Representing RDDs

- A set of partitions
  - Atomic pieces of the dataset
- **A set of dependencies on parent RDDs**
- **A function for computing** the dataset based on its parents
- Metadata about its partitioning scheme
- **Data placement**

Overview of RDD

- **Types**
  - Standard RDD
    - Single column table (list)
  - Pair RDD
    - Key-value pairs
- **Partitioning**
  - Users can specify the partitioning method across machines based on a key in each record
- **Persistence**
  - Users can indicate which RDDs they will reuse and the storage strategy
- **Lineage**
  - How was this RDD derived from other dataset to compute its partitions from data in stable storage?
    - RDDs do not need to be materialized at all times

Spark Programming Interface to RDD

- **Spark does NOT allow any program to modify already existing RDDs**

- **Spark Operations (Programming interface to RDD)**
  - **Transformation**
  - **Action**
  - **Persistence**

Spark Programming Interface to RDD: Type 1. Transformation [1/3]

- **"transformations"**
  - Operations that **create** RDDs
    - Return pointers to new RDDs
    - e.g. `map, filter, and join`
  - RDDs can only be created through **deterministic operations** on either
    - **Data in stable storage**
    - **Other RDDs**

Spark Programming Interface to RDD: Type 2: Action [2/3]

- **"actions"**
  - Operations that **return a value to the application** or **export data to a storage system**
    - e.g. `count`: returns the number of elements in the dataset
    - e.g. `collect`: returns the elements themselves
    - e.g. `save`: outputs the dataset to a storage system

Spark Programming Interface to RDD: Type 3: Persist [3/3]

- **"persist"**
  - Indicates which RDDs they want to **reuse in future operations**
  - Spark keeps persistent RDDs **in memory** by default
  - If there is not enough RAM
    - It can spill them to disk
  - Users are allowed to,
    - store the RDD only on disk
    - replicate the RDD across machines
    - specify a persistence priority on each RDD

Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an **action**
  - Spark internally records metadata to indicate that this operation has been requested

- **Loading data from files into an RDD** is lazily evaluated

- Reduces the number of passes it has to take over our data by grouping operations together

Example: Console Log Mining [1/3]

- Suppose that a web service is experiencing errors and an operator wants to search terabytes of logs in the Hadoop file system (HDFS) to find the cause

- The user load **the error messages** from the logs into the RAM across a set of nodes and query them interactively

Standard RDD

Standard RDD

This RDD will stored in the memory until the end of the current job

```
lines = spark.textFile("hdfs://...")
errors=lines.filter(_.startsWith("ERROR"))
errors.persist()
//To count number of error messages
errors.count()
```

No work has been performed until this point: No "ACTION" so far

The first "Action" operation
Evaluation of all the above lines will be performed now

Example: Console Log Mining [2/3]

- Users can perform further transformations and actions on the RDD

```
//Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

//Return the time fields of errors mentioning
//HDFS as an array (assuming time is field
//number 3 in a tab-separated format
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

This is another transformation. No immediate evaluation

This is another transformation. No immediate evaluation

This is another transformation. No immediate evaluation

The second "Action" operation
Evaluation of all the above lines will be performed now

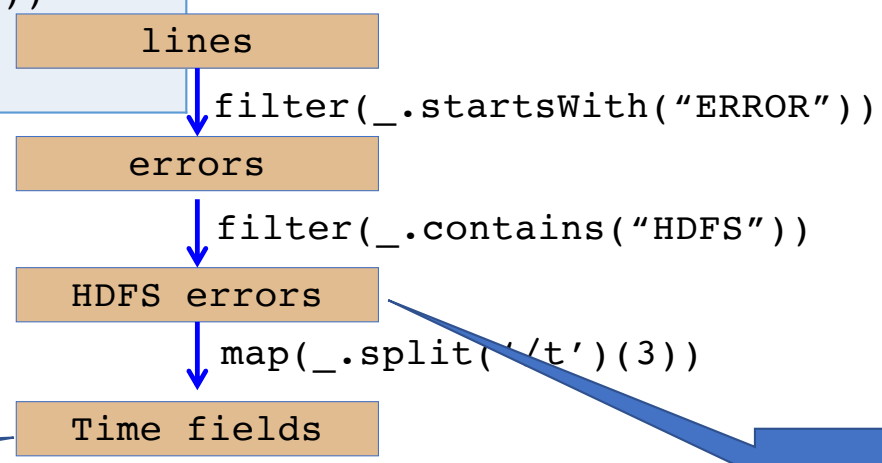
After the first **action** involving `errors` runs, Spark will store the partitions of errors in memory.

Example: Console Log Mining [3/3]

Key operations from the previous Spark code

```
lines = spark.textFile("hdfs://…")
errors=lines.filter(_.startsWith("
ERROR"))
errors.persist()
errors.filter(_.contains("HDFS"))
       .map(_.split('/t')(3))
       .collect()
```

Lineage graph



If a partition of `errors` is lost Spark rebuilds it by applying a filter on only the corresponding partition of `lines`

Internal RDD

Internal RDD

RDD Lineage

- A graph of all the parent RDDs of an RDD
  - A graph of what transformations need to be executed after an action has been called
- A.K.A. Logical execution plan
  - starts with the earliest RDDs (those with no dependencies on other RDDs or reference cached data) and ends with the RDD that produces the result of the action that has been called to execute
- **Fault tolerance with lineage**
  - If any loss occurs, Spark can recover the failure using the lineage graph
    - Finds an available RDD and repeat the computations

Why Key/Value Pairs?

- Pair RDDs
  - Spark provides special operations on RDDs containing key/value pairs
  - Pair RDDS allow you to **act on each key in parallel** or **regroup data across the network**

- e.g.
  - `reduceByKey()`
    - **Aggregates data separately for each key**

  - `join()`
    - Merge two RDDs by grouping elements with the **same key**

Creating Pair RDDs

-- using the first word as the key

- Running `map()` function
  - A transformation that passes each dataset element through a function and returns a new RDD representing the results
  - Returns **key/value pairs**

```
PairFunction < String, String, String > keyData =
   new PairFunction < String, String, String >() {
     public Tuple2 < String, String > call(String x) {
       return new Tuple2(x.split(" ")[0], x);
   }
};

JavaPairRDD <String,String> pairs = lines.mapToPair(keyData);
```

PairFunction: A function that returns key-value pairs (Tuple2<K, V>), and can be used to construct PairRDDs
Tuple2: A function that creates a tuple

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Transformations on Standard RDDs and Pair RDDs

Transformations on an RDD (e.g. rdd={ 1, 3, 6}, other={2,3,4})

Function	purpose	Example	Result
`map``(`*`func`*`)`	Return a new RDD formed by passing each element through a function func	`rdd.map(x*x)`	{1, 9, 36}
`filter``(`*`func`*`)`	Select element(s) on which the func returns true	`rdd.filter(x>1)`	{3, 6}
`flatMap``(`*`func`*`)`	Similar to map but returns 0 or more outputs	`rdd.filter(2..x)`	{2,3,2,3,4,5,6}
`union``(`*`otherDataset`*`)`	Create a union set of two RDDs	`rdd.union(other)`	{1,2,3,4,6}
`mapPartitions``(`*`func`*`)`	Similar to map, but runs separately on each partition (block) of the RDD	`Heavy initialization such as DB connection`	Single DB connection per partition
`intersection``(`*`otherDataset`*`)`	Intersecting two RDDs	`rdd.intersection (other)`	{3}

Pair RDDs are still RDDs

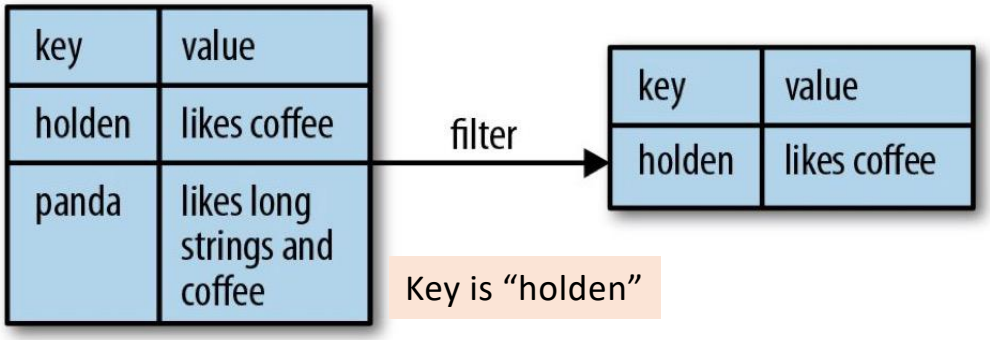
- Supports the **same functions as RDDs**

e.g. filtering out the long words

```
Function < Tuple2 < String, String >, Boolean > longWordFilter =
   new Function < Tuple2 < String, String >, Boolean >() {
     public Boolean call( Tuple2 < String, String > keyValue) {
       return (keyValue._2().length() < 20);
     }
};

JavaPairRDD < String, String > result =
                                    pairs.filter(longWordFilter);
```

`Filter` on Pair RDDs



Transformations on a pair RDD (e.g. rdd={(1, 2), (3, 4), (3, 6)})

Function	purpose	Example	Result
`mapValues(func)`	Apply a function to each value of a pair RDD **without changing the key**	`rdd.mapValues(x=>x+1)`	`{(1,3),(3,5),(3,7)}`
`flatMapValues(func)`	Apply a function that returns an **iterator**	`rdd.flatMapValues(x=>(x to 5)`	`{(1,2),(1,3),(1,4),(1,5),(3,4),(3,5)}`
`keys()`	Return an RDD of just the **keys**	`rdd.keys()`	`{1,3,3}`
`values()`	Return an RDD of just the **values**	`rdd.values()`	`{2,4,6}`
`sortByKey()`	Return an RDD **sorted by the key**	`rdd.sortByKey()`	`{(1,2),(3,4),(3,5)}`

Transformations on a pair RDD (e.g. rdd={(1, 2), (3, 4), (3, 6)})

Function	purpose	Example	Result
`reduceByKey()`	**Combine** values with the **same key**	`rdd.reduceByKey( (x, y) => x + y)`	{(1,2),(3,10)}
`groupByKey()`	**Group** values with the same key	`rdd.groupByKey()`	{(1,[2]), (3,[4,6])}
`combineByKey( createCombiner, mergeValue, mergeCombiners, partitioner)`	Combine values with the same key using a different result type	Slides: 44	

Transformations on two pair RDDs (e.g. rdd={(1, 2), (3, 4), (3, 6)}) other={(3,9)}

Function	purpose	Example	Result
`subtractByKey`	**Remove elements with a key present in the other RDD**	`rdd.subtractByKey(other)`	`{(1,2)}`
`join`	**Inner join**	`rdd.join(other)`	`{(3,(4,9)),(3,(6,9))}`
`rightOuterJoin`	Perform a join where the **key must be present in the other RDD**	`rdd.rightOuterJoin(other)`	`(3,(Some(4),9)),(3,(Some(6),9))}`
`leftOuterJoin()`	Perform a join where the key must be present in the first RDD	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3,(4,Some(9))), (3,(6,Some(9)))}`
`coGroup`	Group data from both RDDs sharing the same key	`rdd.cogroup(other)`	`{(1,([2],[])),(3,([4,6],[9]))}`

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Aggregations with PairRDDs

Aggregations with PairRDDs

- **Aggregate** statistics across all elements **with the same key**
- Example:
- **`reduceByKey()`**
  - Similar to `reduce()`
  - Takes a function and use it to combine values
  - Runs several parallel reduce operations
    - One for each key in the dataset
    - Each operations combines values that have the same keys
  - `reduceByKey()` is **not implemented as an action that returns a value to the user program**
    - **There can be a large number of keys**
    - It **returns a new RDD** consisting of each key and the reduced value for that key
  - **Therefore, `reduceByKey()` is a transformation**

Example

- Key-value pairs are represented using the `Tuple2` class

```
JavaRDD<String> lines = sc.textFile("data.txt");

JavaPairRDD<String, Integer> pairs =
                    lines.mapToPair(s -> new Tuple2(s, 1));

JavaPairRDD<String, Integer> counts =
                    pairs.reduceByKey((a, b) -> a + b);
```

Revisit the word count example

```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
```

`combineByKey()`

- The most **general of the per-key aggregation functions**
  - Most of the other per-key combiners are implemented using it
- Similar to reduceByKey()
  - More flexible
  - Allows the user to return values that are not the same type as the input data
- `createCombiner()`
  - If `combineByKey()` finds a new key
  - This happens the first time a key is found in each **partition**, rather than only the first time the key is found in the RDD
- `mergeValue()`
  - If it is not a new value in that partition
- `mergeCombiners()`
  - Merging the results from each partition

Per-key average using `combineByKey()` [1/2]

```
public static class AvgCount implements Serializable {
   public AvgCount(int total, int num) {
        total_ = total;
        num_ = num; }
   public int total_;
   public int num_;
   public float avg() {
        return total_ / (float) num_;
   }
}

Function <Integer, AvgCount > createAcc = new
   Function < Integer, AvgCount >() {
     public AvgCount call(Integer x) {
       return new AvgCount(x, 1); }
   };

Function2 < AvgCount, Integer, AvgCount > addAndCount = new
   Function2 < AvgCount, Integer, AvgCount >() {
      public AvgCount call(AvgCount a, Integer x) {
          a.total_ + = x;
          a.num_ + = 1;
```

Per-key average using `combineByKey()` [2/2]

```
      return a; }
};
Function2 < AvgCount, AvgCount, AvgCount > combine =
   new Function2 < AvgCount, AvgCount, AvgCount >() {
   public AvgCount call( AvgCount a, AvgCount b) {
      a.total_ + = b.total_;
      a.num_ + = b.num_;
      return a; }
};

AvgCount initial = new AvgCount(0,0);
JavaPairRDD < String, AvgCount > avgCounts =
nums.combineByKey(createAcc, addAndCount, combine);
Map < String, AvgCount > countMap = avgCounts.collectAsMap();
for (Entry < String, AvgCount > entry : countMap.entrySet()) {
        System.out.println( entry.getKey() + ":"
                            + entry.getValue().avg());
}
```

Tuning the level of parallelism

- When performing aggregations or grouping operations, we can ask Spark to use a specific number of partitions
  - `reduceByKey((x, y) = > x + y, 10)`

- **`repartition()`**
  - Shuffles the data across the network to create a new set of partitions
  - Expensive operation
  - Optimized version: **`coalesce()`**
    - Reduces data movement
    - A subset of current nodes will be kept

`groupByKey()`

- Group our data using the key in our RDD
- On an RDD consisting of keys of type `K` and values of type `V`
  - Results will be RDD of type `[K, Iterable[V]]`

- `cogroup()`
  - Grouping data from **multiple RDDs**
  - Over two RDDs sharing the same key type, `K`, with the respective value types `V` and `W` gives us back `RDD[(K, (Iterable[V], Iterable[W]))]`

joins

- **Join operations generates a new RDD**
  - **Join is a transformation operation**

- Inner join
  - Only keys that are present in both pair RDDs are output

- `leftOuterJoin(other)` and `rightOuterJoin(other)`
  - One of the pair RDDs can be missing the key

- `leftOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the source RDD

- `rightOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the other RDD

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Key-Value pairs : Actions available on RDDs

Actions on standard RDDs(example: rdd={1,3,4})

Function	Description	Example	Result
`reduce(`*`func`*`)`	Aggregate the elements of the dataset using a function *func*	`rdd.reduce(x+y)`	`{8}`
`collect()`	**Return all the elements**	`rdd.collect()`	`{1,3,4}`
`count()`	**Return the number of elements**	`rdd.count()`	`3`
`first()`	**Return the first elements**	`rdd.first()`	`1`
`take(`*`n`*`)`	**Return first n elements**	`rdd.take(1)`	`1`
`saveAsTextFile(`*`path`*`)`	Write the elements of the dataset as a text file	`rdd.saveAsTextFile (file_path)`	`Text file`

Actions on pair RDDs(example({(1,2),(3,4),(3,6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key	`rdd.countByKey()`	`{(1,1),(3,2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup at the driver	`rdd.collectAsMap()`	`Map{(1,2),(3,4),(3,6)}`
`lookup(key)`	Return all values associated with the provided key	`rdd.lookup(3)`	`[4,6]`

Revisit: Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an action
  - Spark internally records metadata to indicate that this operation has been requested
- **Loading data from files into an RDD** is lazily evaluated
- Reduces the number of passes it has to take over our data by grouping operations together

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

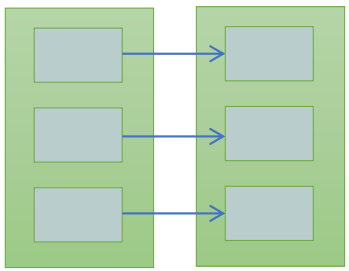
RDD Dependency in Spark

Dependency between RDDs [1/4]

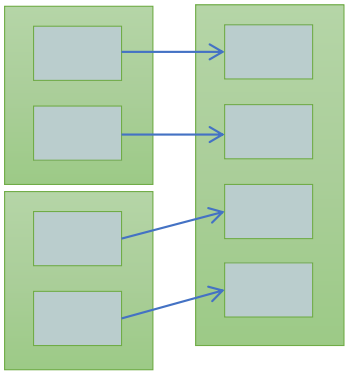
- **Narrow** dependency
- **Wide** dependency

Dependency between RDDs [2/4]

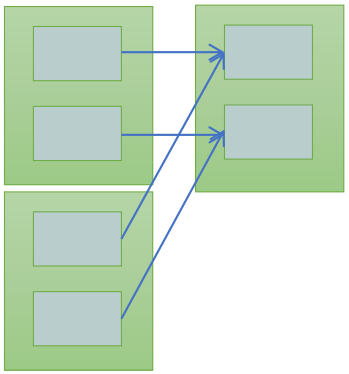
- **Narrow** dependency
  - Each partition of the parent RDD is used by **at most one partition** of the child RDD



map, filter



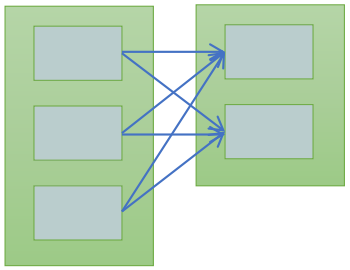
union



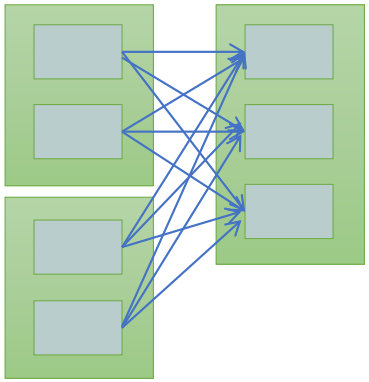
Join with inputs co-partitioned

Dependency between RDDs [3/4]

- **Wide** dependency
  - **Multiple child partitions** may depend on a single partition of parent RDD



groupByKey



Join with inputs not co-partitioned

Dependency between RDDs [4/4]

- Narrow dependency
  - Pipelined execution on one cluster node
  - e.g. a map followed by a filter
  - Failure recovery is more straightforward
- Wide dependency
  - Requires data from all parent partitions to be available and to be shuffled across the nodes
  - Failure recovery could involve a large number of RDDs
    - Complete re-execution may be required

Interface used to represent RDDs in Spark

- `partitions()`
  - Returns a list of partition objects
- `preferredLocations(p)`
  - List nodes where partition p can be accessed faster due to data locality
- `dependencies()`
  - Return a list of dependencies
- `iterator(p, parentIters)`
  - Compute the elements of partition p given iterators for its parent partitions
- `partitioner()`
  - Return metadata specifying whether the RDD is hash/range partitioned

CS435 Introduction to Big Data

Week 11 – Lecture 1 In-Memory Cluster Computing -Apache Spark

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- Resubmission
  - PA0 ,1, 2, and 3(will be applicable after the submission deadline)
  - Deadline: 5PM 11/30/2021
  - Submission: via email (to cs435@cs.colostate.edu)
  - Score deduction: 25% of the score you have received
  - What if I already submitted and my homework has been graded?
    - Any higher score will be selected
- Term project proposal
  - Comments/feedbacks are sent via email
    - If your team did not get any email from me, let me know
  - Unless I have said that your team should change the topic, please continue working on the current topic
  - If your team wants to change the topic, let me know!

1

FAQs - continued

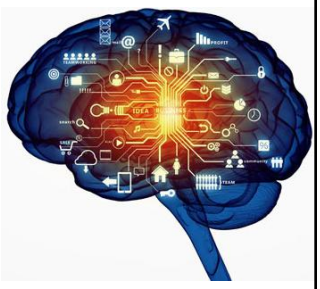
- Final Term Project software submission and demonstration
  - Visit your canvas page
  - Sign up for the software demonstration
    - Link included in the canvas page
- Final Term Project presentation sign up
  - Visit your canvas page
  - Sign up for the software demonstration
    - Link included in the canvas page

Topics discussed in the previous class

- In-memory cluster computing - Continued
  - Apache Spark
  - Introduction to the Apache Spark's Transformations and Actions
  - Lazy evaluation
  - RDD Lineage
  - Transformations (for standard RDD and pair RDD)

Topics to be covered

- In-memory cluster computing - Continued
  - Apache Spark
  - Aggregation
  - Actions



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

This material is built based on

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)
- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015
- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Aggregations with PairRDDs

Aggregations with PairRDDs

- **Aggregate** statistics across all elements **with the same key**
- Example:
- **`reduceByKey()`**
  - Similar to `reduce()`
  - Takes a function and use it to combine values
  - Runs several parallel reduce operations
    - One for each key in the dataset
    - Each operations combines values that have the same keys
  - `reduceByKey()` is **not implemented as an action that returns a value to the user program**
    - **There can be a large number of keys**
    - It **returns a new RDD** consisting of each key and the reduced value for that key
  - **Therefore, `reduceByKey()` is a transformation**

Example

- Key-value pairs are represented using the `Tuple2` class

```
JavaRDD<String> lines = sc.textFile("data.txt");

JavaPairRDD<String, Integer> pairs =
                    lines.mapToPair(s -> new Tuple2(s, 1));

JavaPairRDD<String, Integer> counts =
                    pairs.reduceByKey((a, b) -> a + b);
```

Revisit the word count example

```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
```


`combineByKey()`

- The most **general of the per-key aggregation functions**
  - Most of the other per-key combiners are implemented using it
- Similar to reduceByKey()
  - More flexible
  - Allows the user to return values that are not the same type as the input data
- `createCombiner()`
  - If `combineByKey()` finds a new key
  - This happens the first time a key is found in each **partition**, rather than only the first time the key is found in the RDD
- `mergeValue()`
  - If it is not a new value in that partition
- `mergeCombiners()`
  - Merging the results from each partition

Per-key average using `combineByKey()` [1/2]

```
public static class AvgCount implements Serializable {
   public AvgCount(int total, int num) {
        total_ = total;
        num_ = num; }
   public int total_;
   public int num_;
   public float avg() {
        return total_ / (float) num_;
   }
}

Function <Integer, AvgCount > createAcc = new
   Function < Integer, AvgCount >() {
     public AvgCount call(Integer x) {
       return new AvgCount(x, 1); }
   };

Function2 < AvgCount, Integer, AvgCount > addAndCount = new
   Function2 < AvgCount, Integer, AvgCount >() {
      public AvgCount call(AvgCount a, Integer x) {
          a.total_ + = x;
          a.num_ + = 1;
```


Per-key average using `combineByKey()` [2/2]

```
      return a; }
};
Function2 < AvgCount, AvgCount, AvgCount > combine =
   new Function2 < AvgCount, AvgCount, AvgCount >() {
   public AvgCount call( AvgCount a, AvgCount b) {
      a.total_ + = b.total_;
      a.num_ + = b.num_;
      return a; }
};

AvgCount initial = new AvgCount(0,0);
JavaPairRDD < String, AvgCount > avgCounts =
nums.combineByKey(createAcc, addAndCount, combine);
Map < String, AvgCount > countMap = avgCounts.collectAsMap();
for (Entry < String, AvgCount > entry : countMap.entrySet()) {
        System.out.println( entry.getKey() + ":"
                            + entry.getValue().avg());
}
```

Tuning the level of parallelism

- When performing aggregations or grouping operations, we can ask Spark to use a specific number of partitions
  - `reduceByKey((x, y) = > x + y, 10)`

- **`repartition()`**
  - Shuffles the data across the network to create a new set of partitions
  - Expensive operation
  - Optimized version: **`coalesce()`**
    - Reduces data movement
    - A subset of current nodes will be kept

`groupByKey()`

- Group our data using the key in our RDD
- On an RDD consisting of keys of type `K` and values of type `V`
  - Results will be RDD of type **`[K, Iterable[V]]`**

- `cogroup()`
  - Grouping data from **multiple RDDs**
  - Over two RDDs sharing the same key type, `K`, with the respective value types `V` and `W` gives us back `RDD[(K, (Iterable[V], Iterable[W]))]`

joins

- **Join operations generates a new RDD**
  - **Join is a transformation operation**
- Inner join
  - Only keys that are present in both pair RDDs are output
- `leftOuterJoin(other)` and `rightOuterJoin(other)`
  - One of the pair RDDs can be missing the key
- `leftOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the source RDD
- `rightOuterJoin(other)`
  - The resulting pair RDD has entries for each key in the other RDD

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Key-Value pairs : Actions available on RDDs

Actions on standard RDDs(example: rdd={1,3,4})

Function	Description	Example	Result
`reduce(func)`	Aggregate the elements of the dataset using a function *func*	`rdd.reduce(x+y)`	`{8}` → a collection Note: it is not an RDD
`collect()`	Return all the elements	`rdd.collect()`	`{1,3,4}` → a collection
`count()`	Return the number of elements	`rdd.count()`	3
`first()`	Return the first elements	`rdd.first()`	1
`take(n)`	Return first n elements	`rdd.take(1)`	1
`saveAsTextFile(path)`	Write the elements of the dataset as a text file	`rdd.saveAsTextFile(file_path)`	Text file

reduce(func) vs. reducyByKey()?
reduce() operates on RDD of objects (standard RDD), whereas reduceByKey operates on RDD of key value pairs. The function reduce() is a member of RDD[T] class, while reduceByKey() is member of PairRDDFunctions[K,V] class.
reduce() function returns a **collection** which will be included in DAG for the next level, so it is implemented as action. While reduceByKey() returns another RDD which might a part of sequence of operations in a DAG, so it is implemented as transformation.

Actions on pair RDDs(example({(1,2),(3,4),(3,6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key	`rdd.countByKey()`	`{(1,1),(3,2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup at the driver	`rdd.collectAsMap()`	`Map{(1,2),(3,4),(3,6)}`
`lookup(key)`	Return all values associated with the provided key	`rdd.lookup(3)`	`[4,6]`

countByKey

`public scala.collection.Map<K,Object> countByKey()`

Count the number of elements for each key, collecting the results to a local Map.

Returns:
(undocumented)

Note:
This method should only be used if the resulting map is expected

Revisit: Lazy Evaluation

- Transformations on RDDs are **lazily evaluated**
  - Spark will **NOT** begin to execute until it sees an action
  - Spark internally records metadata to indicate that this operation has been requested
- **Loading data from files into an RDD** is lazily evaluated
- Reduces the number of passes it has to take over our data by grouping operations together

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

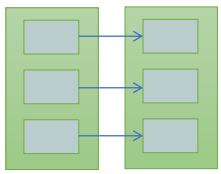
RDD Dependency in Spark

Dependency between RDDs [1/4]

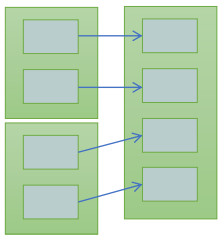
- **Narrow** dependency
- **Wide** dependency

Dependency between RDDs [2/4]

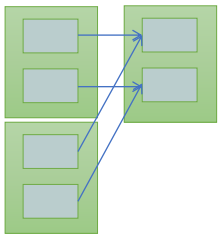
- **Narrow** dependency
  - Each partition of the parent RDD is used by **at most one partition** of the child RDD



map, filter



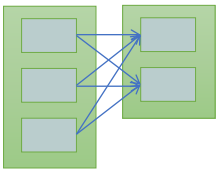
union



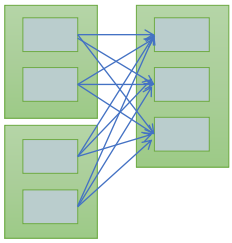
Join with inputs co-partitioned

Dependency between RDDs [3/4]

- **Wide** dependency
  - **Multiple child partitions** may depend on a single partition of parent RDD



groupByKey



Join with inputs not co-partitioned

Dependency between RDDs [4/4]

- Narrow dependency
  - Pipelined execution on one cluster node
    - e.g. a map followed by a filter
  - Failure recovery is more straightforward
- Wide dependency
  - Requires data from all parent partitions to be available and to be shuffled across the nodes
  - Failure recovery could involve a large number of RDDs
    - Complete re-execution may be required

Interface used to represent RDDs in Spark

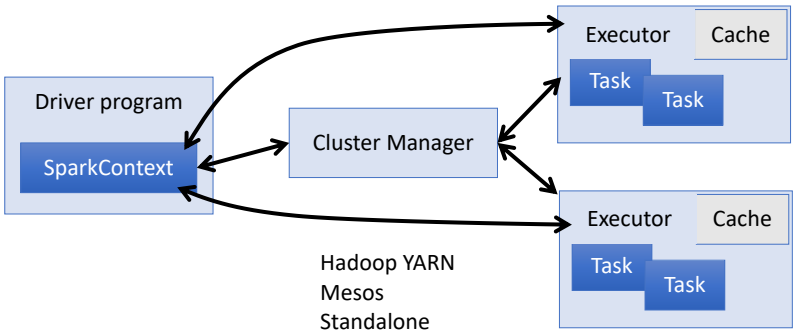
- `partitions()`
  - Returns a list of partition objects
- `preferredLocations(p)`
  - List nodes where partition p can be accessed faster due to data locality
- `dependencies()`
  - Return a list of dependencies
- `iterator(p, parentIters)`
  - Compute the elements of partition p given iterators for its parent partitions
- `partitioner()`
  - Return metadata specifying whether the RDD is hash/range partitioned

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Spark Cluster

Spark cluster and resources



Spark cluster [1/3]

- Each application gets its **own executor processes**
  - Must be up and running for the duration of the entire application
  - Run tasks in multiple threads
  - Isolate applications from each other
    - Scheduling side (each driver schedules its own tasks)
    - Executor side (tasks from different applications run in different JVMs)
  - **Data cannot be shared across different Spark applications** (instances of SparkContext) without writing it to an external storage system

Spark cluster [2/3]

- Spark is agnostic to the underlying cluster manager
  - As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN)

Spark cluster [3/3]

- **Driver program** must **listen for and accept incoming connections** from its executors throughout its lifetime
  - Driver program must be network addressable from the worker nodes
- Driver program should run close to the worker nodes
  - **On the same local area network**

Cluster Manager Types

- **Standalone**
  - Simple cluster manager included with Spark
- **Mesos**
  - Fine-grained sharing option
    - Frequently shared objects for Interactive applications
  - **Mesos master determines the machines that handle the tasks**
- **Hadoop YARN**
  - Resource manager in Hadoop 2

Dynamic Resource Allocation

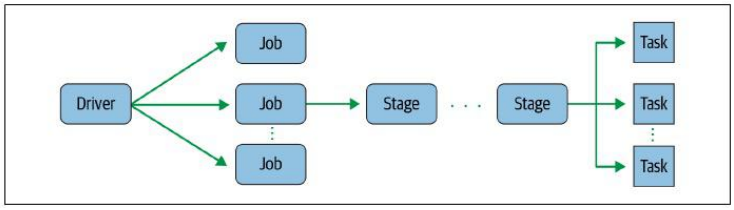
- Dynamically adjust the resources that the applications occupy
  - Based on the workload
  - Your application **may give resources back to the cluster** if they are no longer used
- Only available on **coarse-grained cluster managers**
  - Standalone mode, YARN mode, Mesos coarse grained mode

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Scheduling

Behind your Spark program



- Application
  - A spark submit (A user program)
- Job
  - A group of tasks that need to run to evaluate an action
- Stage
  - Jobs are divided into “stages” based on the shuffle boundary
- Task
  - A unit operation
    - Each stage is further divided into tasks based on the number of partitions in the RDD. So tasks are the smallest units of work for Spark

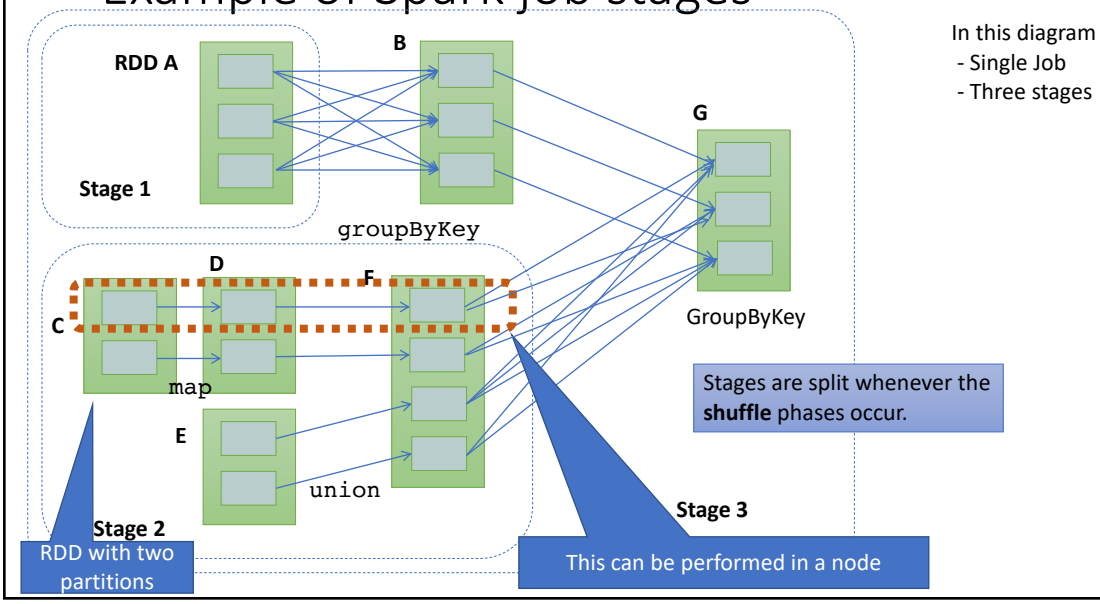
Jobs in Spark application

- **"Job"**
  - **A Spark action (e.g. save, collect) and any tasks that need to run to evaluate that action**
- Within a given Spark application, multiple parallel tasks can run simultaneously
  - If they were submitted from separate threads

Job scheduling

- User runs an **action** (e.g. count or save) on an RDD
  - A job will be created
  - Scheduler examines that **RDD's lineage graph to build a DAG of stages** to execute
- Each stage contains as many pipelined transformations as possible
  - With **narrow** dependencies
- The boundaries of the stages are the **shuffle operations**
  - For **wide** dependencies
  - For any already computed partitions that can short circuit the computation of a parent RDD

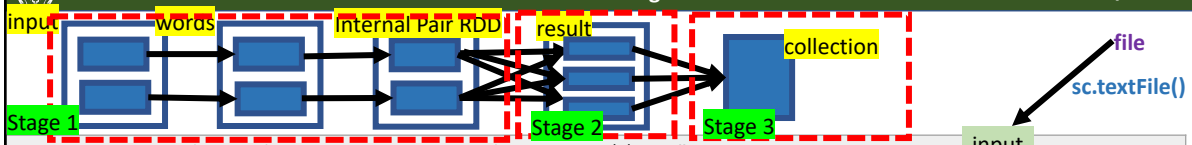
Example of Spark job stages



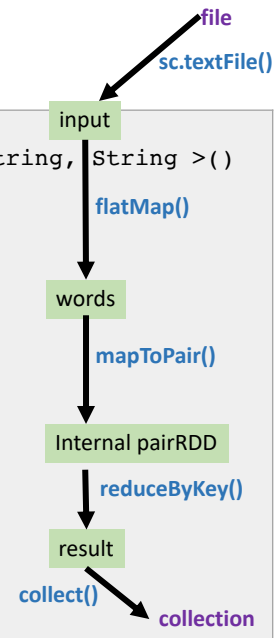
In this diagram
- Single Job
- Three stages

Revisit the word count example

```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
result.collect();
```




```
JavaRDD <String> input = sc.textFile("s3://...")
JavaRDD <String> words = input.flatMap(new FlatMapFunction < String, String >()
{
   public Iterable < String > call( String x) {
      return Arrays.asList(x.split(" "));
   }
});
JavaPairRDD <String,Integer> result = words
    .mapToPair(new PairFunction<String,String,Integer>() {
       public Tuple2 <String,Integer> call(String x) {
         return new Tuple2(x, 1);
      }
     })
    .reduceByKey(new Function2 <Integer,Integer,Integer>(){
       public Integer call(Integer a, Integer b) {
          return a + b;
       }
     });
result.collect();
```



Default FIFO scheduler

- By default, Spark's scheduler runs jobs in **FIFO** fashion
- First job gets the first priority on all available resources
  - Then the second job gets the priority, etc.
  - As long as the resource is available, jobs in the queue will start right away

Fair Scheduler

- Assigns tasks between jobs in a "**round robin**" fashion
  - All jobs get a roughly equal share of cluster resources
- Short jobs that were submitted when a long job is running can start receiving resources right away
  - Good response times, without waiting for the long job to finish
- Best for multi-user settings

Fair Scheduler Pools

- Supports grouping jobs into pools
  - With different options (e.g. weights)
  - "high-priority" pool for more important jobs
- This approach is modeled after the **Hadoop Fair Scheduler**
- **Default behavior of pools**
  - Each pool gets **an equal share of the cluster**
  - Inside each pool, jobs run in **FIFO** order
  - If the Spark cluster creates one pool per user
    - Each user will get an equal share of the cluster
    - Each user's queries will run in order

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Closures

Understanding closures

- To execute jobs, Spark breaks up the processing of RDD operations into tasks to be executed by **an executor**
- Prior to execution, Spark computes the task's **closure**
  - The closure is those **variables and methods** that **must** be **visible for the executor** to perform its computations on the RDD
- This closure is serialized and sent to each executor

Understanding closures

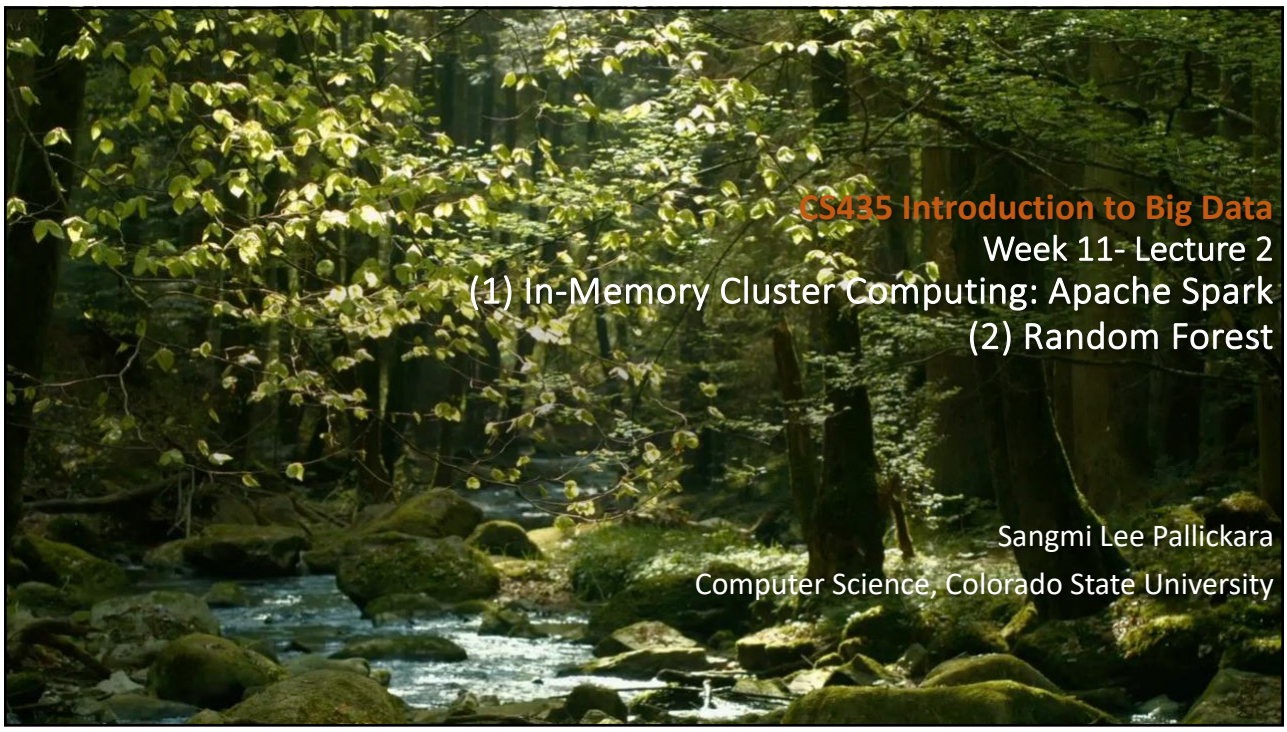
```
1: int counter = 0;
2: JavaRDD<Integer> rdd = sc.parallelize(data);
3:
4: rdd.foreach(x -> counter += x);
5:
6: println("Counter value: " + counter);
```

- **`counter`**(in line 4) is referenced within the `foreach` function, it's no longer the **`counter`** (in line 1) on the driver node
- **`counter`**(in line 1) will still be zero
- In local mode, in some circumstances the `foreach` function will actually execute within the same JVM as the driver
  - **`counter`** may be actually updated

Solutions?

- Closures (e.g. loops or locally defined methods) **should not** be used to mutate some global state
  - Spark does not define or guarantee the behavior of mutations to objects referenced from outside the closures
- **Accumulator** provides a mechanism for safely updating a variable when execution is split up across worker nodes in a cluster

Questions?



0

FAQs

- ReduceByKey() vs. Reduce()?
  - ReduceByKey() is a transformation
  - Reduce() is an action
  - Why?

1

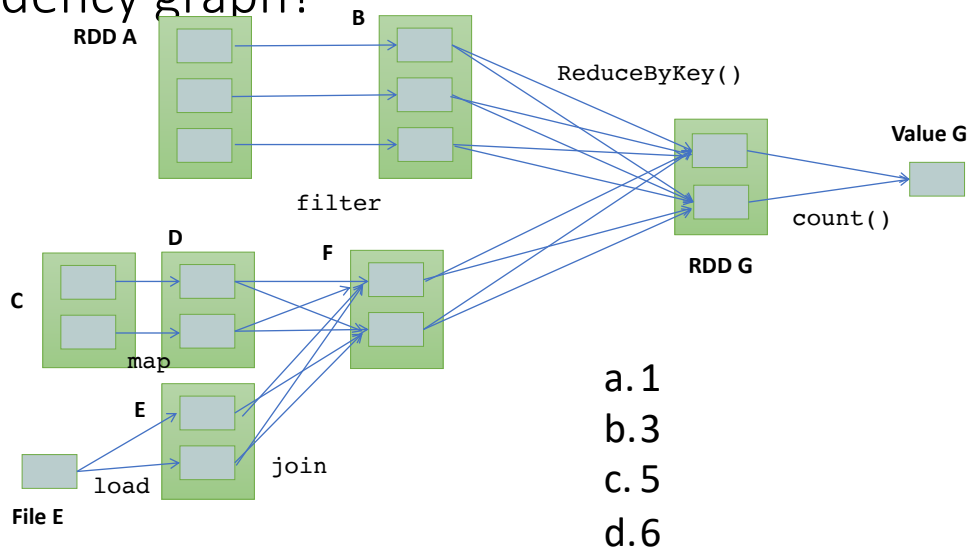
Topics discussed in the previous class

- In-memory cluster computing - Continued
  - Aggregation with PairRDD
  - Actions
  - RDD Dependency
    - Wide dependency vs. Narrow dependency
  - Application/Job/Stage/Task

Topics to be covered

- In-memory cluster computing - Continued
  - Partitions
  - Why RDD? Contrasting RDD to the traditional distributed memory approach

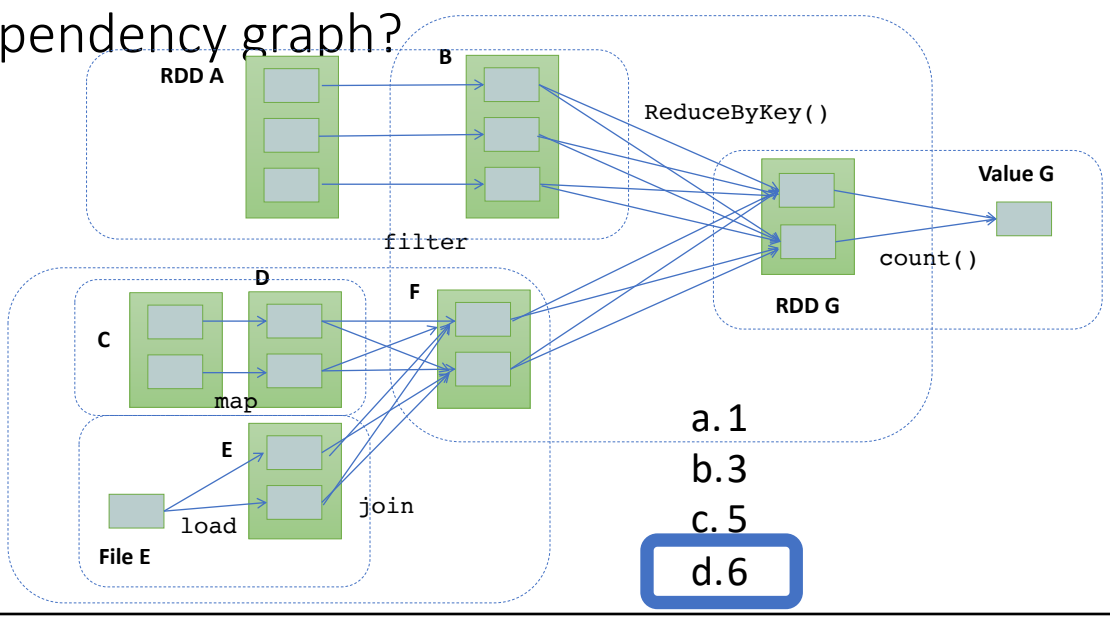
Review Question: How many Stages are in this RDD dependency graph?



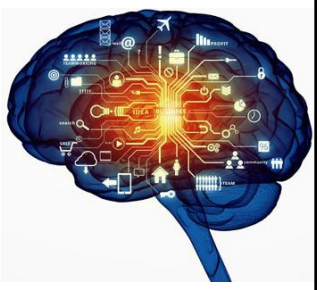
a. 1
b. 3
c. 5
d. 6

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 11-B Fall 2021 Computer Science

Review Question: How many Stages are in this RDD dependency graph?



a. 1
b. 3
c. 5
d. 6



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

This material is built based on

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,*" The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)
- Holden Karau, Andy Komwinski, Patrick Wendell and Matei Zaharia, "Learning Spark", O'Reilly, 2015
- Spark Overview, https://spark.apache.org/docs/2.3.0/
- Spark programming guide
  - Job Scheduling
    - https://spark.apache.org/docs/2.0.0-preview/job-scheduling.html

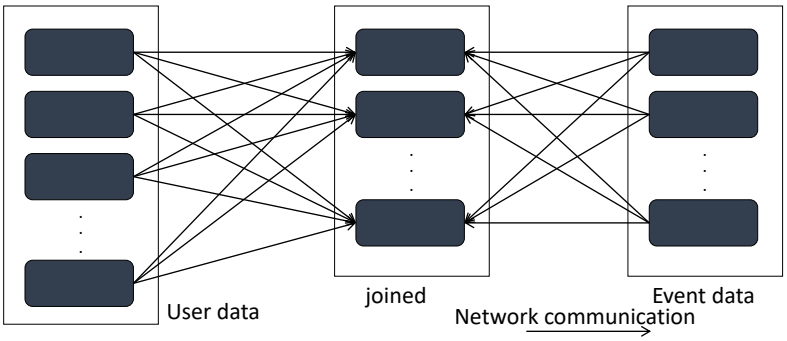
Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Data Partitioning

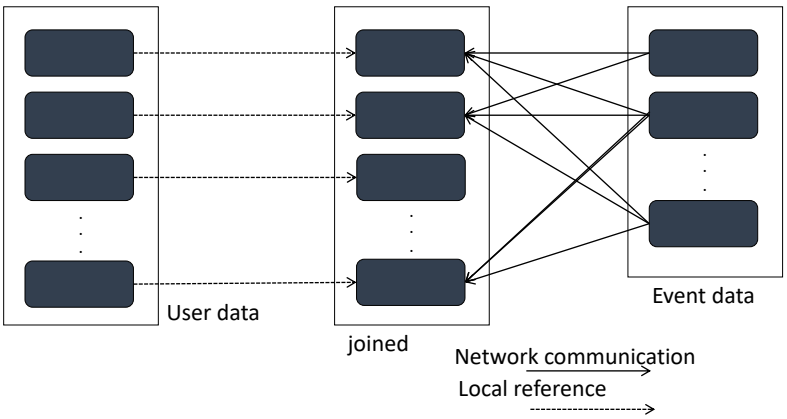
Why partitioning?

- Consider an application that keeps a large table of user information in memory
  - An RDD of `(UserID, UserInfo)` pairs
  - The application periodically combines this table with a smaller file representing events that happened in the last five minutes



Using `partitionBy()`

- Transforms `userData` to hash-partitioned RDD



Spark RDD Persistence

- Spark is persisting (or caching) a dataset in memory across operations
- Reuses them in other actions on that dataset
- Future actions will be much faster
- Key tool for iterative algorithms (>10x faster)
- Mark your RDD with persist() or cache()

Spark RDD persist() with a storage level

Storage Level	Description
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.

More options:
https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-persistence

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Distributed Memory Abstraction: Why RDD?

Benefits of RDDs as a distributed memory abstraction [1/7]

- RDD vs. Distributed Shared Memory (DSM)?
  - How does RDD work differently compared to DSM?
  - Write/Consistency/Fault-Recovery mechanism/Straggler mitigation
- RDDs can only be created ("written") through **coarse-grained transformations**
  - Coarse-grained transformations are applied over an entire dataset
  - Reads on RDDs can still be fine-grained
    - A large read-only lookup table
  - Applications perform bulk writes
- **More efficient fault tolerance**
  - Lineage based bulk recovery

Benefits of RDDs as a distributed memory abstraction [2/7]

RDD vs. DSM- Read operation

- RDD
  - Either coarse grained or fine grained
- DSM
  - The read operation in Distributed shared memory is fine-grained

Benefits of RDDs as a distributed memory abstraction [3/7]

RDD vs. DSM- Write operation

- RDD
  - The write operation in RDD is coarse grained
- DSM
  - The write operation in Distributed shared memory is fine-grained
  - Read and Write happens into the same location of memory

Benefits of RDDs as a distributed memory abstraction [4/7]

RDD vs. DSM- Consistency

- RDD
  - RDD is immutable in nature
    - Any changes on RDD is permanent
  - The level of consistency is high
- DSM
  - If the programmer follows the rules, the memory will be consistent, and the results of memory operations will be predictable

Benefits of RDDs as a distributed memory abstraction [5/7]

RDD vs. DSM- Fault-Recovery Mechanism

- RDD
  - The lost data can be easily recovered in Spark RDD using lineage graph at any moment
  - For each transformation, new RDD is formed
  - RDDs are immutable
- DSM
  - Fault tolerance is achieved by a checkpointing technique which allows applications to roll back to a recent checkpoint rather than restarting

Benefits of RDDs as a distributed memory abstraction [6/7]

RDD vs. DSM- Straggler Mitigation

- Stragglers
  - Nodes taking more time to complete than their peers
  - Due to load imbalance, I/O blocks, garbage collections, etc.
- RDD
  - **Creates backup copies of slow tasks**
    - without accessing the same memory
- DSM
  - It is quite difficult to achieve straggler mitigation

Benefits of RDDs as a distributed memory abstraction [7/7]

- Runtime can schedule tasks **based on data locality**
  - To improve performance
- RDDs degrade gracefully when there is **insufficient memory**
  - **Partitions that do not fit in the RAM are stored on disk**

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 11-B Fall 2021 Computer Science

Applications NOT suitable for RDDs

- RDDs are best suited for **batch applications** that apply the same operations to **all elements of a dataset**
  - Steps are managed by **lineage graph** efficiently
  - Recovery is managed effectively
- RDDs would not be suitable for applications
  - Making asynchronous fine-grained updates to shared state
  - e.g. a **storage system for a web application** or an incremental web crawler

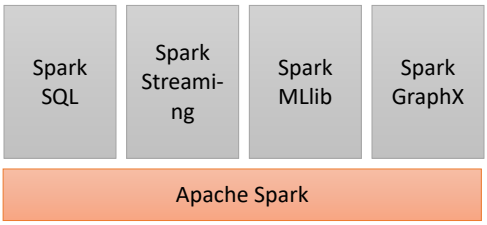
Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Software stack

Spark stack of libraries

- SQL and DataFrames
- Machine learning
- GraphX
- Spark Streaming
- 3rd party libraries
  - https://spark.apache.org/third-party-projects.html



Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

Regressions and Classifications

- **Regression analysis**
  - A statistical process for estimating the relationships among variables
  - Dependent variable vs. independent variable ("predictors")
  - How the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed
  - Predicting numerical values
    - Income, temperature, size
- **Classifications**
  - Identifying which of a set of categories a new observation belongs
  - Predicting a label or category
    - Spam, picture of a cat

Features and Feature Vectors

- `Input` and `output` to regression and classification
- Example
  - **Predicting tomorrows temperature given today's weather**
- Features (dimensions, predictors, or variables) of today's weather
  - Today's **high temperature**
  - Today's **low temperature**
  - Today's **average humidity**
  - **Whether it's cloudy, rainy, or clear today**
  - **The number of weather forecasters predicting a cold snap tomorrow**
- Each of these features can be quantified
- `(13.1, 19.0, 0.73, cloudy, 1)`
  - Feature vector

Features and Feature Vectors

- Feature vector
  - These features together, in order
  - `(13.1, 19.0, 0.73, cloudy, 1)`
  - Features do NOT need to be the same type
- **Categorical features**
  - Features that have no ordering
    - Cloudy, clear
- **Numeric features**
  - Features that can be quantified by a number and have a meaningful ordering
    - 23F, 56F (23F < 56F)

Training Examples

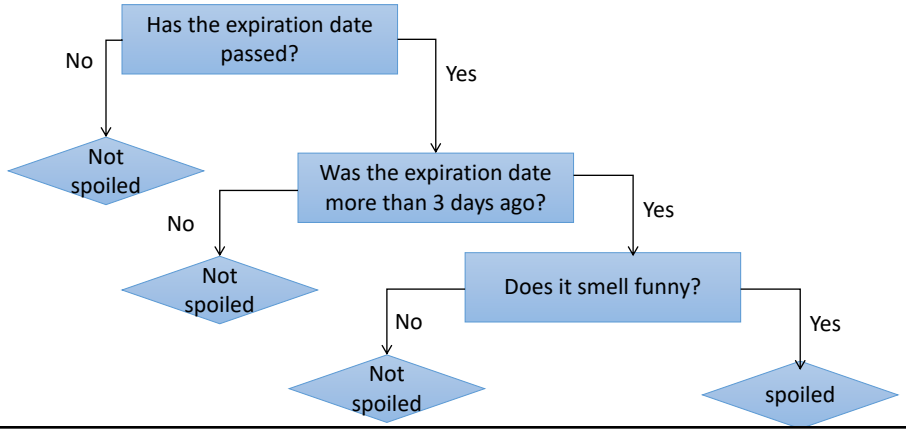
- A learning algorithm needs to train on data in order to make predictions
- Inputs
- Correct outputs (from historical data)
  - "One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was **17.2 degrees**"
  - **Target (output)**
    - 17.2 degree
- Feature vector often includes the target value
  - `(13.1, 19.0, 0.73, cloudy, 1, 17.2)`

Decision Trees and Forests

- Decision trees can naturally handle both categorical and numeric features
  - Easy to parallelize
  - Robust to **outliers**
    - A few extreme and possibly erroneous data points may not affect predictions at all
- Random Decision Forests
  - Extended decision tree algorithm

Decision tree

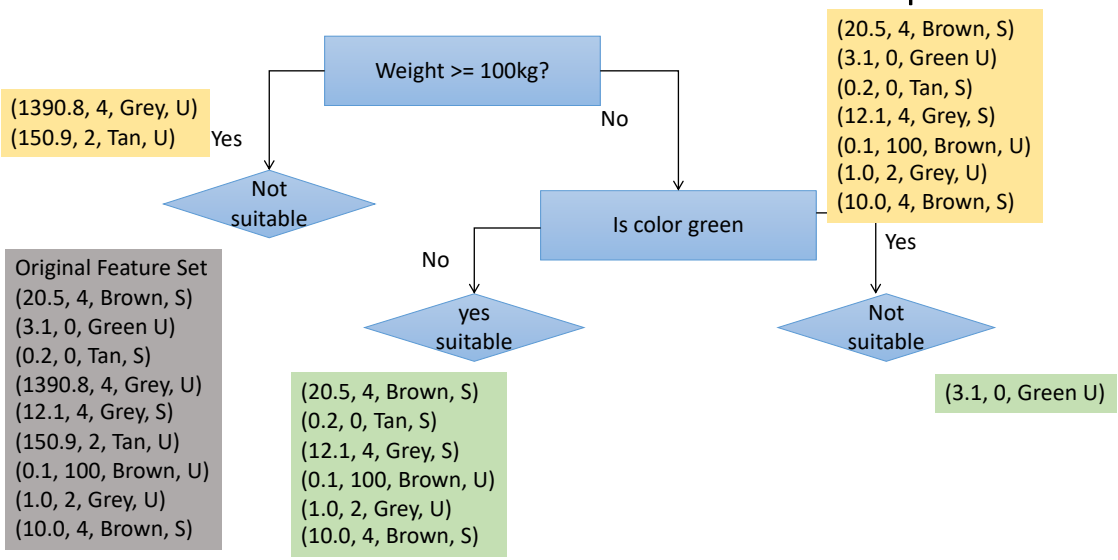
- Spark MLlib's DecisionTree and RandomForest implementation
- Each decision leads to one of two results
  - Prediction or another decision



Example: Finding a good pet for the customer

name	Weight (hg)	# of legs	color	Satisfactory?
Fido	20.5	4	Brown	S
Mr. Slither	3.1	0	Green	U
Nemo	0.2	0	Tan	S
Dumbo	1390.8	4	Grey	U
Kitty	12.1	4	Grey	S
Jim	150.9	2	Tan	U
Millie	0.1	100	Brown	U
McPigeon	1.0	2	Grey	U
Spot	10.0	4	Brown	S

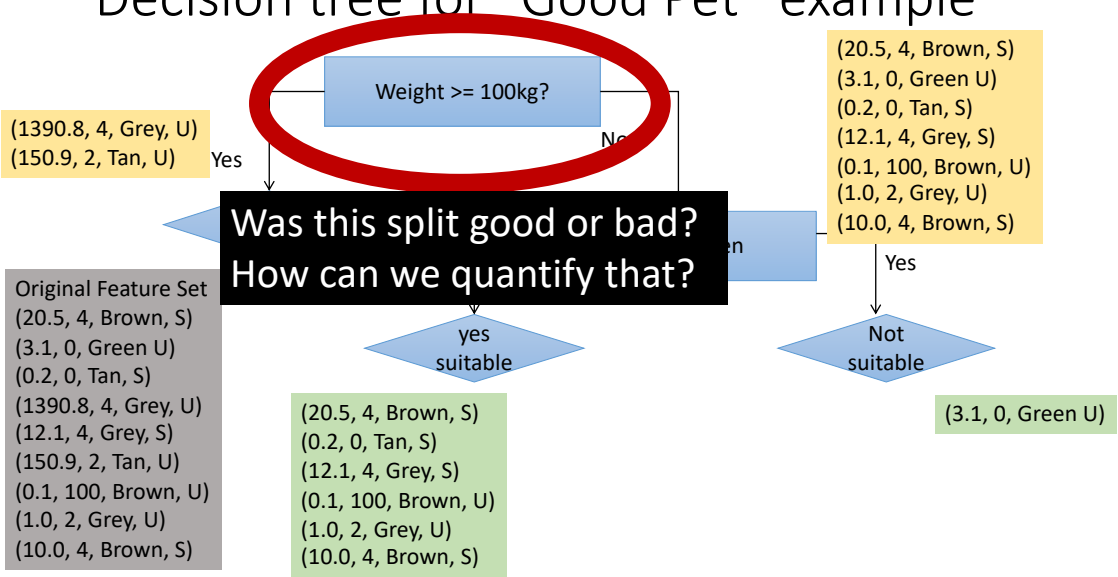
Decision tree for "Good Pet" example



32

32

Decision tree for "Good Pet" example



33

33

Gini impurity

- Gini impurity
  - Measuring impurity degree
  - Within a subset, the probability that a randomly chosen classification of a randomly chosen example is incorrect

Gini impurity – Example 1

- Example 1
  - Calculate the Gini impurity of our entire dataset
  - Random selection: Blue (50%), Green (50%)
  - Classification: it will be also "random"
  - What is the probability we classify our data points incorrectly?
  - **50%**

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

Gini impurity – Example 1: Random split

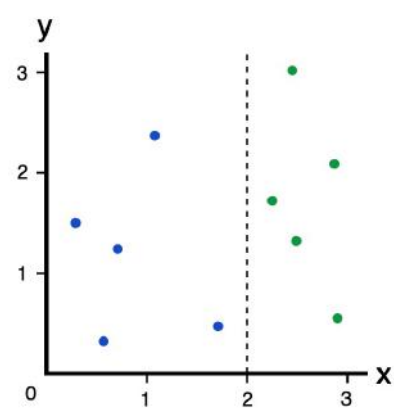
- If we have C total classes and $p(i)$ is the probability of picking a data point with class i
- Gini Impurity
  - $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
  - $C==2$ and $p(1)=p(2)=0.5$ (random classification)

$G=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
$=-0.5 \times (1-0.5)+0.5 \times (1-0.5)$
$=0.5$

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

Gini impurity – Example 2: Perfect split

- Left branch are only blues and right branch are only green
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=1 \times (1-1)+0 \times (1-0)$
  $=0$
- $G_{right}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=0 \times (1-0) + 1 \times (1-1)$
  $=0$
- **Both branches have 0 impurity! (Best Gini impurity)**

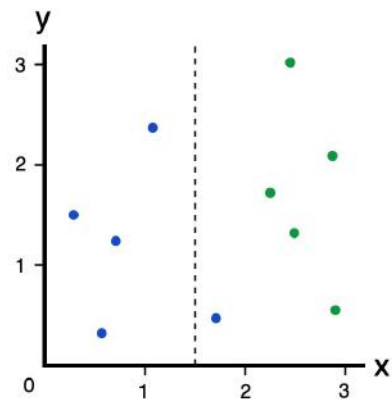


A Perfect Split

Gini impurity – Example 3: Imperfect split

- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $= 1 \times (1-1) + 0 \times (1-0)$
  $= 0$
- $G_{right} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $= 1/6 \times (1-1/6) + 5/6 \times (1-5/6)$
  $= 5/18 = 0.278$
- **Lower Gini impurity means better split**

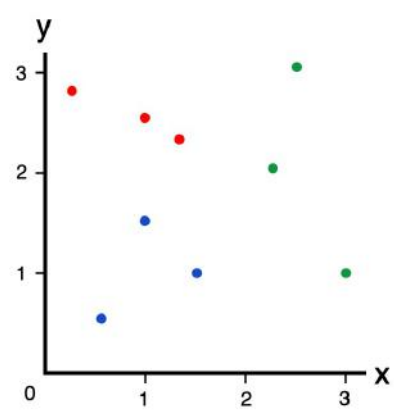
- **Gini Gain**
  - The quality of the split by weighting the impurity of each branch by how many elements it has
- $(0.4 \times 0) + (0.6 \times 0.278) = 0.167$
- **Higher Gini Gain means better split**



An Imperfect Split

Training a Decision Tree with Gini Impurity

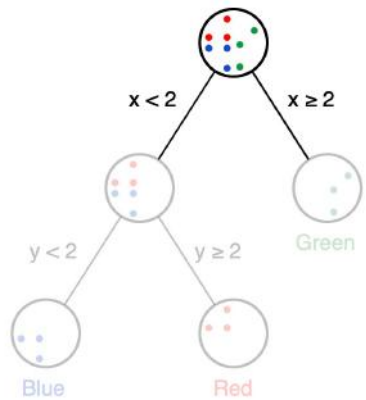
- The 3 class dataset
  - Red/blue/green with two features x and y



The Dataset v2

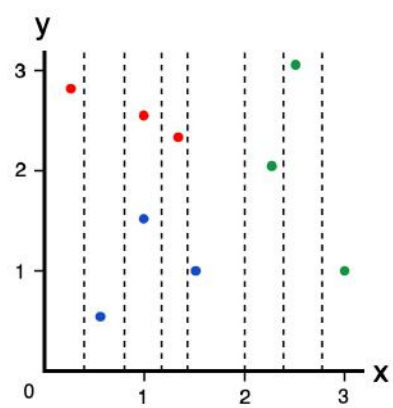
Training a Decision Tree with Gini Impurity : Root Node

- Finding the Root Node
  - Which feature do we have to use? X or Y?
  - What will be the test threshold? (x<2 or x<3)
- Example → see the picture
  - Using x feature with a test threshold of 2
- Good split
  - Separating different classes as much as possible



Training a Decision Tree with Gini Impurity : Root Node

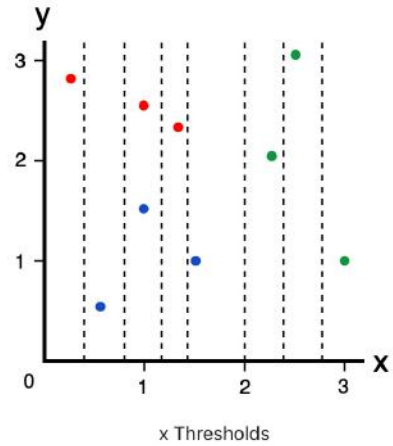
- Find the threshold that provides highest Gini Gain (lowest Gini Impurity)
- We will go over every possible split to find the best threshold
- First example: $x = 0.4$



Split	Left Branch	Right Branch
$x = 0.4$	●	● ● ● ● ● ● ●

Training a Decision Tree with Gini Impurity : Root Node

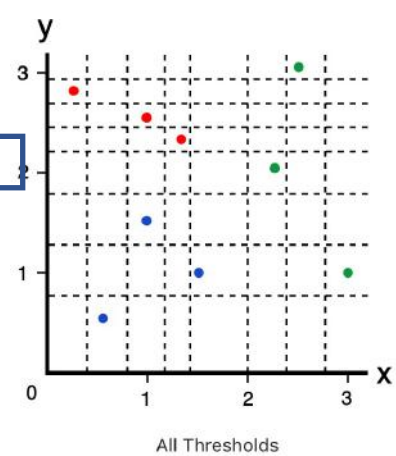
- Gini Impurity of the entire dataset
  - First example: $x = 0.4$
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{initial} = (1/3 \times 2/3) + (1/3 \times 2/3) + (1/3 \times 2/3) = 2/3$
- $G_{left} = 0 \times 1 + 1 \times 0 + 0 \times 1 = 0$
- $G_{right} = 3/8 \times 5/8 + 2/8 \times 6/8 + 3/8 \times 5/8 = 21/32$
- $Gain = G_{initial} - 1/9 G_{left} - 8/9 G_{right}$
  $= 2/3 - (1/9 \times 0) - (8/9 \times 21/32) = 0.083$



Split	Left Branch	Right Branch
$x = 0.4$	•	• • • • • • •

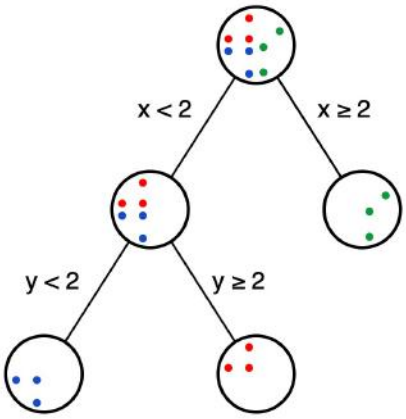
Training a Decision Tree with Gini Impurity : Root Node

Split	Left Branch	Right Branch	Gini Gain
$x = 0.4$	•	• • • • • • •	0.083
$x = 0.8$	• •	• • • • • • •	0.048
$x = 1.1$	• • • •	• • • •	0.133
$x = 1.3$	• • • •	• • • •	0.233
$x = 2$	• • • • • •	• • •	0.333
$x = 2.4$	• • • • • • •	• •	0.191
$x = 2.8$	• • • • • • • •	•	0.083
$y = 0.8$	•	• • • • • • •	0.083
$y = 1.2$	• • •	• • • • • •	0.111
$y = 1.8$	• • • •	• • • • •	0.233
$y = 2.1$	• • • • •	• • • •	0.233
$y = 2.4$	• • • • • •	• • •	0.111
$y = 2.7$	• • • • • • •	• •	0.048
$y = 2.9$	• • • • • • • •	•	0.083



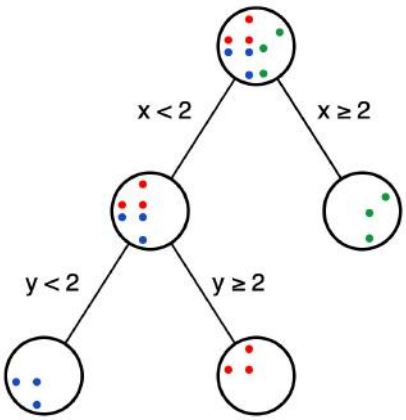
Training a Decision Tree with Gini Impurity : Second Node

- Do The Same Thing..



Training a Decision Tree with Gini Impurity : When to stop?

- Until have a Gini Gain of 0
- We trained a decision tree!



Entropy

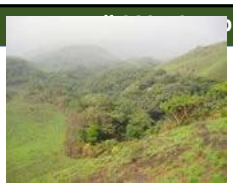
- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?

$$I_E(p) = \sum_{i=1}^{N} p_i \log(1/p) = -\sum_{i=1}^{N} p_i \log(p_i)$$

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests with MLlib



Covertype dataset

- Dataset with records of the types of forest covering parcels of land in Colorado, USA
  - Each example contains several features describing
    - Each parcel of land
    - Elevation, slope, distance to water, shade, and soil type
- The forest cover type is to be predicted
  - From the rest of features
  - Total 54 features
- Used in Kaggle competition
- It includes categorical and numeric features
- 581,012 examples
  - https://archive.ics.uci.edu/ml/datasets/Covertype

Attribute information

Name/ Data Type/ Measurement/ Description

Elevation **/ quantitative** /meters / Elevation in meters
Aspect / quantitative / azimuth / Aspect in degrees azimuth
Slope / quantitative / degrees / Slope in degrees
Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology / quantitative / meters / Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways / quantitative / meters / Horz Dist to nearest roadway
Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer soltice
Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns) **/ qualitative** / 0 (absence) or 1 (presence) / Wilderness area designation
Soil_Type (40 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Soil Type designation
Cover_Type (7 types) / integer **/ 1 to 7** / Forest Cover Type designation

Preparing data

- The `covtype.data` file should be extracted and copied into HDFS
  - File is available at `/user/ds/`

- `LabeledPoint`
  - The **Spark MLlib abstraction for a feature vector**
  - Consists of a Spark MLlib Vector of features, and a target value (label)
  - `LabeledPoint` is only for numeric features
    - It can be used with categorical features, with appropriate encoding

Using `LabeledPoint` for the categorical features

- **One-hot** coding
  - One categorical feature that takes on N distinct values becomes N numeric features, each taking on the value 0 or 1
  - Exactly one of the N values have value 1 and the others are 0
  - `Cloudy, rainy or clear`
    - `Cloudy: 1,0,0`
    - `Rainy: 0,1,0`
    - `Clear: 0,0,1`

- **1-of-n** coding
  - `Cloudy: 1`
  - `Rainy: 2`
  - `Clear: 3`

Question

Using `LabeledPoint` for the categorical features

- There are four categorical features, ”basketball” ,”soccer” ,”volleyball”, and “baseball”. What is the one-hot coding of “soccer”? Suppose that you use a feature vector of ((”basketball” ,”soccer” ,”volleyball”, “baseball”).

a) 1,0,0,0
b) 0,1,0,0
c) 0,0,1,0
d) 1,1,1,0

Question

Using `LabeledPoint` for the categorical features

- There are four categorical features, ”basketball” ,”soccer” ,”volleyball”, and “baseball”. What is the 1-of-n coding of “soccer”? Suppose that you use a code of ((”basketball”=1 ,”soccer”=2 ,”volleyball”=3, “baseball”=4).

a) 1
b) 2
c) 3
d) 4

Categorical values in `Covtype` data set

- The `covtype.info` file says that four of the columns are actually a one-hot encoding of a single categorical feature
  - `Wilderness_Type`, with four values
  - Likewise, 40 of the columns are really one `Soil_Type` categorical feature
- **The target itself is a categorical value encoded as the values 1 to 7**
- The remaining features are numeric features in various units, like meters, degrees, or a qualitative "index" value

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

A First Decision Tree

- Spark MLlib requires input in the form of `LabeledPoint` objects

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression._

val rawData = sc.textFile("hdfs:///user/ds/covtype.data")

val data = rawData.map { line = >
   val values = line.split(',').map(_.toDouble)
   val featureVector = Vectors.dense(values.init)

   val label = values.last - 1
   LabeledPoint(label, featureVector)
}
```


Splitting data

- Training, cross-validation, and test
  - 80% of data for **training** and 10% each **for cross-validation** and **test**
  - Training and CV sets are used to choose a good setting of **hyperparameters** for this data set
  - Test set is used to produce an unbiased evaluation of the expected accuracy of a model built with those **hyperparameters**

```
val Array(trainData, cvData, testData) =
                data.randomSplit(Array(0.8, 0.1, 0.1))

trainData.cache()

cvData.cache()

testData.cache()
```

Building a `DecisionTreeModel` on the training set

- Building a `DecisionTreeModel` on the training set with some default arguments
- Compute some metrics about the resulting model using the CV set

Building a `DecisionTreeModel` on the training set

```
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.tree.model._
import org.apache.spark.rdd._

def getMetrics(model: DecisionTreeModel, data:
RDD[LabeledPoint]):
     MulticlassMetrics = {
   val predictionsAndLabels = data.map(example
  = >
   (model.predict(example.features), example.label))
  new MulticlassMetrics(predictionsAndLabels)
}

val model = DecisionTree.trainClassifier(trainData, 7,
Map[Int, Int](), "gini", 4, 100)

val metrics = getMetrics(model, cvData)
```

Confusion matrix

- 7 x 7 matrix
- The row number corresponds to an actual correct value
- The column number corresponds to a predicted value

```
metrics.confusionMatrix

...
14019.0 6630.0  15.0    0.0   0.0  1.0   391.0
5413.0  22399.0 438.0   16.0  0.0  3.0   50.0
0.0     457.0   2999.0  73.0  0.0  12.0  0.0
0.0     1.0     163.0   117.0 0.0  0.0   0.0
0.0     872.0   40.0    0.0   0.0  0.0   0.0
0.0     500.0   1138.0  36.0  0.0  48.0  0.0
1091.0  41.0    0.0     0.0   0.0  0.0   891.0

metrics.precision
…
0.7030630195577938
```


Precision in multiclass metrics

- Binary classification
  - **Positive vs. negative** class
  - Precision is the fraction of examples that the classifier marked positive that are actually positive
    - $PPV = TP/(TP+FP)$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - $TPR = TP/P=TP/(TP+FN)$
- Multiclass problem
  - **Positive class vs. negative (all else)**

Is 70% accuracy good?

- Classifier that classifies at random in proportion to its prevalence in the training set
- What is the baseline ?
  - A broken clock will be correct twice a day
- Randomly guessing a classification would also be occasionally correct

Decision Tree Hyperparameters [1/2]

- **Hyperparameters**
  - Values we have to choose by building models
  - **Maximum depth, maximum bins, and impurity measure**
- **Maximum depth**
  - Limits the number of levels in the decision tree
  - Useful to avoid overfitting the training data
- **Maximum bins**
  - feature <= value
  - feature in (value1, value 2 ,…)
  - A larger number of bins requires more processing time
    - More optimal decision rule

Decision Tree Hyperparameters [2/2]

- Good rule should **distinguish examples more meaningfully**
- Example
  - E.g. a rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
- A good rule divides the training data's target values into relatively homogeneous or "pure" subsets
- **Minimizing the impurity of the two subsets**

Tuning Decision Trees

- Spark tries a number of combinations of impurity measure, maximum depth or number of bins and reports the results

```
val evaluations =
   for (impurity <- Array(" gini", "entropy");
        depth <- Array( 1, 20);
        bins <- Array( 10, 300))
     yield {
       val model = DecisionTree.trainClassifier(trainData, 7,
                Map[Int, Int](), impurity, depth, bins)
       val predictionsAndLabels = cvData.map(example = >
         (model.predict( example.features), example.label) )
       val accuracy =
          new MulticlassMetrics(predictionsAndLabels).
          precision ((impurity, depth, bins), accuracy)
      }
      evaluations.sortBy(_._2). reverse.foreach( println) …
```

Tuning Decision Trees

- continued

```
(( entropy, 20,300), 0.9125545571245186)
(( gini, 20,300), 0.9042533162173727)
(( gini, 20,10), 0.8854428754813863)
(( entropy, 20,10), 0.8848951647411211)
(( gini, 1,300), 0.6358065896448438)
(( gini, 1,10), 0.6355669661959777)
(( entropy, 1,300), 0.4861446298673513)
(( entropy, 1,10), 0.4861446298673513)
```


Categorical Features Revisited

- `Map[Int, Int]()`
  - Keys
    - **Indices of features** in the input Vector
  - Values
    - Distinct **value counts**
- Empty `Map()`
  - No features should be treated as categorical
  - All are numeric
- Numeric representation of categorical features
  - It can cause errors
  - The algorithm would be trying to learn from an ordering that has no meaning

Treating the categorical features with one-hot encoding

- Encodes the categorical features as several binary 0/1 values
- Any decision rule on the "numeric" features will choose thresholds between 0 and 1
  - All are equivalent since all values are 0 or 1
- Considers the values of the underlying categorical feature **individually**
  - Increases memory usage

Converting one-hot encoding to 1-n encoding [1/3]

```
val data = rawData.map { line = >
  val values = line.split(','). map(_.toDouble)
  val wilderness = values.slice(10, 14).indexOf(1.0). toDouble
  val soil = values.slice(14, 54).indexOf(1.0).toDouble
  val featureVector = Vectors.dense(values.slice(0, 10) :+
wilderness :+ soil)
  val label = values.last - 1
  LabeledPoint( label, featureVector)
}
```

- 4 "wilderness" features
- 40 "soil" features
- Add derived features back to first 10

Converting one-hot encoding to 1-n encoding [2/3]

```
val evaluations =
  for (impurity <- Array("gini", "entropy"); depth <- Array(
10, 20, 30); bins <- Array(40, 300))
  yield {
     val model =
   DecisionTree.trainClassifier(trainData,7,Map(10->4,11->40),
impurity, depth, bins)
     val trainAccuracy = getMetrics(model, trainData).
precision val cvAccuracy = getMetrics( model,cvData).
precision ((impurity, depth, bins),(trainAccuracy,cvAccuracy))
}
```

- Specify value count for categorical features 10, 11
  - Causes these features to be treated as categorical

Converting one-hot encoding to 1-n encoding [3/3]

```
(( entropy, 30,300),( 0.9996922984231909,0.9438383977425239))
(( entropy, 30,40),( 0.9994469978654548,0.938934581368939)) ((
gini, 30,300),( 0.9998622874061833,0.937127912178671)) ((
gini, 30,40),( 0.9995180059216415,0.9329467634811934)) ((
entropy, 20,40),( 0.9725865867933623,0.9280773598540899)) ((
gini, 20,300),( 0.9702347139020864,0.9249630062975326)) ((
entropy, 20,300),( 0.9643948392205467,0.9231391307340239)) ((
gini, 20,40),( 0.9679344832334917,0.9223820503114354)) ((
gini, 10,300),( 0.7953203539213661,0.7946763481193434)) ((
gini, 10,40),( 0.7880624698753701,0.7860215423792973))
…
```

- Tree-building process completes several times **faster**
- By treating categorical features as categorical features, it **improves accuracy by almost 3%**

Ensemble Approach: 1. Bagged Decision Tree

- Bagging
  - Bootstrap Aggregating

Step 1) Sample n data points with replacement

Step 2) Train with n data points

Step 3) repeat step 1 and 2, t times

We have t decision trees

- Take the **majority vote** if our trees produce class labels (like colors)
- Take the **average** if our trees produce numerical values (e.g. when predicting temperature, price, etc)

Ensemble Approach: 2. Random Forests

- Number of trees and **number of features to try when finding the best split**
- Dataset with p features
- Select a subset of features for each tree
  - Usually $\sqrt{p}$ or $p/3$
  - Feature bagging
- This provides randomness that makes
  - Individual tree more unique
  - Reduces correlation between trees

RandomForest

```
val forest = RandomForest.trainClassifier(
       trainData, 7, Map( 10 -> 4, 11 -> 40), 20,
       "auto", "entropy", 30, 300)
```

- Number of trees to build
  - Here, 20
- "auto"
  - The strategy for choosing which features to evaluate at each level of the tree
  - The random decision forest implementation will NOT even consider every feature as the basis of a decision rule
    - Only a subset of all features

Making predictions

- The results of the `DecisionTree` and `RandomForest` training
  - `DecisionTreeModel` and `RandomForestModel` objects
  - `predict()` method
    - Accepts a Vector object
  - We can classify a new example by converting it to a feature vector in the same way and predicting its target class

```
val input = "2709,125,28,67,23,3224,253,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble))
forest.predict(vector)
```

Part 2. Large Scale Data Storage Systems

Distributed File System

Distributed file systems

- Hadoop vs. Google File System?

Part 1. Large scale data analysis using MapReduce

Distributed File System

Google File System (GFS)

A server room in Council Bluffs, Iowa

- Photo: Google/C

Data Center

- https://youtube.googleapis.com/v/avP5d16wEp0%26fs=1%26autoplay=1

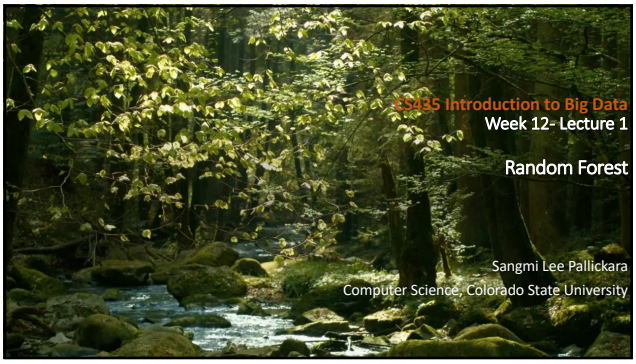
Data centers

- US data centers consumes about 70 billion kilowatt-hours of electricity in 2014
  - 2 percent of country's total energy consumption
  - 6.4 million average American homes

Questions?

Questions?

Questions?



0

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 12-A Fall 2021 Computer Science

FAQs

- Please sign up for the PA3 demo
- Term project
  - Regular meetings are important!

1

1

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 12-A Fall 2021 Computer Science

Topics discussed in the previous class

- In-memory cluster computing - Continued
  - Partitions
  - Why RDD? Contrasting RDD to the traditional distributed memory approach
- Introduction to Random Forest

2

2

Topics to be covered

- Decision Tree and Random Forest

3

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

4

REVIEW

Features and Feature Vectors

- Feature vector
  - These features together, in order
  - `(13.1, 19.0, 0.73, cloudy, 1)`
  - Features do NOT need to be the same type
- **Categorical features**
  - Features that have no ordering
    - Cloudy, clear
- **Numeric features**
  - Features that can be quantified by a number and have a meaningful ordering
    - 23F, 56F (23F < 56F)

5

REVIEW

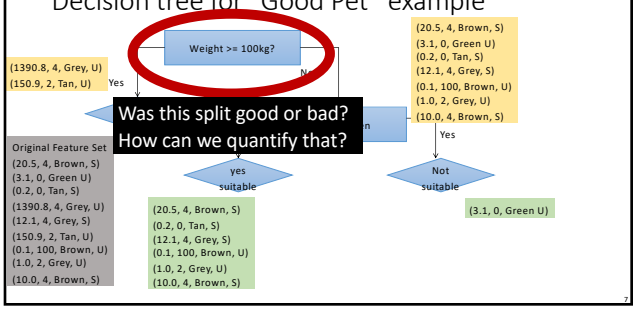
Training Examples

- A learning algorithm needs to train on data in order to make predictions
- Inputs
- Correct outputs (from historical data)
  - "One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was **17.2 degrees**"
  - **Target (output)**
    - 17.2 degree
- Feature vector often includes the target value
  - `(13.1, 19.0, 0.73, cloudy, 1, 17.2)`

6

REVIEW

Decision tree for "Good Pet" example



7

REVIEW

Gini impurity

- Gini impurity
  - Measuring impurity degree
    - Does your classifier collect the similar items?
  - Within a subset, the probability that a randomly chosen classification of a randomly chosen example is incorrect

8

REVIEW

Gini impurity – Example 1

- Example 1
  - Calculate the Gini impurity of our entire dataset
  - Random selection: Blue (50%), Green (50%)
  - Classification: it will be also "random"
  - What is the probability we classify our data points incorrectly?
  - **50%**

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

9

Gini impurity – Example 1: Random split

- If we have C total classes and $p(i)$ is the probability of picking a data point with class i within the split
  - *In this case, C==2 (blue or green)*
  - The probability of picking a data point with class *blue = 0.5 (random)*
  - The probability of picking a data point with class *green = 0.5 (random)*
- Gini Impurity
  - $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
  - $C==2$ and $p(blue)=p(green)=0.5$ (random classification)

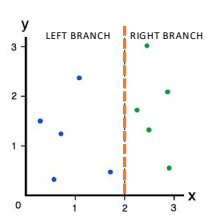
$G=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
$=0.5 \times (1-0.5)+0.5 \times (1-0.5)$
$=0.5$

Event	Probability
Pick blue and classify blue (correct)	25%
Pick blue and classify green (incorrect)	25%
Pick green and classify blue (incorrect)	25%
Pick green and classify green (correct)	25%

10

Gini impurity – Example 2: Perfect split

- Left branch are only blues and right branch are only green
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=1 \times (1-1)+0 \times (1-0)$
  $=0$
- $G_{right}=p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  $=0 \times (1-0) + 1 \times (1-1)$
  $=0$
- **Both branches have 0 impurity! (Best Gini impurity)**



A Perfect Split

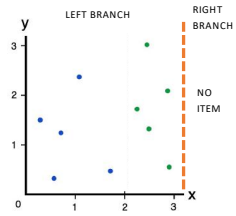
11

Gini impurity – Example 3: Poor split

- Left branch are all the blues and green
- Right branch has no item

- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$

- G_{left}=$p(1)$ x $(1 - p(1)) + p(2)$ x $(1 - p(2))$
  $=5/10$ x $(1-5/10)+4/10$ x $(1-4/10)$
  $=5/10$ x $5/10 + 5/10$ x $5/10 = 50/100 = 0.5$
- G_{right}=$p(1)$ x $(1 - p(1)) + p(2)$ x $(1 - p(2))$
  $=0$x $(1-0) + 0$ x $(1-0)$
  $=0$



A Perfect Split

12

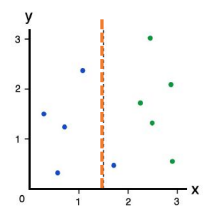
Gini impurity – Example 3: Imperfect split

- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$

- G_{left}=$p(1)$ x $(1 - p(1)) + p(2)$ x $(1 - p(2))$
  $=1$ x $(1-1)+0$x $(1-0)$
  $=0$
- G_{right}=$p(1)$ x $(1 - p(1)) + p(2)$ x $(1 - p(2))$
  $=1/6$ x $(1-1/6) + 5/6$ x $(1-5/6)$
  $=5/18 =0.278$
- **Lower Gini impurity means better split**

- **Gini Gain**
  - The quality of the split by weighting the impurity of each branch by how many elements it has
  - $(0.4$ x $0) + (0.6$ x $0.278) = 0.167$
  - ~~Lower Gini Gain means better split~~
  - Higher Gini Gain means better split
- **NOTE: Spark MLlib uses Information Gain:** the difference between the parent node impurity and the weighted sum of the two child node impurities

$$IG(D,s)=Impurity(D)-NleftNImpurity(Dleft)-NrightNImpurity(Dright)$$

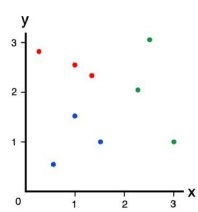


An Imperfect Split

13

Training a Decision Tree with Gini Impurity

- The 3 class dataset
  - Red/blue/green with two features x and y

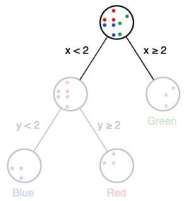


The Dataset v2

14

Training a Decision Tree with Gini Impurity : Root Node

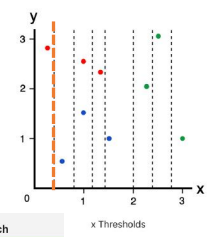
- Finding the Root Node
  - Which feature do we have to use? X or Y?
  - What will be the test threshold? (x<2 or x<3)
- Example → see the picture
  - Using x feature with a test threshold of 2
- Good split
  - Separating different classes as much as possible



15

Training a Decision Tree with Gini Impurity : Root Node

- Find the threshold that provides highest Gini Gain (lowest Gini Impurity)
- We will go over every possible split to find the best threshold
- First example: $x = 0.4$



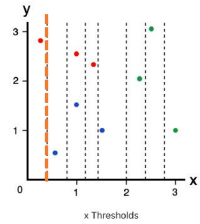
Split	Left Branch	Right Branch
$x = 0.4$	●	●●●●●●●●

16

Training a Decision Tree with Gini Impurity : Root Node

- Gini Impurity of the entire dataset
  - First example: $x = 0.4$
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{initial}$= (1/3 x 2/3) + (1/3 x 2/3) + (1/3 x 2/3) =2/3
- G_{left}=0 x 1 + 1 x 0 + 0 x 1 = 0
- G_{right}=3/8 x 5/8 + 2/8 x 6/8 + 3/8 x 5/8 =21/32
- Gain = $G_{initial}$ -1/9G_{left} – 8/9G_{right}

  =2/3-(1/9 x 0)-(8/9 x 21/32)=0.083



Split	Left Branch	Right Branch
$x = 0.4$	●	●●●●●●●●

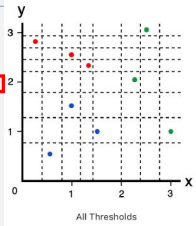
17

Training a Decision Tree with Gini Impurity : Root Node (pick the highest Gini Gain)

Try the feature x

Try the feature y

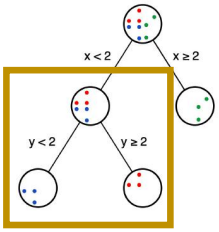
Split	Left Branch	Right Branch	Gini Gain
$x = 0.4$	1	7	0.083
$x = 0.8$	2	7	0.048
$x = 1.1$	4	4	0.133
$x = 1.3$	5	4	0.233
$x = 2$	6	3	0.333
$x = 2.4$	7	2	0.191
$x = 2.8$	8	1	0.083
$y = 0.8$	1	8	0.083
$y = 1.2$	3	6	0.111
$y = 1.8$	4	5	0.233
$y = 2.1$	5	4	0.233
$y = 2.4$	5	3	0.111
$y = 2.7$	7	2	0.048
$y = 2.9$	8	1	0.083



18

Training a Decision Tree with Gini Impurity : Second Node

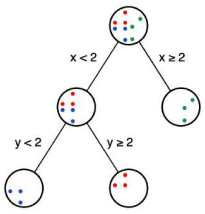
- Do The Same Thing for the child node



19

Training a Decision Tree with Gini Impurity : When to stop?

- Until have a Gini Gain of 0
- We trained a decision tree!



20

Entropy

- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?

$$I_E(p) = \sum_{i=1}^{N} p_i \log(1/p) = -\sum_{i=1}^{N} p_i \log(p_i)$$

- **Information Gain** is the Entropy of the parent node minus the entropy of the child nodes

21

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests with MLlib

22

`Covertype` dataset

- Dataset with records of the types of forest covering parcels of land in Colorado, USA
  - Each example contains several features describing
    - Each parcel of land
    - Elevation, slope, distance to water, shade, and soil type
- The forest cover type is to be predicted
  - From the rest of features
  - Total 54 features
- Used in Kaggle competition
- It includes categorical and numeric features
- 581,012 examples
  - https://archive.ics.uci.edu/ml/datasets/Covertype

23

Attribute information

Name/ Data Type/ Measurement/ Description

Elevation / **quantitative** /meters / Elevation in meters
Aspect / quantitative / azimuth / Aspect in degrees azimuth
Slope / quantitative / degrees / Slope in degrees
Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology / quantitative / meters / Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways / quantitative / meters / Horz Dist to nearest roadway
Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer soltice
Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Wilderness area designation
Soil_Type (40 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Soil Type designation
Cover_Type (7 types) / integer / **1 to 7** / Forest Cover Type designation

24

Preparing data

- The `covtype.data` file should be extracted and copied into HDFS
  - File is available at `/user/ds/`
- `LabeledPoint` (for the RDD version)
  - The **Spark MLlib abstraction for a feature vector**
  - Consists of a Spark MLlib Vector of features, and a target value (label)
  - `LabeledPoint` is only for numeric features
    - It can be used with categorical features, with appropriate encoding

25

Using `LabeledPoint` for the categorical features

- **One-hot** coding
  - One categorical feature that takes on N distinct values becomes N numeric features, each taking on the value 0 or 1
  - Exactly one of the N values have value 1 and the others are 0
  - `Cloudy, rainy or clear`
    - `Cloudy: 1,0,0`
    - `Rainy: 0,1,0`
    - `Clear: 0,0,1`
- **1-of-n** coding
  - `Cloudy: 1`
  - `Rainy: 2`
  - `Clear: 3`

26

Question

Using `LabeledPoint` for the categorical features

- There is a categorical feature with four values, "basketball" ,"soccer" ,"volleyball", and "baseball". What is the one-hot coding of "soccer"? Suppose that you use an encoding scheme of (("basketball" ,"soccer" ,"volleyball", "baseball").

a) 1,0,0,0
b) 0,1,0,0
c) 0,0,1,0
d) 1,1,1,0

27

Question

Using `LabeledPoint` for the categorical features

- There is a categorical feature with four values, "basketball" ,"soccer" ,"volleyball", and "baseball". What is the 1-of-n coding of "soccer"? Suppose that you use an encoding scheme of (("basketball"=1 ,"soccer"=2 ,"volleyball"=3, "baseball"=4).

a) 1
b) 2
c) 3
d) 4

28

Categorical features in `Covtype` data set

- The `covtype.info` file says that four of the columns are actually a one-hot encoding of a single categorical feature
  - `Wilderness_Type`, with four values
  - Likewise, 40 of the columns are really one `Soil_Type` categorical feature
- **The target itself is a categorical value encoded as the values 1 to 7**
- The remaining features are numeric features in various units, like meters, degrees, or a qualitative "index" value

29

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Tree and Random Forest

(1) Decision Tree

30

A First Decision Tree

- Spark MLlib requires input in the form of `LabeledPoint` objects

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression._

val rawData = sc.textFile("hdfs:///user/ds/covtype.data")

val data = rawData.map { line = >
   val values = line.split(',').map(_.toDouble)
   val featureVector = Vectors.dense(values.init)

   val label = values.last - 1
   LabeledPoint(label, featureVector)
}
```


31

Splitting data

- Training, cross-validation, and test
  - 80% of data for **training** and 10% each **for cross-validation** and **test**
  - Training and CV sets are used to choose a good setting of **hyperparameters** for this data set
  - Test set is used to produce an unbiased evaluation of the expected accuracy of a model built with those **hyperparameters**

```
val Array(trainData, cvData, testData) =
                                data.randomSplit(Array(0.8, 0.1, 0.1))

trainData.cache()

cvData.cache()

testData.cache()
```


32

Building a `DecisionTreeModel` on the training set

- Building a `DecisionTreeModel` on the training set with some default arguments
- Compute some metrics about the resulting model using the CV set

33

Building a `DecisionTreeModel` on the training set

```
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.tree.model._
import org.apache.spark.rdd._

def getMetrics(model: DecisionTreeModel, data: RDD[LabeledPoint]):
     MulticlassMetrics = {
   val predictionsAndLabels = data.map(example
  = >
   (model.predict(example.features), example.label))
  new MulticlassMetrics(predictionsAndLabels)
}

val model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int](), "gini",
4, 100)

val metrics = getMetrics(model, cvData)
```


34

Confusion matrix

- 7 x 7 matrix
- The row number corresponds to an actual correct value
- The column number corresponds to a predicted value

```
metrics.confusionMatrix

...
14019.0 6630.0  15.0    0.0   0.0  1.0   391.0
5413.0  22399.0 438.0   16.0  0.0  3.0   50.0
0.0     457.0   2999.0  73.0  0.0  12.0  0.0
0.0     1.0     163.0   117.0 0.0  0.0   0.0
0.0     872.0   40.0    0.0   0.0  0.0   0.0
0.0     500.0   1138.0  36.0  0.0  48.0  0.0
1091.0  41.0    0.0     0.0   0.0  0.0   891.0

metrics.precision
…
0.7030630195577938
```

Example

TP/FP/TN/FN of the type 1

In the first column, out of 20523 items,
14019 items were classified correctly.

5413 items were classified as the second type
1091 items were classified as the seventy type

True positive of the type 1: 14019
False positive of the type 1: (5413+1091)
True negative of the type 1: neither the prediction nor the label take on the value of a given class
- All the items except the first row and the first column
False negative of the type 1:
# of items listed in the first row except the correct label (14019)

35

Precision in multiclass metrics

- Binary classification
  - **Positive vs. negative** class
  - Precision is the fraction of examples that the classifier marked positive that are actually positive
    - $PPV = TP/(TP+FP)$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - $TPR = TP/P=TP/(TP+FN)$
- Multiclass problem
  - **Positive class vs. negative (all else)**
  - **Precision by labels**

$$PPV(\ell) = \frac{TP}{TP+FP} = \frac{\sum_{i=0}^{N-1} \delta(\hat{y}_i = \ell)\delta(y_i = \ell)}{\sum_{i=0}^{N-1} \delta(\hat{y}_i = \ell)}$$

  - https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html

36

Is 70% accuracy good?

- Classifier that classifies at random in proportion to its prevalence in the training set
- What is the baseline ?
  - A broken clock will be correct twice a day
- Randomly guessing a classification would also be occasionally correct

37

Decision Tree Hyperparameters [1/2]

- **Hyperparameters**
  - Values we have to choose by building models
  - **Maximum depth, maximum bins, and impurity measure**
- **Maximum depth**
  - Limits the number of levels in the decision tree
  - Useful to avoid overfitting the training data
- **Maximum bins**
  - feature <= value
  - feature in (value1, value 2 ,…)
  - A larger number of bins requires more processing time
    - More optimal decision rule

38

Decision Tree Hyperparameters [2/2]

- Good rule should **distinguish examples more meaningfully**
- Example
  - E.g. a rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
- A good rule divides the training data's target values into relatively homogeneous or "pure" subsets
- **Minimizing the impurity of the two subsets**

39

Tuning Decision Trees

- Spark tries a number of combinations of impurity measure, maximum depth or number of bins and reports the results

```
val evaluations =
   for (impurity <- Array(" gini", "entropy");
        depth <- Array( 1, 20);
        bins <- Array( 10, 300))
     yield {
       val model = DecisionTree.trainClassifier(trainData, 7,
               Map[Int, Int](), impurity, depth, bins)
       val predictionsAndLabels = cvData.map(example = >
         (model.predict( example.features), example.label) )
       val accuracy =
          new MulticlassMetrics(predictionsAndLabels).
          precision ((impurity, depth, bins), accuracy)
     }
     evaluations.sortBy( . 2). reverse.foreach( println) …
```


40

Tuning Decision Trees

- continued

```
(( entropy, 20,300), 0.9125545571245186)
(( gini, 20,300), 0.9042533162173727)
(( gini, 20,10), 0.8854428754813863)
(( entropy, 20,10), 0.8848951647411211)
(( gini, 1,300), 0.6358065896448438)
(( gini, 1,10), 0.6355669661959777)
(( entropy, 1,300), 0.4861446298673513)
(( entropy, 1,10), 0.4861446298673513)
```


41

Treating the categorical features with one-hot encoding

- Encodes the categorical features as several binary 0/1 values
- Any decision rule on the "numeric" features will choose thresholds between 0 and 1
  - All are equivalent since all values are 0 or 1
- Considers the values of the underlying categorical feature individually
  - Increases memory usage

42

Converting one-hot encoding to 1-n encoding [1/3]

```
val data = rawData.map { line = >
  val values = line.split(','). map(_.toDouble)
  val wilderness = values.slice(10, 14).indexOf(1.0). toDouble
  val soil = values.slice(14, 54).indexOf(1.0).toDouble
  val featureVector = Vectors.dense(values.slice(0, 10) :+
wilderness :+ soil)
  val label = values.last - 1
  LabeledPoint( label, featureVector)
}
```

- 4 "wilderness" features
- 40 "soil" features
- Add derived features back to first 10

43

Converting one-hot encoding to 1-n encoding [2/3]

```
val evaluations =
  for (impurity <- Array("gini", "entropy"); depth <- Array(
10, 20, 30); bins <- Array(40, 300))
  yield {
     val model =
    DecisionTree.trainClassifier(trainData,7,Map(10->4,11->40),
impurity, depth, bins)
     val trainAccuracy = getMetrics(model, trainData).
precision val cvAccuracy = getMetrics( model,cvData).
precision ((impurity, depth, bins),(trainAccuracy,cvAccuracy))
}
```

- Specify value count for categorical features 10, 11
  - Causes these features to be treated as categorical

44

Converting one-hot encoding to 1-n encoding [3/3]

```
(( entropy, 30,300),( 0.9996922984231909,0.9438383977425239))
(( entropy, 30,40),( 0.9994469978654548,0.938934581368939)) ((
gini, 30,300),( 0.9998622874061833,0.937127912178671)) ((
gini, 30,40),( 0.9995180059216415,0.9329467634811934)) ((
entropy, 20,40),( 0.9725865867933623,0.9280773598540899)) ((
gini, 20,300),( 0.9702347139020864,0.9249630062975326)) ((
entropy, 20,300),( 0.9643948392205467,0.9231391307340239)) ((
gini, 20,40),( 0.9679344832334917,0.9223820503114354)) ((
gini, 10,300),( 0.7953203539213661,0.7946763481193434)) ((
gini, 10,40),( 0.7880624698753701,0.7860215423792973))
…
```

- Tree-building process completes several times faster
- By treating categorical features as categorical features, it **improves accuracy by almost 3%**

45

Large Scale Data Analytics

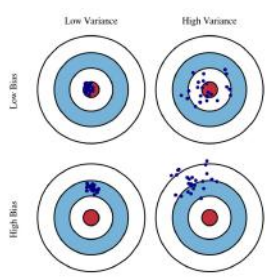
In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Tree and Random Forest
(2) Random Forest

46

Ensemble models

- Combine multiple other models in the prediction process
  - Base estimators
- High variance
  - The model is sensitive to the input values
- Low accuracy
  - One model or one algorithm to fit the entire training data might not be good enough to meet expectations
- Features noise and bias
  - The model relies heavily on one or a few features while making a prediction



Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

47

Ensemble Approach: 1. Bagged Decision Tree

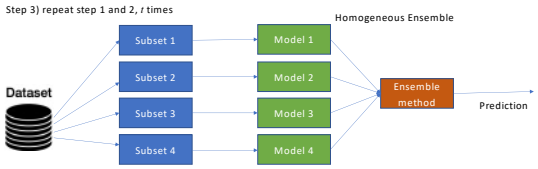
Bagging

Bootstrap Aggregating

Step 1) Sample n data points with replacement

Step 2) Train with n data points

Step 3) repeat step 1 and 2, t times



We have t decision trees

Take the **majority vote** if our trees produce class labels (like colors)

Take the **average** if our trees produce numerical values (e.g. when predicting temperature, price, etc)

48

Ensemble Approach: 2. Random Forests

- Number of trees and **number of features to try when finding the best split**
- Dataset with p features
- Select a subset of features for each tree
  - Usually $\sqrt{p}$ or $p/3$
  - Feature bagging
- This provides randomness that makes
  - Individual tree more unique
  - Reduces correlation between trees

49

RandomForest

```
val forest = RandomForest.trainClassifier(
      trainData, 7, Map( 10 -> 4, 11 -> 40), 20,
      "auto", "entropy", 30, 300)
```

- Number of trees to build
  - Here, 20
- "auto"
  - The strategy for choosing which features to evaluate at each level of the tree
  - The random decision forest implementation will NOT even consider every feature as the basis of a decision rule
    - Only a subset of all features

50

Making predictions

- The results of the `DecisionTree` and `RandomForest` training
  - `DecisionTreeModel` and `RandomForestModel` objects
  - `predict()` method
    - Accepts a Vector object
  - We can classify a new example by converting it to a feature vector in the same way and predicting its target class

```
val input = "2709,125,28,67,23,3224,253,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble))
forest.predict(vector)
```


51

Questions?

52

0

FAQs

- How does Random Forest achieve parallelism with Spark?

1

Topics discussed in the previous class

- Classification model
  - Decision Tree and Random Forest

Topics to be covered

- Predictive analytics
  - Linear Regression
  - MapReduce and Apache Spark

Predictive Analysis

Linear Regression at Scale

This material is developed based on,

- Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. 2006. Map-reduce for machine learning on multicore. In Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06), B. Schölkopf, J. C. Platt, and T. Hoffman (Eds.). MIT Press, Cambridge, MA, USA, 281-288.

Predictive Analysis

Linear Regression: Concepts

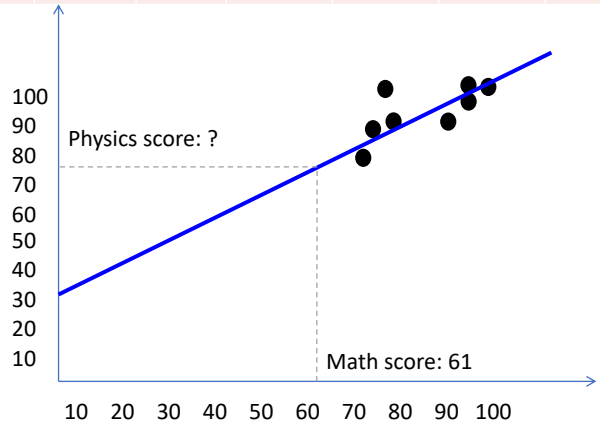
Linear regression [1/3]

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80

- Is there **any correlation** between a student's Math score and Physic score?
- If a student's Math score is known, can we predict his/her Physic score?

Linear regression [2/3]

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80



Linear regression [3/3]

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80

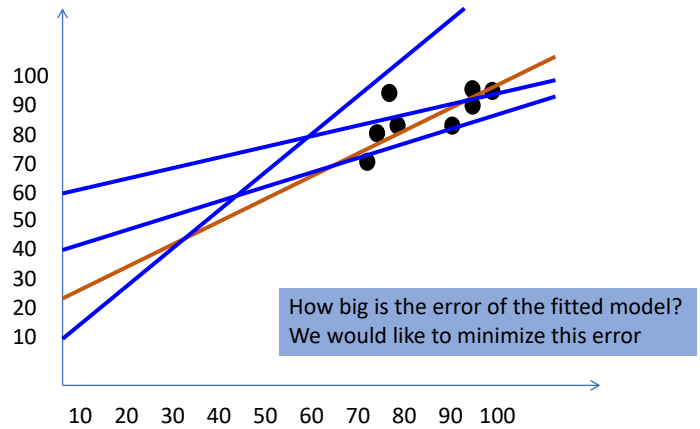
$$h_\theta(x) = \theta_0 + \theta_1 x$$

where, the θs are the **parameter vectors**
$h_\theta(x)$, (Estimated Physics score) is called the regressand or endogeneous variable
x_1 , (Math score) is called regressor, or exogeneous variable

Fitting the linear regression model [1/3]

- The structure

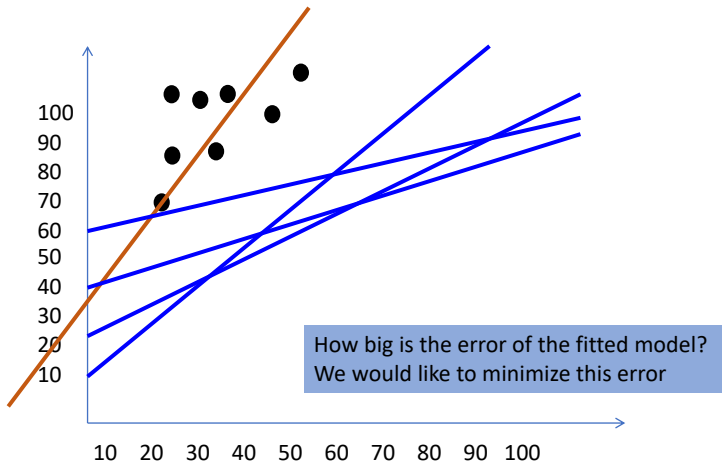
$$h_\theta(x) = \theta_0 + \theta_1 x$$



Fitting the linear regression model [2/3]

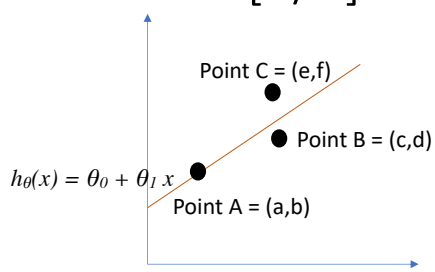
- The structure

$$h_\theta(x) = \theta_0 + \theta_1 x$$



Fitting the linear regression model [3/3]

- How **big is the error** of the fitted model?
  - We would like to minimize this error
  - Absolute error for the point A = 0
  - Absolute error for the point B = $|h_\theta(c) - d|$
  - Absolute error for the point C = $|h_\theta(e) - f|$



- The model that fits the data best
  - The model with **the minimum sum of errors** on the training data
  - e.g. The sum or mean of the squares of the errors
    - Least squares regression

Squared error

- Squared error
  - **Strongly penalizes very large errors**
  - Drawback
    - Is very **sensitive to the data**
    - Erroneous or outlying data points can severely skew the resultant linear function

- We should choose the objective function to optimize

Root Mean Squared Error

- Measures the differences between values predicted by model estimator and the values actually observed

$$RMSD = \sqrt{\frac{\sum_{i=0}^{n-1}(h_\theta(x^{(i)}) - y^{(i)})^2}{n}}$$

Mean Squared Error

- Measures the average of the squares of the errors

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Linear Regression

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \ldots$$

- To simplify the notation:

$$h(x) = \sum_{i=0}^{n} \theta_i x_i$$

Objective function (Cost function)

- For a given training set, how do we pick, or **learn the parameter θ?**
  - **What is the desirable condition of this regression?**

- **Make $h(x)$ close to y**
  - Make your **prediction** close to the **real observation**

- We define the **objective (cost) function**
  - Using Mean Squared Error (MSE) and multiplying ½ for convenience

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m-1} (h_\theta(x^{(i)}) - y^{(i)})^2$$

i is the index of the data points

Minimization problem

- We have an objective function $J(\theta_0, \theta_1)$
- Now, we want to find $\min_{\theta_o,\theta_1} J(\theta_o,\theta_1)$

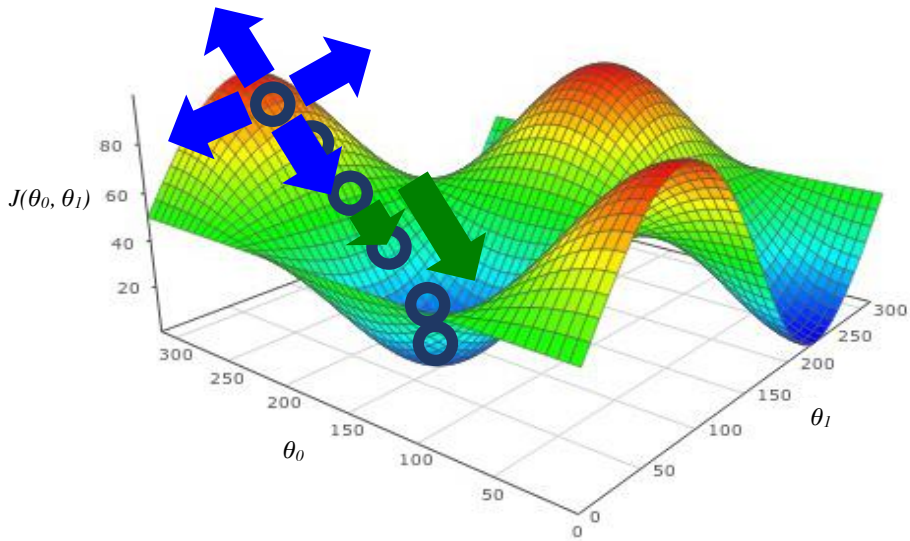
- **Goal: Find parameters to minimize the cost (output of the objective function)**

- Outline of our approach:
  - Start with some θ_0, θ_1
  - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum

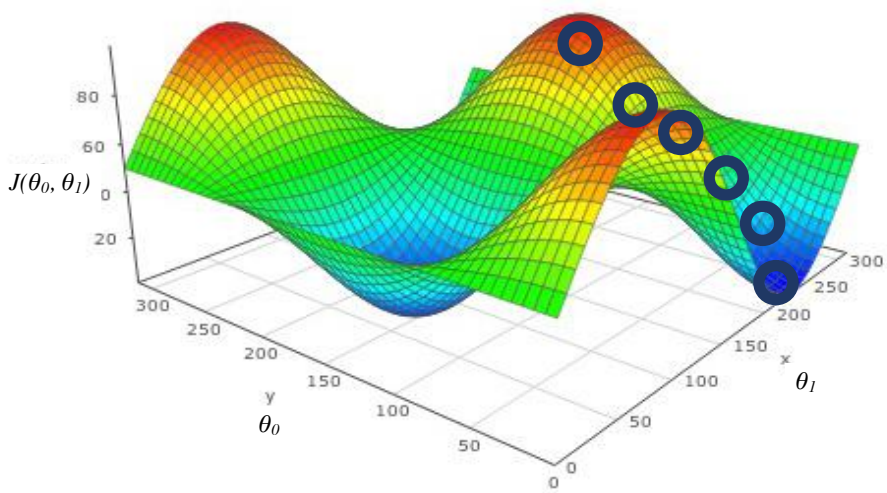
Predictive Analysis

Linear Regression: Gradient Descent Algorithm

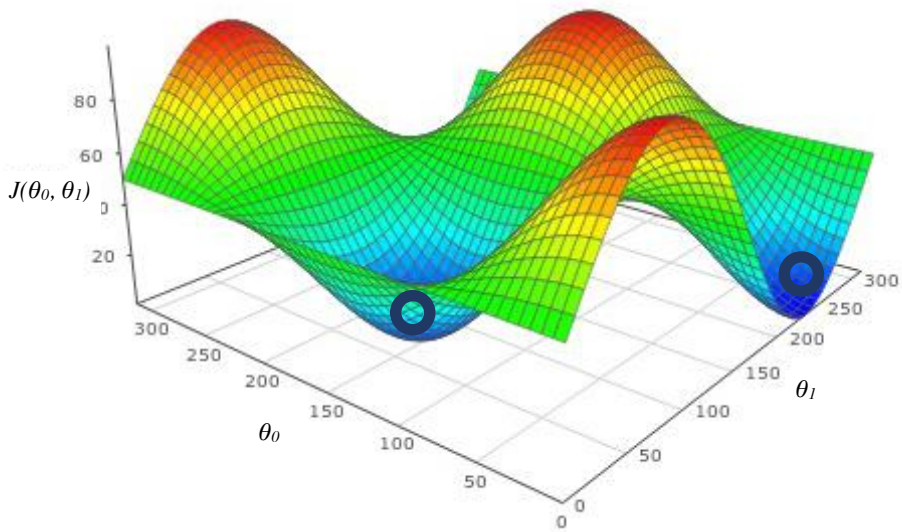
Concept of Gradient descent algorithm (1/2)



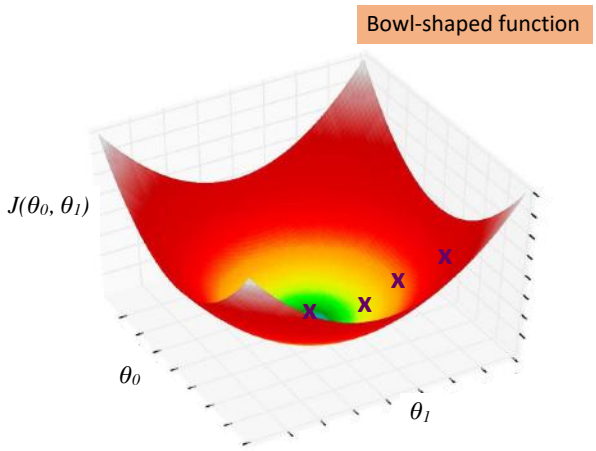
Concept of Gradient descent algorithm (2/2)



Multiple local optimal points



Convex function



Gradient descent algorithm

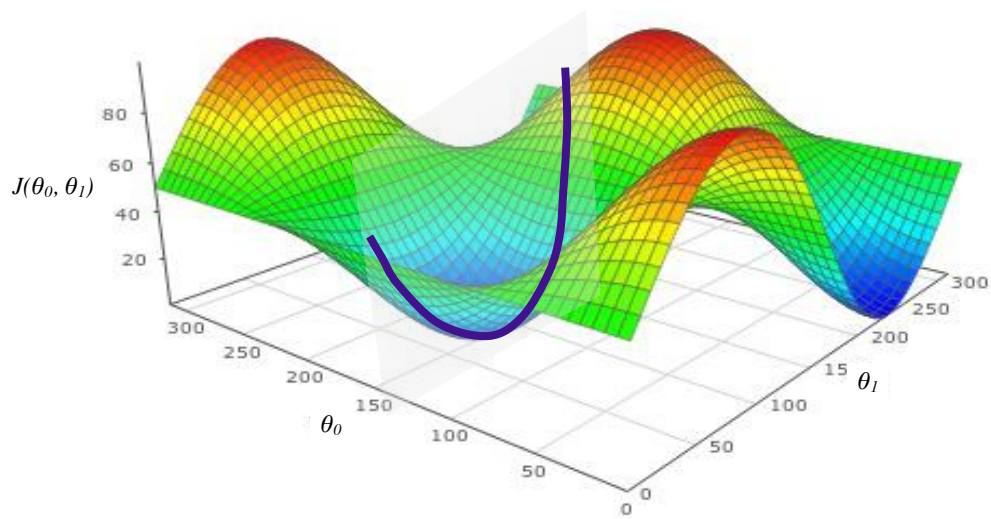
```
Repeat until convergence {
```

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

```
                    (for j=0 and j=1)
}
```

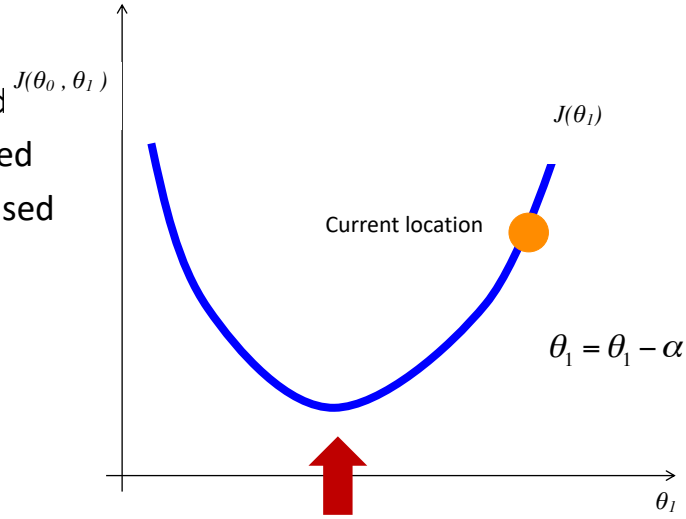
$$\begin{cases} \theta_0 := \theta_0 - \alpha \dfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_1 := \theta_1 - \alpha \dfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \end{cases}$$

1. Finding the direction to move for the next iteration: Decreasing and increasing θ_1



Question) **To reach the minimum point, what should be happened?**

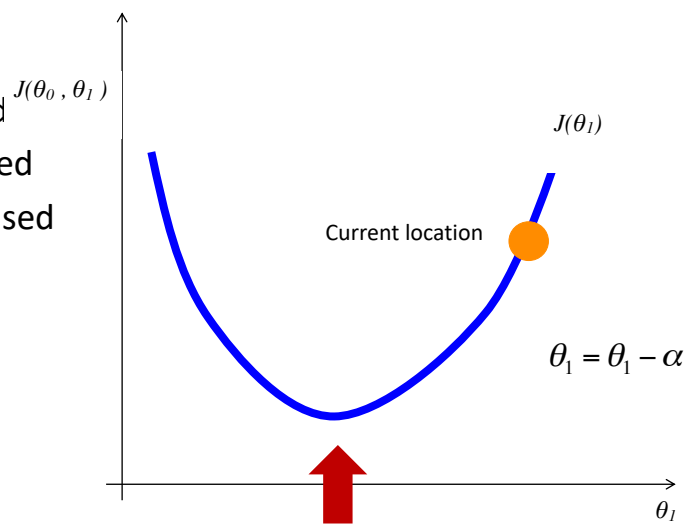
a) θ_1 should be reduced
b) θ_1 should be increased
c) *Alpha* should be reduced
d) *Alpha* should be increased



$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Question) **To reach the minimum point, what should be happened?**

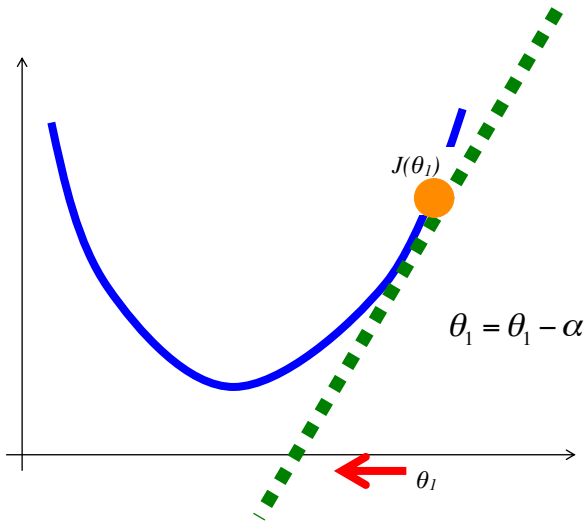
a) **θ_1 should be reduced**
b) θ_1 should be increased
c) *Alpha* should be reduced
d) *Alpha* should be increased



$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Decreasing θ_1

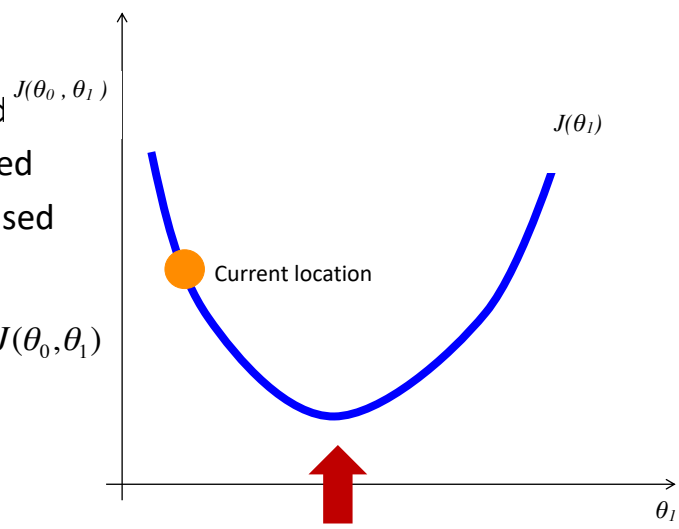
- **Positive** slope



$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Question) **To reach the minimum point, what should be happened?**

a) θ_1 should be reduced
b) θ_1 should be increased
c) *Alpha* should be reduced
d) *Alpha* should be increased



$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Question) **To reach the minimum point, what should be happened?**

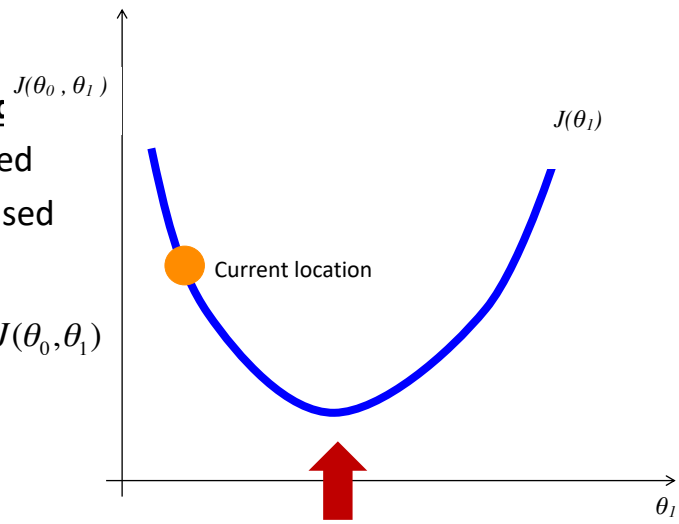
a) θ_1 should be reduced

b) **θ_1 should be increased**

c) *Alpha* should be reduced

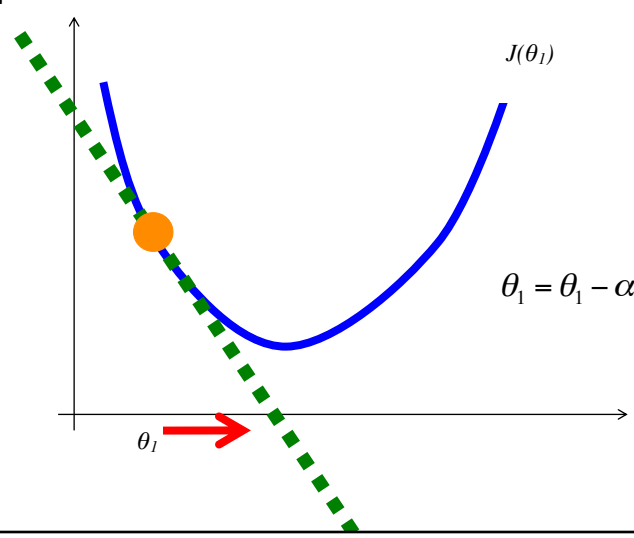
d) *Alpha* should be increased

$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$



Increasing θ_1

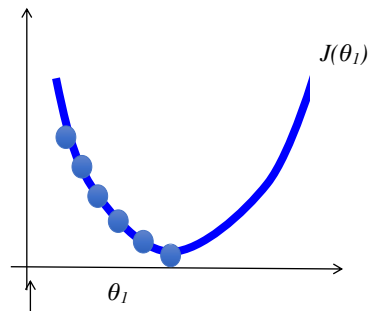
- **Negative** Slope



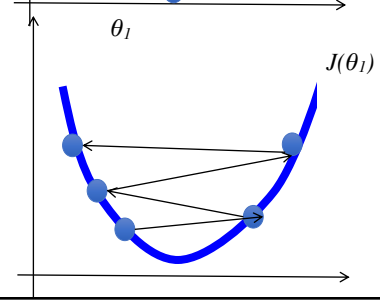
$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

2. Finding the size of steps: Learning rates

- If α(Learning Rate) is too small, gradient descent can be slow



- If α (Learning Rate) is too large, gradient descent can overshoot the minimum



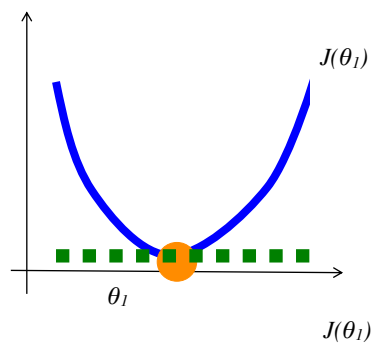
Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required

- **Question**

How will you detect the convergence?

a) Tracking θ_1
b) Tracking θ_0
c) Tracking the slope
d) Randomly testing



Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required

- **Question**

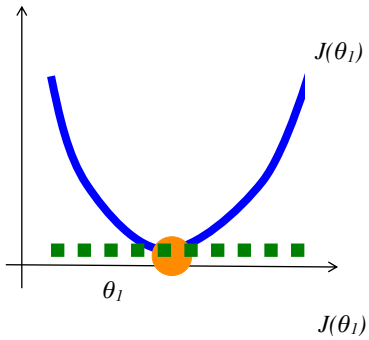
How will you detect the convergence?

a) Tracking θ_1

b) Tracking θ_0

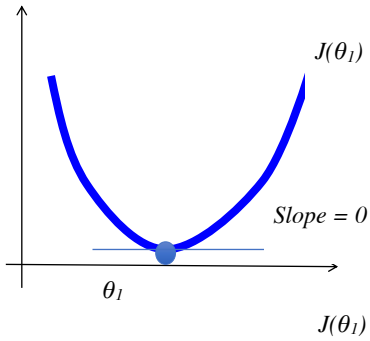
c) <u>Tracking the slope</u>

d) Randomly testing



Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required
  - **<u>Until the slope becomes 0</u>**



Fixed learning rate α

- Gradient descent can converge to a local minimum, even with a fixed learning rate
- As we approach a local minimum, gradient descent will automatically take smaller steps
  - **No need to decrease** α over time

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Simultaneous update

What is the difference between computing A and B?

A: Correct: Simultaneous update

temp0 := $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

temp1 := $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

θ_0 := temp0
θ_1 := temp1

temp1 is calculated before θ_0 is updated

B: Incorrect:

temp0 := $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

θ_0 := temp0

temp1 := $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

θ_1 := temp1

temp1 is calculated after θ_0 is updated

Predictive Analysis

Fitting a Linear Regression model using Gradient Descent Algorithm

Using Gradient Descent Algorithm for fitting a Linear Regression Model

Gradient descent algorithm

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for j=0 and j=1)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m-1} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Continued

$$\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$
$$= \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=0}^{m-1}\left(\theta_0 + \theta_0 x^{(i)} - y^{(i)}\right)^2$$

Case 1, θ_0 $(j = 0)$: $\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$

Case 2, θ_1 $(j = 1)$: $\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_1}\frac{1}{2m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$

Gradient descent for Linear Regression

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

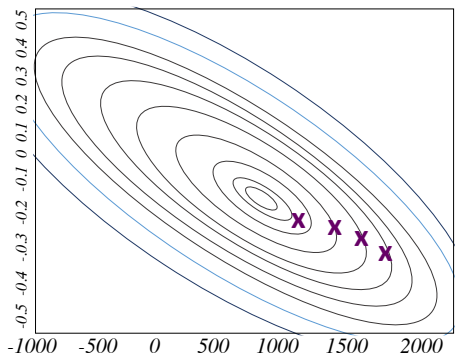
$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=0}^{m-1}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$$

(for j=0 and j=1)

}

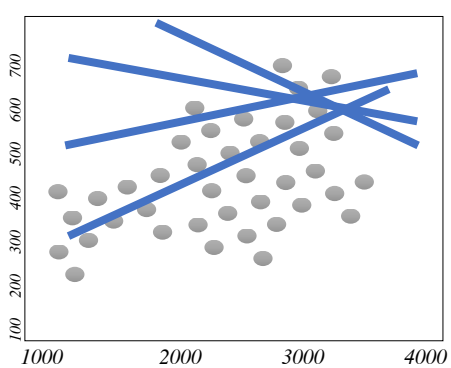
Update θ_0 and θ_1 simultaneously

$J(\theta_0,\theta_1)$

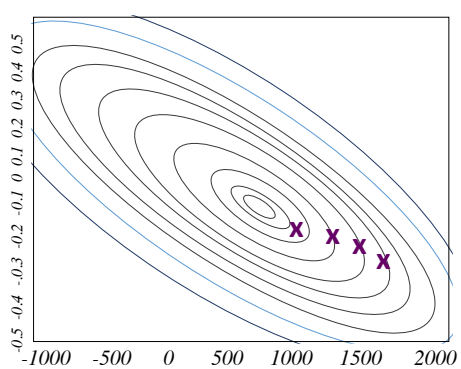


Fitting $h_\theta(x)$

$h_\theta(x)$



$J(\theta_0,\theta_1)$



Predictive Analysis

Linear Regression: Running with MapReduce

"Batch" Gradient Descent

- Batch
  - Each step of gradient descent uses all of the training example

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

Running with MapReduce

- For the sample size 1,000 ($m=1,000$)
- Batch gradient descent:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=0}^{m-1} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

- Using 4 machines

For θ_0

- Step 1. 4 input splits

$(x^{(0)},y^{(0)}),\ldots,((x^{(249)},y^{(249)})$
$(x^{(250)},y^{(250)}),\ldots,((x^{(499)},y^{(499)})$
$(x^{(500)},y^{(500)}),\ldots,((x^{(749)},y^{749})$
$(x^{(750)},y^{(750)}),\ldots,((x^{(999)},y^{(999)})$

$$Temp1 = \sum_{i=0}^{249} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$Temp2 = \sum_{i=250}^{499} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

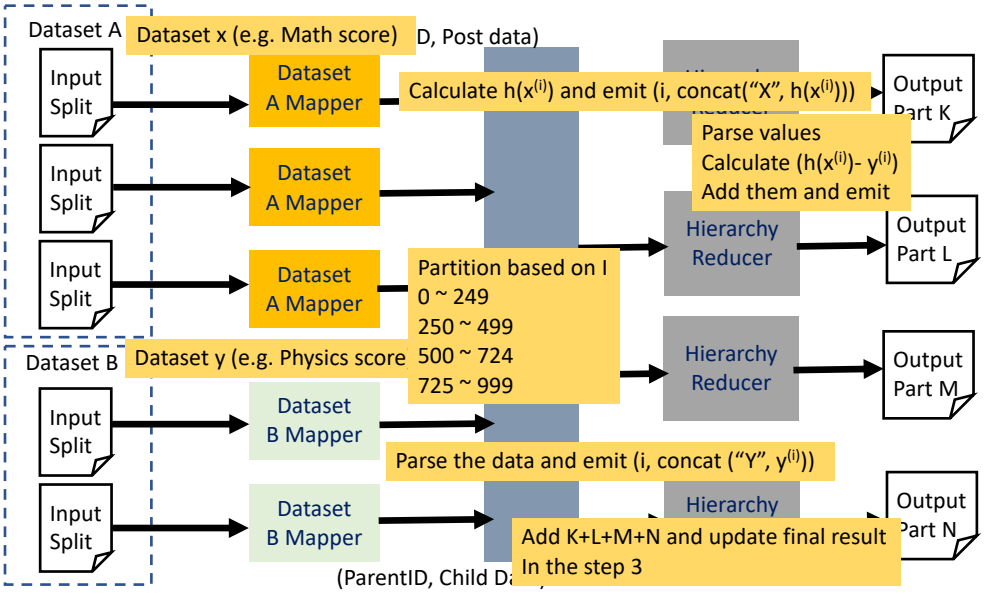
$$Temp3 = \sum_{i=500}^{749} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$Temp4 \sum_{i=750}^{999} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

- Step 2. Calculate temp1 ~ 4
- Step 3. Calculate final results

$$\theta_0 := \theta_0 - \alpha \frac{1}{1,000} (temp1 + temp2 + temp3 + temp4)$$

Structure of the structured to hierarchical pattern



For θ_1

- Step 1. 4 input splits

$(x^{(0)},y^{(0)}),\ldots,((x^{(249)},y^{(249)})$

$(x^{(250)},y^{(250)}),\ldots,((x^{(499)},y^{(499)})$

$(x^{(500)},y^{(500)}),\ldots,((x^{(749)},y^{749)})$

$(x^{(750)},y^{(750)}),\ldots,((x^{(999)},y^{(999)})$

$$Temp1 = \sum_{i=0}^{249} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

$$Temp2 = \sum_{i=250}^{499} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

$$Temp3 = \sum_{i=500}^{749} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

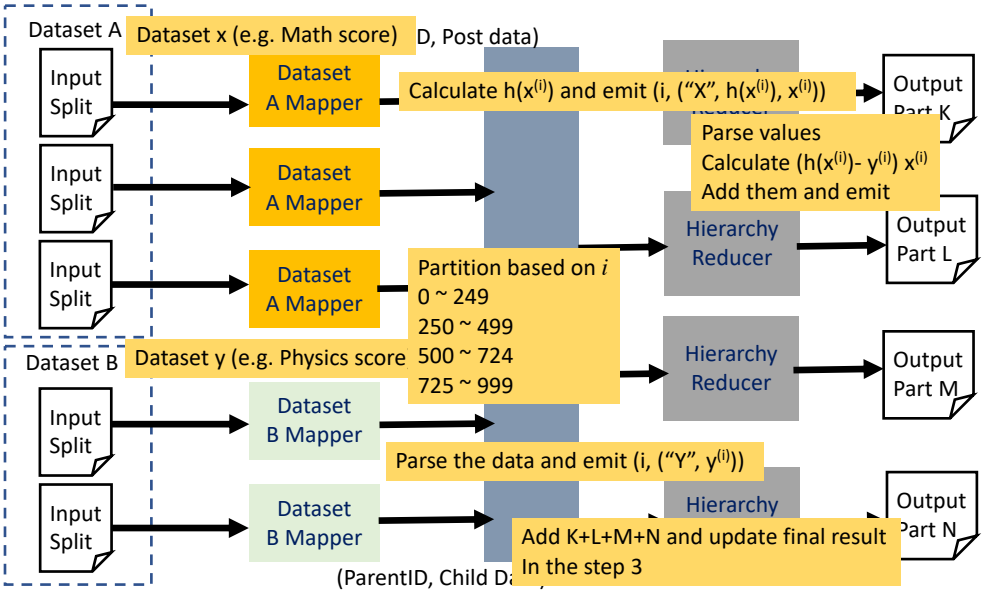
$$Temp4 \sum_{i=750}^{999} (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

- Step 2. Calculate temp1 ~ 4

- Step 3. Calculate final results

$$\theta_1 := \theta_1 - \alpha \frac{1}{1{,}000}(temp1 + temp2 + temp3 + temp4)$$

Structure of the structured to hierarchical pattern



Linear Regression in the Apache Spark MLlib

- Similar steps as in Decision Tree and Random Forest

1. Create a dataset
   1. Generating underlying RDD with a system compatible structure
2. Specify the hyperparameters
3. Train the model
4. Evaluate

```
// Load training data.
Dataset<Row> training = spark.read().format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt");

LinearRegression lr = new LinearRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8);

// Fit the model.
LinearRegressionModel lrModel = lr.fit(training);

// Print the coefficients and intercept for linear regression.
System.out.println("Coefficients: "
  + lrModel.coefficients() + " Intercept: " + lrModel.intercept());
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/ml/JavaLinearRegressionWithElasticNetExample.java" in the Spark repo.

More Models available in Spark MLlib

- Classification and Regression Models
  - https://spark.apache.org/docs/latest/ml-classification-regression.html
- ML Pipelines (e.g., Collaborative Filtering, Frequent Pattern Mining)
  - https://spark.apache.org/docs/latest/ml-pipeline.html
- If your team would like to construct Neural Networks, please consider using TensorFlow or PyTorch
  - Your project must use the distributed version of above framework

0

FAQs

- What is the "batch" gradient descent?
- What if my project does not have direct evaluation method?
- Please fill out the peer-evaluation form for the team participation of PA3
  - **Deadline: 11:59AM 11/16/2021 (see the canvas announcement)**
  - **This evaluation will take less than 2-3 minutes**
  - The same evaluation will be conducted after your team completes the term project. It will account for 3% of your final score
  - Contact me if you would like to improve your team effort/collaboration
    - I will help you to manage your team project

1

1

Topics discussed in the previous class

- Predictive analytics
  - Linear Regression with Gradient descent algorithm
  - Implementation with MapReduce and Apache Spark

Topics to be covered

- Google file system

Large Scale Data Storage Systems

Distributed File System

This material is built based on,

- *SANJAY GHEMAWAT, HOWARD GOBIOFF, SHUN-TAK LEUNG: The Google file system. Proceedings of SOSP 2003: 29-43*
- The link to this publication is listed on the course webpage (schedule)
- References: Tom White, "Hadoop: The definitive guide" , 2012

Distributed file systems

- Hadoop vs. Google File System?
- Where are they in our CS435 software stack?

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Introduction

Demand pulls in GFS [1/2]

- Files are huge by traditional standards
- File mutations predominantly through **appends**
  - Not overwrites
- Applications and File system API designed in **lock-step**
- Component failures are the **norm**
  - **Major reasons for the data center outage**
    - **UPS system failure**
    - **Accident and human error**
    - **Cooling system failure**

Backblaze Hard Drives Quarterly Failure Rates for Q1 2021
Reporting period: 1/1/2021 through 3/31/2021 inclusive

MFG	Model	Drive Size	Drive Count	Avg. Age (months)	Drive Days	Drive Failures	AFR
HGST	HMS5C4040ALE640	4TB	3,163	59.4	281,692	5	0.65%
HGST	HMS5C4040BLE640	4TB	12,738	53.4	1,146,496	10	0.32%
HGST	HUH728080ALE600	8TB	1,077	37.7	97,027	4	1.50%
HGST	HUH721212ALE600	12TB	2,599	18.0	233,948	5	0.78%
HGST	HUH721212ALE604	12TB	5,685	3.7	308,793	6	0.71%
HGST	HUH721212ALN604	12TB	10,825	24.0	974,310	9	0.34%
Seagate	ST4000DM000	4TB	18,882	65.3	1,701,967	59	1.27%
Seagate	ST6000DX000	6TB	886	71.8	79,740	-	0.00%
Seagate	ST8000DM002	8TB	9,744	54.1	878,106	26	1.08%
Seagate	ST8000NM0055	8TB	14,419	44.2	1,297,674	31	0.87%
Seagate	ST10000NM0086	10TB	1,200	41.6	108,057	6	2.03%
Seagate	ST12000NM0007	12TB	13,702	31.9	1,732,307	66	1.39%
Seagate	ST12000NM0008	12TB	20,085	12.3	1,764,318	41	0.85%
Seagate	ST12000NM001G	12TB	9,029	7.5	704,446	12	0.62%
Seagate	ST14000NM001G	14TB	5,977	5.9	538,401	13	0.88%
Seagate	ST14000NM0138	14TB	1,675	3.8	135,157	9	2.43%
Seagate	ST16000NM001G	16TB	2,459	1.7	54,177	1	0.67%
Toshiba	MD04ABA400V	4TB	99	70.3	8,910	-	0.00%
Toshiba	MG07ACA14TA	14TB	27,336	8.6	2,165,421	34	0.57%
Toshiba	MG07ACA14TEY	14TB	405	3.8	33,831	1	1.08%
Toshiba	MG08ACA16TEY	16TB	1,014	4.1	91,260	-	0.00%
WDC	WUH721414ALE6L4	14TB	8,400	3.8	640,767	10	0.57%
WDC	WUH721816ALE6L0	16TB	520	0.4	4,853	-	0.00%
		Totals	171,919		14,981,658	348	0.85%

BACKBLAZE

Source: https://www.backblaze.com/blog/backblaze-hard-drive-stats-q1-2021/

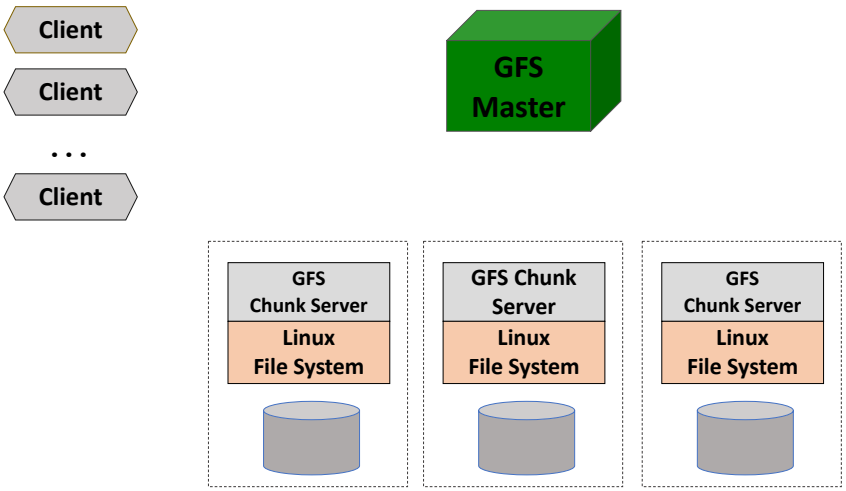
Demand pulls in GFS- [2/2]

- Hundreds of producers will **concurrently append** to a file
  - Many-way merging
- High **sustained bandwidth** is more important
  - than low latency

The file system interface

- Does not implement standard APIs such as POSIX
- Supports **`create, delete, open, close, read`** and **`write`**
- **`snapshot`**
  - Create a fast copy of file and directory tree
- **`record append`**
  - Multiple files can concurrently append records to the same file
    - Without additional locking

Architecture of GFS



Chunks

- Obvious reason
  - The file is too big
- **Set the stage** for computations that operate on this data
  - Parallel I/O
  - I/O seek times are 14 x 10^6 slower than CPU access times

Chunk size

- This is fixed at **128MB (it used to be 64MB)**
  - Much larger than typical FS block sizes
    - Disks are divided with blocks: smallest readable or writable unit
    - All major file systems use a block size of 4096 Bytes or more
    - 512 bytes used to be the standard
- Therefore, in GFS (or in HDFS), 128MB of data chuck is the minimum unit to read/write
  - You cannot randomly access a subset of a chunk
- **Lazy space allocation**
  - Stored as plain Linux file
  - Extended only as needed
  - Avoiding internal fragmentation

Chunk size: But why this big?-Advantage

- **Reduces client interaction** with the master
  - Can cache info for a multi-TB working set
- Reduce network **overhead**
  - With a large chunk, client performs more operations
  - Persistent connections
- Reduce **size of metadata** stored in the master
  - 64 bytes of metadata per 64 MB chunk

Large chunk size: Disadvantage

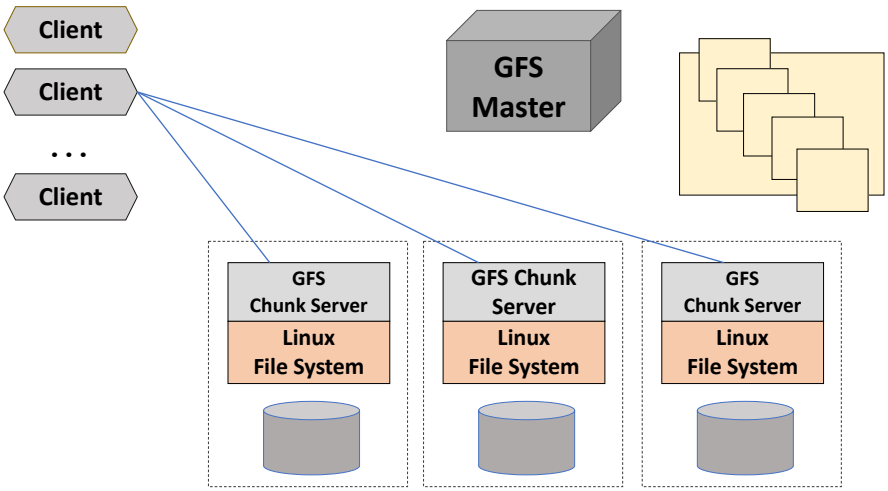
- Small files (with small number of chunks)
  - May **become hot spots**
  - E.g. popular executable files
  - Solution
    - **Assigning a higher replication factor**

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Master Operations

Architecture of GFS



Master operations

- Single master
- Manage system **metadata**
- **Leasing** of chunks
- **Garbage collection** of orphaned chunks
- Chunk **migrations**

Why have a single Master?

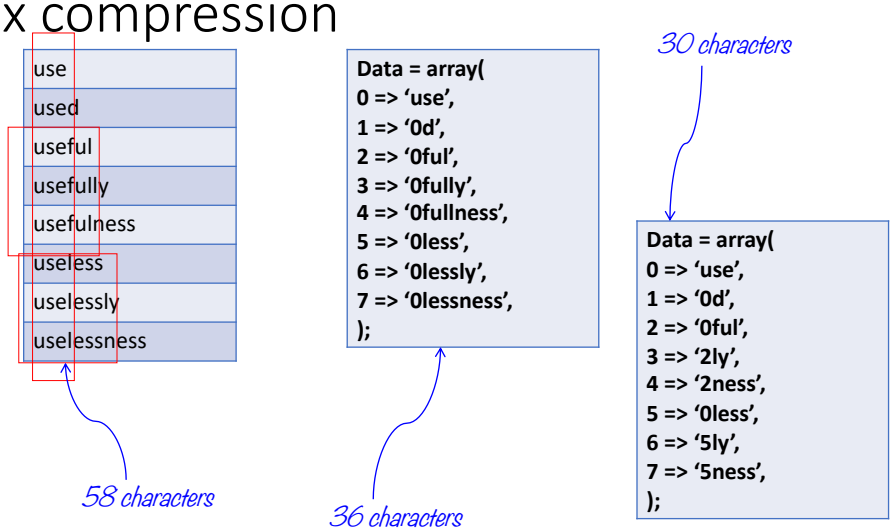
- Vastly **simplifies** design
- Easy to use global knowledge to **reason about**
  - Chunk placements
  - Replication decisions
- Disadvantage
  - **Single-point-of-error**

ALL system metadata is managed by the Master and stored in Main Memory

- File and chunk namespaces
- Mapping from files to chunks
- Location of chunks

Logs mutations into a permanent log

Prefix compression



Why keep the entire metadata in memory?

- **Speed**
- Master can **scan** its **state** in the background
  - Implement chunk garbage collection
  - Re-replicate if there are failures
  - Chunk migration to balance load and space
- **Add** extra memory to increase file system size

Size of the file system with 1 TB of RAM:
Assume file sizes are exact multiples of chunk sizes

- The **master** maintains less than 64 bytes of metadata for each 64MB chunk
- Number of entries = $2^{40}/2^{6}$
- MAXIMUM SIZE of the file system

  = **Number of entries x Chunk size**

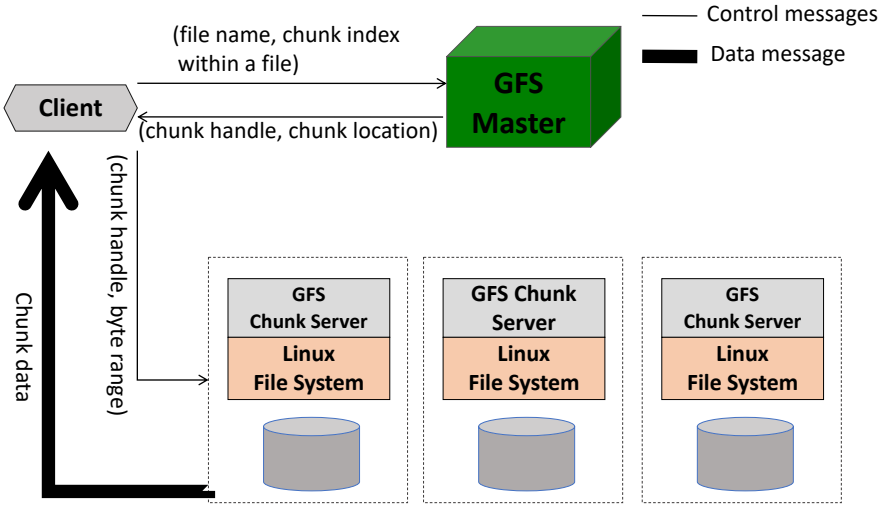
  $= \frac{2^{40}}{2^{6}} \times (2^{6} \times 2^{20})$

  $= 2^{60} = 1$ EB

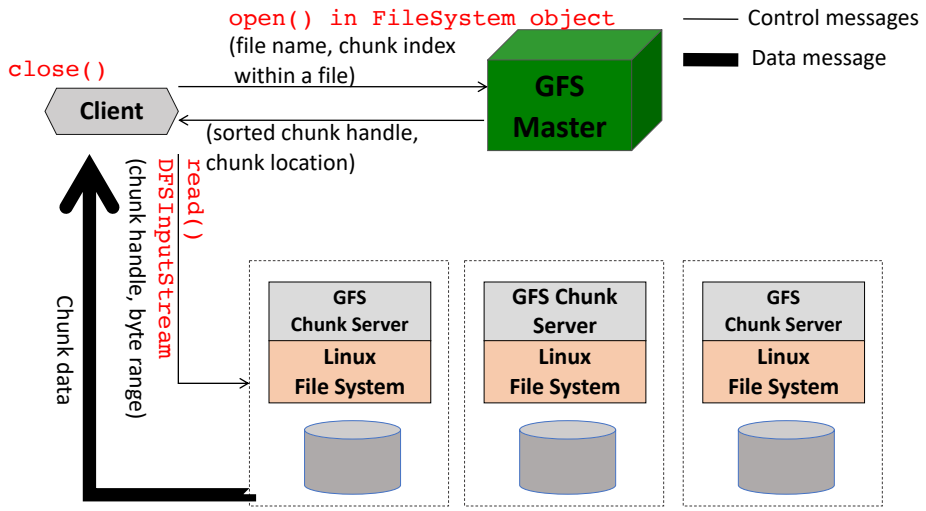
Tracking the chunk servers

- Master **does not** keep a persistent copy of the **location of chunk servers**
- List maintained via **heart-beats**
  - Allows list to be in *sync* with reality despite failures
  - Chunk server has final word on chunks it holds

Simple **read** example



Simple **read** example in HDFS

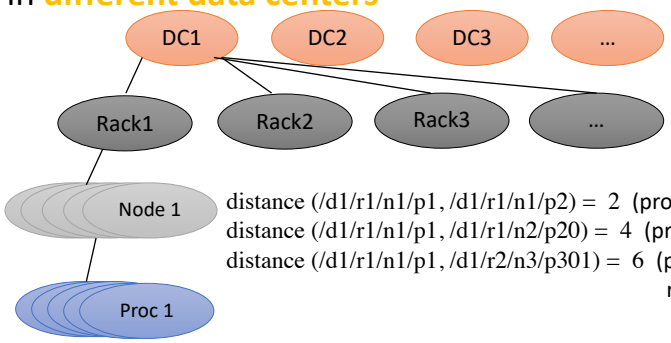


Network topology and HDFS [1/2]

- What does it mean for **two nodes in a local network to be "close"** to each other?
- **Bandwidth between nodes?**
- Hadoop's approach
  - The network is **represented as a tree**
  - The distance between two nodes is **the sum of their distances to their closest common ancestor**

Network topology and HDFS [2/2]

- Processes on the **same node**
- Different nodes on the **same rack**
- Nodes on different racks in the **same data center**
- Nodes in **different data centers**



distance (/d1/r1/n1/p1, /d1/r1/n1/p2) = 2 (processes on the same node)
distance (/d1/r1/n1/p1, /d1/r1/n2/p20) = 4 (processes on the same rack)
distance (/d1/r1/n1/p1, /d1/r2/n3/p301) = 6 (processes on the different rack but in the same data center)

Caching at the client/chunk servers

- Clients **do not cache** file data
  - At client the working set may be *too large*
  - Simplify client; eliminate *cache-coherence* problems
- Chunk servers **do not cache** file data either
  - Chunks are stored as local files
  - Linux's buffer cache *already keeps* frequently accessed data in memory

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Consistency in GFS

What is consistency? (1/4)

(1) **Strong consistency**
- Avoiding all inconsistences

(2) **Update consistency**
- **Write-write conflict**
  - Two people updating the same data
    - at the same time
- Without conflict handling
  - Server will serialize them
  - Decide to apply one then another

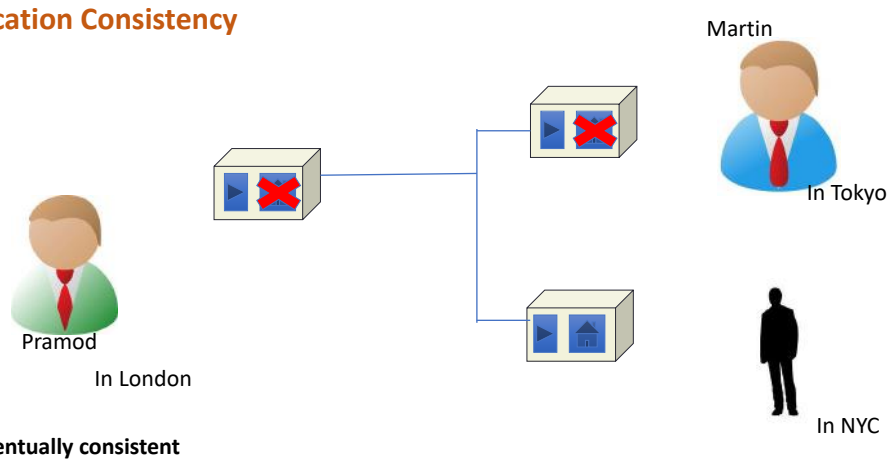
What is consistency? (2/4)

(3) Read consistency

Update consistency does NOT guarantee that readers of that data store will always get consistent responses

What is consistency? (3/4)

(4) Replication Consistency



Eventually consistent

At any time nodes may have replication inconsistencies

-- If there are no more updates, eventually all nodes will be updated to the same value

What is consistency? (4/4)

(5) Session Consistency

- Within a user's session there is read-your-writes consistency
- Sticky session
  - A session that's tied to one node (session affinity)
- Version stamps
  - Ensure every interaction with the data store includes the latest version stamp seen by a session
  - Server node must ensure that it has the updates that include that version stamp before responding to the request

Consistency in the ACID properties

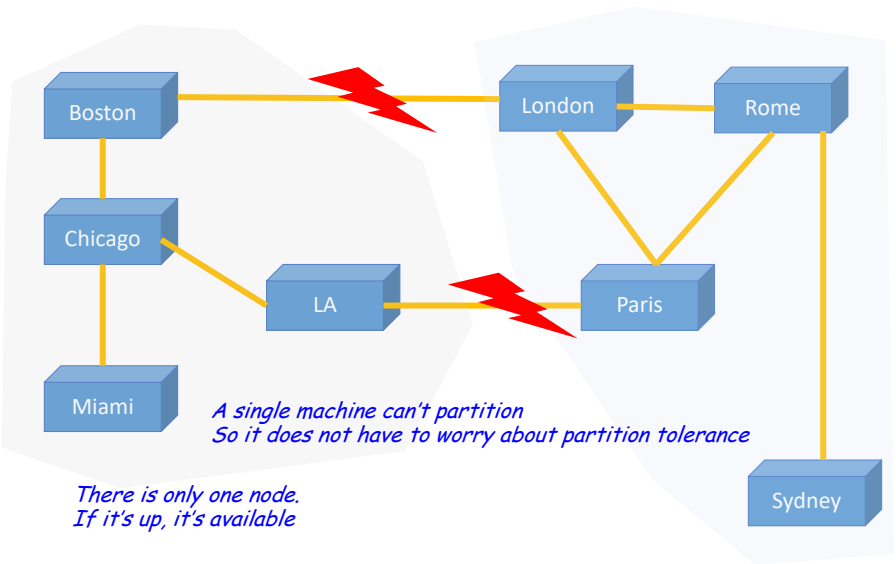
- **Relational databases** allow user to manipulate any combination of rows from any table in a single transaction
- ACID transaction is
  - **Atomic**
  - **Consistent**
  - **Isolated**
  - **Durable**

Relaxed consistency

- **The CAP theorem**
  - Eric Brewer, 2000
  - Seth Gilbert and Nancy Lynch, 2002
    - Formal proof
- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Partition: Two breaks in the communication lines



Relaxed consistency

- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Eventually consistent

- At any time nodes **may have replication inconsistencies**
- If there are no more updates (or updates can be ordered), **eventually all nodes will be updated to the same value**

Mutations in GFS

- **Mutation** changes the content or metadata of a chunk
  - Write
    - Writes data at an application-specific file offset
  - Append
    - Appends atomically *at least once* even in the presence of concurrent mutation
- Each mutation is performed at **all** chunk replicas
- Append is done based **on *record*** (1/4 of a chunk)
  - Offset is specified by GFS

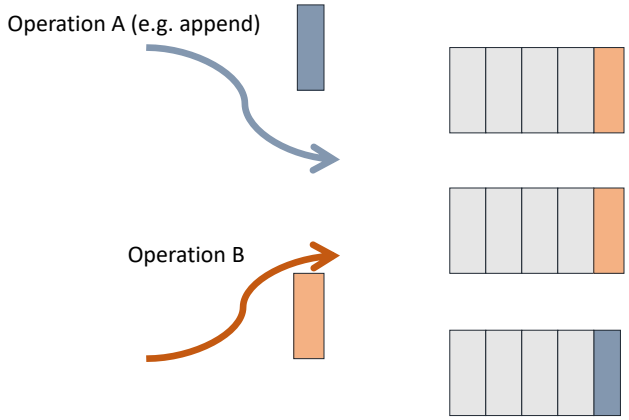
In GFS the state of file region after mutation depends on …

- **Type** of the mutation
- **Success/Failure** of the mutation
- Whether there were **concurren**t mutations

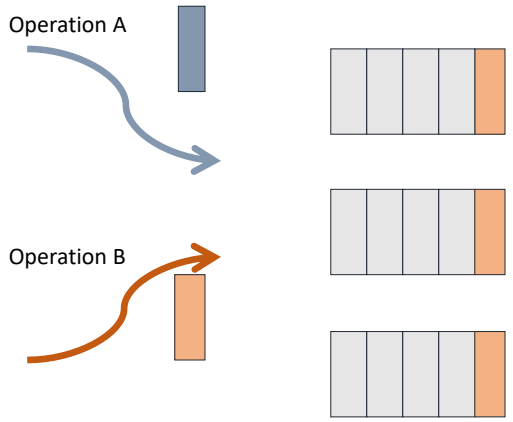
GFS has a relaxed consistency model

- **Consistent**: See the same data
  - On all replicas
- **Defined**: If it is consistent AND
  - Clients see mutation writes in its **entirety**

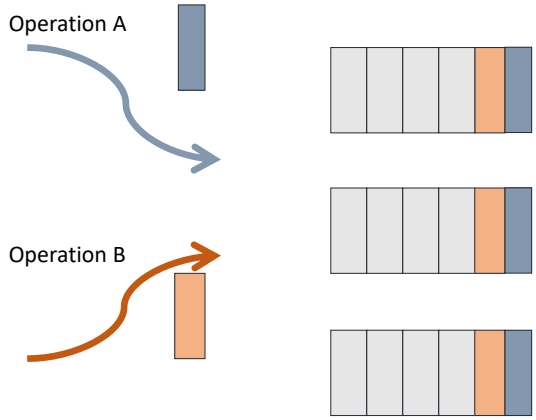
Inconsistent and undefined



Consistent but undefined



Defined



Data mutation

- Write
  - Data to be written at an application-specified **file offset**
- Record appends (in GFS)
  - Data ("record") to be appended **atomically at least once even in the presence of concurrent mutations**
  - **GFS chooses the offset**
  - No need for a distributed lock manger

File state region after a mutation

	Write	Record Append
Serial success	**defined**	**defined** interspersed with *inconsistent*
Concurrent success	Consistent but *undefined*	
Failure	Inconsistent	

For the sequence of successful mutations,

- GFS guarantees the mutated file to be **defined** and to contain the data written by **the last mutation**
  - Applying mutations to a chunk in the **same order on all its replicas**
  - Using chunk version numbers to detect any replica that has become stale
    - If the chunkserver was down

But, clients cache chunk locations…

- What if the chunk location points to a **stale replica** before that information is refreshed?
  - **Cache entry's timeout**
    - Next open to the file will purge all chunk information for that file from the cache
- Most of files are append-only
  - Stale replica usually returns a **premature end** of chunk rather than outdated data

Implications for applications

- Rely on **appends** instead of **overwrites**
- Checkpoint
- Write **records** that are
  - Self-validating
  - Self-identifying

What if many writers concurrently append?

- Append-at-least-once semantics **preserves each writer's output**
- Each record contains extra information like checksums so that its validity can be verified
  - Reader can identify and discard extra padding

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Handling writes and record-appends to a file

GFS uses leases to maintain consistent mutation order across replicas

- Master grants **lease** to one of the replicas
  - **PRIMARY**
- Primary picks **serial-order**
  - For all mutations to the chunk
  - Other replicas *follow* this order
    - When applying mutations

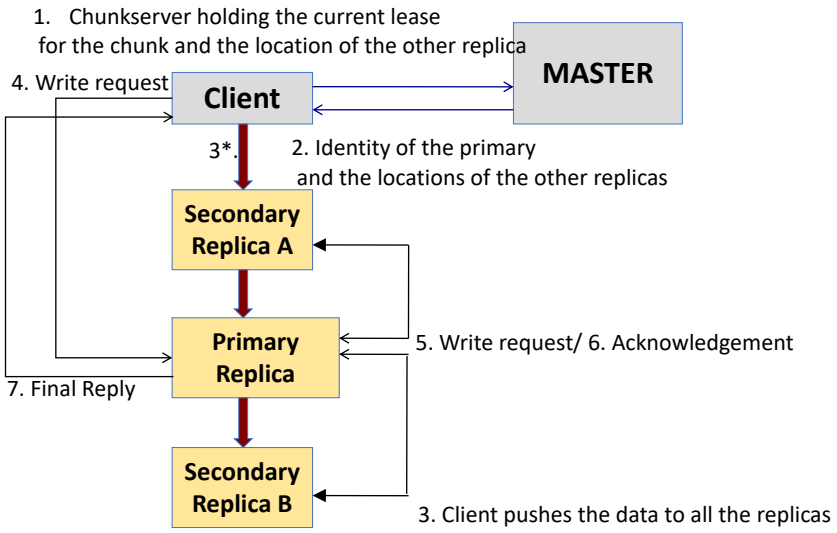
Lease mechanism designed to minimize communications with the master

- Lease has **initial *timeout*** of 60 seconds
- As long as chunk is being mutated
  - Primary can request and receive **extensions**
- Extension requests/grants piggybacked over **heart-beat messages**

Revocation and transfer of leases

- Master may **revoke** a lease before it expires
- If communications lost with primary
  - Master can safely give lease to another replica
    - **ONLY AFTER** the lease period for old primary *elapses*

How a **`write`** is actually performed



Client pushes data to all the replicas [1/2]

- While the data is written, each chunk server stores data in an **LRU buffer** until
  - Data is used
  - Aged out

Client pushes data to all the replicas [2/2]

- When chunk servers acknowledge receipt of data
  - Client sends a ***write*** *request to primary*
- Primary assigns *consecutive serial numbers* to mutations
  - Forwards to replicas

Data flow is decoupled from the control flow to utilize network efficiently [1/2]

- Utilize each machine's network bandwidth
- Avoid network bottlenecks
- Avoid high-latency links
- **Leverage network topology**
  - Estimate distances from IP addresses (in Google's data center)

Data flow is decoupled from the control flow to utilize network efficiently [2/2]

- Pipeline the data transfer
  - Once a chunkserver receives some data, it starts forwarding immediately.
  - For transferring B bytes to R replicas
    - ideal elapsed time will be B/T+RL
    - Where, T is the network throughput
    - L is latency to transfer bytes between two machines

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Record Append and Inconsistency

The control flow for record appends is similar to that of writes

- Google's record appends
  - **Atomic record appends**
- Client pushes data to replicas of the ***last*** chunk of file
- Primary replica checks if the record **fits** in this chunk (64MB)

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 13-A Fall 2021 Computer Science

Primary replica checks if the record append will breach the size (64MB) threshold

- If chunk size would be breached
  - Pad the chunk to maximum size
  - Tell client, that operation should be retried on *next chunk*
- If the record fits, the primary
  - Appends data to its replica
  - Notifies secondaries to **write at the exact offset**

Record sizes and fragmentation

- Size is restricted to ¼ the chunk size (16MB→ 128MB)
- Minimizes worst-case fragmentation
  - Internal fragmentation in each chunk …

What if the secondary replicas could not finish the write operation?

- Client request is considered **failed**
- Modified region is **inconsistent**
  - *No attempt* to delete this from the chunk
  - Client must handle this inconsistency
- Client **retries** the failed mutation

Inconsistent Regions

What if record append fails at one of the replicas

- Client **must retry** the operation
- Replicas of same chunk may contain
  - Different data
  - **Duplicates** of the same record
    - In whole or in part
- Replicas of chunks are **not bit-wise identical**!
  - In most systems, replicas are identical

GFS only guarantees that the data will be written at least once as an atomic unit

- For an operation to return *success*
  - Data must be **written at the same offset** on **all** the replicas
- After the write, all replicas are *as long as* the end of the record
  - Any future record will be assigned a higher offset or a different chunk

GFS client code implements the file system API

- Communications with master and chunk servers done *transparently*
  - On behalf of apps that read or write data
- Interact with master for **metadata**
- **Data-bearing** communications directly to chunk servers

Handling failed "write" in Hadoop HDFS [1/2]

- Different from GFS

1. Pipeline is closed
   - Any packets in the ack queue are added to the front of the data queue
   - Datanodes that are downstream from the failed node will not miss any packets
2. The current block on the good datanodes is given a new identity
   - Reports to the namenode
   - To detect and delete partial block on the failed datanode later on

Handling failed "write" in Hadoop HDFS [2/2]

3. Remainder of the block's data is written to the two good datanodes in the pipeline
4. Namenode notices the block is under-replicated
   - It arranges for a further replica to be created on another node
   - Write quorum
     - `dfs.replication.min` (default to 1)

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Creating Snapshot

Snapshots

- Copying file or directory tree **almost instantaneously**
- Minimizing any **interruptions of ongoing mutations**
- Providing **checkpoint mechanism**
  - Users can commit later
  - Rollback

Snapshots allow you to make a copy of a file very fast

1. Master **revokes** outstanding leases for any chunks of the file (**source**) to be snapshot
2. **Log** the operation to disk
3. Update in-memory state
   1. Duplicate metadata of the source file
4. Newly created file points to the "**same chunks**" as the source

When a client wants to write to a chunk C after the snapshot operation

- Master sees "the *reference count* to **C" > 1**
- Pick **new chunk-handle C'**
- Ask chunk-server with current replica of **C**
  - Create new chunk **C'**
  - Data is *copied locally, not over the network*
- From this point chunk handling of **C'** is no different

GFS does not have a per-directory structure that lists files in the directory

- Name spaces represented as a **lookup** table
  - Maps **full pathnames** to metadata
- Each node has an associated read/write lock
- File creation does not require a lock **on the directory structure**
  - No inode needs to be protected from modification

Each master operation acquires a set of locks before it runs

- **Read lock** prevents a directory/file from being deleted, renamed, or snapshotted
  - Others can still read it
  - Others CANNOT mutate (write/append/delete) this file
- **Write lock** on directory/file names serialize attempts to create a file with the same twice
  - Others CANNOT mutate (write/append/delete) this file
  - Others CANNOT read this file
- If operation involves `/d1/d2/…/dn/leaf`
  - Acquire **read locks** on directory names
    - `/d1, /d1/d2, …, /d1/d2/…/dn`
  - **Read or write lock** on full pathname
    - `/d1/d2/…/dn/leaf`

Locks are used to prevent operations during snapshots

How do we prevent creating /home/user/foo
While /home/user is being snapshotted to /save/user ?

- `/home/user` is being snapshotted to `/save/user`
  - Read locks on `/home` and `/save`
  - Write lock on **`/home/user`** and **`/save/user`**
- To create file, /home/user/foo
  - Read lock on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`
- The two operations will be serialized
  - because they try to obtain **`/home/user`**
- File creation does not require write lock on parent directory … there is no "directory"
  - Read locks on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`

Part 2. Large scale data storage system

Distributed File System

Google File System (GFS): Deletion of files and garbage collection

Garbage collection in GFS

- After a file is deleted, GFS does not reclaim space immediately
- Done **lazily** during garbage collection at
  - File and chunk levels

Master logs a file's deletion immediately

- File is **renamed to a hidden name**
  - Includes deletion timestamp
- Master scans the file system namespace
  - Delete if hidden file existed for <u>more than 3 days</u>
- When file removed from namespace
  - *In memory metadata* is also removed
  - **Severs links** to all its chunks!

Garbage collection: When Master scans its chunk namespace

- Identifies **orphaned chunks**
  - Not reachable from any file
- Erase metadata for these chunks

The role of heart-beats in garbage collection

- Chunk server reports **subset of chunks** it currently has
- Master replies with identity <u>of chunks no longer present</u>
  - Chunk server free to delete its replica of such chunks

Stale chunks and issues

- If a chunk server fails
  - AND misses mutations to the chunk
  - The chunk replica becomes **stale**
- Working with a **stale replica** causes problems with:
  - Correctness
  - Consistency

Aiding the detection of stale chunks

- Master maintains a **chunk version number** for each chunk
  - Distinguish between stale and up-to-date chunks
- When master grants a new lease on chunk
  - Increase version number
  - Inform replicas
  - Record new version persistently

Occurs BEFORE any client can write to chunk

If a replica is unavailable its version number will not be advanced

- When a chunk server restarts, it reports to the Master with the following:
  - Set of Chunks
  - Corresponding version numbers
- Used to **detect** stale replicas
- **Remove stale replicas** in regular garbage collection

Additional safeguards against stale replicas

- Include chunk version number
  - When **client requests** chunk information
    - Client/Chunk server verify version to make sure things are up-to-date
  - During cloning operations
    - Clone the most up-to-date chunk
- Clients and chunk servers expected to **verify** versioning information

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Data Integrity

Data Integrity

- *Impractical* to detect chunk corruptions *across replicas*
  - Not bytewise identical in any case!
- Detection of corruption should be **self-contained**

Data Integrity

- Break chunks into **64 KB** data blocks
- Compute 32-bit checksum for block
  - Keep in chunk server memory
  - Store persistently, **separate** from the data
- Verify checksums of data blocks that **overlap** read range

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Inefficiencies

The master server is a single point of failure

- Master server **restart** takes several seconds
- **Shadow servers** exist
  - Can handle reads of files
    - In place of the master
  - But not writes
- Requires a **massive main memory**

The system is optimized for large files

- But **not for** a very large number of very **small files**
- Primary operation on files
  - Long, sequential reads/writes
  - Large number of **random overwrites** will **clog** things up quite a bit

Consistency Issues: GFS expects clients to resolve inconsistencies

- File chunks may have **gaps or duplicates** of some records
  - The client has to be able to deal with this
- Imagine doing this for a scientific application
  - Portions of a massive array are corrupted
    - Clients would have to detect this
- HDFS does **NOT** have this problem

Security model

- **None**
  - Operation is expected to be in a trusted environment

Questions?

96

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 13-A Fall 2021 Computer Science

Questions?

97

0

FAQs

- Can we use Hadoop/Spark and then process the reduced dataset with python or R or Scikit?
  - YES!

1

1

Topics discussed in the previous class

- Google file system
  - Overview
  - Consistency
  - Master operations

Topics to be covered

- Google file system
  - continued

Large Scale Data Storage Systems

Distributed File System

This material is built based on,

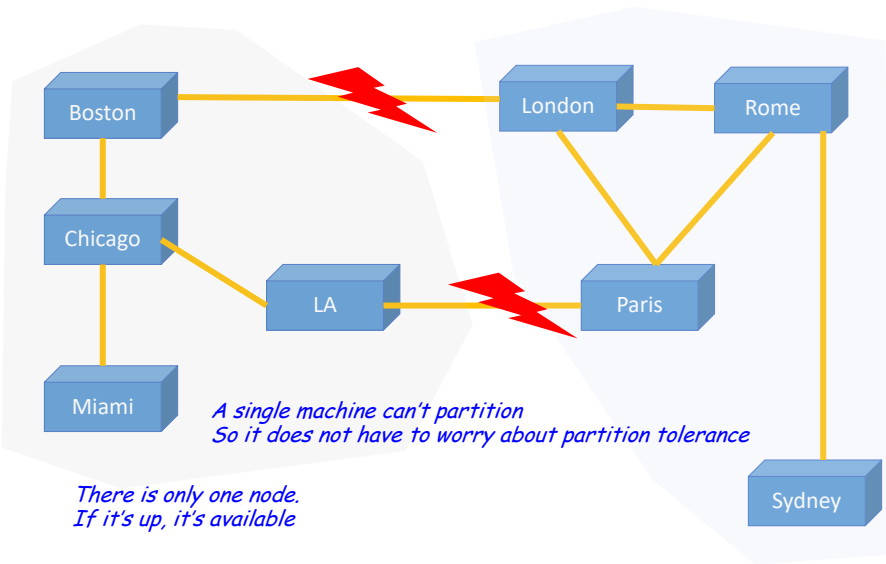
- *SANJAY GHEMAWAT, HOWARD GOBIOFF, SHUN-TAK LEUNG: The Google file system. Proceedings of SOSP 2003: 29-43*
- The link to this publication is listed on the course webpage (schedule)
- References: Tom White, "Hadoop: The definitive guide" , 2012

Relaxed consistency

- **The CAP theorem**
  - Eric Brewer, 2000
  - Seth Gilbert and Nancy Lynch, 2002
    - Formal proof
- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Partition: Two breaks in the communication lines



Relaxed consistency

- Given the three properties of **Consistency, Availability**, and **Partition tolerance**, you can only get two

Consistency
Every read receives the most recent write or an error
Availability
Every request receives by a non-failing node in the system must result in a response – without guarantee that it contains the most recent write
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Eventually consistent

- At any time nodes **may have replication inconsistencies**
- If there are no more updates (or updates can be ordered), **eventually all nodes will be updated to the same value**

Mutations in GFS

- **Mutation** changes the content or metadata of a chunk
  - Write
    - Writes data at an application-specific file offset
  - Append
    - Appends atomically *at least once* even in the presence of concurrent mutation
    - HDFS: data will be appeared exactly once
- Each mutation is performed at **all** chunk replicas
- Append is done based **on *record*** (1/4 of a chunk)
  - Offset is specified by GFS

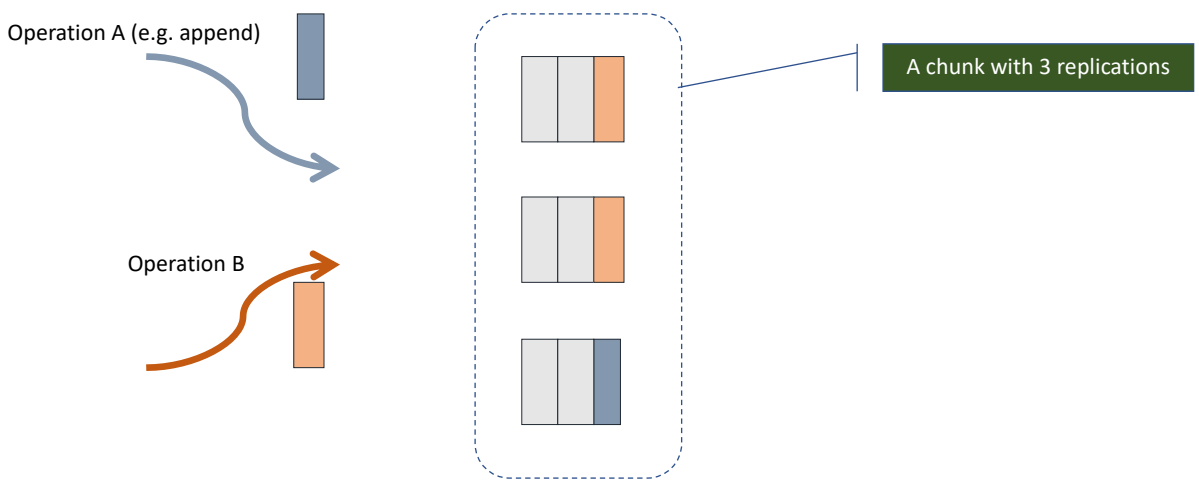
In GFS the state of file region after mutation depends on …

- **Type** of the mutation
- **Success/Failure** of the mutation
- Whether there were **concurren**t mutations

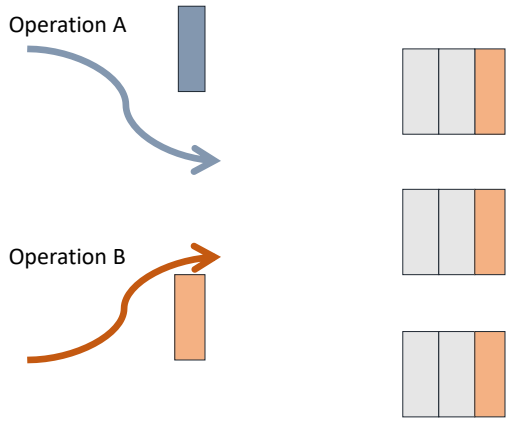
GFS has a relaxed consistency model

- **Consistent**: See the same data
  - On all replicas
- **Defined**: If it is consistent AND
  - Clients see mutation writes in its **entirety**

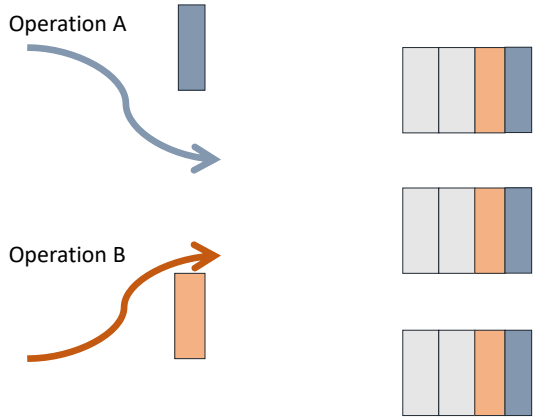
Inconsistent and undefined



Consistent but undefined



Defined



Data mutation

- Write
  - Data to be written at an application-specified **file offset**
- Record appends (in GFS)
  - Data ("record") to be appended **atomically at least once even in the presence of concurrent mutations**
  - **GFS chooses the offset**
  - No need for a distributed lock manger

File state region after a mutation in GFS

	Write	Record Append	HDFS
Serial success	**defined**	**defined** interspersed with *inconsistent*	Consistent and defined
Concurrent success	Consistent but *undefined*		
Failure	Inconsistent		inconsistent

For the sequence of successful mutations,

- GFS guarantees the mutated file to be **defined** and to contain the data written by **the last mutation**
  - Applying mutations to a chunk in the **same order on all its replicas**
  - Using chunk version numbers to detect any replica that has become stale
    - If the chunkserver was down

But, clients cache chunk locations…

- What if the chunk location points to a **stale replica** before that information is refreshed?
  - **Cache entry's timeout**
    - Next open to the file will purge all chunk information for that file from the cache
- Most of files are append-only
  - Stale replica usually returns a **premature end** of chunk rather than outdated data

Implications for applications

- Rely on **appends** instead of **overwrites**
- Checkpoint
- Write **records** that are
  - Self-validating
  - Self-identifying

What if many writers concurrently append?

- Append-at-least-once semantics **preserves each writer's output**
- Each record contains extra information like checksums so that its validity can be verified
  - Reader can identify and discard extra padding

Large-Scale Data Storage Systems

Distributed File System

Google File System (GFS): Handling writes and record-appends to a file

GFS uses leases to maintain consistent mutation order across replicas

- Master grants **lease** to one of the replicas
  - **PRIMARY**
- Primary picks **serial-order**
  - For all mutations to the chunk
  - Other replicas *follow* this order
    - When applying mutations

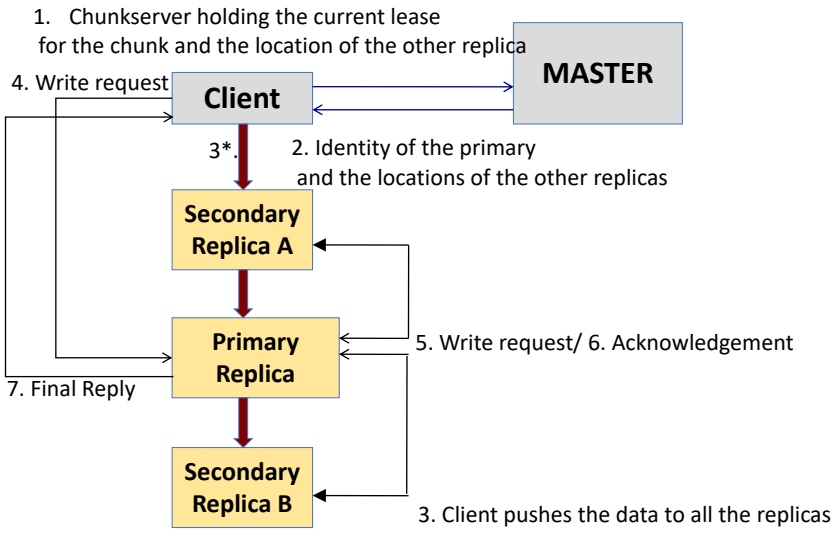
Lease mechanism designed to minimize communications with the master

- Lease has **initial *timeout*** of 60 seconds
- As long as chunk is being mutated
  - Primary can request and receive **extensions**
- Extension requests/grants piggybacked over **heart-beat messages**

Revocation and transfer of leases

- Master may **revoke** a lease before it expires
- If communications lost with primary
  - Master can safely give lease to another replica
    - **ONLY AFTER** the lease period for old primary *elapses*

How a **`write`** is actually performed



Client pushes data to all the replicas [1/2]

- While the data is written, each chunk server stores data in an **LRU buffer** until
  - Data is used
  - Aged out

Client pushes data to all the replicas [2/2]

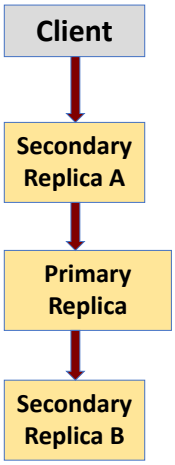
- When chunk servers acknowledge receipt of data
  - Client sends a ***write*** *request to primary*
- Primary assigns *consecutive serial numbers* to mutations
  - Forwards to replicas

Data flow is decoupled from the control flow to utilize network efficiently [1/2]

- Utilize each machine's network bandwidth
- Avoid network bottlenecks
- Avoid high-latency links
- **Leverage network topology**
  - Estimate distances from IP addresses (in Google's data center)

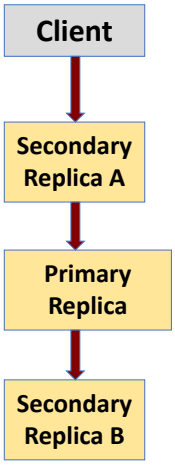
Data flow is decoupled from the control flow to utilize network efficiently [2/2]

- Pipeline the data transfer
  - Once a chunkserver receives some data, it starts forwarding immediately.
  - For transferring B bytes to R replicas
    - ideal elapsed time will be B/T+2RL
    - Where, T is the network throughput
    - L is latency to transfer bytes between two machines



Example

- Assume that the network links are 1 Gbit/sec and 3 replicas
- The latency to transfer bytes between two machines is far below 1ms
- To write 1MB of data
- 8x1MB/1G +2x1(ms) = 10 ms



Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Record Append and Inconsistency

The control flow for record appends is similar to that of writes

- Google's record appends
  - **Atomic record appends**
- Client pushes data to replicas of the ***last*** chunk of file
- Primary replica checks if the record **fits** in this chunk (64MB)

Primary replica checks if the record append will breach the size threshold

- If chunk size would be breached
  - Pad the chunk to maximum size
  - Tell client, that operation should be retried on *next chunk*
- If the record fits, the primary
  - Appends data to its replica
  - Notifies secondaries to **write at the exact offset**

Record sizes and fragmentation

- Size is restricted to ¼ the chunk size (16MB→ 128MB)
- Minimizes worst-case fragmentation
  - Internal fragmentation in each chunk …

What if the secondary replicas could not finish the write operation?

- Client request is considered **failed**
- Modified region is **inconsistent**
  - *No attempt* to delete this from the chunk
  - Client must handle this inconsistency
- Client **retries** the failed mutation

Inconsistent Regions

What if record append fails at one of the replicas

- Client **must retry** the operation
- Replicas of same chunk may contain
  - Different data
  - **Duplicates** of the same record
    - In whole or in part
- Replicas of chunks are **not bit-wise identical**!
  - In most systems, replicas are identical

GFS only guarantees that the data will be written at least once as an atomic unit

- For an operation to return *success*
  - Data must be **written at the same offset** on **all** the replicas
- After the write, all replicas are *as long as* the end of the record
  - Any future record will be assigned a higher offset or a different chunk

GFS client code implements the file system API

- Communications with master and chunk servers done *transparently*
  - On behalf of apps that read or write data
- Interact with master for **metadata**
- **Data-bearing** communications directly to chunk servers

Handling failed "write" in Hadoop HDFS [1/2]

- Different from GFS

1. Pipeline is closed
   - Any packets in the ack queue are added to the front of the data queue
   - Datanodes that are downstream from the failed node will not miss any packets
2. The current block on the good datanodes is given a new identity
   - Reports to the namenode
   - To detect and delete partial block on the failed datanode later on

Handling failed "write" in Hadoop HDFS [2/2]

3. Remainder of the block's data is written to the two good datanodes in the pipeline

4. Namenode notices the block is under-replicated
   - It arranges for a further replica to be created on another node
   - Write quorum
     - `dfs.replication.min` (default to 1)

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Creating Snapshot

Snapshots

- Copying file or directory tree **almost instantaneously**
- Minimizing any **interruptions of ongoing mutations**
- Providing **checkpoint mechanism**
  - Users can commit later
  - Rollback

Snapshots allow you to make a copy of a file very fast

1. Master **revokes** outstanding leases for any chunks of the file (**source**) to be snapshot
2. **Log** the operation to disk
3. Update in-memory state
   1. Duplicate metadata of the source file
4. Newly created file points to the "**same chunks**" as the source

When a client wants to write to a chunk C after the snapshot operation

- Master sees "the *reference count* to **C" > 1**
- Pick **new chunk-handle C'**
- Ask chunk-server with current replica of **C**
  - Create new chunk **C'**
  - Data is *copied locally, not over the network*
- From this point chunk handling of **C'** is no different

GFS does not have a per-directory structure that lists files in the directory

- Name spaces represented as a **lookup** table
  - Maps **full pathnames** to metadata
- Each node has an associated read/write lock
- File creation does not require a lock **on the directory structure**
  - No inode needs to be protected from modification

Each master operation acquires a set of locks before it runs

- **Read lock** prevents a directory/file from being deleted, renamed, or snapshotted
  - Others can still read it
  - Others CANNOT mutate (write/append/delete) this file
- **Write lock** on directory/file names serialize attempts to create a file with the same twice
  - Others CANNOT mutate (write/append/delete) this file
  - Others CANNOT read this file
- If operation involves `/d1/d2/…/dn/leaf`
  - Acquire **read locks** on directory names
    - `/d1, /d1/d2, …, /d1/d2/…/dn`
  - **Read or write lock** on full pathname
    - `/d1/d2/…/dn/leaf`

Locks are used to prevent operations during snapshots

How do we prevent creating /home/user/foo
While /home/user is being snapshotted to /save/user ?

- `/home/user` is being snapshotted to `/save/user`
  - Read locks on `/home` and `/save`
  - Write lock on **`/home/user`** and **`/save/user`**
- To create file, /home/user/foo
  - Read lock on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`
- The two operations will be serialized
  - because they try to obtain **`/home/user`**
- File creation does not require write lock on parent directory … there is no "directory"
  - Read locks on `/home` and **`/home/user`**
  - Write lock on `/home/user/foo`

Part 2. Large scale data storage system

Distributed File System
Google File System (GFS): Deletion of files and garbage collection

Garbage collection in GFS

- After a file is deleted, GFS does not reclaim space immediately
- Done **lazily** during garbage collection at
  - File and chunk levels

Master logs a file's deletion immediately

- File is **renamed to a hidden name**
  - Includes deletion timestamp
- Master scans the file system namespace
  - Delete if hidden file existed for more than 3 days
- When file removed from namespace
  - *In memory metadata* is also removed
  - **Severs links** to all its chunks!

Garbage collection: When Master scans its chunk namespace

- Identifies **orphaned chunks**
  - Not reachable from any file
- Erase metadata for these chunks

The role of heart-beats in garbage collection

- Chunk server reports **subset of chunks** it currently has
- Master replies with identity of chunks no longer present
  - Chunk server free to delete its replica of such chunks

Stale chunks and issues

- If a chunk server fails
  - AND misses mutations to the chunk
  - The chunk replica becomes **stale**
- Working with a **stale replica** causes problems with:
  - Correctness
  - Consistency

Aiding the detection of stale chunks

- Master maintains a **chunk version number** for each chunk
  - Distinguish between stale and up-to-date chunks
- When master grants a new lease on chunk
  - Increase version number
  - Inform replicas
  - Record new version persistently

Occurs BEFORE any client can write to chunk

If a replica is unavailable its version number will not be advanced

- When a chunk server restarts, it reports to the Master with the following:
  - Set of Chunks
  - Corresponding version numbers
- Used to **detect** stale replicas
- **Remove stale replicas** in regular garbage collection

Additional safeguards against stale replicas

- Include chunk version number
  - When **client requests** chunk information
    - Client/Chunk server verify version to make sure things are up-to-date
  - During cloning operations
    - Clone the most up-to-date chunk
- Clients and chunk servers expected to **verify** versioning information

Questions?

CS435 Introduction to Big Data
Week 1-A
Introduction

- Sangmi Lee Pallickara, Ph.D.
- Computer Science, Colorado State University
- http://www.cs.colostate.edu/~cs435

0

COLORADO STATE UNIVERSITY

CS435 Introduction to Big Data

What is Big Data?

1

1

"Big Data"

- The term "big data" refers to data that is so large, fast or complex that it's difficult or impossible to process using traditional methods
- Big Data is about analytics of huge quantities of data in order to infer probabilities
  - Big Data is NOT about trying to "teach" a computer to "think" like humans
- Providing a ***quantitative dimension*** it never had before

2

2

The Three Vs in Big Data

- ***Volume***
  - Voluminous
  - It does not have to be certain number of petabytes or quantity.
- ***Velocity***
  - How fast the data is coming in
  - How fast you need to be able to analyze and utilize it
- ***Variety***
  - Number of sources or incoming vectors

3

3

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

Who uses the Big Data?

- Uber
  - Surge pricing
    - Rate of demands
    - If you are running late for an important appointment, and you book a uber in a crowded place, you must be ready to pay twice to amount
    - On New Year's Eve, the price for driving for one mile can go from $200 to $1,000
    - Machine learning algorithms are used to determine where the demand is strong

4

4

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

Who uses the Big Data?

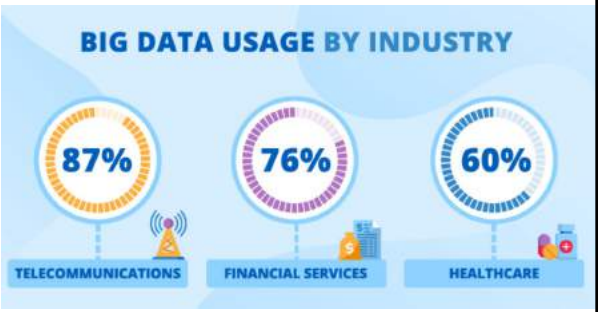
- Netflix
  - Predicts what exactly their customers will enjoy watching with Big Data
  - Recommendation engine
  - Analyzing customer's preferences for the content creating
- Government
  - Public services
  - FDA uses Big Data to detect and study patterns of food-related illnesses and diseases

5

5

Who uses the Big Data?

- 70% of very large organizations (5000+ employees) are using big data analytics
- 43-45% of small, mid-sized and large organizations (fewer than 5000 employees) are using big data
- Top 3 big data use cases for mid-sized, large and very large organizations are data warehouse optimization, predictive maintenance and customer analytics

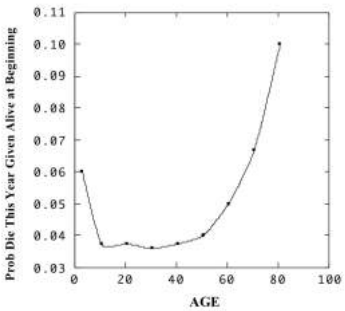


https://www.scnsoft.com/blog/big-data-use-cases-stats-and-examples

Use all of the data, or just a little? [1/5]

John Graunt

London, 17th century



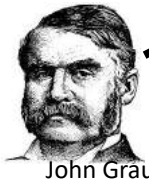
- Founder of the science of demography: *Bills of Mortality*
  - Estimate the population of London, which is then used to estimate the number of eligible 'fighting men' available to the king
  - Dispel the myth that the plague was rifer in years of monarchy changes
  - Investigate how London was expanding

http://www.theactuary.com/archive/old-articles/part-3/who-was-captain-john-graunt-3F/ 7

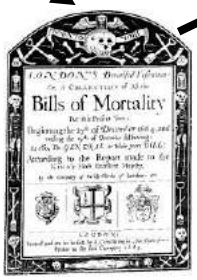
7

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

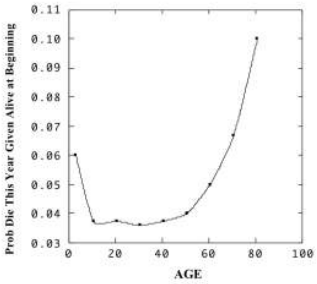
Use all of the data, or just a little? [2/5]



John Graunt



Estimating population of London
384,000



- Using a 1658 map he assumed that there were 54 families living in each space of 100 yards square
- Within the walls of London there were 220 such squares, making 11,880 families there
- The number of deaths in London as a whole, both within and outside the walls, was four times the number of deaths within the walls alone

http://www.theactuary.com/archive/old-articles/part-3/who-was-captain-john-graunt-3F/

8

8

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

Use all of the data, or just a little? [3/5]

- John Graunt's population example
- "Usually [Graunt] explained his steps in solving problems, but he seldom included the actual calculations; and sometimes he omitted important information. Furthermore, his indirect approach sometimes went beyond the reliable use of his data, and the accuracy of some of his answers was difficult to evaluate." -- the *Dictionary of Scientific Biography,* Frank N. Egerton
- How do we generate "The Life Tables" now?

9

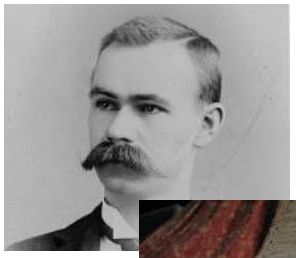
9

Use all of the data, or just a little? [4/5]

- **Conducting censuses**
  - Costly and time-consuming
- The US Constitution mandated one every decade
  - As the growing country measured itself in millions
- By the late nineteenth century, even that was proving problematic
  - Data outstripped the Census Bureau's ability to keep up.
- The 1880 census took **8 years** to complete
  - The information was obsolete even before it became available!
- The 1890 census was estimated to require full **13 years**

Use all of the data, or just a little? [5/5]

- The Census Bureau contracted with Herman Hollerith
  - An American inventor used his idea of punch cards and tabulation
  - machines for the 1890 census
  - Reduced duration to less than one year



- **Methods of acquiring and analyzing big data is very expensive**

Now "Big Data" is available

- ***Volume***
  - Every day 2.5 Quintillion bytes of data are created
  - 90% of world's data today was created in the last two years
- ***Velocity***
  - Every minute there are
    - 216,000 Instagrams
    - 204,000,000 emails sent
    - 277,000 tweets
- ***Variety***
  - 90% of the data is unstructured
  - Tweets, photos, product information, etc.

Modern Big Data Case Study: Google Flu

- The Centers for Disease Control and Prevention (CDC)
  - Requested doctors to inform the CDC about new flu cases
  - Tabulated the numbers once a week
- Typically 1-2 week reporting lag

Detecting influenza epidemics using search engine query data

- Google
  - More than three billion search queries every day
  - Saves all the queries
- Looking at what people were searching for on the Internet
- Predict the spread of the **winter flu** in the United States
  - Nationally
  - Specific regions and states

J. Ginsberg, et al., "Detecting influenza epidemics using search engine query data" Nature 457 pp. 1012 ~ 1014, February 2009
http://www.nature.com/nature/journal/v457/n7232/full/nature07634.html

How did they build a prediction model? (1/3)

- On the spread of seasonal flu between 2003 and 2008
  - Identify areas infected by the flu virus by what people searched for on the Internet
- Aggregating historical logs of online web search queries
  - Between 2003 and 2008
  - Computed a time series of weekly counts for 50 million of the most common search queries in the US
  - Weekly counts were kept for every query in each state
- Simple model that estimates the probability that a random physician visit in a particular region is related to an influenza-like illness(ILI)

How did they build a prediction model? (2/3)

- The probability that a random search query submitted from the same region is ILI-related

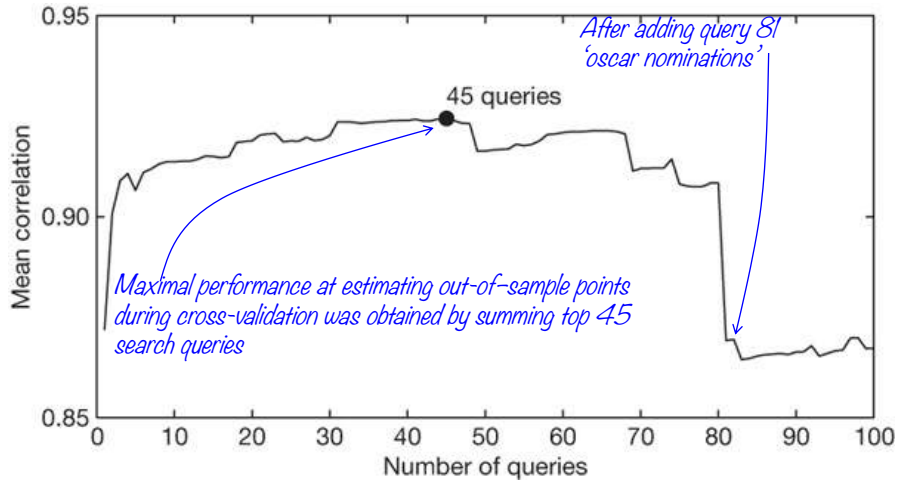
- $logit(I(t)) = \alpha logit(Q(t)) + \varepsilon$ where:
  - $I(t)$ is the percentage of ILI physician visits at time t
  - $Q(t)$ is the ILI-related query fraction at time t
  - α is the multiplicative coefficient
  - ε is the error term
  - $logit(p) = ln(p/(1-p))$

How did they build a prediction model? (3/3)

- Each of the 50 million candidate queries was separately tested
  - To identify the search queries that could most accurately model the CDC ILI visit percentage in each region

- Generated the **highest scoring search queries**

Evaluation of combining top-scoring queries



nature

What makes this work innovative?

- There have been other approaches:
  - CDC's weekly report
  - Call volume to telephone triage advice lines
  - Tracking over-the-counter drug sales
  - Online activity for influenza surveillance
    - Search queries submitted to a Swedish medical website
    - Tracking search keys ('flu', or 'influenza') in Yahoo search queries

- ***What is the innovation in Google's approach?***

Was it successful?

- Google Flu missed H1N1
  - Google Flu was wrong 100 out of 108 weeks since August 2011
- Google Flu was a failure

"Image of the newly identified H1N1 influenza virus",
taken in the CDC Influenza Laboratory.
Source: http://www.cdc.gov/h1n1flu/images.htm?s_cid=cs_001

Why Google Flu is a failure?

- Assumption
  - "People with the flu (the influenza virus, that is) will probably go online to find out how to treat it, or to search for other information about the flu"
  - Google tracks such behavior to predict the trend of influenza
- Big Data hubris
  - Assumption that big data are a substitute for, rather than a supplement to, traditional data collection and analysis
  - Most people don't know what "the flu" is
  - Most people who think they have "the flu" do not
  - *Only 8.8%* of specimens tested positive for influenza

Lazer, D., Kennedy, R., King, G. and Vespignani, A., 2014. The parable of Google Flu: traps in big data analysis. *Science*, *343*(6176), pp.1203-1205.

What are the main challenges for Big Data analytics?

- Ensuring data quality
- Determining type of information product
- Selecting current computing environment and tools
- Scaling up and down based on the current demand: data and model
- Collecting and integrating massive and diverse datasets
- Maintaining data integrity, security, and privacy

Course Introduction

Goals of CS435

- Understanding fundamental concepts in Big Data
- Learn about effective Big Data technologies and how to apply them
  - Computing paradigms
  - Data retrieval methodologies
  - Data analytics patterns
  - Scalable algorithms

Related research areas

- Data Warehousing
  - How can we efficiently resolve queries on massive amounts of input data?
  - The input dataset may be presented in the form of a distributed data stream
- Machine learning
  - How can we efficiently solve large-scale machine learning problems?
  - The input data may be massive; stored in a distributed cluster of machines
- Distributed computing
  - How can we efficiently solve large-scale optimization problems in distributed computing environments?
  - For example, how can we efficiently solve large-scale combinatorial problems, e.g. processing of large scale graphs?

Related courses in CS at CSU

- Machine Learning (CS445, CS545)
- Distributed Systems (CS455, CS555)
- Database Systems (CS430, CS530)
- Computer Communications (CS457, CS557)
- Computer Security (CS356, CS456, CS556)

About me

- Research area
  - Big Data
    - Predictive analysis, Storage, retrievals, metadata management and visual analytics
- Cochran Family Professor
- Associate Editor of the IEEE Transactions of Parallel and Distributed Systems
- NSF CAREER recipient
- IEEE TCSC Award for *Excellence in Scalable Computing*(Mid-Career)
- Research has been funded by
  - The National Science Foundation, The National Institute of Food and Agriculture, Department of Homeland Security, the Advanced Research Projects Agency-Energy (ARPA-E), the Environmental Defense Fund, Google, Amazon, and HP

National Science Foundation WHERE DISCOVERIES BEGIN
USDA United States Department of Agriculture National Institute of Food and Agriculture

Research Projects

- Spectral
  - Predictive analysis at scale
- FineET
  - Estimating evapotranspiration for effective irrigation management
- Sustain
  - Urban sustainability research cyberinfrastructure
- Galileo
  - Distributed data storage system for large scale geospatial time-series datasets
- Helios
  - Visual Analytics
  - Graduate and undergraduate research

Undergraduate Research Opportunity in the Big Data Lab
(1) Research credits and program
(2) Participating in the undergraduate research team
(3) Undergraduate Research Assistantship

Communications (1/3)

- Canvas
  - Major course gateway
  - Modules
  - Announcements
  - Grade
  - Quizzes
  - Assignments
- MS Teams
  - Help sessions
  - Discussion board
  - Online office hours
- Course Website
  - http://www.cs.colostate.edu/~cs435
  - Policies

Communications (2/3)

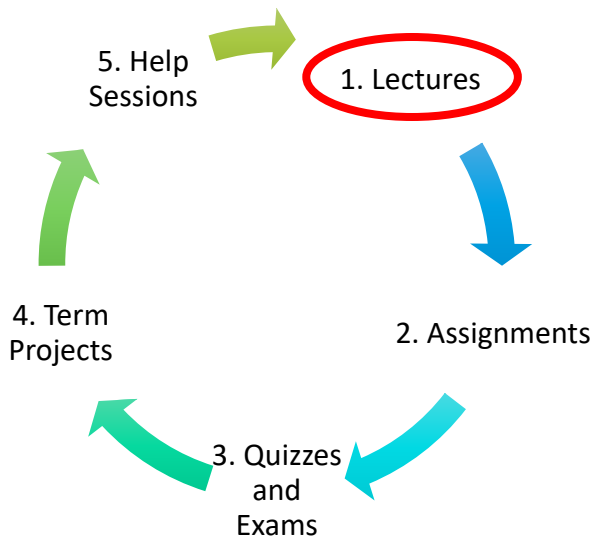
- Professor
  - sangmi@colostate.edu
  - Office hours
    - Friday 9:00AM ~ 10:00AM and by appointment
  - Office: CS374 (and via MSTeams)
  - URL: http://www.cs.colostate.edu/~sangmi

Communications (2/3)

- GTA Helpdesk: CSB120 and MSTeams
  - Abdul Matin
    - Mon: 4 -5PM, Wed: 4 - 5PM, Thur: 11:00 AM -1:00 PM
  - Ramya Patchava
    - Mon : 9:00 AM - 10:00 AM, Tue : 10:00 AM - 11:00 AM, Thu: 9:00 AM - 11:00 AM
- Microsoft TEAMS
  - Recitation and office hours
  - If you are not able to access the team "CompSci CS435 Fall2021" please send a request to cs435@cs.colostate.edu ASAP
- Recitation: Fridays 4:00pm-4:50pm
  - CS130
  - Attendance is optional
  - Recordings will be provided

COLORADO STATE UNIVERSITY — CS435 Introduction to Big Data

Course Structure

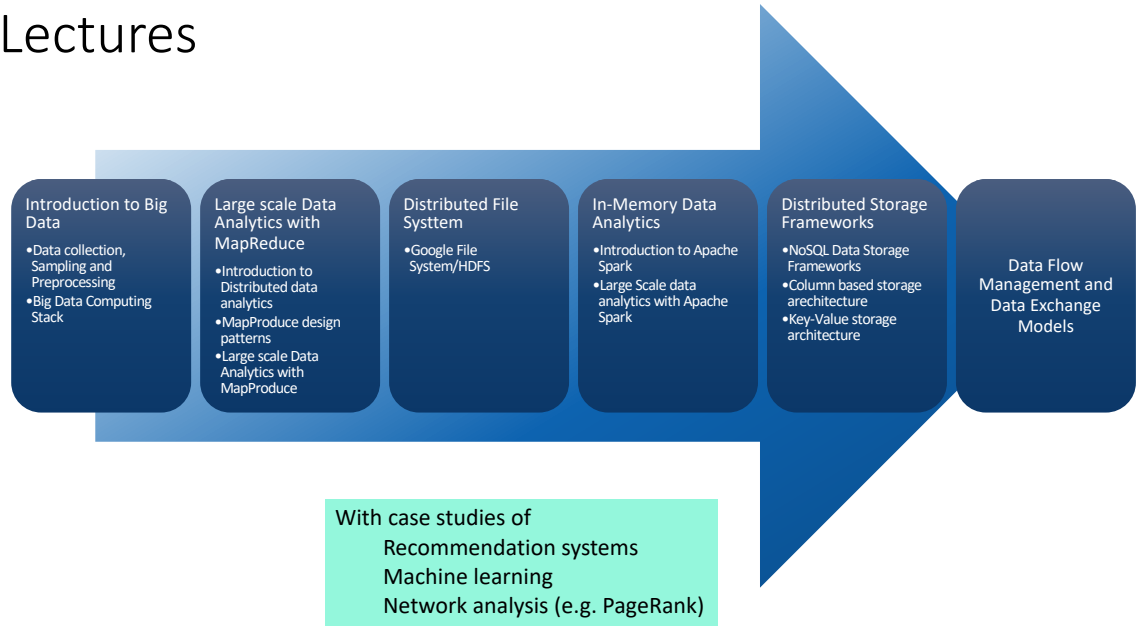


32

32

COLORADO STATE UNIVERSITY — CS435 Introduction to Big Data

Lectures

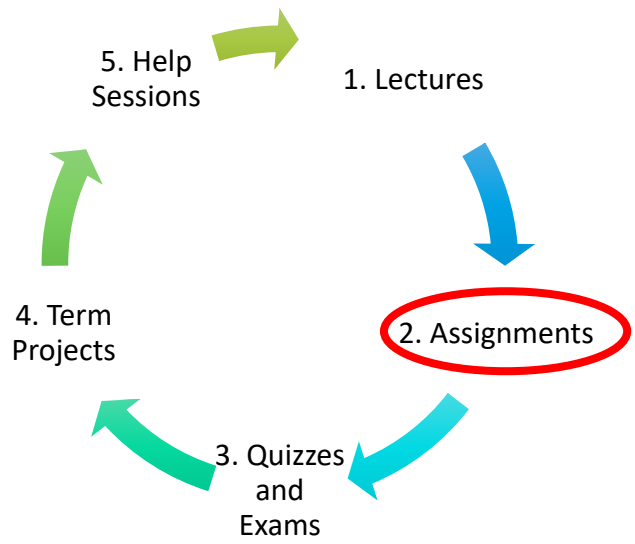


With case studies of
- Recommendation systems
- Machine learning
- Network analysis (e.g. PageRank)

33

33

Course Structure



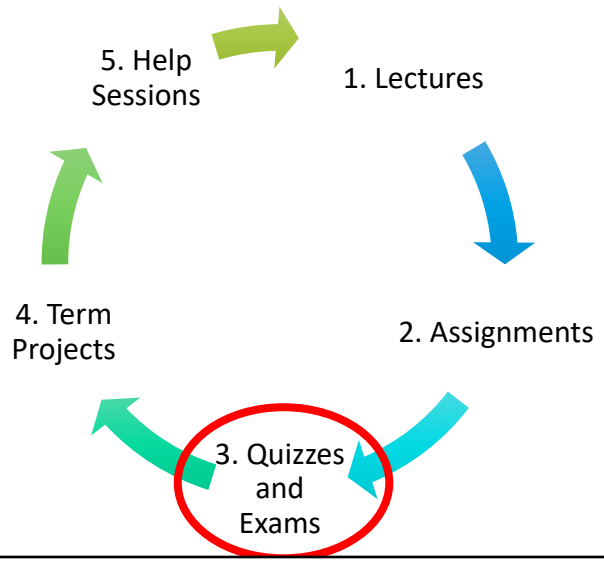
Programming Assignments (1/2)

- Providing hands-on experience
- Assignment 0: Set up your Hadoop cluster and a simple example
  - **Available on the course web**
  - **Due on Sept. 6 5PM**
  - **3 points**
  - **Go to the Assignment #0 in Module Week 1**
- Assignment 1: Profiling an Ego Network in Social Network Service with Scalable Graph Algorithms
- Assignment 2: Is Your Social Network a 'Small-World' network?
- Assignment 3: Large scale data analytics using Apache Spark

Programming Assignments (2/2)

- All assignments will be **individual** submissions
- Required programming language: **Java only**
- Grading will be based on **demo/interview**
- Late submission policy
  - Up to **2 days** of late submission is accepted
  - **10% deduction per-day** will be applied
- Assignment submission
  - Canvas

Course Structure



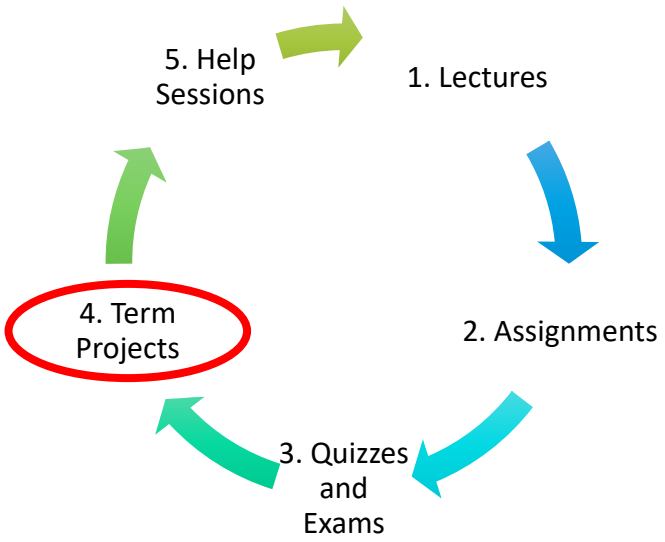
Quizzes and Exams (1/2)

- About 12 quizzes this semester
- Simple questions
- **2 lowest scores will be eliminated at the end of the semester**

Quizzes and Exams (2/2)

- Exams
  - Midterm (20%) and Final (20%)
  - 40% of the course grade
  - Mixture of simple questions and comprehensive questions

Course Structure



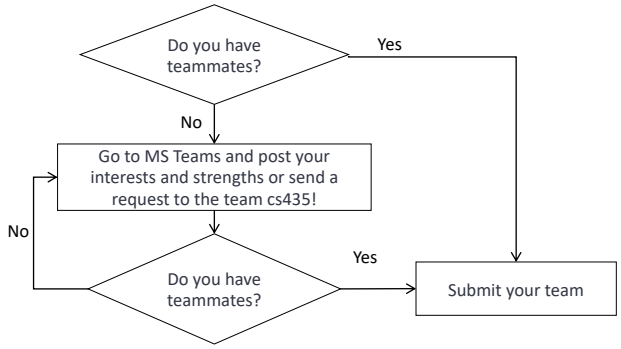
Term Project

- Comprehensive Big Data Analytics experiences
- Phase 0: Find your teammate!
- Phase 1: Term Project Proposal
- Phase 2: Final submission: Software and Report
- Phase 3: Presentation (Class presentation) and demonstration of your software

- Term project is a team effort **3-4 students per team**
  - No single member team or two member team allowed!

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

How do I find my Term Project teammates?

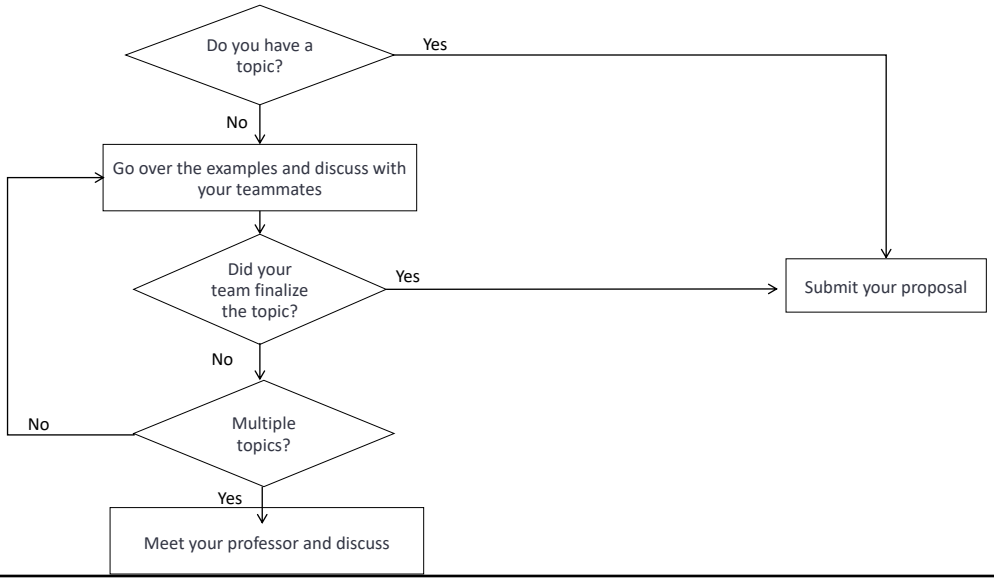


42

42

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

How do I select my Term Project Topic?



43

43

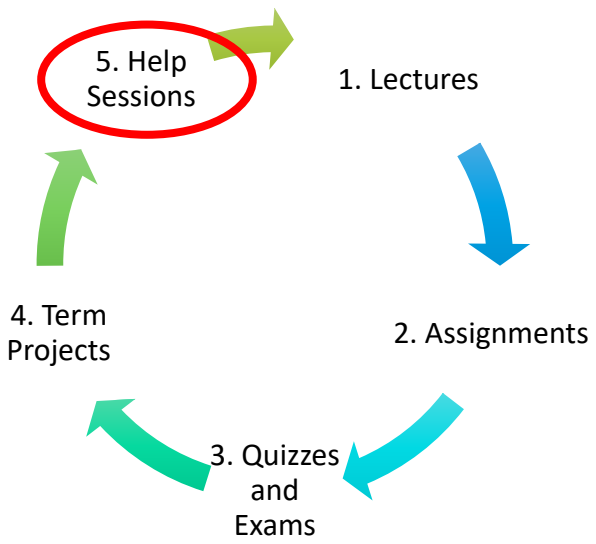
Example topics from previous courses

- Similarity and Clustering on The Million Song Dataset
- Methods Used to Analyze Eclipse and Mozilla Bug Data
- Movie Recommendations using Collaborative Filtering
- Climate Visualization and Predictive Analysis
- Wikipedia Page Traffic Statistics Analysis
  Trending Topics and Page Count Prediction on Wikipedia Traffic Log Data
- Supporting Emergency Response During Natural Disasters with Twitter Data

Example topics from previous courses

- "Time to Answer" for Questions on stackoverflow.com using Map Reduce
- Analysis of words for spell-checking in search queries using digitized books and articles
- Trend Analysis To Improve Donations
- Trends in baby names and predicting popular names
- Regression Analysis of New York City Taxi Fares
- Predicting Stock Similarity Using K-Means clustering
- Who is Building Wikipedia?

Course Structure



Recitation (Optional)

- Friday 4:00PM ~ 4:50PM, CS130
  - Recitation will be led by GTA and recorded
  - Hands-on sessions
    - Bring your laptop and finish the task!
- Video clips are available:
  - Hadoop Standalone Mode Configuration?
  - Hadoop Cluster Configuration?
  - Remote working environment?
  - Go to your Canvas Module and find Programming Assignment 0!

Grading Policy

Category	Points	Percentage of Final Grade
Programming Assignments	**Total: 30** PA0: 3 PA1: 7 PA2: 10 PA3: 10	30 %
Term Project	**Total: 20** Team assignment: 1 Proposal: 2 Final report: 12 Software: 1 Final presentation: 4	20 %
Quizzes and participation	**10**	10 %
Exams (Midterm, Final)	**40**	40%
Bonus	**2 (attendance)**	2%

Final Letter Grade

Letter grades will be based on the following standard breakpoints

- >= 90 is an A,
- >= 89 is an A-,
- >=88 is a B+,
- >=80 is a B,
- >=79 is a B-,
- >=78 is a C+,
- >=70 is a C,
- >=60 is a D,
- and <60 is an F

Course Policy

- No make-up for missed exams
  - Except in extraordinary circumstances (e.g., major illness, family emergency)
- No make-up for missed quizzes
  - Except for the case where there is an emergency

Online vs. On-Campus

- Online section will be operated 100% online
- If an on-campus student would like "virtual-access" to the course, please provide an approval from CSU
- Attendance Bonus score (2%)
  - Attendance for the On-Campus students: in class
  - Attendance for the Online students: if you have watched today's class recording before the next class begins, that class will count

Course Policy

- No text book.
  - I provide several on-line materials and research papers.
  - Rely on the lecture notes
- Accommodation/honor requirements
  - Week 1 and Week 2 Only

Plagiarism

- **We use plagiarism detection software**
- Please do not copy code or sentences from your references or your colleague's work
- All the course components (except for the term project) require individual submission

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

No laptop, hand-held device, or phone in class

- If you would like to use any device, please sit in the last row in the classroom
  - ATTENTION is OUR RESOURCE!
- Attend the class, ask questions, and discuss
- Check the Canvas modules, MS Teams discussion board, and announcement regularly
- Try new technologies and apply them
- Share your experiences with other students in class

54

54

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

Questions?

55

55

WEEK 1- LECTURE 2

OVERVIEW OF BIG DATA ANALYTICS AND INTRODUCTION TO MAPREDUCE

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

- Do we need to use Java for the term project?
  - No. For your term project, you may use Spark or MapReduce or both with any language that the platform supports
- Do we have help session this week?
  - Yes. 4-5pm on Fridays via MS Teams
  - Laksheen will go over the PA0 on Friday (8/28)
  - Help desk will be open Tuesday 2.00 PM ~ 4.00 PM, Thursday 3.00 PM ~ 5.00 PM
- Where can I find the course information and video clips?
  - Visit your Canvas Modules!
    - We collect all the materials there

Topics to be covered

- Overview of Big Data Analytics
- Introduction to MapReduce

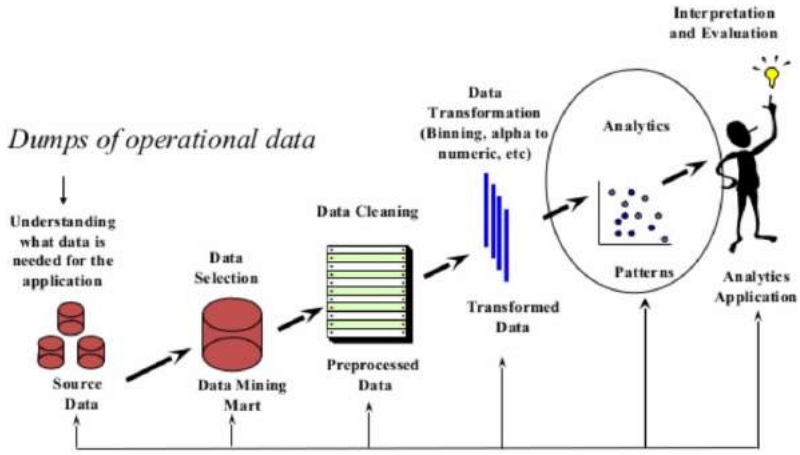
Part 1. Large Scale Data Analytics

TOPIC-1: Overview of Big Data Analytics

This Material is Built Based on,

- Analytics in a Big Data World: The Essential Guide to Data Science and Its Applications, Bart Baesens, 2014, Wiley

Analytics Process Model



The most time-consuming step is the data selection and preprocessing step
- This is usually around 80% of the total time needed to build an analytical model

Analytics in a Big Data World: The Essential Guide to Data Science and Its Applications, Bart Baesens, 2014, Wiley

Types of Analytics

- Analytics is a term that is often used interchangeably with
  - Data science
  - Data mining
  - Knowledge discovery
- **Descriptive analytics**
  - No target variable
  - e.g. Clustering, association rules
- **Predictive analytics**
  - A target variable is typically available
  - E.g. linear/logistic regression, decision trees, neural networks, support vector machines

Types of Data Sources

- **Transactions**
  - Structured, low-level, detailed information
  - Customer transactions
    - Purchase, claim, cash transfer, credit card payment
  - Stored in massive online transaction processing (OLTP) relational database
  - Can be summarized over longer time horizons (e.g. averages, relative trends, Max/Min values)
- **Unstructured data embedded in text documents**
  - emails, web pages, claim forms, audio and video, images
  - Requires extensive preprocessing
- **Qualitative, expert-based data**
  - Requires subject matter experts' (SME) analysis
  - Scientific data

Sampling

- Taking a **subset** of data for analytics
  - Generating hypothesis
  - Model selection
  - Feature selection
  - Speculative process
  - Building analytics model

- **Stratified sampling**
  - Taking samples according to predefined strata
  - e.g. Fraud detection with very skewed (99 percent non-fraud customers, 1 percent fraud customers)
    - Sample should contain the same percentage of fraud customers as in the original data

Types of Data Elements

- **Continuous**
  - Data elements that are defined on an interval that can be limited or unlimited
    - e.g. income, sales, temperature
- **Categorical Nominal**
  - Data elements that can only take on a limited set of values with no meaningful ordering between them
    - e.g. marital status, profession, purpose of loan
- **Ordinal**
  - Data elements that can only take on a limited set of values with a meaningful ordering between them
    - e.g. credit rating, age coded as young, middle age and old
- **Binary**
  - Data elements that can only take on two values
    - e.g. Having child, allowed to drive

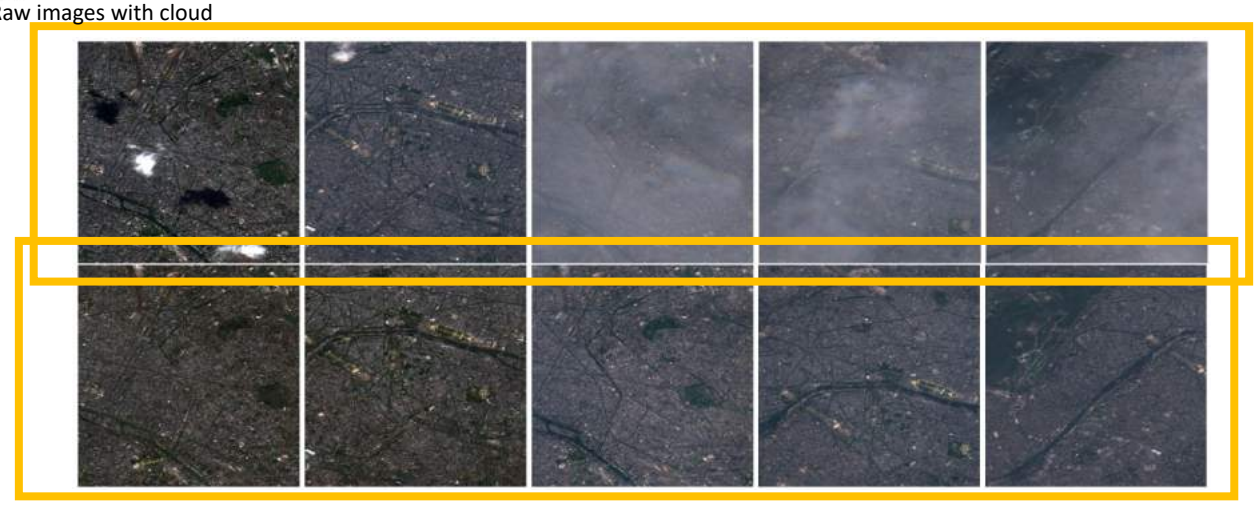
Missing Values

- **Missing values** can occur because of various reasons
  - The information can be **non-applicable**
  - The information can be **undisclosed**
  - The information can be **unavailable**

Missing Values --continued

- **Replace (impute)**
  - Replaces the missing value with a computed/selected value
  - Imputation algorithm examples
    - Hot-deck: replaces with a randomly selected similar records
    - Cold-deck: selects replacement from another dataset
    - Mean substitution: replaces with the mean of that variable for all other cases
    - Regression: predicts missing values of a variable based on other variables.
- **Delete**
  - Deletes observations with lots of missing values
  - This assumes that information is missing at random and has no meaningful interpretation and/or relationship to the target
- **Keep**
  - Missing values can be meaningful
    - e.g. a customer did not disclose the income for current condition

Raw images with cloud



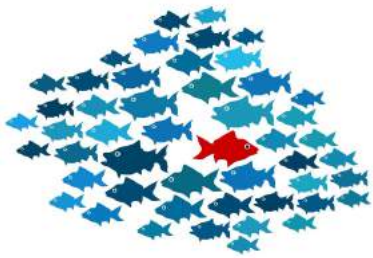
Cloud removed image using GAN

Singh, P. and Komodakis, N., 2018, July. Cloud-gan: Cloud removal for sentinel-2 imagery using a cyclic consistent generative adversarial networks. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*(pp. 1772-1775). IEEE.

12

Outliers of Dataset

- **Outliers** are extreme observations that are very dissimilar to the rest of the population
  - **Valid observation**
    - Salary of boss
  - **Invalid observation**
    - Age is 300



- **Multivariate outliers**
  - Observations that are outlying in multiple dimensions
    - e.g: Temperature in Fort Collins is 100 degrees but on a midnight in December

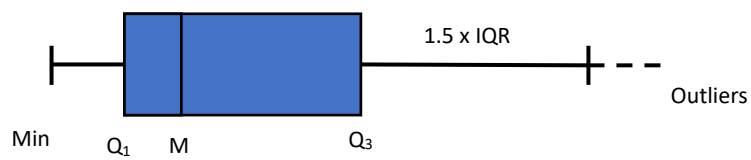
13

13

Identifying Outliers using Box Plots

- A box plot represents three key quartiles of the data
  - **Q_1**: 25% of the observations have a lower value
  - **Q_2**: 50% of the observations have a lower value
  - **Q_3**: 75% of the observations have a lower value
  - The **minimum** and **maximum** values are added

- **Too far away** is now quantified as more than **1.5 x Interquartile Range** ($IQR = (Q_3 - Q_1)$)



Identifying Outliers using *Z*-Score

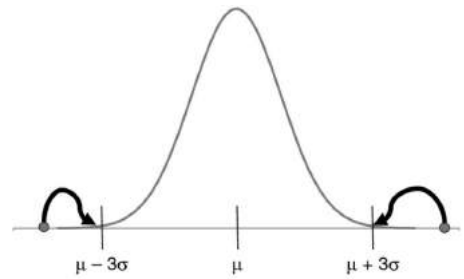
- Measuring how many standard deviations an observation is away from the mean
  - $z_i = \frac{x_i - \mu}{\sigma}$
    where μ represents the average of the variable and σ its standard deviation

- A practical rule of thumb then defines outliers when the absolute value of the z-score | z | is bigger than 3

ID	Age	Z-Score
1	30	(30-40)/10=-1
2	50	(50-40)/10=+1
3	10	(10-40)/10=-3
4	40	(40-40)/10=0
5	60	(60-40)/10=+2
6	80	(80-40)/10=+4
--	..	…
	$\mu = 40$ $\sigma = 10$	$\mu = 0$ $\sigma = 1$

Dealing with Outliers

- Treat outliers as **missing values**
- Popular schemes
  - **Truncation**
    - Taking only values that are within the limits
  - **Winsorizing**
    - Limiting extreme values to reduce the effect of possible spurious outliers
    - {92, 19, **101**, 58, **1053**, 91, 26, 78, 10, 13, **-40**, **101**, 86, 85, 15, 89, 89, 28, **-5**, 41} (N = 20, mean = 101.5)
      → {92, 19, **101**, 58, **101**, 91, 26, 78, 10, 13, **-5**, **101**, 86, 85, 15, 89, 89, 28, **-5**, 41} (N = 20, mean = 55.65)



Using the Z-Scores for truncation

Standardizing Data

- **Scaling variables to a similar range**
  - e.g. predicting credit score using two variables: education and income
  - Elementary school (1), middle school (2), high school (3), college (4), graduate school (5)
  - Income: 0 ~ $5M
  - When building logistic regression models, the coefficient for education might become very small.
- **Min/Max standardization**
  - $X_{new} = \frac{X_{old} - \min(X_{old})}{\max(X_{old}) - \min(X_{old})}(newmax - newmin) + newmin$
  - Where $newmax$ and $newmin$ are the newly imposed $maximum$ and $minimum$ (e.g. 1 and 0)

Standardizing Data. -- continued

- **Z-Score based**
  - Calculate the z-scores
- **Decimal scaling**
  - $X_{new} = \frac{X_{old}}{10^n}$
  - Dividing by a power of 10

- Standardization is useful for regression-based approaches
  - It is not needed for decision trees

Part 1. Large Scale Data Analytics

TOPIC-2: Introduction to MapReduce

COLORADO STATE UNIVERSITY | CS435 Introduction to Big Data: Week 1-B | Fall 2021 Computer Science

Big Data Analytics
-Big Data Technology Stack

20

20

21

In a nutshell

Security and Governance

Data Presentation Layer
Apache Kibana

Data Integration Layer
Apache Flume, Apache Kafka, Apache Sqoop

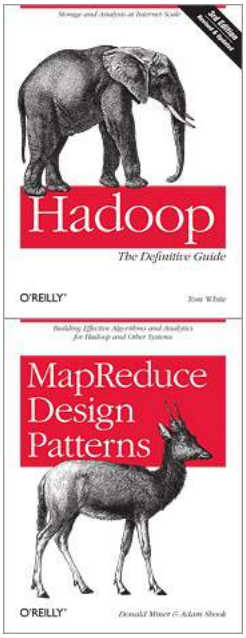
Operations and Scheduling Layer
Apache Ambari
Apache Oozie
Apache Zookeeper

Data Processing Layer
Apache Hadoop MapReduce, Pig, Apache Spark, Cassandra, Storm, Mahout, MLLib, TensorFlow, PyTorch

Data Layer
Apache HDFS, Amazon AWS's S3, IBM GPFS, Microsoft Azure

This material is developed based on,

- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012 --Chapter 2
  - Download this chapter from the CS435 schedule page
- Hadoop: The definitive Guide, Tom White, O'Reilly, 3rd Edition, 2014
- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013



What is MapReduce?

MapReduce [1/2]

- **MapReduce** is **inspired** by the concepts of *map* and *reduce* in Lisp.
- ***"Modern"* MapReduce**
  - Developed within Google as a mechanism for processing large amounts of raw data.
    - Crawled documents or web request logs
    - Distributes these data across thousands of machines
    - **Same computations** are performed on each CPU with different dataset

Sorting 1PB of numbers with a single machine

- What will be the challenges?

Sorting 1PB of numbers with multiple machines

- What will be the requirements to perform this sorting?

MapReduce [2/2]

- **MapReduce** provides an abstraction that allows engineers to perform simple computations while hiding the details of **parallelization**, **data distribution**, **load balancing** and **fault tolerance**

Mapper

- **Mapper** maps **input key/value pairs** to a set of **intermediate key/value pairs**
  - Maps are the individual tasks that transform input records into intermediate records
  - The transformed intermediate records **do not need to be of the same type** as the input records
  - A given input pair may map to **zero or many** output pairs
  - The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job

Reducer

- Reducer reduces **a set of intermediate values which share a key** to a smaller set of values
- Reducer has 3 primary phases
  - Shuffle, sort and reduce
- **Shuffle**
  - Input to the reducer is the sorted output of the mappers
  - The framework fetches the relevant partition of the output of all the mappers via HTTP
- **Sort**
  - The framework groups input to the reducer by keys

MapReduce 101 with an Example

Example 1: WordCount [1/5]

- For text files stored under `usr/joe/wordcount/input`, count the number of occurrences of each word
- How do files and directory look?

```
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/
/usr/joe/wordcount/input/file01
/usr/joe/wordcount/input/file02

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World, Bye World!

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop, Goodbye to hadoop.
```

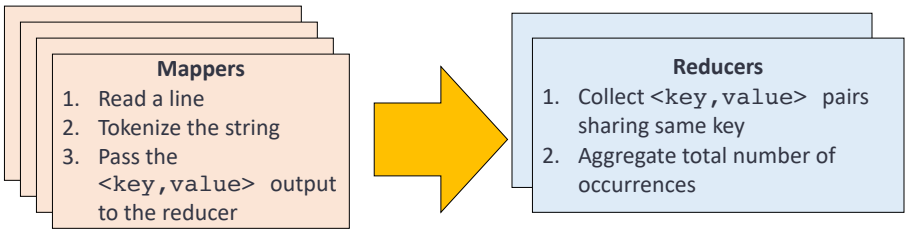

Example 1: WordCount [2/5]

- Run the MapReduce application

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount
/usr/joe/wordcount/input /usr/joe/wordcount/output


$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop, 1
Hello 2
World! 1
World, 1
hadoop. 1
to 1
```

Example 1: WordCount [3/5]



What do you have to pass from the Mappers?

Example 1: WordCount [4/5]

```
public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
           while (tokenizer.hasMoreTokens()) {
                 word.set(tokenizer.nextToken());
                 context.write(word, one);
             }
  }
}
```

Example 1: WordCount [5/5]

```
public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable> {

       public void reduce(Text key, Iterable<IntWritable> values,
Context context)
         throws IOException, InterruptedException {
           int sum = 0;
           for (IntWritable val : values) {
               sum += val.get();
           }
           context.write(key, new IntWritable(sum));
       }
    }
```


Questions?

WEEK 2- LECTURE 1

INTRODUCTION TO MAPREDUCE AND MR DESIGN PATTERNS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

- Quizzes
  - Less than 1 point
  - Lowest 2 scores will be eliminated
  - From Quiz 2, open notes
  - No Internet, No communication with your classmates
- Team assignment
  - If you could not find your team, please send a request to cs435@cs.colostate.edu
- Programming Assignment 0
  - Help desks this week: Online + in-Person
  - Submission deadline: 5PM on September 6 via Canvas
  - **Individual submission**
  - Recitation this week: 4-5PM Friday, CSB130

FAQs

- Many questions about the term project!
- How to do a successful term project
  1. Term project proposal
  2. Final report/software
  3. Presentation
  4. Software demo
- Evaluation Criteria (RCL metric)
  - **Relevance**: Is this a big data problem? Does this team use a big data technology to solve the problem?
  - **Completeness**: Does the project provide sufficient Big Data components? Methodology? Evaluation method?
  - **Level of Challenge**: Does your project involve adequate level of challenges as a senior level CS term project?

Topics that we have discussed in the previous class

- Big Data analytics model
  - How to use samples in BD analytics
  - How do we manage outliers and missing data?
- Very first Introduction to MapReduce
  - What do we need to perform a large-scale computation over a cluster?
  - Simple example: Word Count
    - Mapper
    - Reducer
    - <key, value>

Topics to be covered

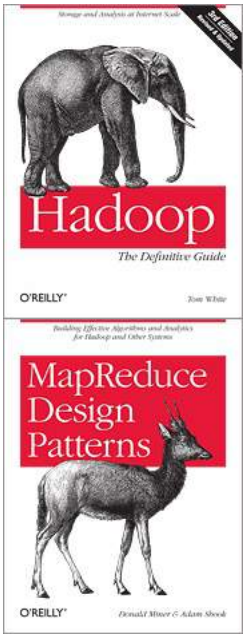
- Topic 1: Introduction to MapReduce: example 2
- Topic 2: MapReduce Design Pattern I. Summarization Patterns

Part 1. Large Scale Data Analytics

TOPIC-1: Introduction to MapReduce

This material is developed based on,

- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012 --Chapter 2
  - Download this chapter from the CS435 schedule page
- Hadoop: The definitive Guide, Tom White, O'Reilly, 3rd Edition, 2014
- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013



MapReduce Example 2

- Focusing on the MapReduce Components

NCDC data example

- A national climate data center record
- Find the maximum temperature of a year (1900 ~ 1999)

```
0057
332130 # USAF weather station identifier
99999 # WBAN weather station identifier
19500101 # observation date
0300 # observation time 4
+ 51317 # latitude (degrees x 1000)
+ 028783 # longitude (degrees x 1000)
FM-12
+ 0171 # elevation (meters)
99999
V020
320 # wind direction (degrees)
1 # quality code
N
```


The first entries for 1990

```
% ls raw/ 1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```

Analyzing the data with Unix Tools (1/2)

- A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/ usr/ bin/ env bash
for year in all/*
do
 echo -ne ` basename $ year .gz `"\t"
 gunzip -c $ year |
   \awk '{ temp = substr( $ 0, 88, 5) + 0;
      q = substr( $ 0, 93, 1);
      if (temp != 9999 && q ~ /[01459]/ && temp > max)
          max = temp
      }
  END { print max }'
Done
```


Analyzing the data with Unix Tools (2/2)

- The script loops through the compressed year files
  - Printing the year
  - Processing each file using `awk`
    - Extracts two fields
    - Air temperature and the quality code
    - Check if it is greater than the maximum value seen so far

```
% ./ max_temperature.sh
1901 317
1902 244
1903 289
1904 256
1905 283
…
```

Results?

- The complete run for the century took **42 minutes**
- To speed up the processing
  - We need to run parts of the program in parallel
  - Process different years in different processes

Challenges

- Dividing the work into equal-size pieces
  - Data size per year?
- Combining the results from independent processes
  - Combining results and sorting by year?
- You are still limited by the processing capacity of a single machine (**the worst one**)!

Map and Reduce

- MapReduce works by breaking the processing into two phases
  - The map phase
  - The reduce phase
- Each phase has key-value pairs as input and output
- Programmers should specify
  - **Types of input/output key-values**
  - **The map function**
  - **The reduce function**

Visualizing the way the MapReduce works (1/3)

Sample lines of input data

```
0067011990999991950051507004... 9999999N9 + 00001 +99999999999...
0043011990999991950051512004... 9999999N9 + 00221 +99999999999...
0043011990999991950051518004... 9999999N9-00111 +99999999999...
0043012650999991949032412004... 0500001N9 + 01111 +99999999999...
0043012650999991949032418004... 0500001N9 + 00781 +99999999999…
```

These lines are presented to the map function as the key-value pairs

```
(0, 0067011990999991950051507004...9999999N9 + 00001+
99999999999...)
(106, 0043011990999991950051512004...9999999N9 + 00221+
99999999999...)
(212, 0043011990999991950051518004...9999999N9-0011 1 +
99999999999...)
```

The keys are the line offsets within the file (optional)

Visualizing the way the MapReduce works (2/3)

The map function extracts the year and the air temperature and emit them as its output

```
(1950, 0)
(1950, 22)
(1950, – 11)
(1949, 111)
(1949, 78)
```

This output key-value pairs will be sorted (by key) and grouped by key

Values passed to each reducer are NOT sorted

Our reduce function will see the following input:

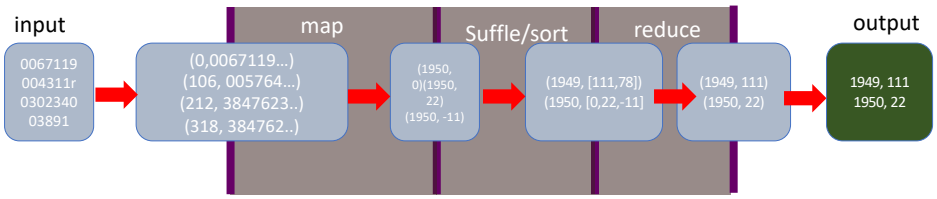
```
(1949, [111, 78])
(1950, [0, 22, –11])
```


Visualizing the way the MapReduce works (3/3)

Reduce function iterates through the list and pick up the maximum reading

```
(1949, 111)
(1950, 22)
```

This is the final output



Comparison with other systems

- MPI vs. MapReduce
  - MapReduce tries to collocate the data with the compute node
  - Data access is fast
    - Data is local!
- Volunteer computing vs. MapReduce
  - SETI@home
    - Using donated CPU time

MapReduce Example 3

Dataset

- Network communication log file

Source	Destination
220.4.5.20	160.33.1.3
220.4.5.20	100.33.1.5
160.33.1.3	79.45.66.9
160.33.1.3	100.33.1.5
100.33.1.5	79.45.66.9

- There are two attributes
  - *From* and *To*
  - *(IP1, IP2)* : There was a communication from IP1 to IP2

Retrieve records with “220.4.5.20” as a source IP without any duplicate record

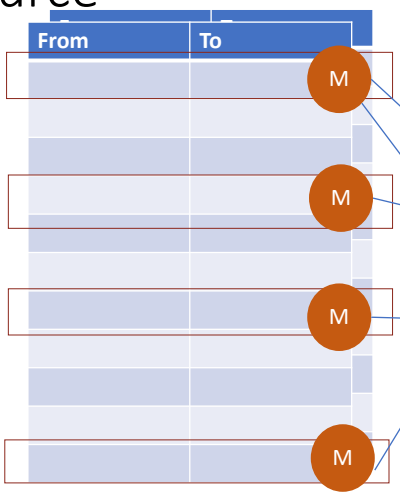
- Check each record and provide as output only those records containing “220,4,5,20” as a source IP

220.4.5.20	160.33.1.3
220.4.5.20	100.33.1.5
160.33.1.3	79.45.66.9
160.33.1.3	100.33.1.5
100.33.1.5	79.45.66.9

→

220.4.5.20	160.33.1.3
220.4.5.20	100.33.1.5

Retrieve records with "220.4.5.20" as the source



Map function
For each record, check the condition (source == "220.4.5.20" and produce the key-value pair ("220.4.5.20", record)

Question: What if the current record does not match with "220.4.5.20"?

a. Emit a dummy tuple, e.g. (a random number, record)
b. Emit the current value
c. Emit ("220.4.5.20", record)
d. Do not emit any tuple

Reduce function
For the shuffled pairs (("220.4.5.20", [recordA, recordA, recordB])
Returns ("220.4.5.20", ,[recordA, recordB]) as the result of eliminating duplications

Worksheet #2-A-1

- *With a dataset in the same format, retrieve all the local accesses per IP using MapReduce*

Source	Destination
220.4.5.20	160.33.1.3
220.4.5.20	220.4.5.20
160.33.1.3	79.45.66.9
160.33.1.3	100.33.1.5
100.33.1.5	79.45.66.9

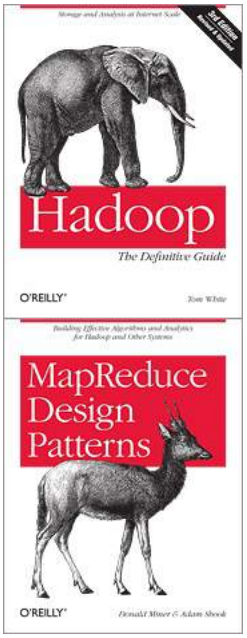
1. What should be the input for your Mapper?
2. What should be the output from your Mapper?
3. What should be the input for your Reducer?
4. What should be the output from your Reducer?

Part 1. Large Scale Data Analytics

Design Pattern 1: Summarization Patterns

This material is developed based on,

- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012 --Chapter 2
  - Download this chapter from the CS435 schedule page
- Hadoop: The definitive Guide, Tom White, O'Reilly, 3rd Edition, 2014
- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013



Why Design Patterns?

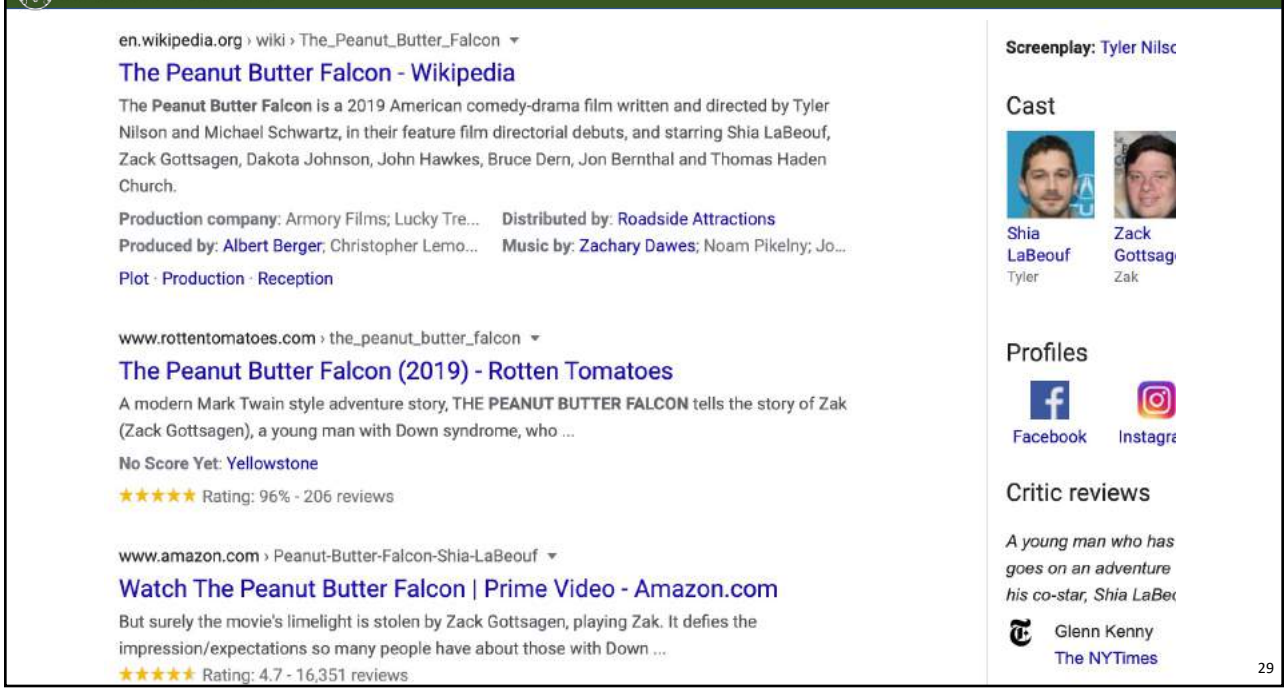
- Tools for solving problems
  - Reusable and providing a general framework
  - Developers can spend less time figuring out how she/he is going to solve the problem

MapReduce Design Patterns to be covered

1. Summarization
2. Filtering Patterns
3. Data Organization Patterns
4. Join Patterns
5. I/O Patterns

Summarization Patterns

- Design patterns that produce a top-level, summarized view of your data
  - Helps you to glean insights not available from looking at a localized set of records alone
- **Numerical summarizations**
  - Max/Min, Count, Average, Median, Standard Deviation
- **Inverted index**
- **Text summarization**



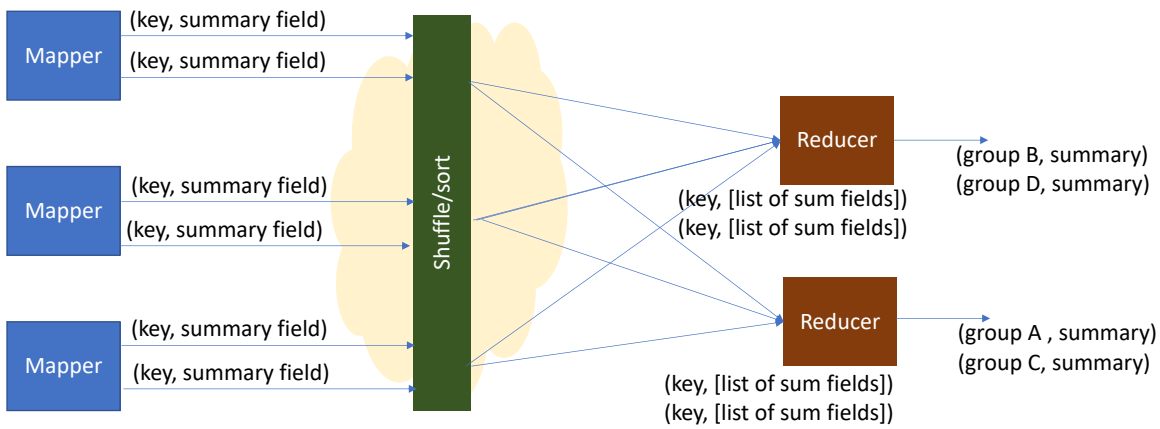
Design Pattern 1: Summarization Patterns

1. Numerical Summarizations

Introduction

- Pattern Description
  - General pattern for **calculating aggregate statistical values over data**
  - **E.g. What is the total expense of each customer?**
- Intent
  - Group records together by a key field and calculate a **numerical aggregate** per group to get a top-level view of the larger data set
- Applicability
  - Numerical summarizations should be used when both of the followings are true
    - Numerical data or counting
    - **The data can be grouped by specific fields**

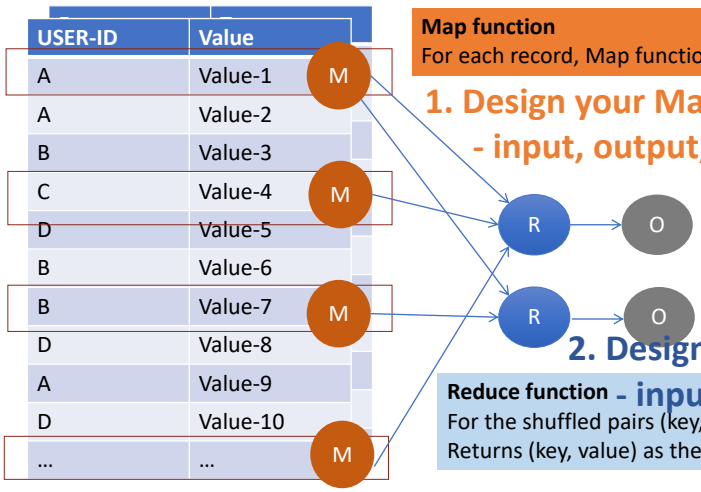
Structure



Numerical Summarization
Example 1. Minimum, Maximum and Count

- Calculating the Min, Max and Count of ***Values*** per user

USER-ID	Value
A	Value-1
A	Value-2
B	Value-3
C	Value-4
D	Value-5
B	Value-6
B	Value-7
D	Value-8
A	Value-9
D	Value-10
…	…



Map function
For each record, Map function produces the key-value pair (key, value)

1. Design your Map function
- input, output, functionality

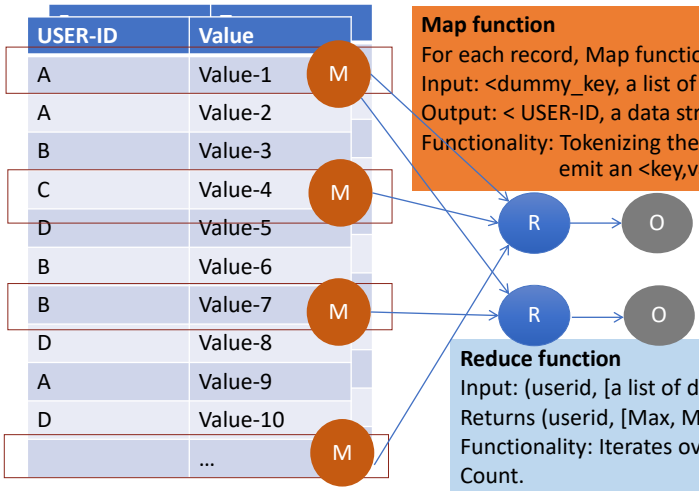
2. Design your Reduce function

Reduce function - input, output, functionality
For the shuffled pairs (key, [a list of values])
Returns (key, value) as the result

Numerical Summarization
Example 1. Minimum, Maximum and Count

- Calculating the Min, Max and Count of ***Value*** per user

USER-ID	Value
A	Value-1
A	Value-2
B	Value-3
C	Value-4
D	Value-5
B	Value-6
B	Value-7
D	Value-8
A	Value-9
D	Value-10
	…



Map function
For each record, Map function produces the key-value pair (key, value)
Input: <dummy_key, a list of values (a line of string)>
Output: < USER-ID, a data structure contains" values">
Functionality: Tokenizing the string, fill out the output data structure, emit an <key,value> pair

Reduce function
Input: (userid, [a list of data values])
Returns (userid, [Max, Min, Count]) as the result
Functionality: Iterates over the list and calculate Min, Max and Count.

34

34

Mapper Code

```
public static class MinMaxCountMapper extends Mapper < Object, Text, Text,
MinMaxCountTuple > {
        // Our output key and value Writables
        private Text outUserId = new Text();
        private MinMaxCountTuple outTuple = new MinMaxCountTuple();
        public void map( Object key, Text value, Context context) throws IOException,
InterruptedException {
                Map < String, String > parsed = transformXmlToMap( value.toString());
                // Assume we have developed a class, parse.
                // Grab the "Value" field since it is what we are finding
                // the min and max value of
                String value = parsed.get("value");

                // Grab the "UserID" since it is what we are grouping by
                String userId = parsed.get("UserId");

                // Set the minimum and maximum date values to the value
                outTuple.setValue(value);


                // Set the comment count to 1
                outTuple.setCount(1);
                // Set our user ID as the output key
                outUserId.set(userId);
                // Write out the hour and the average comment length
                context.write(outUserId, outTuple);
```

35

35

Reducer Code

```
public static class MinMaxCountReducer extends Reducer < Text, MinMaxCountTuple,
Text, MinMaxCountTuple > {

        // Our output value Writable
        private MinMaxCountTuple result = new MinMaxCountTuple();
        public void reduce(Text key, Iterable < MinMaxCountTuple > values, Context
context) throws IOException, InterruptedException {
                // Initialize our result
                result.setMin(null);
                result.setMax(null);
                result.setCount(0);
                int sum = 0;
```

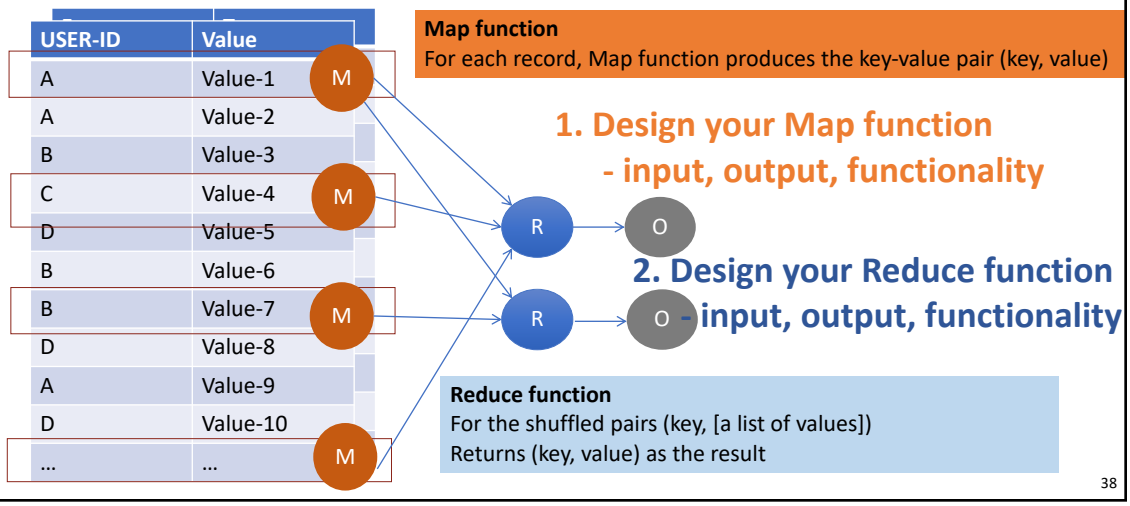

Reducer Code --continued

```
        // Iterate through all input values for this key
        for (MinMaxCountTuple val : values) {
                // If the value is less than the result's min
                // Set the result's min to value
                if (result.getMin()==null || val.getValue().compareTo(result.getMin())<0) {
                        result.setMin(val.getValue());
                }
                // If the value is more than the result's max
                // Set the result's max to value
                if (result.getMax()==null || val.getValue(). compareTo(result.getMax())>0) {
                        result.setMax(val.getValue());
                }
                // Add to our sum the count for value
                sum += val.getCount();
                }
                //Set our count to the number of input values
                result.setCount(sum); context.write(key, result);
        }
    }
```

Numerical Summarization Example 2. Average

- Calculating an average of ***Values*** per user

USER-ID	Value
A	Value-1
A	Value-2
B	Value-3
C	Value-4
D	Value-5
B	Value-6
B	Value-7
D	Value-8
A	Value-9
D	Value-10
…	…



Map function
For each record, Map function produces the key-value pair (key, value)

1. Design your Map function
- input, output, functionality

2. Design your Reduce function
- input, output, functionality

Reduce function
For the shuffled pairs (key, [a list of values])
Returns (key, value) as the result

38

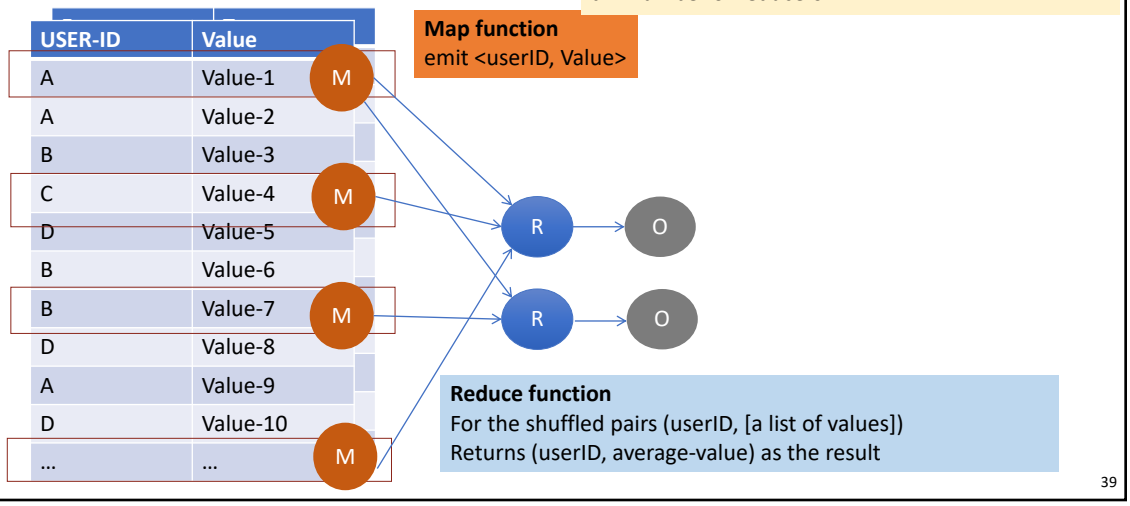
38

Numerical Summarization Example 2. Average

- Calculating an average of ***Values*** per user

Question. How many "value"s will be delivered from the mappers to the reducers?
a. 1
b. Number of total records
c. Number of total distinctive users
d. Number of reducers

USER-ID	Value
A	Value-1
A	Value-2
B	Value-3
C	Value-4
D	Value-5
B	Value-6
B	Value-7
D	Value-8
A	Value-9
D	Value-10
…	…



Map function
emit <userID, Value>

Reduce function
For the shuffled pairs (userID, [a list of values])
Returns (userID, average-value) as the result

39

39

Design Pattern 1: Summarization Patterns

Combiner Functions

Hadoop Combiner

- **Minimizes the data transferred** between map and reduce tasks
- Users can specify a **combiner function**
  - To be run on the map output
  - To replace the map output with the combiner output
- Hadoop does **NOT** guarantee how many times it will call combiner for a particular map output record

Example: Find the maximum temperature

- First map produces
  (1950, 0)
  (1950, 20)
  (1950, 10)
- Second map produces
  (1950, 25)
  (1950, 15)
- Input to the reduce function
  (1950, **[0, 20, 10, 25, 15]**)
- Output
  (1950, 25)

If a combiner finds the maximum temperature for each map output:

- First map produces
  (1950, 0)
  (1950, 20)
  (1950, 10)
  → **(1950, 20)**
- Second map produces
  (1950, 25)
  (1950, 15)
  → **(1950, 25)**
- Input to the reduce function
  (1950, [0, 20, 10, 25, 15])
  →**(1950, [20, 25])**
- Output
  (1950, 25)→(1950, 25)

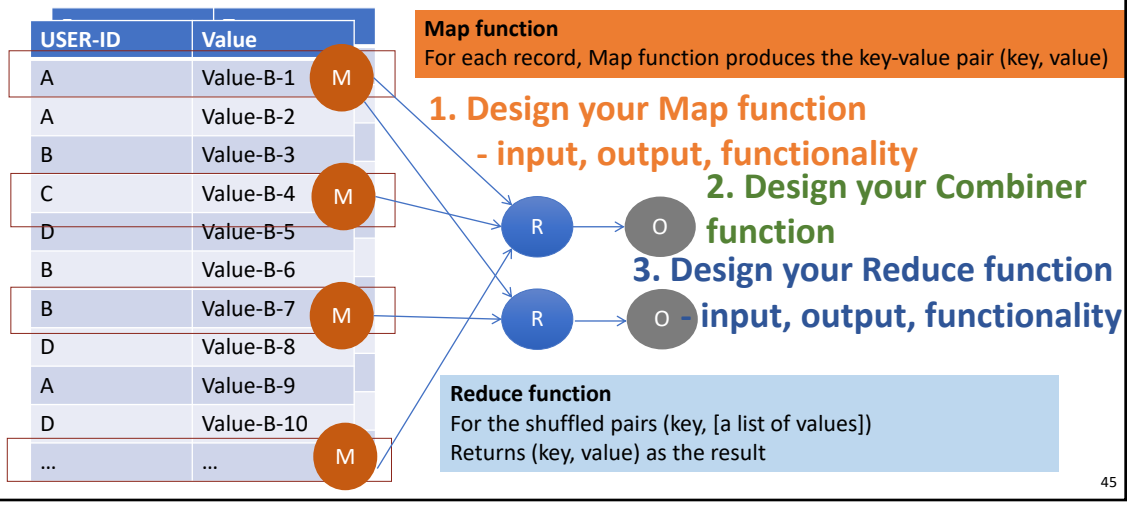
Question

- Can we use a combiner function for finding average value?
  - Yes
  - No

Numerical Summarization Example 2. Average

- Calculating the Average of ***Values*** per user

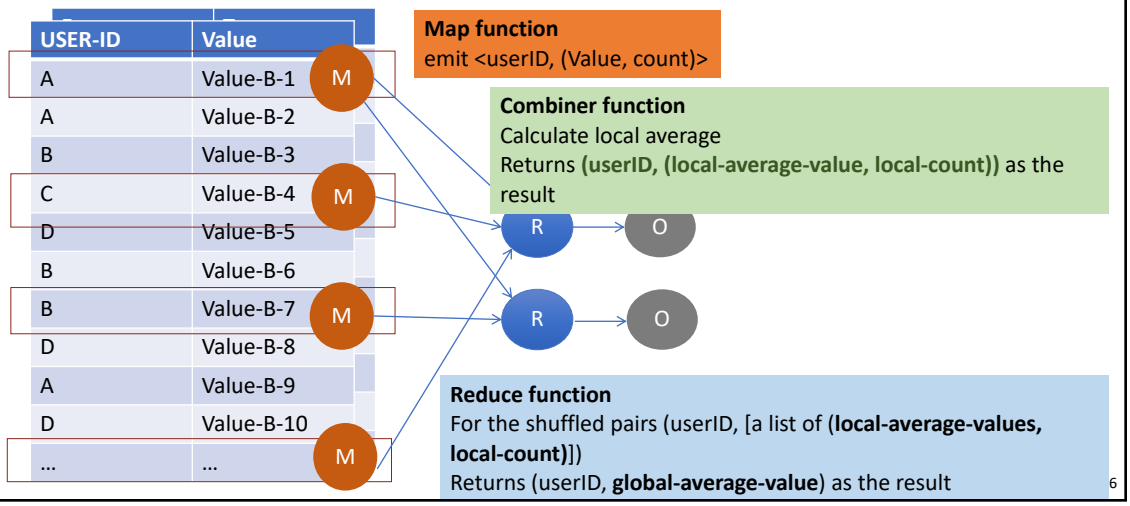
USER-ID	Value
A	Value-B-1
A	Value-B-2
B	Value-B-3
C	Value-B-4
D	Value-B-5
B	Value-B-6
B	Value-B-7
D	Value-B-8
A	Value-B-9
D	Value-B-10
…	…



Numerical Summarization Example 2. Average

- Calculating the Average of ***Values*** per user

USER-ID	Value
A	Value-B-1
A	Value-B-2
B	Value-B-3
C	Value-B-4
D	Value-B-5
B	Value-B-6
B	Value-B-7
D	Value-B-8
A	Value-B-9
D	Value-B-10
…	…



Design Pattern 1: Summarization Patterns

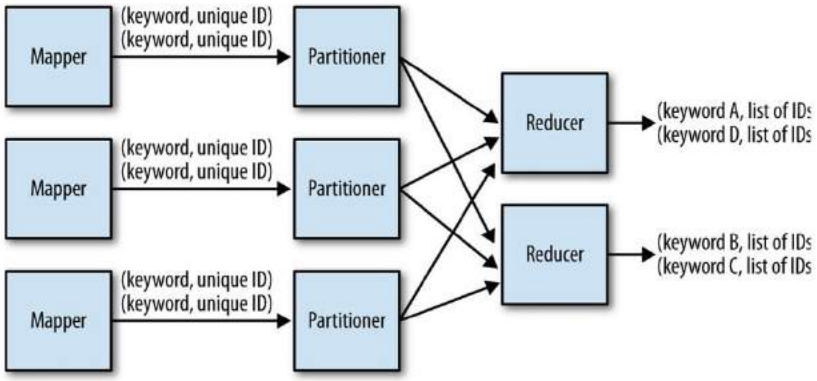
Inverted Index

Inverted Index

- Generate Index from a data dataset to map from **contents**, such as words or numbers
  - Reduces the amount of time to find related items
  - Keyword based search, Web search, and document search
  - e.g. Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment

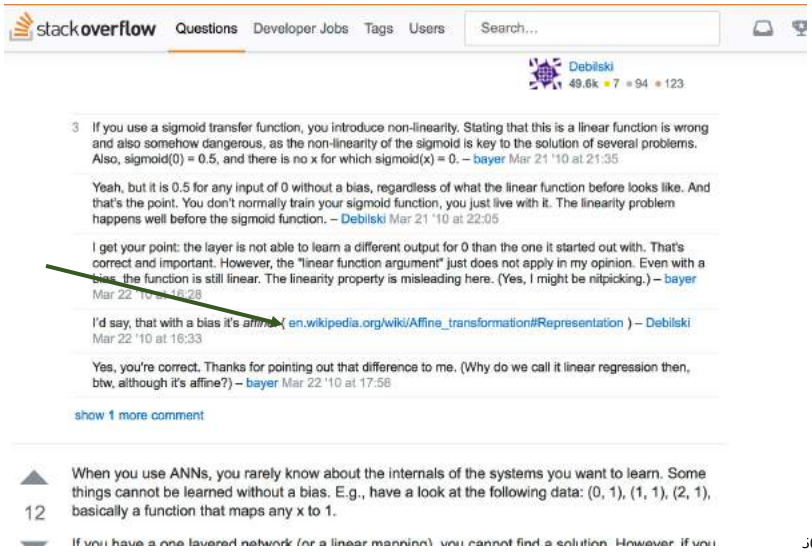
*** Example of an inverted index for document 1 and 2 (D1, and D2)**
D1= "Colorado State University"
{Colorado, State, University}
D2= "University of Colorado"
{University, of, Colorado}
Inverted Index
Colorado – {D1, D2}
State - {D1}
University – {D1, D2}
of – {D2}

Structure



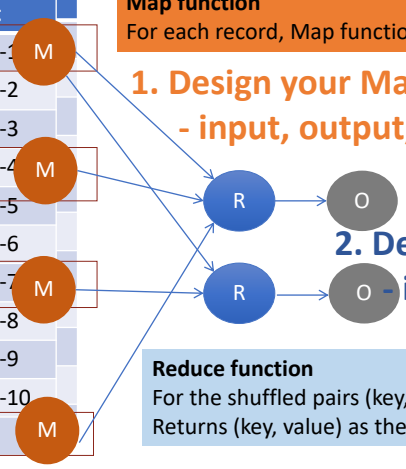
Inverted index of StackOverflow links to Wikipedia

Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment



Inverted index of StackOverflow links to Wikipedia

CommentID	Comment
A	Comment-1
B	Comment-2
C	Comment-3
D	Comment-4
E	Comment-5
F	Comment-6
G	Comment-7
H	Comment-8
I	Comment-9
J	Comment-10
…	…



Map function
For each record, Map function produces the key-value pair (key, value)

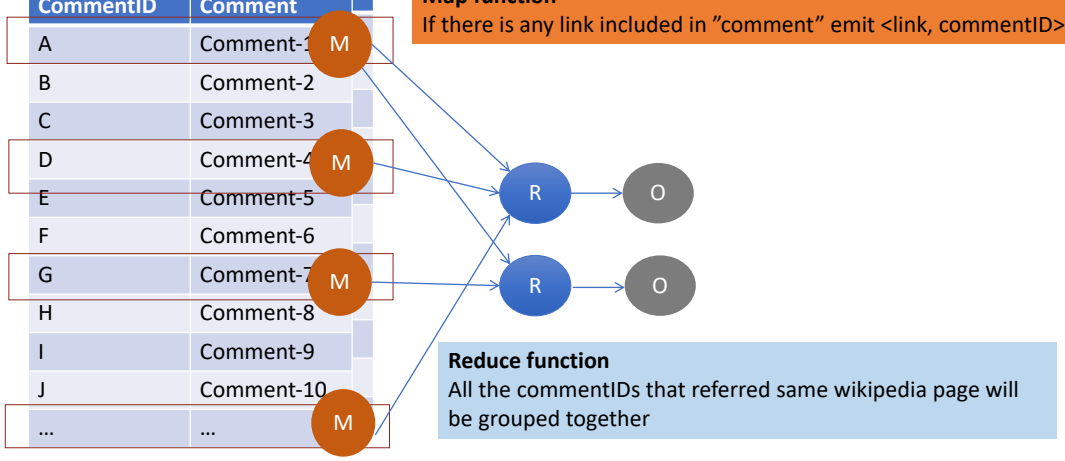
1. Design your Map function
- input, output, functionality

2. Design your Reduce function
- input, output, functionality

Reduce function
For the shuffled pairs (key, [a list of values])
Returns (key, value) as the result

Inverted index of StackOverflow links to Wikipedia

CommentID	Comment
A	Comment-1
B	Comment-2
C	Comment-3
D	Comment-4
E	Comment-5
F	Comment-6
G	Comment-7
H	Comment-8
I	Comment-9
J	Comment-10
…	…



Map function
If there is any link included in "comment" emit <link, commentID>

Reduce function
All the commentIDs that referred same wikipedia page will be grouped together

Mapper Code

```
public static class WikipediaExtractor extends Mapper < Object, Text, Text, Text > {
        private Text link = new Text();
        private Text outkey = new Text();
        public void map(Object key, Text value, Context context)
                                        throws IOException, InterruptedException {
            Map <String, String> parsed = MRDPUtils.transformXmlToMap(value .toString());
            // Grab the necessary XML attributes
            String txt = parsed.get("Body");
            String posttype = parsed.get("PostTypeId");
            String row_id = parsed.get("Id");
            // if the body is null, or the post is a question (1), skip
            if (txt == null || (posttype != null && posttype.equals("1"))) { return; }
            // Unescape the HTML because the SO data is escaped.
            txt = StringEscapeUtils.unescapeHtml(txt.toLowerCase());
            link.set(getWikipediaURL(txt));
            outkey.set(row_id);
            context.write(link, outkey);
        }
}
```

Questions?

Term project proposal

- Submission: October 31 2019 By 5:00 PM via Canvas
- **Title**
- **Problem formulation**
- **Your strategy to solve the problem**
- **Your dataset**
- **Project timeline (weekly plan and roles)**
- **Bibliography**
- **Can we change our topic after the proposal?**
  - **Yes. But you should let me know before you submit your final report. Otherwise I will grade your final report based on your original proposal.**
- **Can we discuss with prof. about topic/strategy/systems?**
  - **WHY NOT??**

More info: http://www.cs.colostate.edu/~cs435/TP1.html

Datasets

- **Examples:**
  - AWS open data: https://registry.opendata.aws
  - Google public data: https://cloud.google.com/public-datasets/
  - Graph/network: http://snap.stanford.edu
  - Kaggle competition:*note that many of datasets in this site are not large enough* https://www.kaggle.com/datasets?utm_medium=paid&utm_source=google.com+search&utm_campaign=datasets&&gclid=EAIaIQobChMIg8Ttpsq35AIVARx9Ch1sDgC7EAAYASAAEgJbEfD_BwE
- **How can I explore these datasets?**
  - Narrow down your interest: Art? Music? Movie? Healthcare? Education? Environment? Food? Finance? Spam data?
  - Search if there are "good" datasets available.
  - Retrieve the dataset and see if you can use it.
    - Was it widely used?
    - Was it well-formatted?
    - Was it self-contained?

FAQs

- Do we need to use Java for the term project?
  - No. For your term project, you may use Spark or MapReduce or both with any language that the platform supports
- Do we have help session this week?
  - Yes. 4-5pm on Fridays via MS Teams
  - Laksheen will go over the PA0 on Friday (8/28)
  - Help desk will be open Tuesday 2.00 PM ~ 4.00 PM, Thursday 3.00 PM ~ 5.00 PM
- Where can I find the course information and video clips?
  - Visit your Canvas Modules!
    - We collect all the materials there

Topics to be covered

- Overview of Big Data Analytics
- Introduction to MapReduce

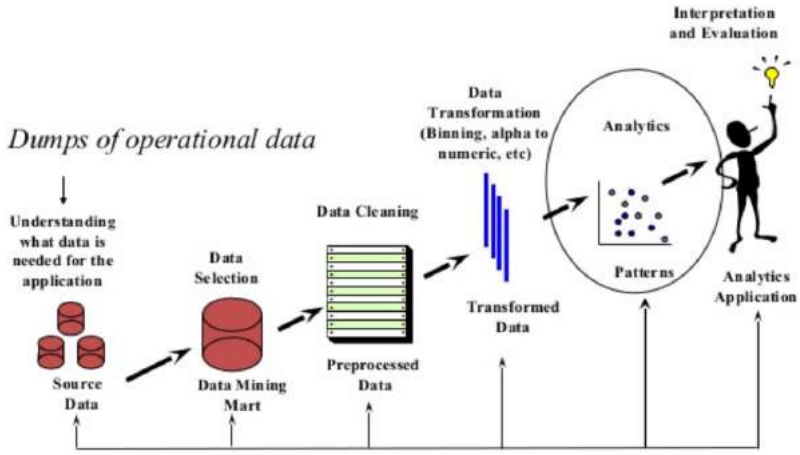
Part 1. Large Scale Data Analytics

TOPIC-1: Overview of Big Data Analytics

This Material is Built Based on,

- Analytics in a Big Data World: The Essential Guide to Data Science and Its Applications, Bart Baesens, 2014, Wiley

Analytics Process Model



The most time-consuming step is the data selection and preprocessing step
- This is usually around 80% of the total time needed to build an analytical model

Analytics in a Big Data World: The Essential Guide to Data Science and Its Applications, Bart Baesens, 2014, Wiley

Types of Analytics

- Analytics is a term that is often used interchangeably with
  - Data science
  - Data mining
  - Knowledge discovery
- **Descriptive analytics**
  - No target variable
  - e.g. Clustering, association rules
- **Predictive analytics**
  - A target variable is typically available
  - E.g. linear/logistic regression, decision trees, neural networks, support vector machines

Types of Data Sources

- **Transactions**
  - Structured, low-level, detailed information
  - Customer transactions
    - Purchase, claim, cash transfer, credit card payment
  - Stored in massive online transaction processing (OLTP) relational database
  - Can be summarized over longer time horizons (e.g. averages, relative trends, Max/Min values)
- **Unstructured data embedded in text documents**
  - emails, web pages, claim forms, audio and video, images
  - Requires extensive preprocessing
- **Qualitative, expert-based data**
  - Requires subject matter experts' (SME) analysis
  - Scientific data

Sampling

- Taking a **subset** of data for analytics
  - Generating hypothesis
  - Model selection
  - Feature selection
  - Speculative process
  - Building analytics model

- **Stratified sampling**
  - Taking samples according to predefined strata
  - e.g. Fraud detection with very skewed (99 percent non-fraud customers, 1 percent fraud customers)
    - Sample should contain the same percentage of fraud customers as in the original data

Types of Data Elements

- **Continuous**
  - Data elements that are defined on an interval that can be limited or unlimited
    - e.g. income, sales, temperature
- **Categorical Nominal**
  - Data elements that can only take on a limited set of values with no meaningful ordering between them
    - e.g. marital status, profession, purpose of loan
- **Ordinal**
  - Data elements that can only take on a limited set of values with a meaningful ordering between them
    - e.g. credit rating, age coded as young, middle age and old
- **Binary**
  - Data elements that can only take on two values
    - e.g. Having child, allowed to drive

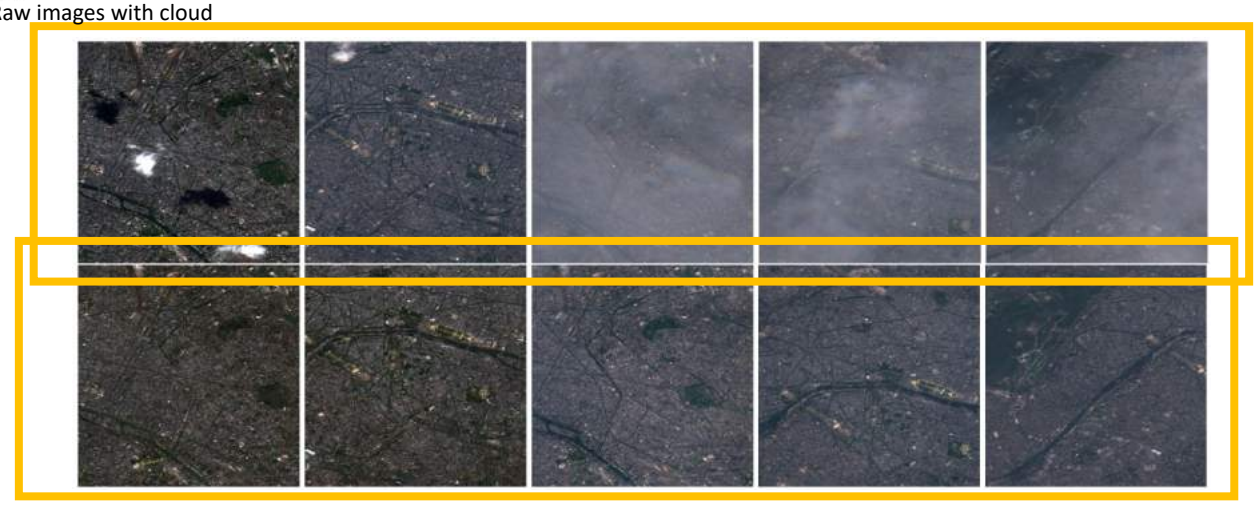
Missing Values

- **Missing values** can occur because of various reasons
  - The information can be **non-applicable**
  - The information can be **undisclosed**
  - The information can be **unavailable**

Missing Values --continued

- **Replace (impute)**
  - Replaces the missing value with a computed/selected value
  - Imputation algorithm examples
    - Hot-deck: replaces with a randomly selected similar records
    - Cold-deck: selects replacement from another dataset
    - Mean substitution: replaces with the mean of that variable for all other cases
    - Regression: predicts missing values of a variable based on other variables.
- **Delete**
  - Deletes observations with lots of missing values
  - This assumes that information is missing at random and has no meaningful interpretation and/or relationship to the target
- **Keep**
  - Missing values can be meaningful
    - e.g. a customer did not disclose the income for current condition

Raw images with cloud



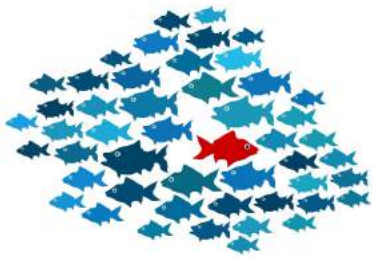
Cloud removed image using GAN

Singh, P. and Komodakis, N., 2018, July. Cloud-gan: Cloud removal for sentinel-2 imagery using a cyclic consistent generative adversarial networks. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*(pp. 1772-1775). IEEE.

68

Outliers of Dataset

- **Outliers** are extreme observations that are very dissimilar to the rest of the population
  - **Valid observation**
    - Salary of boss
  - **Invalid observation**
    - Age is 300



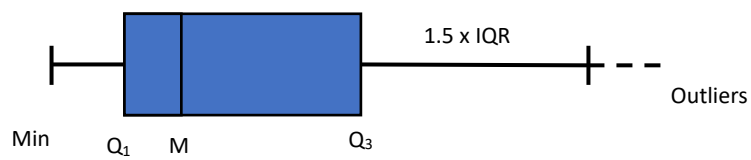
- **Multivariate outliers**
  - Observations that are outlying in multiple dimensions
    - e.g: Temperature in Fort Collins is 100 degrees but on a midnight in December

69

69

Identifying Outliers using Box Plots

- A box plot represents three key quartiles of the data
  - Q_1: 25% of the observations have a lower value
  - Q_2: 50% of the observations have a lower value
  - Q_3: 75% of the observations have a lower value
  - The **minimum** and **maximum** values are added
- **Too far away** is now quantified as more than **1.5 x Interquartile Range** ($IQR = (Q_3 - Q_1)$)



Identifying Outliers using *Z*-Score

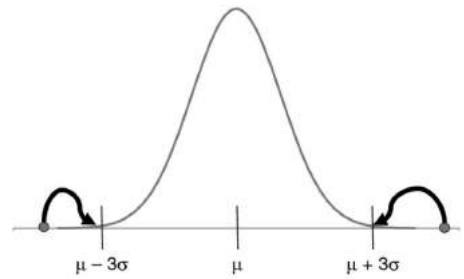
- Measuring how many standard deviations an observation is away from the mean
  - $z_i = \frac{x_i - \mu}{\sigma}$

    where μ represents the average of the variable and σ its standard deviation
- A practical rule of thumb then defines outliers when the absolute value of the z-score | z | is bigger than 3

ID	Age	Z-Score
1	30	(30-40)/10=-1
2	50	(50-40)/10=+1
3	10	(10-40)/10=-3
4	40	(40-40)/10=0
5	60	(60-40)/10=+2
6	80	(80-40)/10=+4
--	..	…
	$\mu = 40$ $\sigma = 10$	$\mu = 0$ $\sigma = 1$

Dealing with Outliers

- Treat outliers as **missing values**
- Popular schemes
  - **Truncation**
    - Taking only values that are within the limits
  - **Winsorizing**
    - Limiting extreme values to reduce the effect of possible spurious outliers
    - {92, 19, **101**, 58, **1053**, 91, 26, 78, 10, 13, **-40**, **101**, 86, 85, 15, 89, 89, 28, **-5**, 41} (N = 20, mean = 101.5)
      → {92, 19, **101**, 58, **101**, 91, 26, 78, 10, 13, **-5**, **101**, 86, 85, 15, 89, 89, 28, **-5**, 41} (N = 20, mean = 55.65)



Using the Z-Scores for truncation

Standardizing Data

- **Scaling variables to a similar range**
  - e.g. predicting credit score using two variables: education and income
  - Elementary school (1), middle school (2), high school (3), college (4), graduate school (5)
  - Income: 0 ~ $5M
  - When building logistic regression models, the coefficient for education might become very small.
- **Min/Max standardization**
  - $X_{new} = \frac{X_{old} - \min(X_{old})}{\max(X_{old}) - \min(X_{old})}(newmax - newmin) + newmin$
  - Where $newmax$ and $newmin$ are the newly imposed $maximum$ and $minimum$ (e.g. 1 and 0)

Standardizing Data. -- continued

- **Z-Score based**
  - Calculate the z-scores
- **Decimal scaling**
  - $X_{new} = \frac{X_{old}}{10^n}$
  - Dividing by a power of 10

- Standardization is useful for regression-based approaches
  - It is not needed for decision trees

Part 1. Large Scale Data Analytics

TOPIC-2: Introduction to MapReduce

COLORADO STATE UNIVERSITY | CS435 Introduction to Big Data: Week 2-A | Fall 2021Computer Science

Big Data Analytics
-Big Data Technology Stack

76

76

77

In a nutshell

Security and Governance

Data Presentation Layer
Apache Kibana

Data Integration Layer
Apache Flume, Apache Kafka, Apache Sqoop

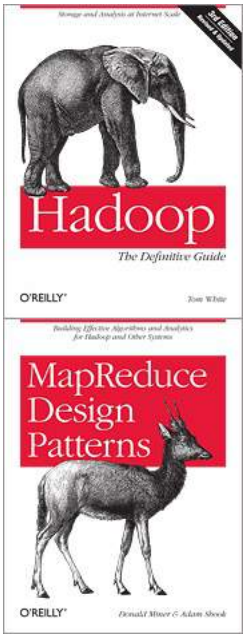
Operations and Scheduling Layer
Apache Ambari
Apache Oozie
Apache Zookeeper

Data Processing Layer
Apache Hadoop MapReduce, Pig, Apache Spark, Cassandra, Storm, Mahout, MLLib, TensorFlow, PyTorch

Data Layer
Apache HDFS, Amazon AWS's S3, IBM GPFS, Microsoft Azure

This material is developed based on,

- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012 --Chapter 2
  - Download this chapter from the CS435 schedule page
- Hadoop: The definitive Guide, Tom White, O'Reilly, 3rd Edition, 2014
- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013



What is MapReduce?

MapReduce [1/2]

- **MapReduce** is **inspired** by the concepts of *map* and *reduce* in Lisp.
- ***"Modern"* MapReduce**
  - Developed within Google as a mechanism for processing large amounts of raw data.
    - Crawled documents or web request logs
    - Distributes these data across thousands of machines
    - **Same computations** are performed on each CPU with different dataset

Sorting 1PB of numbers with a single machine

- What will be the challenges?

Sorting 1PB of numbers with multiple machines

- What will be the requirements to perform this sorting?

MapReduce [2/2]

- **MapReduce** provides an abstraction that allows engineers to perform simple computations while hiding the details of **parallelization**, **data distribution**, **load balancing** and **fault tolerance**

Mapper

- **Mapper** maps **input key/value pairs** to a set of **intermediate key/value pairs**
  - Maps are the individual tasks that transform input records into intermediate records
  - The transformed intermediate records **do not need to be of the same type** as the input records
  - A given input pair may map to **zero or many** output pairs
  - The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job

Reducer

- Reducer reduces **a set of intermediate values which share a key** to a smaller set of values
- Reducer has 3 primary phases
  - Shuffle, sort and reduce
- **Shuffle**
  - Input to the reducer is the sorted output of the mappers
  - The framework fetches the relevant partition of the output of all the mappers via HTTP
- **Sort**
  - The framework groups input to the reducer by keys

MapReduce 101 with an Example

Example 1: WordCount [1/5]

- For text files stored under `usr/joe/wordcount/input`, count the number of occurrences of each word
- How do files and directory look?

```
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/
/usr/joe/wordcount/input/file01
/usr/joe/wordcount/input/file02

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World, Bye World!

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop, Goodbye to hadoop.
```

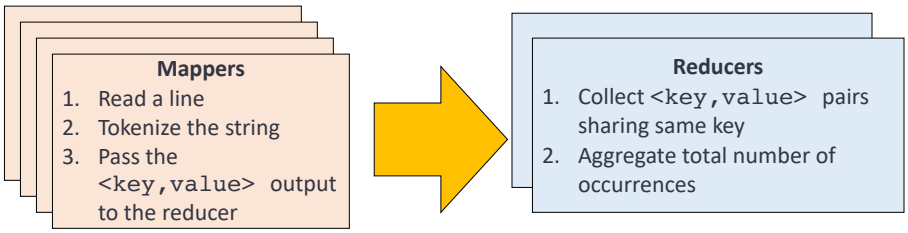

Example 1: WordCount [2/5]

- Run the MapReduce application

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount
/usr/joe/wordcount/input /usr/joe/wordcount/output


$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop, 1
Hello 2
World! 1
World, 1
hadoop. 1
to 1
```

Example 1: WordCount [3/5]



What do you have to pass from the Mappers?

Example 1: WordCount [4/5]

```
public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
           while (tokenizer.hasMoreTokens()) {
                 word.set(tokenizer.nextToken());
                 context.write(word, one);
            }
  }
}
```

Example 1: WordCount [5/5]

```
public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable> {

       public void reduce(Text key, Iterable<IntWritable> values,
Context context)
         throws IOException, InterruptedException {
           int sum = 0;
           for (IntWritable val : values) {
               sum += val.get();
           }
           context.write(key, new IntWritable(sum));
       }
    }
```


Questions?

WEEK 2- LECTURE 2

MAPREDUCE DESIGN PATTERNS 1: SUMMARIZATION PATTERN

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

- Programming Assignment 1 will be posted tomorrow (9/2)
  - We will discuss PA0/PA1 in the recitation (Sept 3)
  - Discussion in the MS TEAMs page – please join if you have any question!
- Attendance bonus score
  - If you attend,
  - Greater than or equal to 80% of lecture sessions: 2 points
  - Greater than 70% of lecture sessions: 1.5 points
  - Greater than 60% of the lecture sessions: 1 point
  - I will use the Echo360 report for the online students
- **MapReduce to compute average (with/without combiner)**

Topics discussed in the previous class

- Topic 1: Introduction to MapReduce: example 2
  - NCDC data: finding max temperature per year
  - Analysis of the network log file
- Topic 2: MapReduce Design Pattern I. Summarization Patterns
  - Finding Max/Min/sum?
  - Average
  - Combiner

Topics to be covered today

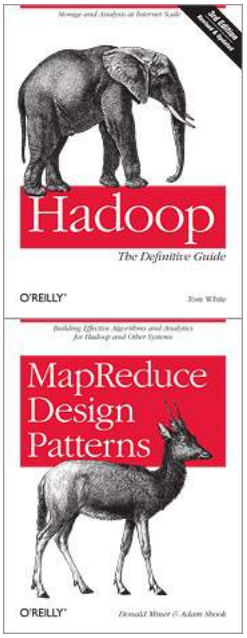
- Topic 1: MapReduce Design Pattern I. Summarization Patterns
- Topic 2: MapReduce Design Pattern II. Filter Patterns

Part 1. Large Scale Data Analytics

Design Pattern 1: Summarization Patterns

This material is developed based on,

- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012 --Chapter 2
  - Download this chapter from the CS435 schedule page
- Hadoop: The definitive Guide, Tom White, O'Reilly, 3rd Edition, 2014
- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013



Design Pattern 1: Summarization Patterns

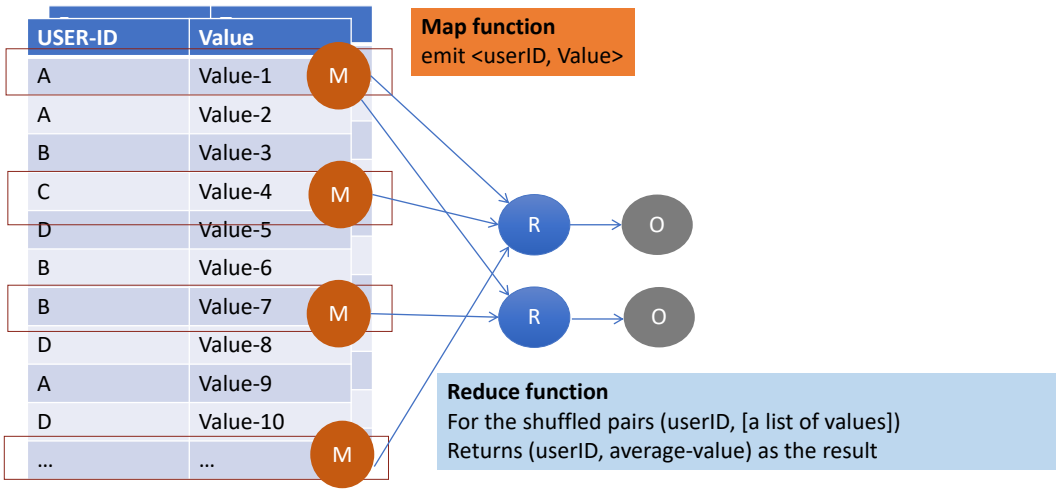
1. Numerical Summarizations

REVIEW

Numerical Summarization Example 2. Average

- Calculating an average of ***Values*** per user

USER-ID	Value
A	Value-1
A	Value-2
B	Value-3
C	Value-4
D	Value-5
B	Value-6
B	Value-7
D	Value-8
A	Value-9
D	Value-10
…	…



REVIEW

Hadoop Combiner

- **Minimizes the data transferred** between map and reduce tasks
- Users can specify a **combiner function**
  - To be run on the map output
  - To replace the map output with the combiner output
- Hadoop does **NOT** guarantee how many times it will call combiner for a particular map output record
- The function should be **cumulative** and **associative**

REVIEW

Example: Find the maximum temperature

Without Combiner

- First map produces
  (1950, 0)
  (1950, 20)
  (1950, 10)
- Second map produces
  (1950, 25)
  (1950, 15)
- Input to the reduce function
  (1950, **[0, 20, 10, 25, 15]**)
- Output
  (1950, 25)

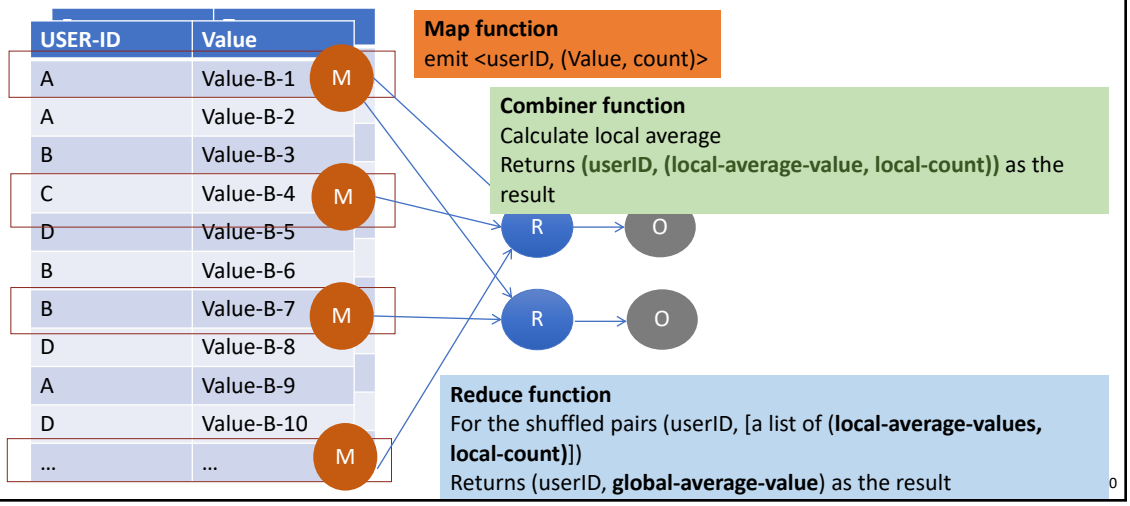
With Combiner

- First map produces
  (1950, 0)
  (1950, 20)
  (1950, 10)
  → **(1950, 20)**
- Second map produces
  (1950, 25)
  (1950, 15)
  → **(1950, 25)**
- Input to the reduce function
  (1950, [0, 20, 10, 25, 15])
  →**(1950, [20, 25])**
- Output
  (1950, 25)→(1950, 25)

REVIEW

Numerical Summarization Example 2. Average

- Calculating the Average of ***Values*** per user



Worksheet #2-2: Question 1

Assume that you analyze a set of e-commerce transaction records with n users. Each user has at least m number of transaction records that contains `date/time`, `amount`, and `productID`. Suppose you calculate average of the amount of transactions per user using MapReduce with M mappers and R reducers.

What will be the total number of records (e.g. the `amount` attribute) that is delivered to the reducers in your MepReduce cluster? In this computation, we **do not** use combiner.

a. Approximately m
b. Approximately n x M
c. Approximately m x n
d. At least m

Answer: C

Worksheet #2-2: Question 2

Assume that you analyze a set of e-commerce transaction records with n users. Each user has at least m number of transaction records that contains `date/time`, `amount`, and `productID`. Suppose you calculate average of the amount of transactions per user using MapReduce with M mappers and R reducers.

What will be the closest total number of records (e.g. the `amount` attribute) that is delivered to the reducers in your MepReduce cluster? In this computation, we use combiner. Assume that the data is sorted based on the transaction time.

a. Approximately m
b. Approximately n x M
c. Approximately m x n
d. Approximately M

Answer: B

Design Pattern 1: Summarization Patterns

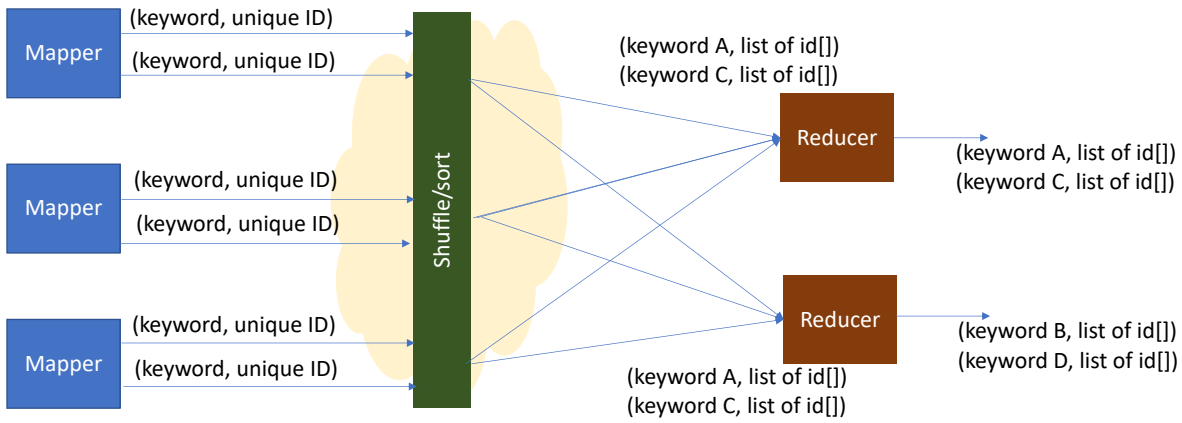
Inverted Index

Inverted Index

- Generate Index from a data dataset to map from **contents**, such as words or numbers
  - Reduces the amount of time to find related items
  - Keyword based search, Web search, and document search
  - e.g. Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment

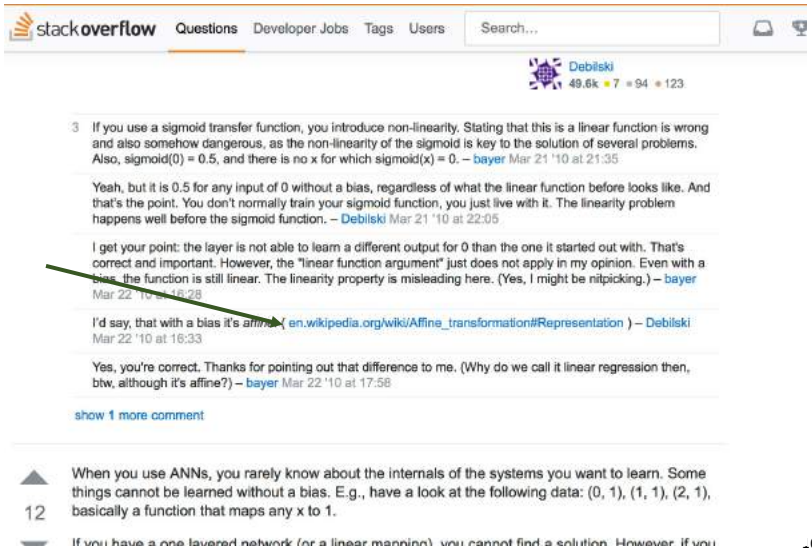
*** Example of an inverted index for document 1 and 2 (D1, and D2)**
D1= "Colorado State University"
{Colorado, State, University}
D2= "University of Colorado"
{University, of, Colorado}
Inverted Index
Colorado – {D1, D2}
State - {D1}
University – {D1, D2}
of – {D2}

Structure



Inverted index of StackOverflow links to Wikipedia

Adding StackOverflow links to each Wikipedia page that is reference in a StackOverflow comment



16

Inverted index of StackOverflow links to Wikipedia

CommentID	Comment
A	Link-1
B	Link-2
C	Link-3
D	Link-4
E	Link-5
F	Link-6
G	Link-7
H	Link-8
I	Link-9
J	Link-10
…	…

Map function
For each record, Map function produces the key-value pair (key, value)

1. Design your Map function
- input, output, functionality

2. Design your Reduce function
- input, output, functionality

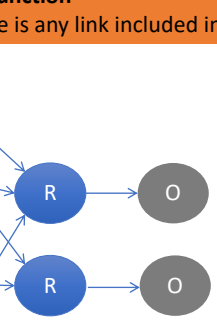
Reduce function
For the shuffled pairs (key, [a list of values])
Returns (key, value) as the result

17

17

Inverted index of StackOverflow links to Wikipedia

CommentID	Comment
A	Link-1
B	Link-2
C	Link-3
D	Link-4
E	Link-5
F	Link-6
G	Link-7
H	Link-8
I	Link-9
J	Link-10
…	…



Map function
If there is any link included in "comment" emit <link, commentID>

Reduce function
All the commentIDs that referred same wikipedia page will be grouped together

Mapper Code

```
public static class WikipediaExtractor extends Mapper < Object, Text, Text, Text > {
        private Text link = new Text();
        private Text outkey = new Text();
        public void map(Object key, Text value, Context context)
                                        throws IOException, InterruptedException {
            Map <String, String> parsed = MRDPUtils.transformXmlToMap(value .toString());
            // Grab the necessary XML attributes
            String txt = parsed.get("Body");
            String posttype = parsed.get("PostTypeId");
            String row_id = parsed.get("Id");
            // if the body is null, or the post is a question (1), skip
            if (txt == null || (posttype != null && posttype.equals("1"))) { return; }
            // Unescape the HTML because the SO data is escaped.
            txt = StringEscapeUtils.unescapeHtml(txt.toLowerCase());
            link.set(getWikipediaURL(txt));
            outkey.set(row_id);
            context.write(link, outkey);
        }
}
```

Worksheet #2-2: Question 3

- What should be in your Reducer?

a. Mechanism to access the underlying file system and store output
b. Process to group all different CommentIDs
c. Process to merging the results
d. Nothing to do

Map-Only Job in MapReduce

- Some MR jobs do not require reducer(s)
- To configure a job without reducer
  - Set `job.setNumreduceTasks(0)` in the configuration in a driver

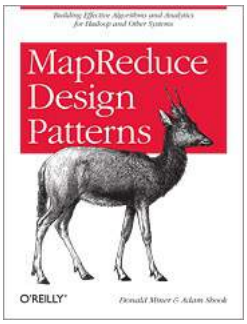
- **Advantages of using Map-Only job**
  - Between map and reduce
    - Sort and Shuffle
      - Expensive computation
  - The Map-only jobs can avoid sort and shuffle
  - Output of mapper is written to local disk
    - Your mapper can write output to the HPDF files directly

Part 1. Large Scale Data Analytics

Design Pattern 2: Filtering Patterns

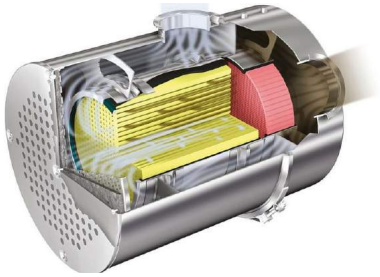
This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012



Filtering pattern

- Providing an **abstract** of existing data
- Many data filtering do **NOT** require the "`reduce`" part of MapReduce
  - **It does not produce an aggregation**
- Known uses
  - Tracking a thread of events
  - Distributed grep
  - Data cleaning
  - Closer view of data
  - Simple random sampling
  - Removing low scoring data



COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 2-B Fall 2021Computer Science

Filtering patterns covered in this class

1. Simple Random Sampling
2. Bloom filter
3. Top 10
4. Distinct

MapReduce Design Patterns II: Filtering Patterns

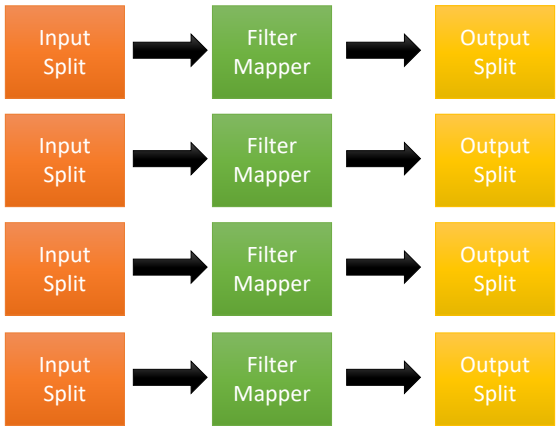
1. Simple Random Sampling

Filtering Pattern 1.

Simple Random Sampling

- Each record has an **equal probability** of being selected
- Useful for **sizing down** a data set
  - For representative analysis

The structure of the simple filter pattern



Writing a Simple Random Sampling filter

```
public static class SRSMapper
  extends Mapper < Object, Text, NullWritable, Text > {
    private Random rands = new Random();
    private Double percentage;
    protected void setup(Context context) throws IOException, InterruptedException {
    // Retrieve the percentage that is passed in via the configuration
    // like this: conf.set(" filter_percentage", .5);
    // for .5%
    String strPercentage = context.getConfiguration().get("filter_percentage");
    percentage = Double.parseDouble(strPercentage) / 100.0;
    }

    public void map( Object key, Text value, Context context) throws
  IOException, InterruptedException {
        if (rands.nextDouble() < percentage) {
           context.write(NullWritable.get(), value); // otherwise, drop it.
        }
    }
}
```

Stratified Sampling with MapReduce

- Implementing the Stratified Sampling
- Proportion allocation
  - The size of portion in each stratum is taken in proportion to the size of the stratum
- e.g. Sampling from Olympic athletes per event
  - E.g. 11,656 athletes participated in Tokyo Olympic, 2021.

1. Stratification of data
   Collectively exhaustive and mutually exclusive
   Every element in the population must be assigned to one and only one stratum
2. Apply Simple Random sampling within each stratum

MapReduce Design Patterns II: Filtering Patterns

2. Bloom Filter

Filtering Pattern 2.
Bloom Filter

- Checking the **membership** of a set
- Known uses
  - Removing most of the non-membership values
  - Prefiltering a data set for an expensive set membership check

What is a Bloom Filter?

- Burton Howard Bloom in 1970
- **Probabilistic data structure** used to test **whether a member is an element** of a set
- **Strong space advantage**

Building a Bloom filter

- m
  - The number of bits in the filter
- n
  - The number of members in the set
- p
  - The desired false positive rate
- k
  - The number of different hash functions used to map some element to one of the m bits with a uniform random distribution

Building a Bloom filter

- $m = 8$
  - The number of bits in the filter
- $n = 3$
  - The number of members in the set T={5, 10,15}
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$

Initial bloom filter	0	0	0	0	0	0	0	0

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(5) = 7$, $h2(5) = 5$, $h3(5) = 5$

7th bit 0th bit

Initial bloom filter	0	0	0	0	0	0	0	0
After h1(5) = 7	1	0	0	0	0	0	0	0
After h2(5) = 5	1	0	1	0	0	0	0	0
After h3(5) = 5	1	0	1	0	0	0	0	0

36

36

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(10) = 6$, $h2(10) = 7$, $h3(10) = 2$

7th bit 0th bit

After encoding 5	1	0	1	0	0	0	0	0
After h1(10) = 6	1	1	1	0	0	0	0	0
After h2(10) = 7	1	1	1	0	0	0	0	0
After h3(10) = 2	1	1	1	0	0	1	0	0

37

37

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(15) = 6$, $h2(15) = 1$, $h3(15) = 7$

7th bit 0th bit

After encoding 5 and 10	1	1	1	0	0	1	0	0

Question

What will be the bloom filter after encoding an integer 15? Use the current Bloom filter discussed in this slide.

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 2-B Fall 2021Computer Science

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(15) = 6$, $h2(15) = 1$, $h3(15) = 7$

7th bit 0th bit

After encoding 5 and 10	1	1	1	0	0	1	0	0

After h1(15) = 5	1	1	1	0	0	1	0	0
After h2(15) = 1	1	1	1	0	0	1	1	0
After h3(15) = 7	1	1	1	0	0	1	1	0

Applying a Bloom filter

- Is 5 part of set T?
- *h1(5), h2(5), h3(5)'th bits are 1*
  - *5 is **probably** a part of set T*

h1(x) = 3x mod 8
h2(x) = (2x +3) mod 8
h3(x) = x mod 8

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(5) = 7	1	1	1	0	0	1	1	0
Check h2(5) = 5	1	1	1	0	0	1	1	0
Check h3(5) = 5	1	1	1	0	0	1	1	0

Applying a Bloom filter

- Is 8 part of set T?
- *h1(8), h2(8), h3(8)*
- *8 is **NOT** a part of set T*

h1(x) = 3x mod 8
h2(x) = (2x +3) mod 8
h3(x) = x mod 8

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(8) = 0	1	1	1	0	0	1	1	0
Check h2(8) = 3	1	1	1	0	0	1	1	0
Check h3(8) = 0	1	1	1	0	0	1	1	0

Applying a Bloom filter

Question

- Is 9 part of set T?
  a. Yes
  b. No

$h1(x) = 3x \bmod 8$
$h2(x) = (2x + 3) \bmod 8$
$h3(x) = x \bmod 8$

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0

Applying a Bloom filter

- Is 9 part of set T?
- $h1(9), h2(9), h3(9)$
- *9 is **NOT** a part of set T*

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(9) = 3	1	1	1	0	0	1	1	0
Check h2(9) = 5	1	1	1	0	0	1	1	0
Check h3(9) = 1	1	1	1	0	0	1	1	0

Applying a Bloom filter

Question

- Is 7 part of set T?
  a. Yes
  b. No

$h1(x) = 3x \bmod 8$
$h2(x) = (2x + 3) \bmod 8$
$h3(x) = x \bmod 8$

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0

Applying a Bloom filter

- Is 7 part of set T?
- *h1(7), h2(7), h3(7)'th bits are 1*
  - *7 is **probably** a part of set T*

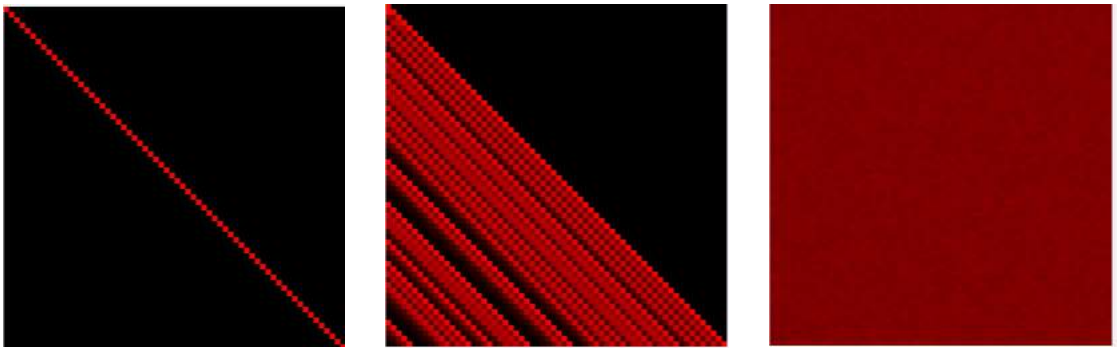
After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(7) = 7	1	1	1	0	0	1	1	0
Check h2(7) = 1	1	1	1	0	0	1	1	0
Check h3(7) = 7	1	1	1	0	0	1	1	0

Hash functions

- k Hash functions
  - Uniform random distribution in [1..m)
- Cryptographic hash functions
  - MD5, SHA-1, SHA-256, Tiger, Whirlpool..
- Murmur Hashes (non-cryptographic), Jenkins, Java hashCode(), Spooky, CityHash

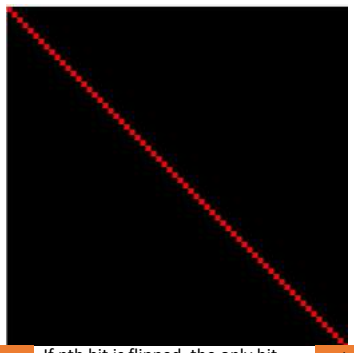
Evaluating Hash Functions: Avalanche Effect

- Each bit should have **50% chances** to change if you **change 1 bit** of the input
- Avalanche Diagram
  - A grid such that the (x,y) cell's color represents the probability that flipping x'th bit of input will result of yth bit being flipped in the output

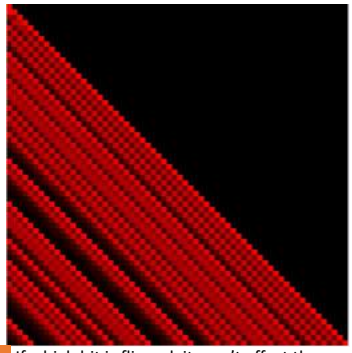


Avalanche Effect

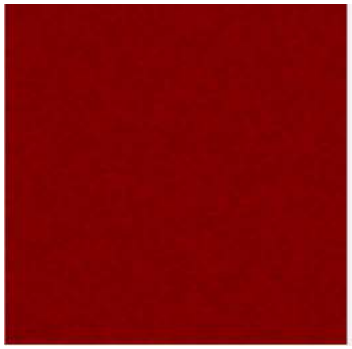
- Each bit should have **50% chances** to change if you **change 1 bit** of the input
- Avalanche Diagram
  - A grid such that the (x,y) cell's color represents the probability that flipping x'th bit of input will result of yth bit being flipped in the output



n^{th} bit — If nth bit is flipped, the only bit flipped in the output will be n^{th} bit



0th bit — If a high bit is flipped, it won't affect the lower bits.



Perfect bit independence

False positive rate (1/2)

$$fpr = (1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-kn/m})^k$$

m=number of bits in the filter
n=number of elements
k=number of hashing functions

False positive rate (2/2)

- A bloom filter with an optimal value for k and 1% error rate only needs **9.6** bits per key.
- Add **4.8 bits/key** and the error rate decreases by **10 times**
- 10,000 words with 1% error rate and 7 hash functions
  - **~12KB of memory**
- 10,000 words with 0.1% error rate and 11 hash functions
  - **~18KB of memory**

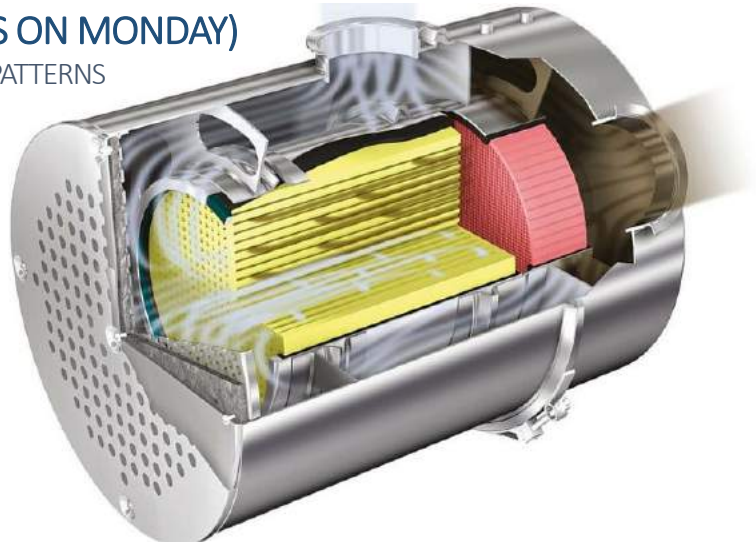
How big should I make my Bloom Filter?

- Try various values of **k and m**
  - To achieve target false–positive rate ($(1-e^{-kn/m})^k$)
- **Then, how many hash functions should I use?**
  - The **more hash functions** you have
    - the **slower** your bloom filter
    - The quicker it fills up
  - If you have **few hash functions**
    - **Too many false positives**
  - Given an m and an n, the **optimal value of k**
    - $(m/n)ln(2)$

Questions?

WEEK 3- LECTURE 2 (NO CLASS ON MONDAY)

MAPREDUCE DESIGN PATTERN 2: FILTER PATTERNS



Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

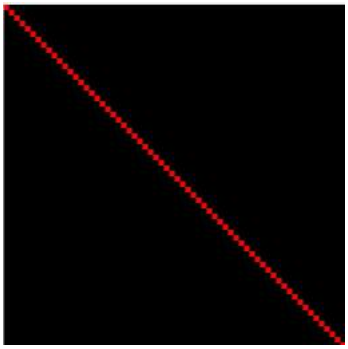
- Avalanche diagram
- Term project

Non-Cryptographic Hash function

- What is the hash function?
  - Any function that can be used to map a value to an arbitrary size to fixed-size value
  - Properties
    - Uniformity
    - A good hash function should map the expected inputs as evenly as possible over its output range
    - E.g. Hashtable, Hashmap
- What is the "cryptographic" hash function?
  - Properties
    - Pre-image resistance, Second pre-image resistance, collision resistance
- OK. Then, what is the "Non-Cryptographic Hash function?

Evaluating Hash Functions: Avalanche Effect

- Each bit should have **50% chances** to change if you **change 1 bit** of the input
- Avalanche Diagram
  - A grid such that the (x,y) cell's color represents the probability that flipping x'th bit of input will result of yth bit being flipped in the output (e.g. bright red: high probability)

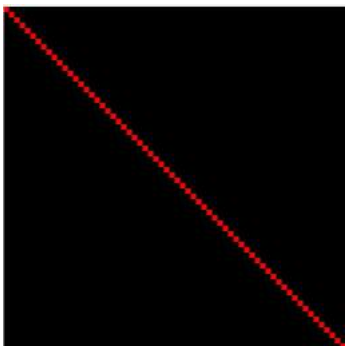


n^{th} bit

0th bit

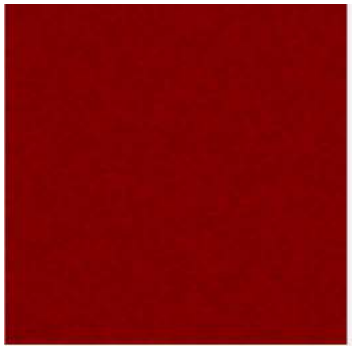
Evaluating Hash Functions: Avalanche Effect

- Each bit should have **50% chances** to change if you **change 1 bit** of the input
- Avalanche Diagram
  - A grid such that the (x,y) cell's color represents the probability that flipping x'th bit of input will result of yth bit being flipped in the output (e.g. bright red: high probability)



n^{th} bit

0th bit



Topics discussed in the previous class

- Filtering patterns
  - Simple Random Sampling
  - Bloom Filter

Topics to be covered

- Topic 1: Term Project planning and exploring topics and datasets
- Topic 2: MapReduce Design Pattern II. Filter Patterns

Upcoming deliverable: Term project proposal

- **Title**
- **Problem formulation**
- **Your strategy to solve the problem**
- **Your dataset**
- **Project timeline (weekly plan and roles)**
- **Bibliography**
- **Can we change our topic after the proposal?**
  - **Yes. But you should consult with me before you submit your final report. Otherwise I will grade your final report based on your original proposal.**
- **Can we discuss with prof. about topic/strategy/systems?**
  - **WHY NOT??**

Datasets

- **Examples:**
  - Sustain data: https://urban-sustain.org/. https://urban-sustain.org/documents/videoTutorials.php
  - AWS open data: https://registry.opendata.aws
  - Google public data: https://cloud.google.com/public-datasets/
  - Graph/network: http://snap.stanford.edu
  - Kaggle competition:*note that many of datasets in this site are not large enough* https://www.kaggle.com/datasets?utm_medium=paid&utm_source=google.com+search&utm_campaign=datasets&&gclid=EAIaIQobChMIg8Ttpsq35AIVARx9Ch1sDgC7EAAYASAAEgJbEfD_BwE
  - SNAP: http://snap.stanford.edu
- **How can I explore these datasets?**
  - Narrow down your interest: Art? Music? Movie? Healthcare? Education? Environment? Food? Finance? Spam data?
  - Search if there are "good" datasets available. Consider the missing data rate, attributes, density, and coverage
  - Retrieve the dataset and see if you can use it.
    - Was it widely used?
    - Was it well-formatted?
    - Was it self-contained?

How to be a successful team member

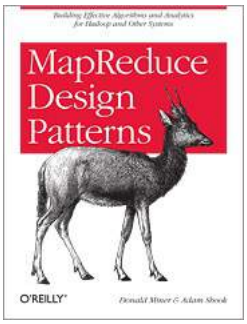
- **Your participation within your team will be assessed by your team members**
- Participation
  1. Not active for the task
  2. Actively participated
  3. Significant contribution
  4. Taking initiative
- Task implementation
  - Achieving the individual goals successfully within the time frame that the team agreed and initiates improvements
- Communication
  - Comments always insightful and constructive. Comments balanced between general impression, opinions and specific, thoughtful criticisms or contributions

Part 1. Large Scale Data Analytics

Design Pattern 2: Filtering Patterns

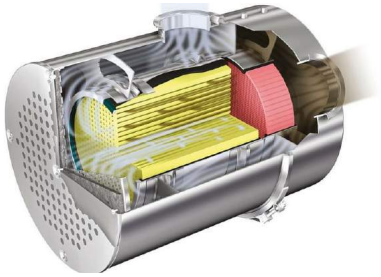
This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012



Filtering pattern

- Providing an **abstract** of existing data
- Many data filtering do **NOT** require the "`reduce`" part of MapReduce
  - **It does not produce an aggregation**
- Known uses
  - Tracking a thread of events
  - Distributed grep
  - Data cleaning
  - Closer view of data
  - Simple random sampling
  - Removing low scoring data



COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 3-B Fall 2021Computer Science

Filtering patterns covered in this class

1. Simple Random Sampling
2. Bloom filter
3. Top 10
4. Distinct

MapReduce Design Patterns II: Filtering Patterns

2. Bloom Filter

Building a Bloom filter

- m
  - The number of bits in the filter
- n
  - The number of members in the set
- p
  - The desired false positive rate
- k
  - The number of different hash functions used to map some element to one of the m bits with a uniform random distribution

Building a Bloom filter

- $m = 8$
  - The number of bits in the filter
- $n = 3$
  - The number of members in the set T={5, 10,15}
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$

Initial bloom filter	0	0	0	0	0	0	0	0

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(5) = 7$, $h2(5) = 5$, $h3(5) = 5$

7th bit 0th bit

Initial bloom filter	0	0	0	0	0	0	0	0
After h1(5) = 7	1	0	0	0	0	0	0	0
After h2(5) = 5	1	0	1	0	0	0	0	0
After h3(5) = 5	1	0	1	0	0	0	0	0

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(10) = 6$, $h2(10) = 7$, $h3(10) = 2$

7th bit 0th bit

After encoding 5	1	0	1	0	0	0	0	0
After h1(10) = 6	1	1	1	0	0	0	0	0
After h2(10) = 7	1	1	1	0	0	0	0	0
After h3(10) = 2	1	1	1	0	0	1	0	0

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{ 5, 10, 15 \}$
- $k = 3$
  - $h1(x) = 3x \bmod 8$
  - $h2(x) = (2x + 3) \bmod 8$
  - $h3(x) = x \bmod 8$
  - $h1(15) = 6$, $h2(15) = 1$, $h3(15) = 7$

7th bit 0th bit

After encoding 5 and 10	1	1	1	0	0	1	0	0
After h1(15) = 5	1	1	1	0	0	1	0	0
After h2(15) = 1	1	1	1	0	0	1	1	0
After h3(15) = 7	1	1	1	0	0	1	1	0

Applying a Bloom filter

- Is 5 part of set *T*?
- *h1(5), h2(5), h3(5)'th bits are 1*
  - *5 is **probably** a part of set T*

h1(x) = 3x mod 8
h2(x) = (2x +3) mod 8
h3(x) = x mod 8

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(5) = 7	1	1	1	0	0	1	1	0
Check h2(5) = 5	1	1	1	0	0	1	1	0
Check h3(5) = 5	1	1	1	0	0	1	1	0

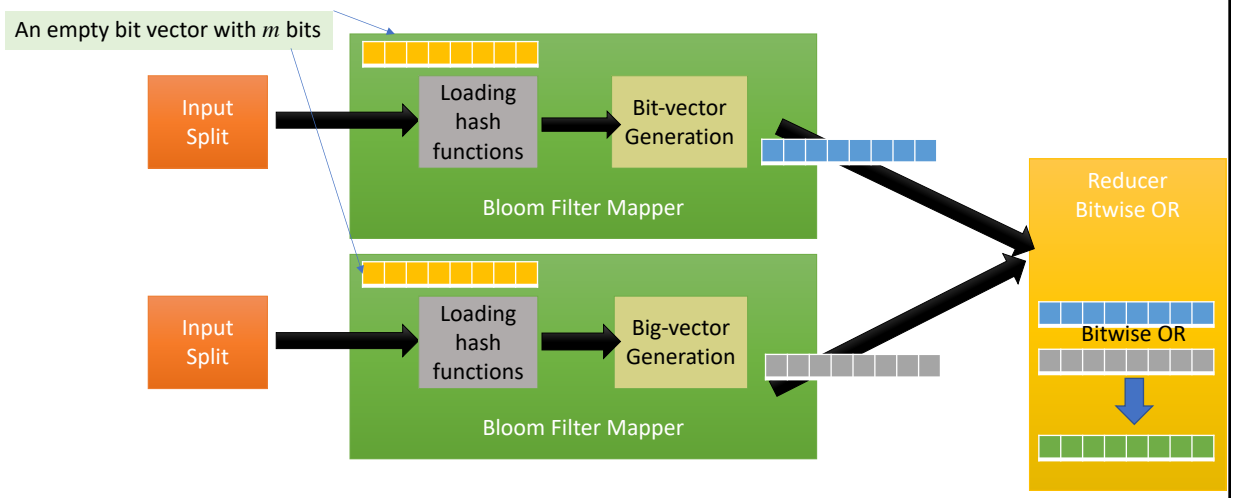
Applying a Bloom filter

- Is 8 part of set *T*?
- *h1(8), h2(8), h3(8)*
- *8 is **NOT** a part of set T*

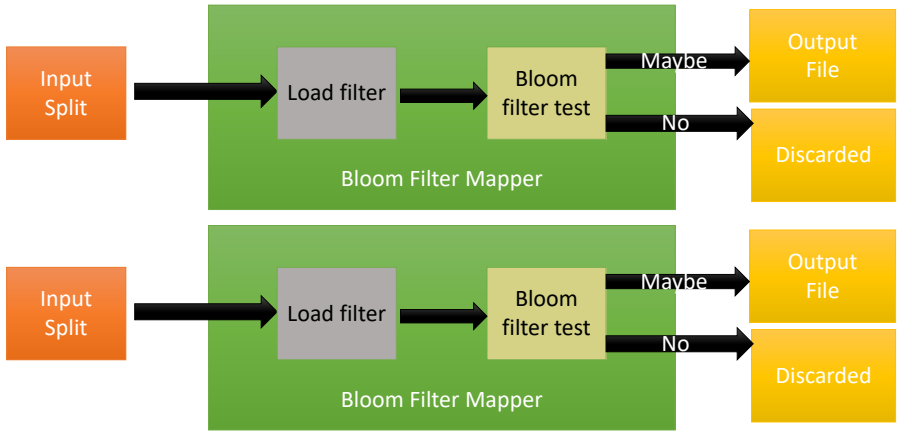
h1(x) = 3x mod 8
h2(x) = (2x +3) mod 8
h3(x) = x mod 8

After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(8) = 0	1	1	1	0	0	1	1	0
Check h2(8) = 3	1	1	1	0	0	1	1	0
Check h3(8) = 0	1	1	1	0	0	1	1	0

Building Bloom Filter with MapReduce



Running Bloom Filter with MapReduce (to filter out any negative membership)



Bloom Filtering mapper (checking) [1/2]

```
public static class BloomFilteringMapper extends Mapper < Object, Text, Text,
NullWritable > {
      private BloomFilter filter = new BloomFilter();
      protected void setup(Context context) throws
      IOException, InterruptedException
      {
        // Get file from the DistributedCache
        URI[] files=DistributedCache.getCacheFiles(
             context.getConfiguration());
        System.out.println(
             "Reading Bloom filter from: "
                   + files[0].getPath());
        // Open local file for read.
        DataInputStream strm =
      new DataInputStream(new FileInputStream(files[0].getPath()));
        // Read into our Bloom filter.
        filter.readFields(strm);
        strm.close();
      }
```

HDFS's Distributed Cache?
MapReduce framework copies read-only files or archives or jar files to the worker nodes, before the execution of any tasks for the job on that node

Bloom Filtering mapper (Checking) [2/2]

```
   public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
      Map < String, String > parsed=transformXmlToMap(value.toString());
      // Get the value for the comment
      String comment = parsed.get("Text");
      StringTokenizer tokenizer = new StringTokenizer(comment);
      // For each word in the comment
      while (tokenizer.hasMoreTokens()) {
            // If the word is in the filter,
            // output the record and break
            String word = tokenizer.nextToken();
            if (filter.membershipTest(new Key(word.getBytes())))
            {
                context.write(value, NullWritable.get());
                break;
            }
      }
   }     }
```

MapReduce Design Patterns II: Filtering Patterns

3. Top 10 (Top K)

Filtering Pattern 3. `Top 10`

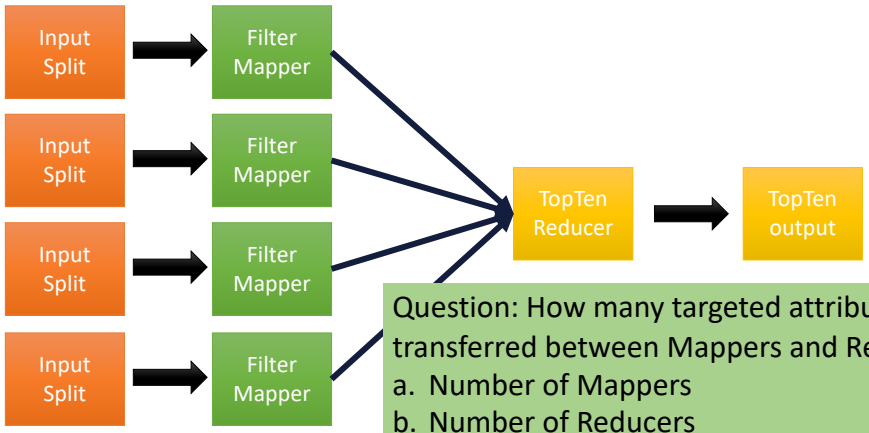
- Retrieves a **relatively small number (top K)** of records, according to a ranking scheme in your dataset, no matter how large the data

- E.g. generate top 10 clients
- E.g. generate top 10 clients per zip code ..
- E.g. generate top 10 clients per day ..
- E.g. generate top 10 clients per age ..

- Known uses
  - Outlier analysis
  - Selecting interesting data
  - Data summarization
  - Catchy dashboards

The structure of Non Top10 pattern

e.g. What are the top 10 best selling albums of 2021?



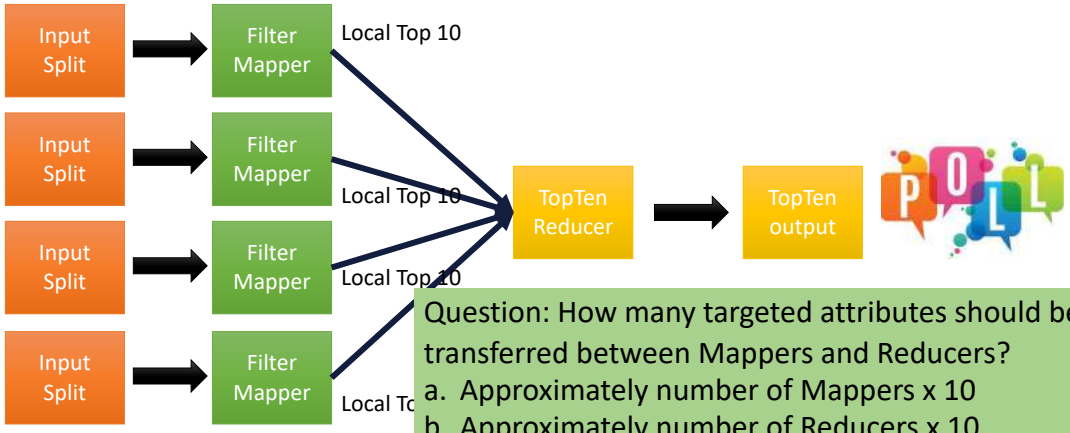
Question: How many targeted attributes should be transferred between Mappers and Reducers?

a. Number of Mappers
b. Number of Reducers
c. 1
d. Number of records

28

The structure of Top 10 pattern with a single group

e.g. What are the top 10 best selling albums of 2021?

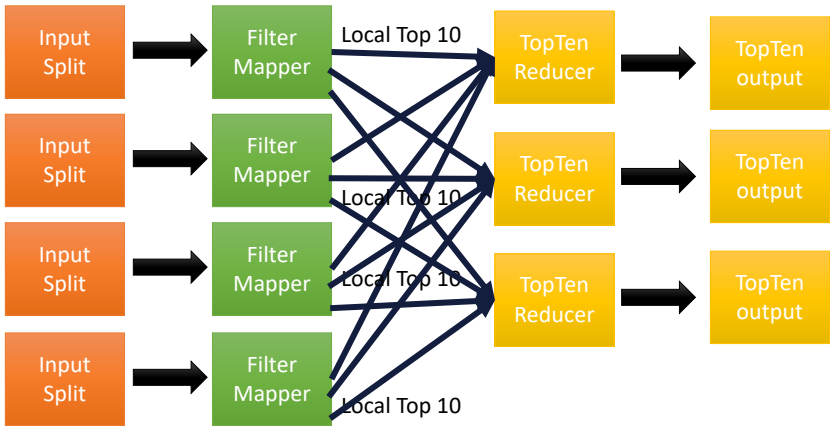


Question: How many targeted attributes should be transferred between Mappers and Reducers?

a. Approximately number of Mappers x 10
b. Approximately number of Reducers x 10
c. 1
d. Approximately number of records

29

The structure of Top 10 pattern with multiple groups



Mapper [1/2]

```
public static class TopTenMapper extends Mapper < Object, Text, NullWritable, Text > {
   // Stores a map of user reputation to the record
   private TreeMap < Integer, Text >
       repToRecordMap = new TreeMap < Integer, Text >();
   public void map( Object key, Text value, Context context) throws IOException, InterruptedException {
      Map < String, String > parsed = transformXmlToMap( value.toString());
      String userId = parsed.get("Id");
      String reputation = parsed.get("Reputation");
      // Add this record to our map with the reputation as the key
      repToRecordMap.put(Integer.parseInt(reputation), new Text(value));
      // If we have more than ten records, remove the one with the lowest rep
      // As this tree map is sorted in descending order, the user with
      // the lowest reputation is the last key.
      if (repToRecordMap.size() > 10) {
         repToRecordMap.remove(repToRecordMap.firstKey());
      }
```

Mapper [2/2]

```
  protected void cleanup(Context context)
              throws IOException, InterruptedException {
        // Output our ten records to the reducers with a null key
        for (Text t : repToRecordMap.values()) {
              context.write(NullWritable.get(), t);
        }
    }
}
```

Note: `setup()` and `cleanup()` are called for each Mapper and Reducer. So, if there Are 20 mappers running (10,000 inputs each), the setup/cleanup will be called 20 times.

Example:

```
public void run(Context context) throws IOException, InterruptedException {
     setup(context);
     try { while (context.nextKey()) {
            reduce(context.getCurrentKey(), context.getValues(), context); } }
     finally { cleanup(context); }
}
```


Reducer

```
public static class TopTenReducer extends Reducer < NullWritable, Text, NullWritable, Text > {
  // Stores a map of user reputation to the record
  // Overloads the comparator to order the reputations in descending order
  private TreeMap < Integer, Text > repToRecordMap = new TreeMap < Integer, Text >();
  public void reduce(NullWritable key, Iterable < Text > values, Context context) throws IOException,
InterruptedException {
      for (Text value : values) {
         Map < String, String > parsed = transformXmlToMap(value.toString());
        repToRecordMap.put(Integer.parseInt( parsed.get("Reputation")), new Text( value));
        // If we have more than ten records, remove the one with the lowest rep
        // As this tree map is sorted in descending order, the user with
        // the lowest reputation is the last key.
        if (repToRecordMap. size() > 10) {
            repToRecordMap.remove(repToRecordMap.firstKey());
         }
      }
      for (Text t : repToRecordMap.descendingMap(). values()) {
          // Output our ten records to the file system with a null key
          context.write(NullWritable.get(), t);
      }   }      }
```

MapReduce Design Patterns II: Filtering Patterns

4. Distinct

Filtering Pattern 4. `Distinct`

- You have data that contains **similar records** and you want to find a **unique set** of values
  - e.g. Generate a list of distinct user ids

Mapper Code

```
public static class DistinctUserMapper extends Mapper < Object, Text, Text, NullWritable > {
    private Text outUserId = new Text();
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map < String, String > parsed = transformXmlToMap(value.toString());
    // Get the value for the UserId attribute
    String userId = parsed.get("UserId");
    // Set our output key to the user's id
    outUserId.set(userId);
    // Write the user's id with a null value
    context.write(outUserId, NullWritable.get());
   }
}
```


Reducer code

```
public static class DistinctUserReducer extends Reducer < Text, NullWritable, Text, NullWritable > {
    public void reduce(Text key, Iterable < NullWritable > values, Context context) throws IOException, InterruptedException {

    // Write the user's id with a null value
    context.write(key, NullWritable.get());
    }
}
```

Using `NullWritable`
`NullWritable` is a special type of Writable, as it has a zero-length serialization. No bytes are written to, or read from, the stream. It is used as a placeholder; for example, in MapReduce, a key or a value can be declared as a `NullWritable` when you don't need to use that position—it effectively stores a constant empty value

Part 1. Large Scale Data Analytics

Design Pattern 3: Data Organization Patterns

Data Organization Patterns

- Reorganizing data
  - Partitioning, Sharding and Sorting

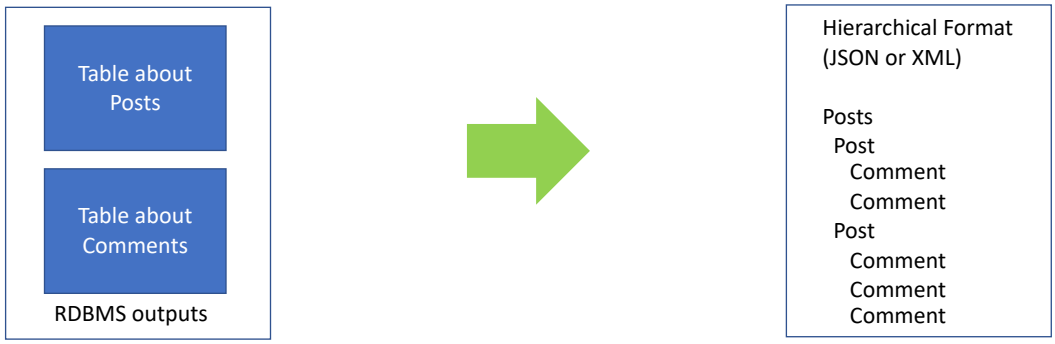
1. "Structured" to "hierarchical" pattern
2. Partitioning and binning patterns
3. Total order sorting

MapReduce Design Patterns III: Data Organization Patterns

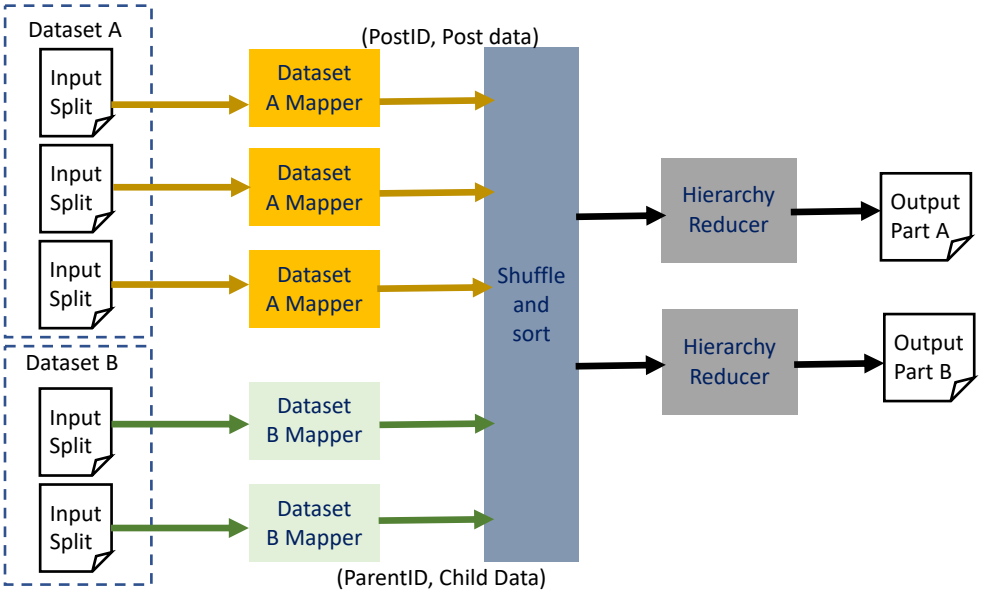
1. "Structured" to "hierarchical" pattern

"Structured" to "Hierarchical"

- Creates **new records** from data with a very **different structure**
  - e.g. transforms your row-based data to a **hierarchical format such as JSON or XML**
- Organizing StackOverflow data



Structure of the "structured" to "hierarchical" pattern



Driver Code

```
public static void main( String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = new Job(conf, "PostCommentHierarchy");
  job.setJarByClass(PostCommentBuildingDriver.class);
  MultipleInputs.addInputPath(job, new Path(args[0]),
                        TextInputFormat.class, PostMapper.class);
  MultipleInputs.addInputPath(job, new Path(args[1]),
                        TextInputFormat.class, CommentMapper.class);
  job.setReducerClass(UserJoinReducer.class);
  job.setOutputFormatClass(TextOutputFormat.class);
  TextOutputFormat.setOutputPath(job, new Path(args[2]));
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);
  System.exit(job.waitForCompletion(true) ? 0 : 2);
}
```

Mapper Code (for posts)

```
public static class PostMapper extends Mapper < Object, Text, Text, Text > {
  private Text outkey = new Text();
  private Text outvalue = new Text();
  public void map( Object key, Text value, Context context) throws IOException,
InterruptedException {
    Map < String, String > parsed =
MRDPUtils.transformXmlToMap(value.toString());
    // The foreign join key is the post ID
    outkey.set(parsed.get("Id"));
    // Flag this record for the reducer and then output
    outvalue.set("P" + value.toString());
    context.write(outkey, outvalue);
  }
}
```


Mapper Code (for comments)

```
public static class CommentMapper extends Mapper < Object, Text, Text, Text > {
  private Text outkey = new Text();
  private Text outvalue = new Text();
  public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
    Map < String, String > parsed = MRDPUtils.transformXmlToMap(value
.toString());
    // The foreign join key is the post ID
    outkey.set(parsed.get("PostId"));
    // Flag this record for the reducer and then output
    outvalue.set("C" + value.toString());
    context.write(outkey, outvalue);
  }
}
```

Reducer Code

```
public static class PostCommentHierarchyReducer extends Reducer < Text, Text,
Text, NullWritable > {
  private ArrayList < String > comments = new ArrayList < String >();
  private DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  private String post = null;
  public void reduce( Text key, Iterable < Text > values, Context context)
throws IOException, InterruptedException {
    // Reset variables
    post = null;
    comments.clear();
    // For each input value
    for (Text t : values) {
      // If this is the post record, store it, minus the flag
      if (t.charAt(0) == 'P') { post = t.toString().substring(1, t.toString().
length())
          .trim();
```


Reducer Code

```
      } else {
        // Else, it is a comment record. Add it to the list, minus
        // the flag
        comments.add(t.toString().substring(1, t.toString().length()).trim());
             }
       }
      // If there are no comments, the comments list will simply be empty.
      // If post is not null, combine post with its comments.
      if (post != null) {
      // nest the comments underneath the post element
         String postWithCommentChildren = nestElements(post, comments);
         // write out the XML
         context.write(new Text(postWithCommentChildren), NullWritable.get());
      }
    } ...
```

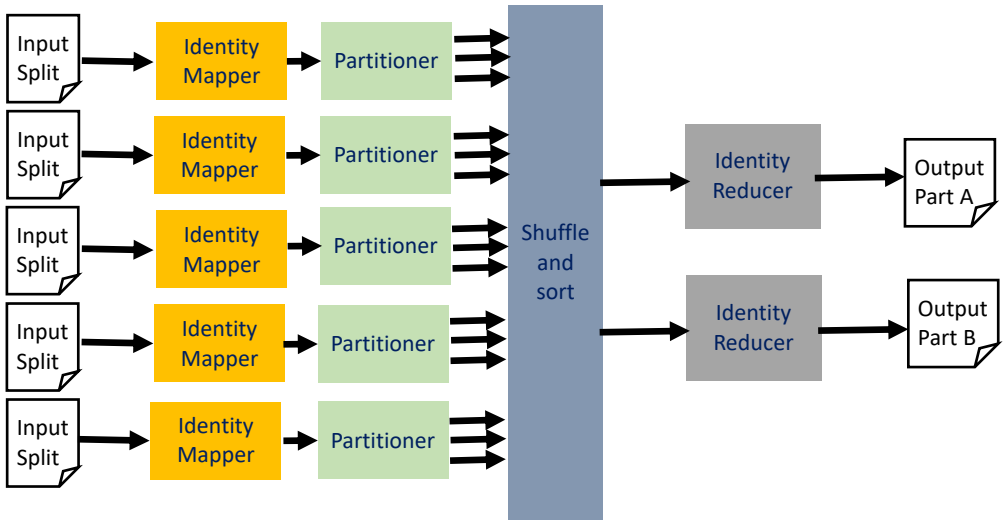
MapReduce Design Patterns III: Data Organization Patterns

2. Partitioning pattern

Partitioning Pattern

- Moves the records into **categories**
  - But it doesn't really care about the order of records
  - Shards, partitions, or bins
- e.g. Partitioning **by date**
  - Groups data based on date
  - Given a set of user information, **partition the records** based on **the year of last access date**, one partition per year

Structure of the partitioning pattern



Partitioner [1/4]

- Partitions the key-value pairs of **intermediate Map-outputs**
  - Uses a **user-defined condition**
- e.g. Process the input dataset to find the **highest salaried employee** by **gender** in different **age groups**

1201	James	45	Male	50000
1202	Matthew	40	Male	51000
1203	Kevin	34	Male	45000
1204	Maria	30	Female	43000
1205	Julia	20	Female	23000
1206	Lavanya	25	Female	42000
1207	Joseph	19	Male	19000
1208	Steve	22	Male	32000
1209	Max	24	Male	33000
1210	Jenifer	28	Female	37000
1211	Nick	18	Male	19000
1212	MaryAnn	33	Female	56000
1213	Kelvin	39	Male	65000

Partitioner [2/4]

- Map Tasks
  - Input
    - Dummy key, data
    - (Dummy_key, "1201 \t James \t 45 \t male \t\ 50000")
  - Method
    - Read the value and extract gender information

```
String[] str = value.toString().split("\t", -3);
String gender = str[3];
```

  - Output
    - Gender data and value

```
context.write(new Text(gender), value);
```


Partitioner [3/4]

- Partitioner Task
  - Dividing the data from the map task into **segments**
  - Input
    - A collection of key-value pairs from the map task
    - Key: gender, value: whole record data
  - Method
    - Read the **age field** and apply conditions
  - Output
    - The **data of key-value pairs are segmented into three collections** of key-value pairs
    - The reducer works **individually on each collection**

```
int age = Integer.parseInt(str[2]);
if(age<=20) {
   return 0;
} else if(age>20 && age<=30) {
   return 1 % numReduceTasks;}
else {
   return 2 % numReduceTasks;
}
```

Partitioner [4/4]

- Reduce Task
  - The **number of partitions(segments)** is **equal** to the **number of reduce tasks**
  - The reducer will execute three times (in our example) with different collection of key-value pairs

Driver Code (last access date partitioner)

```
…
// Set custom partitioner and min last access date
job.setPartitionerClass(LastAccessDatePartitioner.class);
LastAccessDatePartitioner.setMinLastAccessDate(job, 2008);

// Last access dates span between 2008-2011, or 4 years
job.setNumReduceTasks(4);
…
```

Mapper Code (last access date partitioner)

```
public static class LastAccessDateMapper extends Mapper < Object, Text, IntWritable, Text > {
  // This object will format the creation date string into a Date object
  private final static SimpleDateFormat frmt =
     new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS");
  private IntWritable outkey = new IntWritable();
  protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map < String, String > parsed =
          MRDPUtils.transformXmlToMap(value .toString());
    // Grab the last access date
    String strDate = parsed.get("LastAccessDate");
    // Parse the string into a Calendar object
    Calendar cal = Calendar.getInstance();
    cal.setTime(frmt.parse(strDate));
    outkey.set(cal.get(Calendar.YEAR)); // Write out the year with the input value
    context.write(outkey, value);
} }
```


Partitioner code (last access date partitioner)

```
public static class LastAccessDatePartitioner extends Partitioner < IntWritable, Text > implements Configurable {
  private static final String MIN_LAST_ACCESS_DATE_YEAR =
                                          "min.last.access.date.year";
  private Configuration conf = null;
  private int minLastAccessDateYear = 0;
  public int getPartition(IntWritable key, Text value, int numPartitions) {
    return key.get() - minLastAccessDateYear;
  }
  public Configuration getConf() {
    return conf;
  }
  public void setConf(Configuration conf) {
    this.conf = conf;
    minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
  }
```

Partitioner code (last access date partitioner)

```
  public static void setMinLastAccessDate(Job job, int
minLastAccessDateYear) {
    job.getConfiguration()
            .setInt(MIN_LAST_ACCESS_DATE_YEAR,
minLastAccessDateYear);
  }
}
```


Reducer Code (last access date partitioner)

```
public static class ValueReducer extends Reducer < IntWritable, Text,
Text, NullWritable > {
      protected void reduce( IntWritable key, Iterable < Text > values,
Context context) throws IOException, InterruptedException {
            for (Text t : values) {
                  context.write(t, NullWritable.get());
            }
      }
}
```

Unevenly distributed partitions

- Observation
  - Recent years will have more users
- Provide finer grained segmentations to the recent years
  - e.g. Monthly partitions for recent 3 years

MapReduce Design Patterns II: Filtering Patterns

3. Total Order Sorting Pattern

Total Order Sorting Pattern

- Sorts your data
  - e.g. Sorting 1TB of numeric values
  - e.g. Sorting **comments by userID** and you have a **million users**

Structure of Total Order Sorting Pattern

- Two phases
  - **Analysis** phase
    - Determines the ranges
  - **Sorting** phase
    - Actually sorts the data

Structure of the Total Order Sorting Pattern
- Analysis phase

- Performs a **simple random sampling**
- Generates outputs with **the sort key as its output keys**
  - Data will show up as sorted at the reducer
- Sampling rate?
  - Assume that the number of records in the entire dataset is known (or can be estimated)
  - If you plan on running the order with a thousand **reducers**
    - Sampling about a hundred thousand records will be enough
- Only **one reducer** will be used
  - Collects the sort keys together into a sorted list
  - The list of **sorted keys will be sliced into the data range boundaries**

Structure of Total Order Sorting Pattern
- Sorting phase

- Mapper extracts the sort key
  - Stores the sort key to the "value"
- Custom partitioner
  - Use `TotalOrderPartitioner` (Hadoop API)
  - Takes the data ranges from the partition file and decides which reducer to send the data
  - Dynamic and load balanced
- Reducer
  - The number of reducers needs to be **equal to the number of partitions**

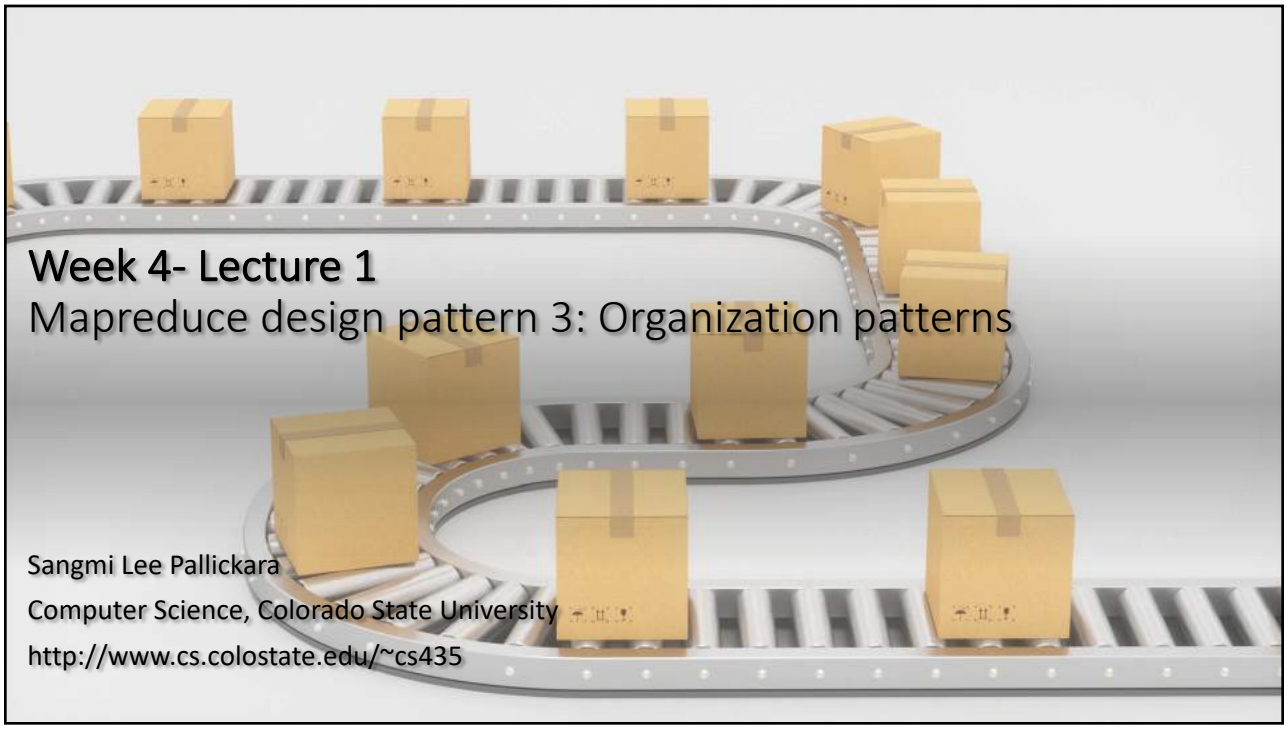
TeraSort Benchmark

- The most well-known Hadoop benchmark
- In 2008, Yahoo! Set a record by sorting 1 TB of data in 209 seconds
  - Hadoop cluster with 910 nodes
  - Owen O'Malley of the Yahoo!
- In 2009, Yahoo! Sorted 1PB of data in 16 hours
  - Hadoop cluster of 3800 nodes
  - For 1TB, it took 62 seconds
  - http://sortbenchmark.org/YahooHadoop.pdf

TeraSort Benchmark APIs

- TeraGen
  - MR to generate the data
- TeraSort
  - Samples the input data and uses MR to sort the data into a total order
- TeraValidate
  - MR that validates the output
- TeraSort is a standard MapReduce with a custom partitioner that uses a sorted list of N-1 sorted sampled keys that define the key range for each reduce
- $sample[i-j]<=key<sample[i]$ are sent to reducer i
  - Total 1,000 lines of java code

Questions?



0

FAQs

- Quizzes
  - Discussed in the recitation class
  - Questions/solutions will be posted before exams
- How does “Reducer” know which output they should use?

1

Design your **map** and **reduce function** to perform following data processing.

Find the top 100 clients who spent the most electricity (kilowatt) for the last month. Assume that all the electricity usage measures are unique.

Files contain information about the last month only. The data is formatted as follows:

{customerID, TAB, address, TAB, zipcode, TAB, electricity_usage, LINEFEED}. Assume that your MapReduce program is using the Top-N design pattern.

Topics discussed in the previous class

- Topic 1: Term Project planning and exploring topics and datasets
- Topic 2: MapReduce Design Pattern II. Filter Patterns
  - Bloomfilter
  - Top N
  - Distinct
- Topic 3: MapReduce Design Pattern III. Organization Patterns
  - “Structure” to “Hierarchical”

Topics to be covered

- MapReduce Design Pattern III. Organization Patterns

Part 1. Large Scale Data Analytics

Design Pattern 3: Data Organization Patterns

Data Organization Patterns

- Reorganizing data
  - Partitioning, Sharding and Sorting

1. "Structured" to "hierarchical" pattern
2. Partitioning and binning patterns
3. Total order sorting

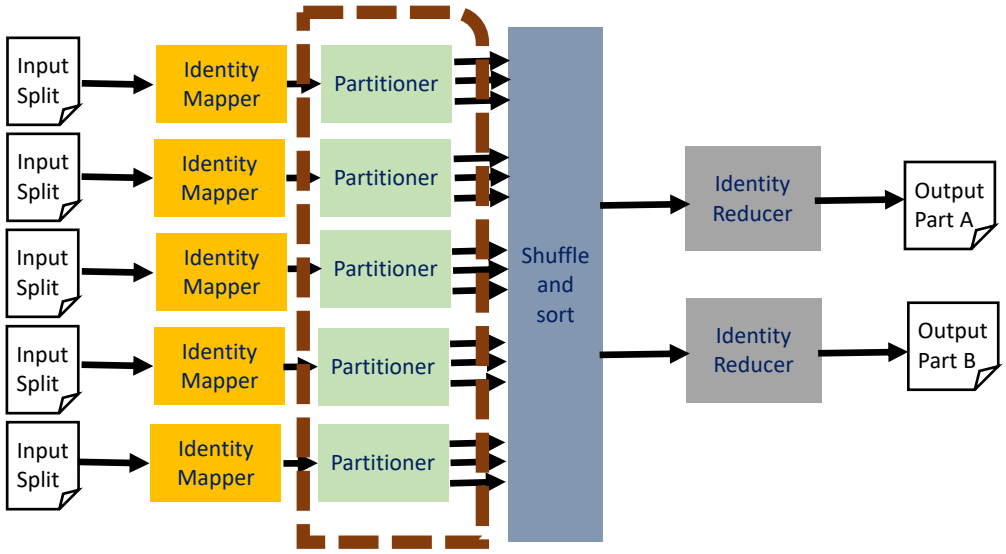
MapReduce Design Patterns III: Data Organization Patterns

2. Partitioning pattern

Partitioning Pattern

- Moves the records into **categories**
  - But it doesn't really care about the order of records
  - Shards, partitions, or bins
- e.g. Partitioning **by date**
  - Groups data based on date
  - Given a set of user information, **partition the records** based on **the year of last access date**, one partition per year

Structure of the partitioning pattern



What is a "Partitioner"?

- Partitioner allows distributing how outputs from the map stage are send to the reducers
- The total number of partitions is same as the total number of ~~reducers~~ reducer tasks
- It reduces the amount of time of reducer job

```
public int getPartition(K key, V value,
    int numReduceTasks) {
         return(key.hashCode()
                & Integer.MAX_VALUE) % numReduceTasks;
}
```


Custom Partitioner

- Parameters
  - Key, Value, numPartitions
- Optimize the mapreduce programs and executed the given problem as well as possible
- Can divide the output files how it required

```
public abstract class Partitioner<KEY, VALUE> {
       public abstract int getPartition (KEY key, VALUE value,
                        int numPartitions){
        return;
       }
}
```

Partitioner [1/4]

- Partitions the key-value pairs of **intermediate Map-outputs**
  - Uses a **user-defined condition**
- e.g. Process the input dataset to find the **highest salaried employee** by **gender** in different **age groups**

1201	James	45	Male	50000
1202	Matthew	40	Male	51000
1203	Kevin	34	Male	45000
1204	Maria	30	Female	43000
1205	Julia	20	Female	23000
1206	Lavanya	25	Female	42000
1207	Joseph	19	Male	19000
1208	Steve	22	Male	32000
1209	Max	24	Male	33000
1210	Jenifer	28	Female	37000
1211	Nick	18	Male	19000
1212	MaryAnn	33	Female	56000
1213	Kelvin	39	Male	65000

Partitioner [2/4]

- Map Tasks
  - Input
    - Dummy key, data
    - `(Dummy_key, “1201 \t James  \t 45 \t male \t\ 50000”)`
  - Method
    - Read the value and extract gender information

```
String[] str = value.toString().split(“\t”, -3);
String gender = str[3];
```

  - Output
    - Gender data and value

```
context.write(new Text(gender), value);
```

Partitioner [3/4]

- Partitioner Task
  - Dividing the data from the map task into **segments**
  - Input
    - A collection of key-value pairs from the map task
    - Key: gender, value: whole record data
  - Method
    - Read the **age field** and apply conditions
  - Output
    - The **data of key-value pairs are segmented into three collections** of key-value pairs
    - The reducer works **individually on each collection**

```
int age = Integer.parseInt(str[2]);
if(age<=20) {
   return 0;
} else if(age>20 && age<=30) {
   return 1 % numReduceTasks;}
else {
   return 2 % numReduceTasks;
}
```


Partitioner [4/4]

- Reduce Task
  - The **number of partitions(segments)** is **equal** to the **number of reduce tasks**
    - **NOTE: "keys" are still sorted and passed to the reducer**
    - **In this example, there will be 3 reducers and each reducer will get <"male", [list of values]> and <"female", [list of values]>**

Partitioning Pattern: Example

- Partitioning **by date**
  - Groups data based on date
  - Given a set of user information, **partition the records** based on **the year of last access date**, one partition per year

Driver Code (last access date partitioner)

```
…
// Set custom partitioner and min last access date
job.setPartitionerClass(LastAccessDatePartitioner.class);
LastAccessDatePartitioner.setMinLastAccessDate(job, 2008);

// Last access dates span between 2008-2011, or 4 years
job.setNumReduceTasks(4);
…
```

Mapper Code (last access date partitioner)

```
public static class LastAccessDateMapper extends Mapper < Object, Text, IntWritable, Text > {
  // This object will format the creation date string into a Date object
  private final static SimpleDateFormat frmt =
     new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS");
  private IntWritable outkey = new IntWritable();
  protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map < String, String > parsed =
          MRDPUtils.transformXmlToMap(value .toString());
    // Grab the last access date
    String strDate = parsed.get("LastAccessDate");
    // Parse the string into a Calendar object
    Calendar cal = Calendar.getInstance();
    cal.setTime(frmt.parse(strDate));
    outkey.set(cal.get(Calendar.YEAR)); // Write out the year with the input value
    context.write(outkey, value);
} }
```


Partitioner code (last access date partitioner)

```
public static class LastAccessDatePartitioner extends Partitioner < IntWritable, Text > implements Configurable {
  private static final String MIN_LAST_ACCESS_DATE_YEAR =
                                          "min.last.access.date.year";
  private Configuration conf = null;
  private int minLastAccessDateYear = 0;
  public int getPartition(IntWritable key, Text value, int numPartitions) {
    return key.get() - minLastAccessDateYear;
  }
  public Configuration getConf() {
    return conf;
  }
  public void setConf(Configuration conf) {
    this.conf = conf;
    minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
  }
```

Partitioner code (last access date partitioner)

```
  public static void setMinLastAccessDate(Job job, int
minLastAccessDateYear) {
    job.getConfiguration()
            .setInt(MIN_LAST_ACCESS_DATE_YEAR,
minLastAccessDateYear);
  }
}
```


Reducer Code (last access date partitioner)

```
public static class ValueReducer extends Reducer < IntWritable, Text,
Text, NullWritable > {
      protected void reduce( IntWritable key, Iterable < Text > values,
Context context) throws IOException, InterruptedException {
            for (Text t : values) {
                  context.write(t, NullWritable.get());
            }
      }
}
```

Worksheet #4-1

- Assume that your application has more 80% of new users in recent years
- Design your partitioner to resolve this problem.

Unevenly distributed partitions

- Observation
  - Recent years will have more users
- Provide finer grained segmentations to the recent years
  - e.g. Monthly partitions for recent 3 years

MapReduce Design Patterns II: Filtering Patterns

3. Total Order Sorting Pattern

Total Order Sorting Pattern

- Sorts your data
  - e.g. Sorting 1TB of numeric values
  - e.g. Sorting **comments by userID** and you have a **million users**

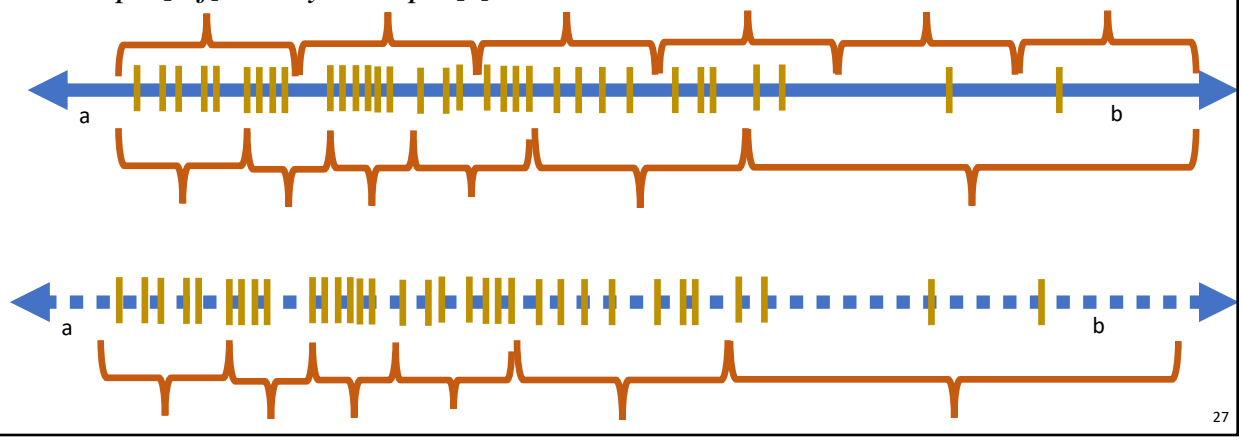
Structure of Total Order Sorting Pattern

- Two phases
  - **Analysis** phase
    - Determines the ranges
  - **Sorting** phase
    - Actually sorts the data

With only one MapReduce Job?
No. You can use multiple MapReduce Jobs for this problem!

What do we do in the "Analysis phase"?

- We will get "splits" of Tera Bytes of numbers
- *sample[i-j]<=key<sample[i]* are sent to reducer *i*

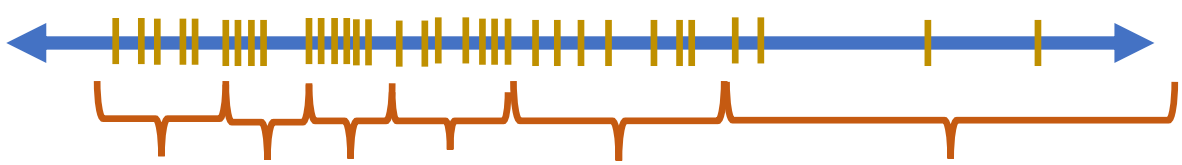


Structure of the Total Order Sorting Pattern
- Analysis phase

- Performs a **simple random sampling**
- Generates outputs with **the sort key as its output keys**
  - Data will show up as sorted at the reducer
- Sampling rate?
  - Assume that the number of records in the entire dataset is known (or can be estimated)
  - If you plan on running the order with a thousand **reducers**
    - Sampling about a hundred thousand records will be enough
- Only **one reducer** will be used
  - Collects the sort keys together into a sorted list
  - The list of **sorted keys will be sliced into the data range boundaries**

What do we do in the ”Sorting phase”?

- We will sort data within the splits



Structure of Total Order Sorting Pattern
- Sorting phase

- Mapper extracts the sort key
  - Stores the sort key to the "value"
- Custom partitioner
  - Use **`TotalOrderPartitioner`** (Hadoop API)
  - Takes the data ranges from the partition file and decides which reducer to send the data
  - Dynamic and load balanced
  - **`setPartitionerFile(Configration conf)`**
    - Get the path to the SequenceFile string the sorted partition keyset
- Reducer
  - The number of reducers needs to be **equal to the number of partitions**

TeraSort Benchmark

- The most well-known Hadoop benchmark
- In 2008, Yahoo! Set a record by sorting 1 TB of data in 209 seconds
  - Hadoop cluster with 910 nodes
  - Owen O'Malley of the Yahoo!
- In 2009, Yahoo! Sorted 1PB of data in 16 hours
  - Hadoop cluster of 3800 nodes
  - For 1TB, it took 62 seconds
  - http://sortbenchmark.org/YahooHadoop.pdf

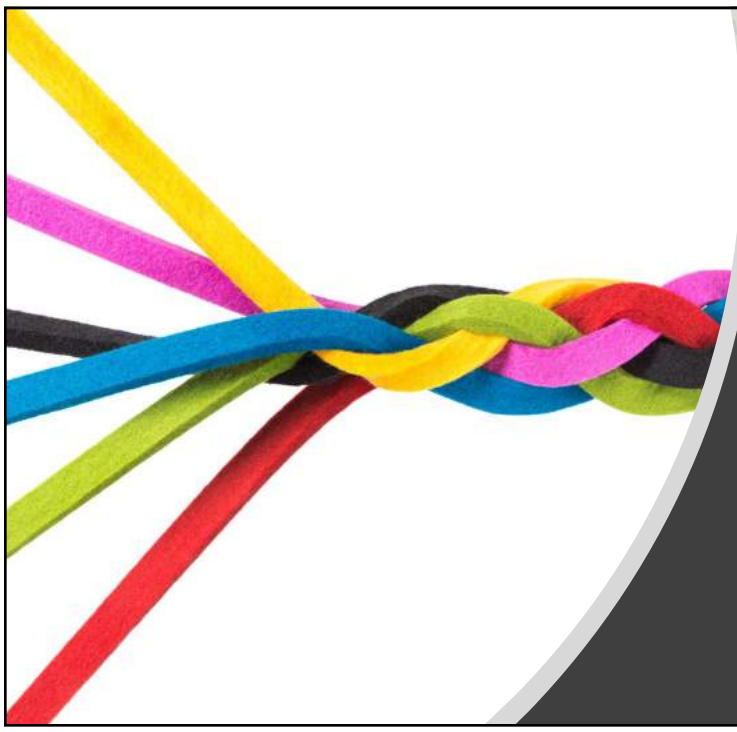
TeraSort Benchmark APIs

- TeraGen
  - MR to generate the data
- TeraSort
  - Samples the input data and uses MR to sort the data into a total order
- TeraValidate
  - MR that validates the output

- TeraSort is a standard MapReduce with a custom partitioner that uses a sorted list of N-1 sorted sampled keys that define the key range for each reduce
- *sample[i-j]<=key<sample[i]* are sent to reducer *i*

  - Total 1,000 lines of java code

Questions?



Week 4 – Lecture 2

MapReduce Design Pattern IV: Join Patterns

Sangmi Lee Pallickara
Computer Science, Colorado State University

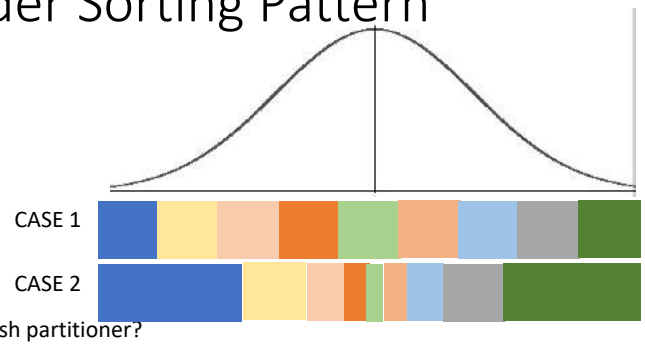
FAQs

- Please sign on the attendance sheet
- Quiz #3
  - Score will be available on Sunday (9/19)
- Midterm
  - 10/6 Wednesday (5:30AM – 6:15PM) in class
  - 75 minutes
  - Review session on 10/4 with sample questions
- Why do we need the analysis phase for the TeraSort problem?

REVIEW

Structure of Total Order Sorting Pattern

- Two phases
  - **Analysis** phase
    - Determines the ranges
  - **Sorting** phase
    - Actually sorts the data



Can we use a single reducer?

If we will use multiple reducers, can we use a random hash partitioner?

Each partition must ensure that all items are neighboring (in a sorted list) and there is no item from other partitions To complete the portion of sorted list.

What if we partition the data based on (Max-Min)/(# of reducers)?

We need a data-specific ranges to assign roughly the same number of consecutive items

→ **Analysis Phase!**

Topics discussed in the previous class

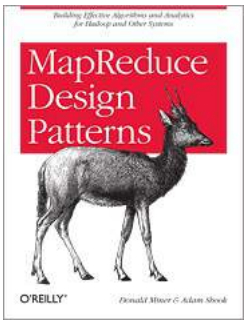
- MapReduce Design Pattern III: Organization Patterns
  - Partitioning
  - Total order sorting (TeraSort)

Topics to be covered

- Hands-on session for Sustain - Data exploration
  - For your term project
  - Enhancing your term project
  - Daniel Reynolds
    - Undergraduate Research Assistant
- MapReduce Design Pattern IV. Join Patterns

This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012



Part 1. Large Scale Data Analytics

Design Pattern 4: Join Patterns

Join Patterns

- Data is **all over** the place
- "Joins" allow users to **create a smaller reference set** or **filter out** or **select dataset to discover interesting relationships** across datasets
- Joining a terabyte of data onto another terabyte dataset could require up to **two terabytes of bandwidth!**
  - That's before any actual join logic can be done!

1. Reduce Side Join Pattern
2. Replicated Join Pattern
3. Composite Join Pattern
4. Cartesian Product Pattern

Brief review on **joins**

- A join is an operation that combines records from two or more datasets **based on a field or set of fields**
  - Foreign key
- The **foreign key** is the field in a relational table that matches the column of another table
  - Used as a means to cross-reference between tables

Example

Dataset A

UserID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diago, CA
9	3443	Oakland, CA

Dataset B

UserID	PostID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hey, of course, it's all true!
5	44921	Please see my post below
5	44920	Thank you very much for your reply
8	48678	HTML is not a subset of XML!

Inner Join

Dataset A

UserID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diego, CA
9	3443	Oakland, CA

Datatset B

UserID	PostID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hey, of course, it's all true!
5	44921	Please see my post below
5	44920	Thank you very much for your reply
8	48678	HTML is not a subset of XML!

Default join style

Records from both A and B that contain **identical values for the given foreign key** are brought together

Worksheet Question:
As the result of "inner join of A and B on user ID", how many unique tuples will be generated?
a. 3. b. 4. c. 5. d. 6

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 4-B Fall 2021Computer Science

Inner Join

Dataset A

UserID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diego, CA
9	3443	Oakland, CA

Datatset B

UserID	PostID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hey, of course, it's all true!
5	44921	Please see my post below
5	44920	Thank you very much for your reply
8	48678	HTML is not a subset of XML!

Default join style

Records from both A and B that contain **identical values for the given foreign key** are brought together

Inner join of A+B on User id

A. UserID	A.Reputation	A.Location	B.UserID	B.PostID	B.Text
3	3738	New York, NY	3	35314	Not sure why this is getting downvoted
3	3738	New York, NY	3	48002	Hey, of course, it's all true!
5	17556	San Diego, CA	5	44921	Please see my post below
5	17556	San Diego, CA	5	44920	Thank you very much for your reply

Outer Join

Dataset A

UserID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diego, CA
9	3443	Oakland, CA

Datatset B

UserID	PostID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hey, of course, it's all true!
5	44921	Please see my post below
5	44920	Thank you very much for your reply
8	48678	HTML is not a subset of XML!

Records from a foreign key not present in both table will **be also in the final table**

Left Outer Join

Unmatched records in the "left" table will be in the final table

Null values in the columns of the right table that did not match

Right Outer Join

The right table records are kept and the left table values are null where appropriate

Full outer join

contains all unmatched records from both tables

12

12

Left Outer Join

Dataset A

UserID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diego, CA
9	3443	Oakland, CA

Datatset B

UserID	PostID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hey, of course, it's all true!
5	44921	Please see my post below
5	44920	Thank you very much for your reply
8	48678	HTML is not a subset of XML!

Left Outer join of A+B on User id

A. UserID	A.Reputation	A.Location	B.UserID	B.PostID	B.Text
3	3738	New York, NY	3	35314	Not sure why this is getting downvoted
3	3738	New York, NY	3	48002	Hey, of course, it's all true!
4	12946	New York, NY	null	null	null
5	17556	San Diego, CA	5	44921	Please see my post below
5	17556	San Diego, CA	5	44920	Thank you very much for your reply
9	3443	Oakland, CA	null	null	null

13

13

Anti Join

Dataset A

UserID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diego, CA
9	3443	Oakland, CA

Datatset B

UserID	PostID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hey, of course, it's all true!
5	44921	Please see my post below
5	44920	Thank you very much for your reply
8	48678	HTML is not a subset of XML!

Anti Join

Full outer join minus the inner join

Anti join of A+B on User id

A. UserID	A.Reputation	A.Location	B.UserID	B.PostID	B.Text
4	12946	New York, NY	null	null	null
null	null	null	8	48775	HTML is not a subset of XML!
9	3443	Oakland, CA	null	null	null

MapReduce Design Patterns IV: Join Patterns

1. Reduce Side Join Pattern

Reduce Side Join Pattern

- **Most straightforward implementation** of a join in MapReduce
- Requires a **large amount of network bandwidth**
  - Bulk of the data is sent to the reduce phase
  - If you have resources available this will be a possible solution

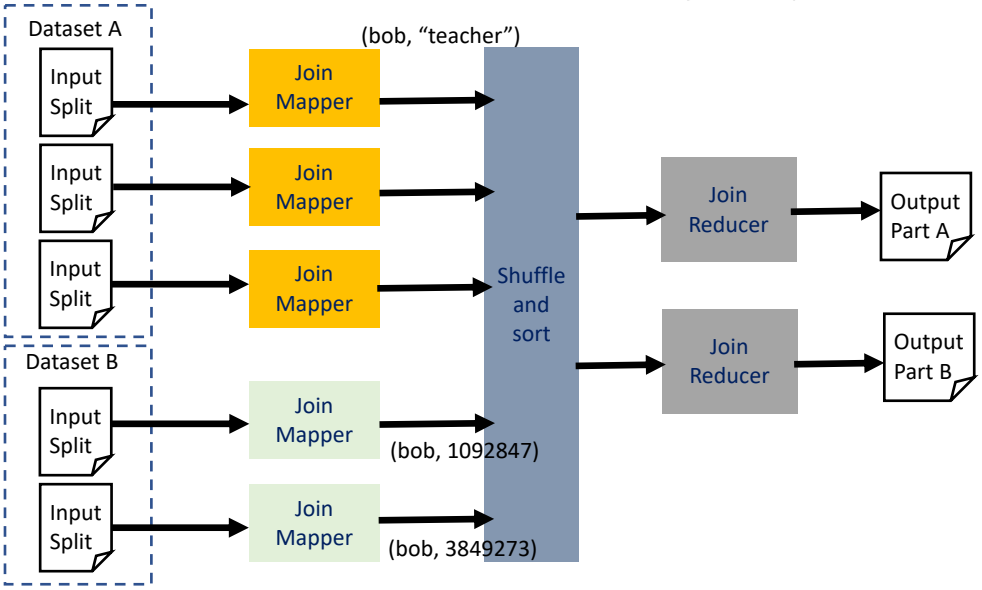
- Worksheet question 2:

Assume that you perform join operation between two very large tables A and B using MapReduce. Table A contains the following attributes: `student_ID, semester, GPA, number_of_credits`

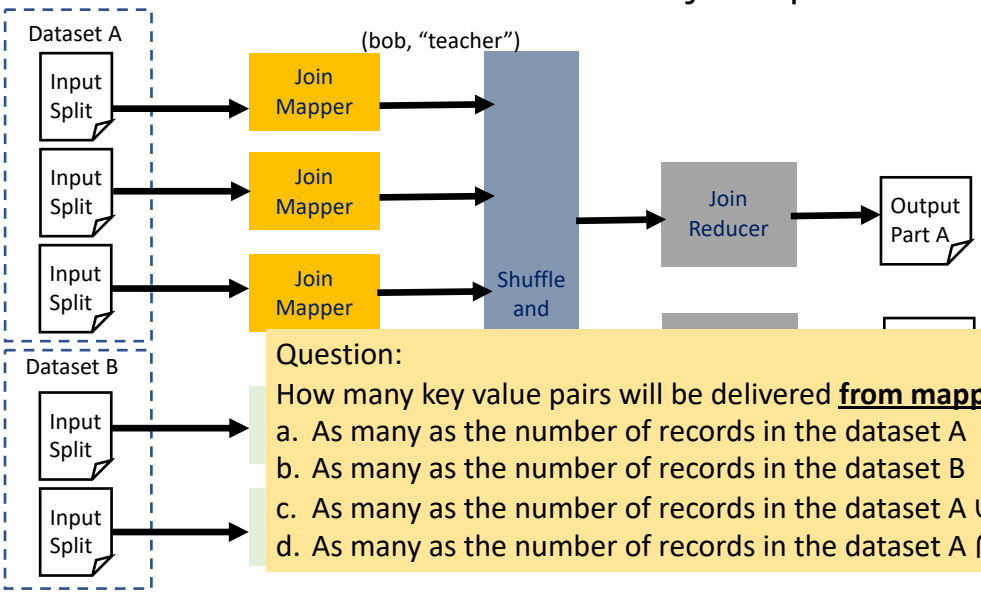
Table B contains the following attributes: `student_ID, address, phone_number, DOB`

Here, `student_ID` is the primary key for the join operation. Design a MapReduce application for this operation.

Structure of the reduce side join pattern



Structure of the reduce side join pattern



Question:
How many key value pairs will be delivered **from mappers**?

a. As many as the number of records in the dataset A
b. As many as the number of records in the dataset B
c. As many as the number of records in the dataset A ∪ B
d. As many as the number of records in the dataset A ∩ B

Performance analysis

- The reducer side join puts a lot of strain on the cluster's network
- The **foreign key and output record** of each input record are extracted
  - No data can be filtered ahead of time
  - Almost all of the data will be sent to the shuffle and sort step
- Reduce side joins will typically utilize **relatively more reducers** than your typical analytics

Driver Code

```
...
// Use MultipleInputs to set which input uses what mapper
// This will keep parsing of each data set separate from a logical
standpoint
// The first two elements of the args array are the two inputs
MultipleInputs.addInputPath(job, new Path(args[0]),
      TextInputFormat.class, UserJoinMapper.class);
MultipleInputs.addInputPath( job, new Path(args[1]),
      TextInputFormat.class, CommentJoinMapper.class);
job.getConfiguration()
...
```


”User” Mapper Code

```
public static class UserJoinMapper extends Mapper < Object, Text, Text, Text > {
  private Text outkey = new Text();
  private Text outvalue = new Text();
  public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
    // Parse the input string into a nice map
    Map < String, String > parsed =
          MRDPUtils.transformXmlToMap(value.toString());
    String userId = parsed.get("Id");
    // The foreign join key is the user ID
    outkey.set(userId);
    // Flag this record for the reducer and then output
    outvalue.set("A" + value.toString());
    context.write(outkey, outvalue);
  }
}
```

"Comment" mapper code

```
public static class CommentJoinMapper extends Mapperlt; Object, Text, Text, Text > {
  private Text outkey = new Text();
  private Text outvalue = new Text();
  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map < String, String > parsed = transformXmlToMap( value.toString());
    // The foreign join key is the user ID
    outkey.set(parsed.get("UserId"));
    // Flag this record for the reducer and then output
    outvalue.set("B" + value.toString());
    context.write(outkey, outvalue);
  }
}
```


Reducer Code

```
public static class UserJoinReducer extends Reducer < Text, Text, Text, Text > {
  private static final Text EMPTY_TEXT = Text("");
  private Text tmp = new Text();
  private ArrayList < Text > listA = new ArrayList < Text >();
  private ArrayList < Text > listB = new ArrayList < Text >();
  private String joinType = null;
  public void setup( Context context) {
    // Get the type of join from our configuration
    joinType = context.getConfiguration().get("join.type");
  }
  public void reduce( Text key, Iterable < Text > values, Context context) throws IOException, InterruptedException {
    // Clear our lists
    listA.clear();
```

Reducer Code

```
listB.clear();
// iterate through all our values, binning each record based on what
// it was tagged with. Make sure to remove the tag! while (values.hasNext()) {
  tmp = values.next();
  if (tmp.charAt(0) == 'A') {
    listA.add(new Text(tmp.toString().substring(1)));
  } else if (tmp.charAt('0') == 'B') {
      listB.add(new Text(tmp.toString().substring(1)));
  }
}
// Execute our join logic now that the lists are filled
executeJoinLogic(context);
}
private void executeJoinLogic(Context context) throws IOException,
InterruptedException {
   ...
}
```


Inner Join Code

```
if (joinType.equalsIgnoreCase("inner")) {
  // If both lists are not empty, join A with B
  if (!listA.isEmpty() && !listB.isEmpty()) {
     for (Text A : listA) {
        for (Text B : listB) {
           context.write(A, B);
        }
      }
    }
} ...
```

??? Join Code

```
...
else if (joinType.equalsIgnoreCase(" ??? ")) {
  // For each entry in A,
  for (Text A : listA) {
      // If list B is not empty, join A and B
      if (! listB.isEmpty()) {
           for (Text B : listB) {
                context.write(A, B);
           }
       } else {
            // Else, output A by itself
            context.write( A, EMPTY_TEXT);
       }
    }
 } ...
```

Question:
What is the operation performed in this reducer?

a. Inner Join
b. Left outer join
c. Right outer join
d. Anti join


```
...
else if (joinType.equalsIgnoreCase("leftouter")) {
  // For each entry in A,
  for (Text A : listA) {
      // If list B is not empty, join A and B
      if (! listB.isEmpty()) {
           for (Text B : listB) {
                context.write(A, B);
           }
       } else {
            // Else, output A by itself
            context.write( A, EMPTY_TEXT);
       }
    }
 } ...
```

Right outer Join Code

```
...
else if (joinType.equalsIgnoreCase("rightouter")) {
  // For each entry in B,
  for (Text B : listB) {
      // If list A is not empty, join A and B
      if (! listA.isEmpty()) {
           for (Text A : listA) {
                context.write( A, B);
           }
       } else {
            // Else, output B by itself
            context.write( EMPTY_TEXT, B);
       }
    }
 } ...
```

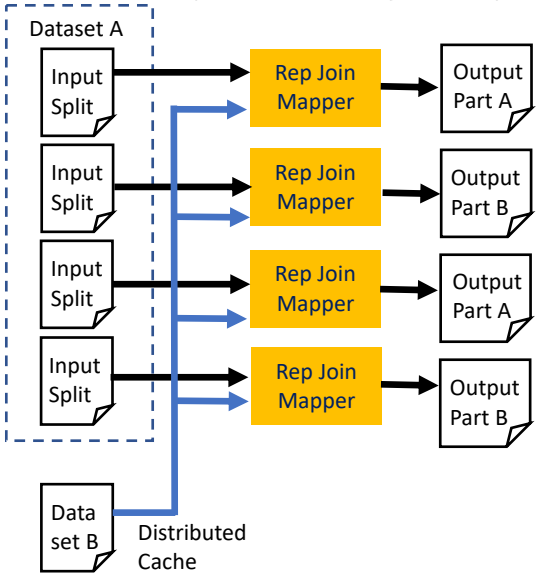

MapReduce Design Patterns IV: Join Patterns

2. Replicated Join

Replicated Join

- Special type of join operation between **one large and small data set** that can be performed on the map-side
  - e.g. Mapping with the zip code, or area code
- Mapper
  - Reads all files from the **distributed cache** during the **setup phase**
    - **Sorting them in to in-memory lookup tables**
  - Performs mapper process
    - Joining data
  - If the foreign key is not found in the in-memory structure?
    - The record is either omitted or output (based on the join type)
- **No combiner/partitioner/reducer needed**

Structure of the replicated join pattern



Hadoop `DistributedCache`

- Provided by the Hadoop MapReduce Framework
- Caches **read only text files, archives, jar files etc.**
- Once a file is cached for a job using Distributed cache
  - Data will be available on **each data node** where **map/reduce tasks are runni**ng

Working with `DistributedCache`

- Make sure
  - Your file is available and accessible via http:// or hdfs://

- Setup the application's **`JobConf`** in your **`Driver` class**

```
DistributeCache.addFileToClasspath(new Path("/usr/datafile/XYZ")
```

Size of `DistributedCache` in Hadoop

- Size
  - Default size of the Hadoop distributed cache is 10GB
  - Configurable in **mapred-site.xml**
- Data consistency
  - Hadoop Distributed Cache tracks the modification of timestamps of the cache file
  - Cache files should not be modified by the application or externally while the job is executing.
- Overhead
  - Object serialization

Using `DistributedCache` for replicated join

- A **small file** is pushed to **all map tasks** using **`DistributedCache`**
- Useful for join between **a small set and a large set of data**
  - e.g. user information vs. transaction records, user information vs. comment history
- Mapper Code
  - **Setup phase**
    - User data is read from the DistributedCache and stored in memory
    - (userID, records) pairs are stored in a HashMap for data retrieval during the map process
  - **Map phase**
    - For each input record (from the large dataset), the user information is retrieved from the HashMap
    - Assemble a joined record

Mapper Code [1/3]

-Join (Comment, user, on userID)

```
public static class ReplicatedJoinMapper extends Mapper < Object, Text, Text, Text > {
  private static final Text EMPTY_TEXT = new Text("");
  private HashMap < String, String > userIdToInfo = new HashMap < String, String >();
  private Text outvalue = new Text();
  private String joinType = null;



  public void setup(Context context) throws IOException,InterruptedException {
    Path[] files = DistributedCache.getLocalCacheFiles( context.getConfiguration());
```


Mapper Code [2/3]

-Join (Comment, user, on userID)

```
// Read all files in the DistributedCache
for (Path p : files) {
  BufferedReader rdr = new BufferedReader(new InputStreamReader(new GZIPInputStream(new FileInputStream(new File(p.toString()))))); 
  String line = null;
  // For each record in the user file
  while ((line = rdr.readLine()) != null) {
    // Get the user ID for this record
    Map < String, String > parsed = transformXmlToMap(line);
    String userId = parsed.get("Id");
    // Map the user ID to the record
    userIdToInfo.put(userId, line);
  }
}
// Get the join type from the configuration
joinType = context.getConfiguration(). get(" join.type"); }
```

Mapper Code [3/3]

-Join (Comment, user, on userID)

```
public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
  Map <String, String> parsed = transformXmlToMap(value.toString());
  String userId = parsed.get("UserId");
  String userInformation = userIdToInfo.get(userId);
  // If the user information is not null, then output
  if (userInformation != null) {
    outvalue.set(userInformation);
    context.write(value, outvalue);
  } else if (joinType.equalsIgnoreCase("leftouter")) {
    // If we are doing a left outer join,
    // output the record with an empty value
    context.write(value, EMPTY_TEXT);
  }
 }
}
```

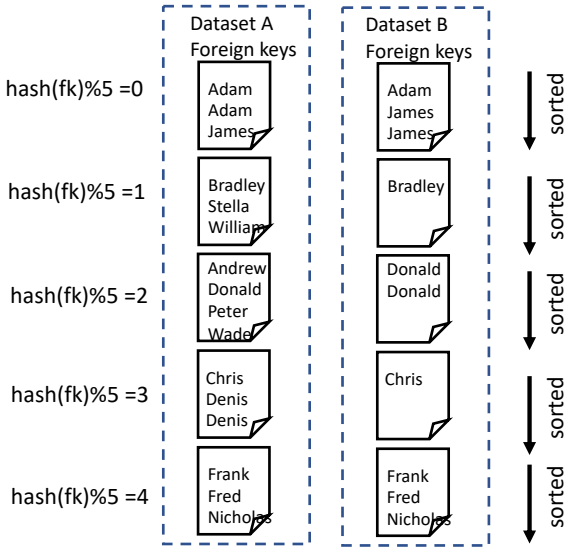

MapReduce Design Patterns IV: Join Patterns

3. Composite Join

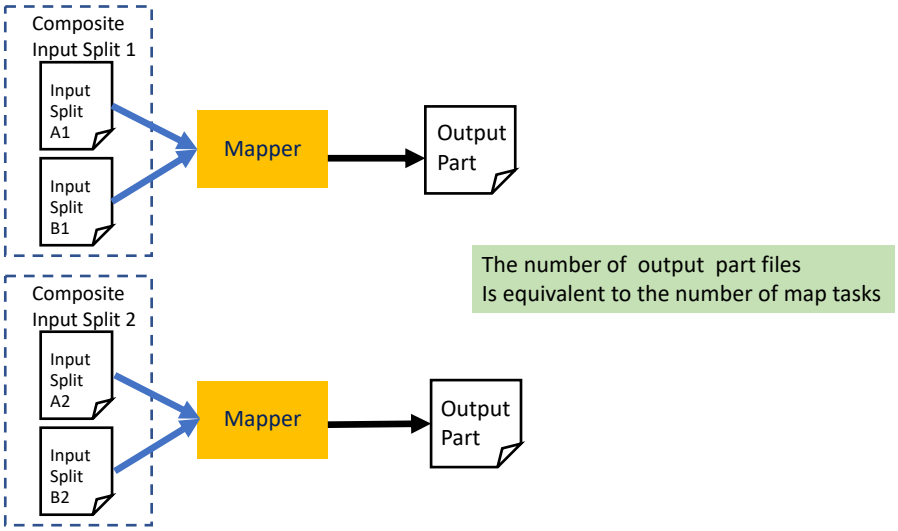
Composite Join

- Joins **very large datasets together**
  - And if the datasets are sorted by foreign key
- **No shuffle and sort needed**
- Each input dataset must be **partitioned and sorted in a specific way** and **divided into the same number of partitions**

Structure of the composite join pattern



Structure of the composite join pattern



Joining process (Inner Join)- Within a group

UserID	Credit Card
12	Master
13	Master
14	Visa
15	Visa
15	Discover
17	Master
17	Visa
17	Discover
20	Visa
20	American Express

UserID	Payments
11	FC Utility
12	FC Utility
13	FC Utility
13	Coloradoan
13	Netflix
13	Xcel
13	Honda Finance
14	Xcel
19	FC Utility
20	Xcel

Joining process (Inner Join)-Between groups

UserID	Credit Card
12	Master
…	…
20	American Express

UserID	Payments
11	FC Utility
…	…
13	FC Utility

UserID	Credit Card
21	Visa
…	…
29	American Express

UserID	Payments
22	FC Utility
…	…
27	FC Utility

UserID	Credit Card
32	Master
…	…
38	Visa

UserID	Payments
34	FC Utility
…	…
49	FC Utility

Hash function h(fk)= fk mod 10
There is no overlapping range
Between groups.
No computation across the
sub-tables will be needed

Joining process (Inner Join)

For N items in the left dataset and M items in the right dataset
Naïve approach (without sorting and exact matching partitioning) takes O(NM)
With the composite join, it takes O(N)

UserID	Credit Card
12	Master
13	Master
14	Visa
15	Visa
15	Discover
17	Master
17	Visa
17	Discover
20	Visa
20	American Express

UserID	Payments
11	FC Utility
12	FC Utility
13	FC Utility
13	Coloradoan
13	Netflix
13	Xcel
13	Honda Finance
14	Xcel
19	FC Utility
20	Xcel

Example: Composite user comment join

- Preconditions
  - User and comment datasets should be **preprocessed by MapReduce**
  - **`CompositeInputFormat`** will be used
  - Output should use the **`TextOutputFormat`**
    - Key: userID
    - Value: userXML or comment XML
    - Hadoops `KeyValueTextOutputFormat` can parse these information
- Note: Since **`CompositeInputFormat`** uses `Text` objects as the key for comparisons
  - **Align your keys as `Text` objects** (rather than `LongWritable`)
  - e.g. "12345" comes before "2"

Driver Code [1/2]

```
public static void main( String[] args) throws Exception {
  Path userPath = new Path(args[0]);
  Path commentPath = new Path(args[1]);
  Path outputDir = new Path(args[2]);
  String joinType = args[3];
  JobConf conf = new JobConf("CompositeJoin");
  conf.setJarByClass(CompositeJoinDriver.class)
  conf.setMapperClass(CompositeMapper.class);
  conf.setNumReduceTasks(0);
  // Set the input format class to a CompositeInputFormat  ass.
  // The CompositeInputFormat will parse all of our input  les
and output
  // records to our mapper.
  conf.setInputFormat(CompositeInputFormat.class);
```

This input format scans through the left dataset and Identifies matching in the right dataset. It generates input value, (item from The left dataset, item from the right Dataset)

Driver Code [1/2]

```
  // The composite input format join expression
  // will set how the records
  // are going to be read in, and in what input format.
  conf.set("mapred.join.expr",
          CompositeInputFormat.compose(joinType,
          KeyValueTextInputFormat.class, userPath, commentPath));
  TextOutputFormat.setOutputPath(conf, outputDir);
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);
  RunningJob job = JobClient.runJob(conf);
  while (!job.isComplete()) {
     Thread.sleep(1000);
  }
  System.exit(job.isSuccessful() ? 0 : 1);
}
```


Mapper Code

```
public static class CompositeMapper extends
MapReduceBase implements Mapper < Text, TupleWritable,
Text, Text > {
  public void map(Text key, TupleWritable value,
OutputCollector < Text, Text > output, Reporter
reporter) throws IOException {
    // Get the first two elements in the tuple and
output them
    output.collect((Text)value.get(0),
                   (Text)value.get(1));
  }
}
```

All of the matching has been done in CompositeInputFormat already.

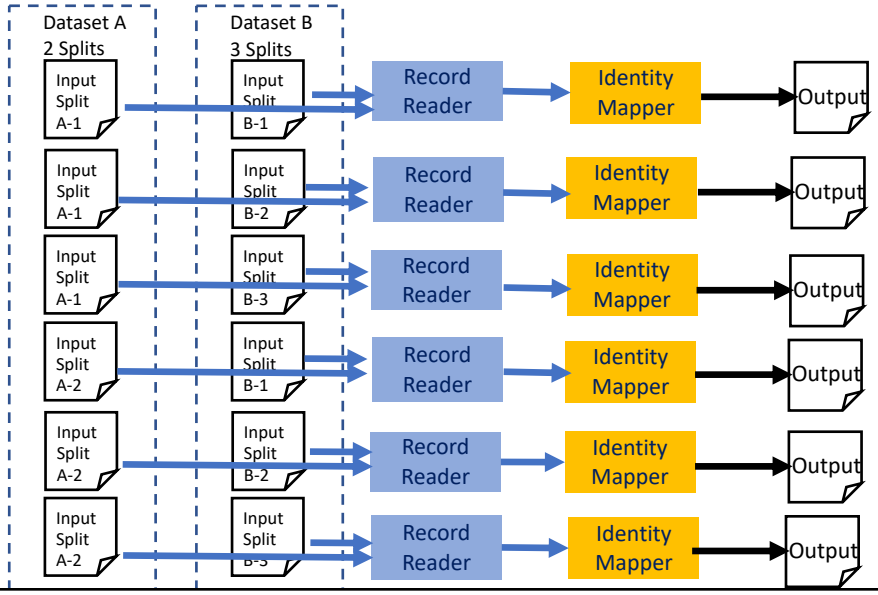
MapReduce Design Patterns IV: Join Patterns

4. Cartesian Product

Cartesian Product

- Pairs **every record** from multiple inputs
- Applicability
  - **You want to analyze relationships between all pairs of records**
  - e.g. Correlation analysis

Structure of the Cartesian Product



Performance Analysis

- Cartesian product requires a massive data space
  - To compute a Cartesian product between N records and M records
  - N x M data points are created
    - Requires many Map slots and a very long time

Example: Cartesian Product

- Comments comparison
  - Inspects a pair of comments and determines how similar they are
  - If they are similar enough, the pair is stored in the output file
- During the setup phase
  - `getInputSplits`
    - Creates the **cross-product of the input splits** and sets into a list of **CompositeInputSplits**

HDFS Block Size vs. Hadoop Input Split

- Block Size
  - Physical Location where the data been stored (default size of the HDFS block is 128MB)
  - All blocks of the file are of the same size except the last block
  - HDFS distributes these blocks (chucks) across the cluster
  - Achieves more parallelism and fault tolerance (with replication)
- Input Split
  - Logical representation of block
    - More or less than a Block Size
  - Used during data processing in MapReduce program or other processing techniques
  - InputSplit does not contain actual data but a reference to the data
  - Split acts as a broker between the block and mapper
    - During MapReduce execution, Hadoop scans through the blocks and create InputSplits and assign them to individual mapper

Example of Using Input Split [1/2]

- 1.28GB file
  - Divided into 10 blocks (128MB)
  - InputFormat.getSplits() is responsible for generating the input splits
    - Default: creates one input split for each HDFS block
- Case 1. If input split is not specified
  - HDFS block size will be the split size
  - 10 mappers are initialized to load the file
  - Each mapper loads one block
- Case 2. If the start and end positions of the records are not in the same block?
  - InputSplit provides the Start and End positions (offsets)
  - Mapper is going to load the block of data according to start and end offset values

Example of Using Input Split [2/2]

- 1.28GB file
  - Divided into 10 blocks (128MB)
  - InputFormat.getSplits() is responsible for generating the input splits
    - Default: creates one input split for each HDFS block
- Case 3. If the specified split size was not correct
  - The whole file will form one input split and processed by one map
  - Requires processing very large file
- Case 4. If your resource is limited and you want to limit the number of maps?
  - Specify the Split size as 256MB
  - Only 5 maps will be executed

From Input Splits to Mapper tasks

- Hadoop's Interface `InputFormat<K,V>`
  - `RecordReader<K,V>`
  - `InputSplit[]`
- Validate the input-specification of the job
- Split-up the input file(s) into logical `InputSplits`
  - Assign to an individual Mapper
- Provide the `RecordReader` implementation to the user to glean input records from the logical `InputSplit` for processing Mapper

`RecordReader<K,V>`

Modifier and Type	Method and Description
void	close()Close this InputSplit to future operations.
K	createKey()Create an object of the appropriate type to be used as a key.
V	createValue()Create an object of the appropriate type to be used as a value.
long	getPos()Returns the current position in the input.
float	getProgress()How much of the input has the RecordReader consumed i.e.
boolean	next(K key, V value)Reads the next key/value pair from the input for processing.

InputSplit

Modifier and Type	Method and Description
long	getLength()Get the total number of bytes in the data of the InputSplit.
String[]	getLocations()Get the list of hostnames where the input split is located.

Input Format Code

```
public static class CartesianInputFormat extends FileInputFormat {
  public static final String LEFT_INPUT_FORMAT = "cart.left.inputformat";
  public static final String LEFT_INPUT_PATH = "cart.left.path";
  public static final String RIGHT_INPUT_FORMAT = "cart.right.inputformat";
  public static final String RIGHT_INPUT_PATH = "cart.right.path";

  public static void setLeftInputInfo(JobConf job, Class <? extends
FileInputFormat > inputFormat, String inputPath) {
    job.set(LEFT_INPUT_FORMAT, inputFormat.getCanonicalName());
    job.set(LEFT_INPUT_PATH, inputPath);
  }
  public static void setRightInputInfo(JobConf job, Class <? extends
FileInputFormat > inputFormat, String inputPath) {
    job.set(RIGHT_INPUT_FORMAT, inputFormat.getCanonicalName());
    job.set(RIGHT_INPUT_PATH, inputPath);
  }
```

```
  public InputSplit[] getSplits(JobConf conf, int numSplits) throws IOException
{
    // Get the input splits from both the left and right datasets
    InputSplit[] leftSplits = getInputSplits(conf, conf.get(LEFT_INPUT_FORMAT),
    conf.get(LEFT_INPUT_PATH), numSplits);
    InputSplit[] rightSplits = getInputSplits(conf,
conf.get(RIGHT_INPUT_FORMAT), conf.get( RIGHT_INPUT_PATH), numSplits);
    // Create our CompositeInputSplits, size equal to
    // left.length * right.length
    CompositeInputSplit[] returnSplits =
              new CompositeInputSplit[leftSplits.length * rightSplits.length];
```



```
int i = 0;
    // For each of the left input splits
    for (InputSplit left : leftSplits) {
      // For each of the right input splits
      for (InputSplit right : rightSplits) {
        // Create a new composite input split composing of the two
        returnSplits[i] = new CompositeInputSplit(2);
        returnSplits[i].add(left);
        returnSplits[i].add(right);
        ++i;
} }
```

```
  // Return the composite splits
  LOG.info("Total splits to process: " + returnSplits.length);
  return returnSplits;
}

public RecordReader getRecordReader(InputSplit split, JobConf conf,
Reporter reporter) throws IOException {
  // Create a new instance of the Cartesian record reader
  return new CartesianRecordReader((CompositeInputSplit)split, conf,
reporter);
}
```



```
private InputSplit[] getInputSplits(JobConf conf, String
inputFormatClass, String inputPath, int numSplits) throws
ClassNotFoundException, IOException {
  // Create a new instance of the input format
  FileInputFormat inputFormat =
              (FileInputFormat)ReflectionUtils
                 .newInstance(Class.forName(inputFormatClass), conf);
  // Set the input path for the left data set
  inputFormat.setInputPaths(conf, inputPath);
  // Get the left input splits
  return inputFormat.getSplits(conf, numSplits);
  }
}
```

Record reader

```
public static class CartesianRecordReader < K1, V1, K2, V2 > implements
RecordReader < Text, Text > {
  // Record readers to get key value pairs
  private RecordReader leftRR = null, rightRR = null;
  // Store configuration to re-create the right record reader
  private FileInputFormat rightFIF;
  private JobConf rightConf;
  private InputSplit rightIS;
  private Reporter rightReporter;
  // Helper variables
  private K1 lkey;
  private V1 lvalue;
  private K2 rkey;
  private V2 rvalue;
  private boolean goToNextLeft = true, alldone = false;
```



```
public CartesianRecordReader(CompositeInputSplit split, JobConf conf,
Reporter reporter) throws IOException {
  this.rightConf = conf;
  this.rightIS = split.get(1);
  this.rightReporter = reporter;
  // Create left record reader
  FileInputFormat leftFIF = (FileInputFormat)ReflectionUtils
                .newInstance(Class.forName(conf.get(CartesianInputFormat
                .LEFT_INPUT_FORMAT)),conf);

  leftRR = leftFIF.getRecordReader(split.get(0),conf, reporter);
```

```
// Create right record reader
  rightFIF = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(conf
.get(CartesianInputFormat.RIGHT_INPUT_FORMAT)), conf);
  rightRR = rightFIF.getRecordReader(rightIS, rightConf, rightReporter);
  // Create key value pairs for parsing
  lkey = (K1)this.leftRR.createKey();
  lvalue = (V1)this.leftRR.createValue();
  rkey = (K2)this.rightRR.createKey();
  rvalue = (V2)this.rightRR.createValue();
}
```



```
public boolean next(Text key, Text value) throws IOException {
  do {
  // If we are to go to the next left key/value pair
  if (goToNextLeft) {
    // Read the next key value pair, false means no more pairs
    if (!leftRR.next(lkey, lvalue)) {
      // If no more, then this task is nearly finished
      alldone = true; break;
    }else {
      // If we aren't done, set the value to the key and set
      // our flags
      key.set(lvalue.toString());
      goToNextLeft = alldone = false;
      // Reset the right record reader
      this.rightRR = this.rightFIF.getRecordReader(this.rightIS, this.rightConf,
this.rightReporter);
     }
}
```

```
  // Read the next key value pair from the right data set
  if (rightRR.next(rkey, rvalue)) {
  // If success, set the value
  value.set(rvalue.toString());
  } else {
    // Otherwise, this right data set is complete
    // and we should go to the next left pair
    goToNextLeft = true;
  }
   // This loop will continue if we finished reading key/value
   // pairs from the right data set
  } while (goToNextLeft);
  // Return true if a key/value pair was read, false otherwise
  return !alldone;
 }
}
```


Driver Code

```
public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {
  // Configure the join type
  JobConf conf = new JobConf("Cartesian Product");
  conf.setJarByClass(CartesianProduct.class);
  conf.setMapperClass(CartesianMapper.class);
  conf.setNumReduceTasks(0);
  conf.setInputFormat(CartesianInputFormat.class);
  // Configure the input format
  CartesianInputFormat.setLeftInputInfo(conf, TextInputFormat.class,
args[0]);
  CartesianInputFormat.setRightInputInfo(conf, TextInputFormat.class,
args[0]);
  TextOutputFormat.setOutputPath(conf, new Path(args[1]));
```

Driver Code

```
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);
  RunningJob job = JobClient.runJob(conf);
  while (!job.isComplete()) {
    Thread.sleep(1000);
  }
  System.exit(job.isSuccessful() ? 0 : 1);
}
```


Mapper Code (comparing comments)

```
public static class CartesianMapper extends MapReduceBase implements
Mapper < Text, Text, Text, Text > {
  private Text outkey = new Text();
  public void map(Text key, Text value, OutputCollector < Text, Text >
output, Reporter reporter) throws IOException {
    // If the two comments are not equal
    if (!key.toString().equals(value.toString())) {
      String[] leftTokens = key.toString().split("\\s");
      String[] rightTokens = value.toString().split("\\s");
      HashSet < String > leftSet =
             new HashSet < String >(Arrays.asList(leftTokens));
      HashSet < String > rightSet =
             new HashSet < String >(Arrays.asList(rightTokens));
      int sameWordCount = 0;
      StringBuilder words = new StringBuilder();
```

```
      for (String s : leftSet) {
          if (rightSet.contains(s)) {
             words.append(s + ",");
             ++sameWordCount;
           }
      }
      // If there are at least three words, output
      if (sameWordCount > 2) {
        outkey.set(words + "\t" + key);
        output.collect(outkey, value);
      }
    }
  }
}
```


Questions?

Questions?

Topics to be covered

- MapReduce Design Pattern III. Organization Patterns

Part 1. Large Scale Data Analytics

Design Pattern 3: Data Organization Patterns

Data Organization Patterns

- Reorganizing data
  - Partitioning, Sharding and Sorting

1. "Structured" to "hierarchical" pattern
2. Partitioning and binning patterns
3. Total order sorting

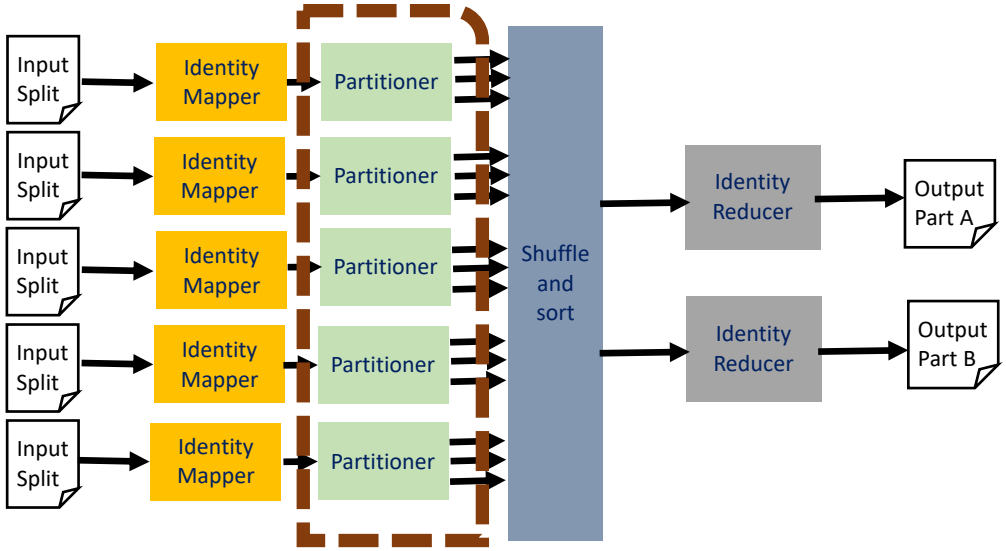
MapReduce Design Patterns III: Data Organization Patterns

2. Partitioning pattern

Partitioning Pattern

- Moves the records into **categories**
  - But it doesn't really care about the order of records
  - Shards, partitions, or bins
- e.g. Partitioning **by date**
  - Groups data based on date
  - Given a set of user information, **partition the records** based on **the year of last access date**, one partition per year

Structure of the partitioning pattern



What is a "Partitioner"?

- Partitioner allows distributing how outputs from the map stage are send to the reducers
- The total number of partitions is same as the total number of reducers
- It reduces the amount of time of reducer job

```
public int getPartition(K key, V value,
    int numReduceTasks) {
         return(key.hashCode()
                & Integer.MAX_VALUE) % numReduceTasks;
}
```

Custom Partitioner

- Parameters
  - Key, Value, numPartitions
- Optimize the mapreduce programs and executed the given problem as well as possible
- Can divide the output files how it required

```
public abstract class Partitioner<KEY, VALUE> {
       public abstract int getPartition (KEY key, VALUE value,
                        int numPartitions){
        return;
       }
}
```


Partitioner [1/4]

- Partitions the key-value pairs of **intermediate Map-outputs**
  - Uses a **user-defined condition**
- e.g. Process the input dataset to find the **highest salaried employee** by **gender** in different **age groups**

1201	James	45	Male	50000
1202	Matthew	40	Male	51000
1203	Kevin	34	Male	45000
1204	Maria	30	Female	43000
1205	Julia	20	Female	23000
1206	Lavanya	25	Female	42000
1207	Joseph	19	Male	19000
1208	Steve	22	Male	32000
1209	Max	24	Male	33000
1210	Jenifer	28	Female	37000
1211	Nick	18	Male	19000
1212	MaryAnn	33	Female	56000
1213	Kelvin	39	Male	65000

Partitioner [2/4]

- Map Tasks
  - Input
    - Dummy key, data
    - (Dummy_key, "1201 \t James \t 45 \t male \t\ 50000")
  - Method
    - Read the value and extract gender information

```
String[] str = value.toString().split("\t", -3);
String gender = str[3];
```

  - Output
    - Gender data and value

```
context.write(new Text(gender), value);
```


Partitioner [3/4]

- Partitioner Task
  - Dividing the data from the map task into **segments**
  - Input
    - A collection of key-value pairs from the map task
    - Key: gender, value: whole record data
  - Method
    - Read the **age field** and apply conditions
  - Output
    - The **data of key-value pairs are segmented into three collections** of key-value pairs
    - The reducer works **individually on each collection**

```
int age = Integer.parseInt(str[2]);
if(age<=20) {
   return 0;
} else if(age>20 && age<=30) {
   return 1 % numReduceTasks;}
else {
   return 2 % numReduceTasks;
}
```

Partitioner [4/4]

- Reduce Task
  - The **number of partitions(segments)** is **equal** to the **number of reduce tasks**
    - **NOTE: "keys" are still sorted and passed to the reducer**
    - **In this example, there will be 3 reducers and each reducer will get <"male", [list of values]> and <"female", [list of values]>**

Partitioning Pattern: Example

- Partitioning **by date**
  - Groups data based on date
  - Given a set of user information, **partition the records** based on **the year of last access date**, one partition per year

Driver Code (last access date partitioner)

```
…
// Set custom partitioner and min last access date
job.setPartitionerClass(LastAccessDatePartitioner.class);
LastAccessDatePartitioner.setMinLastAccessDate(job, 2008);

// Last access dates span between 2008-2011, or 4 years
job.setNumReduceTasks(4);
…
```


Mapper Code (last access date partitioner)

```
public static class LastAccessDateMapper extends Mapper < Object, Text, IntWritable,
Text > {
  // This object will format the creation date string into a Date object
  private final static SimpleDateFormat frmt =
     new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS");
  private IntWritable outkey = new IntWritable();
  protected void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
    Map < String, String > parsed =
          MRDPUtils.transformXmlToMap(value .toString());
    // Grab the last access date
    String strDate = parsed.get("LastAccessDate");
    // Parse the string into a Calendar object
    Calendar cal = Calendar.getInstance();
    cal.setTime(frmt.parse(strDate));
    outkey.set(cal.get(Calendar.YEAR)); // Write out the year with the input value
    context.write(outkey, value);
} }
```

Partitioner code (last access date partitioner)

```
public static class LastAccessDatePartitioner extends Partitioner < IntWritable,
Text > implements Configurable {
  private static final String MIN_LAST_ACCESS_DATE_YEAR =
                                          "min.last.access.date.year";
  private Configuration conf = null;
  private int minLastAccessDateYear = 0;
  public int getPartition(IntWritable key, Text value, int numPartitions) {
    return key.get() - minLastAccessDateYear;
  }
  public Configuration getConf() {
    return conf;
  }
  public void setConf(Configuration conf) {
    this.conf = conf;
    minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
  }
```


Partitioner code (last access date partitioner)

```
  public static void setMinLastAccessDate(Job job, int
minLastAccessDateYear) {
    job.getConfiguration()
            .setInt(MIN_LAST_ACCESS_DATE_YEAR,
minLastAccessDateYear);
  }
}
```

Reducer Code (last access date partitioner)

```
public static class ValueReducer extends Reducer < IntWritable, Text,
Text, NullWritable > {
      protected void reduce( IntWritable key, Iterable < Text > values,
Context context) throws IOException, InterruptedException {
            for (Text t : values) {
                  context.write(t, NullWritable.get());
            }
      }
}
```


Worksheet #4-1

- Assume that your application has more 80% of new users in recent years
- Design your partitioner to resolve this problem.

Unevenly distributed partitions

- Observation
  - Recent years will have more users
- Provide finer grained segmentations to the recent years
  - e.g. Monthly partitions for recent 3 years

MapReduce Design Patterns II: Filtering Patterns

3. Total Order Sorting Pattern

Total Order Sorting Pattern

- Sorts your data
  - e.g. Sorting 1TB of numeric values
  - e.g. Sorting **comments by userID** and you have a **million users**

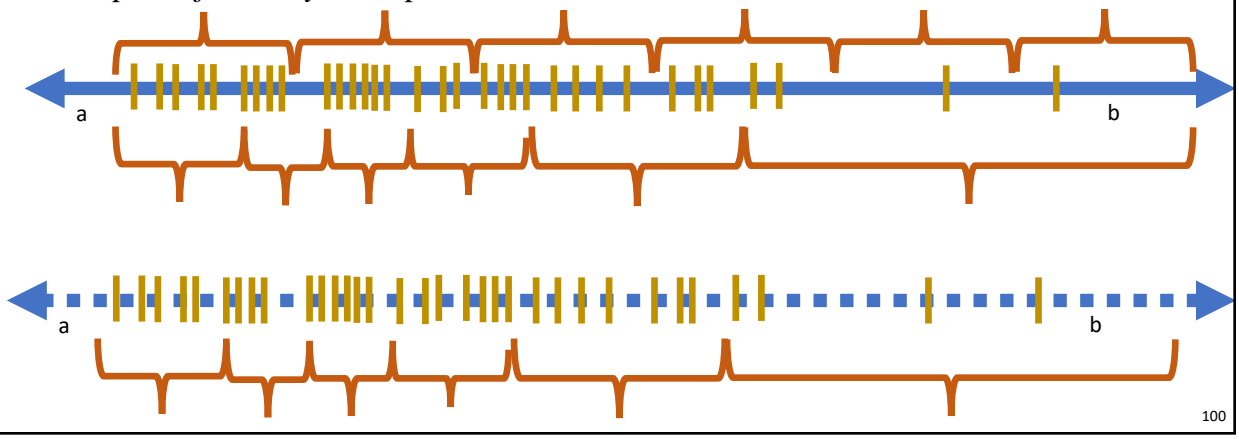
Structure of Total Order Sorting Pattern

- Two phases
  - **Analysis** phase
    - Determines the ranges
  - **Sorting** phase
    - Actually sorts the data

With only one MapReduce Job?
No. You can use multiple MapReduce Jobs for this problem!

What do we do in the "Analysis phase"?

- We will get "splits" of Tera Bytes of numbers
- *sample[i-j]<=key<sample[i]* are sent to reducer *i*

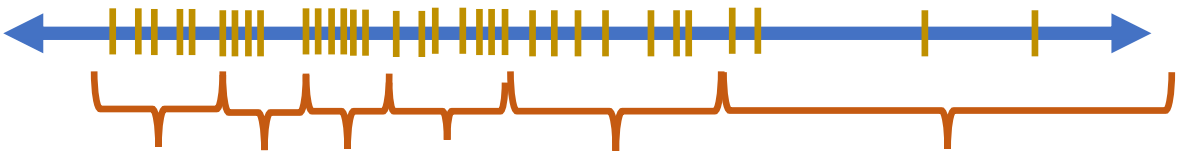


Structure of the Total Order Sorting Pattern
- Analysis phase

- Performs a **simple random sampling**
- Generates outputs with **the sort key as its output keys**
  - Data will show up as sorted at the reducer
- Sampling rate?
  - Assume that the number of records in the entire dataset is known (or can be estimated)
  - If you plan on running the order with a thousand **reducers**
    - Sampling about a hundred thousand records will be enough
- Only **one reducer** will be used
  - Collects the sort keys together into a sorted list
  - The list of **sorted keys will be sliced into the data range boundaries**

What do we do in the "Sorting phase"?

- We will sort data within the splits



Structure of Total Order Sorting Pattern
- Sorting phase

- Mapper extracts the sort key
  - Stores the sort key to the "value"
- Custom partitioner
  - Use `TotalOrderPartitioner` (Hadoop API)
  - Takes the data ranges from the partition file and decides which reducer to send the data
  - Dynamic and load balanced
  - **`setPartitionerFile(Configration conf)`**
    - Get the path to the SequenceFile string the sorted partition keyset
- Reducer
  - The number of reducers needs to be **equal to the number of partitions**

TeraSort Benchmark

- The most well-known Hadoop benchmark
- In 2008, Yahoo! Set a record by sorting 1 TB of data in 209 seconds
  - Hadoop cluster with 910 nodes
  - Owen O'Malley of the Yahoo!
- In 2009, Yahoo! Sorted 1PB of data in 16 hours
  - Hadoop cluster of 3800 nodes
  - For 1TB, it took 62 seconds
  - http://sortbenchmark.org/YahooHadoop.pdf

TeraSort Benchmark APIs

- TeraGen
  - MR to generate the data
- TeraSort
  - Samples the input data and uses MR to sort the data into a total order
- TeraValidate
  - MR that validates the output
- TeraSort is a standard MapReduce with a custom partitioner that uses a sorted list of N-1 sorted sampled keys that define the key range for each reduce
- $sample[i-j]<=key<sample[i]$ are sent to reducer i
  - Total 1,000 lines of java code

Questions?

Week 5 – Lecture 1
MapReduce Design Pattern IV: Join Patterns

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- Reducer vs. Reducer Task?
- Programming Assignment 1
  - Demo for grading
- Attendance
  - We update once a week
  - Contact GTAs: cs435@cs.colostate.edu

1

1

Topics discussed in the previous class

- MapReduce Design Pattern IV. Join Patterns

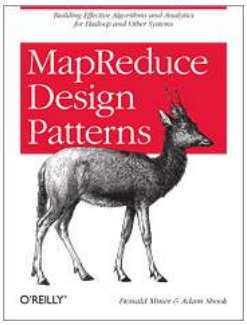
1. Reduce Side Join Pattern
2. Replicated Join Pattern

Topics to be covered

- MapReduce Design Pattern IV. Join Patterns
  - Composite Join Pattern
  - Cartesian Product Pattern

This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012



Part 1. Large Scale Data Analytics

Design Pattern 4: Join Patterns

Join Patterns

- Data is **all over** the place
- "Joins" allow users to **create a smaller reference set** or **filter out** or **select dataset to discover interesting relationships** across datasets
- Joining a terabyte of data onto another terabyte dataset could require up to **two terabytes of bandwidth!**
  - That's before any actual join logic can be done!

1. Reduce Side Join Pattern
2. Replicated Join Pattern
3. Composite Join Pattern
4. Cartesian Product Pattern

REVIEW

Brief review on `join`s

- A join is an operation that combines records from two or more datasets **based on a field or set of fields**
  - Foreign key
- The **foreign key** is the field in a relational table that matches the column of another table
  - Used as a means to cross-reference between tables
- Inner join
- Left outer join
- Right outer join
- Full outer join
- Anti join

REVIEW

Reduce Side Join Pattern

- **Most straightforward implementation** of a join in MapReduce
- Requires a **large amount of network bandwidth**
  - Bulk of the data is sent to the reduce phase
  - If you have resources available this will be a possible solution
- Worksheet question 2:

Assume that you perform join operation between two very large tables A and B using MapReduce. Table A contains the following attributes:

`user_ID, address, phone_number, DOB`

Table B contains the following attributes: `user_ID, post_ID, comment_ID`

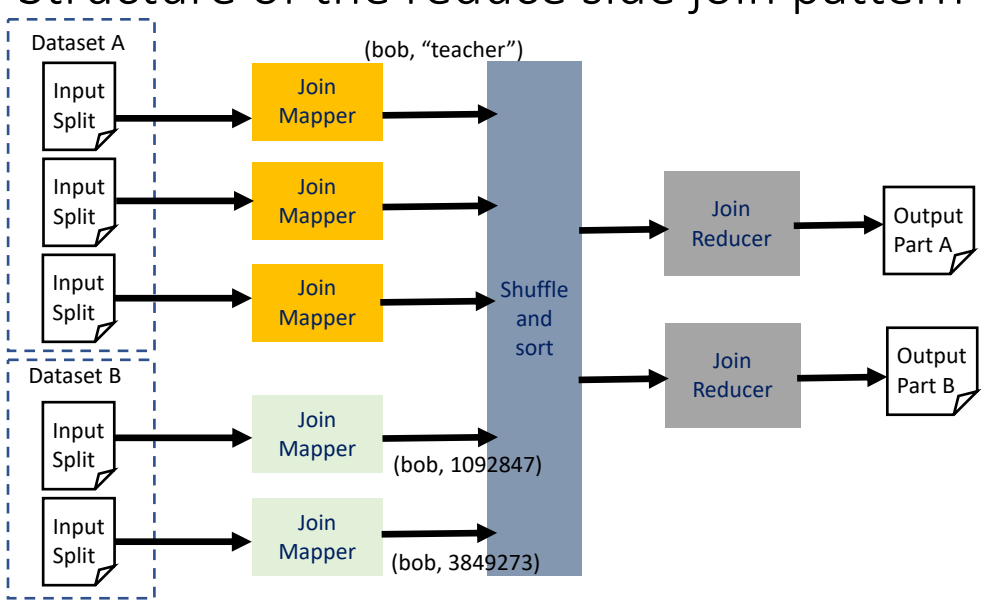
Here, `user_ID` is the primary key for the join operation. Design a MapReduce application for this operation.

8

8

REVIEW

Structure of the reduce side join pattern



9

9

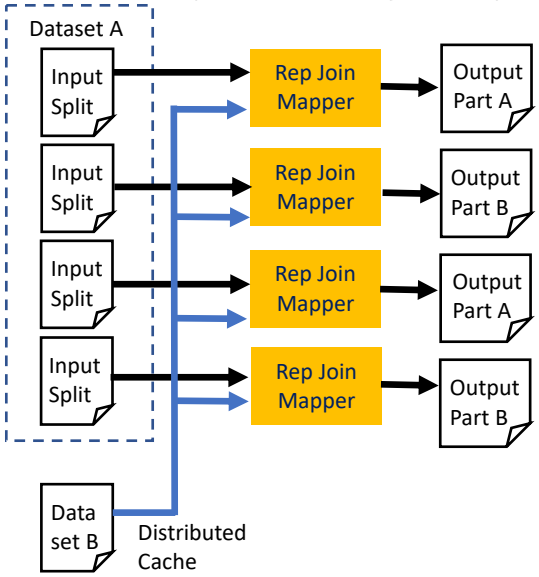
REVIEW

Replicated Join

- Special type of join operation between **one large and small data set** that can be performed on the map-side
  - e.g. Mapping with the zip code, or area code
- Mapper
  - Reads all files from the **distributed cache** during the **setup phase**
    - **Sorting them in to in-memory lookup tables**
  - Performs mapper process
    - Joining data
  - If the foreign key is not found in the in-memory structure?
    - The record is either omitted or output (based on the join type)
- **No combiner/partitioner/reducer needed**

REVIEW

Structure of the replicated join pattern



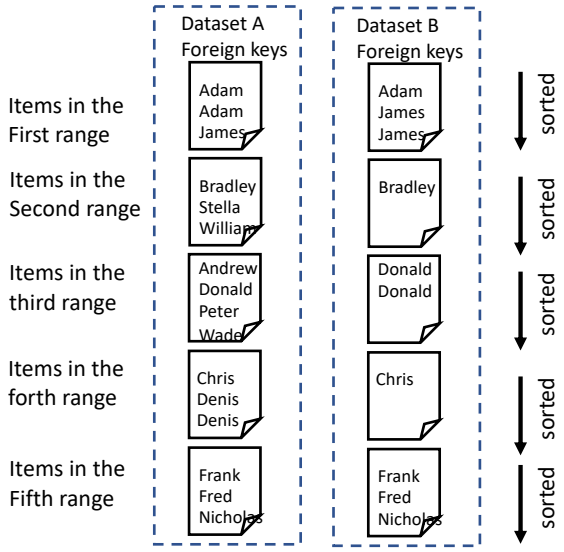
MapReduce Design Patterns IV: Join Patterns

3. Composite Join

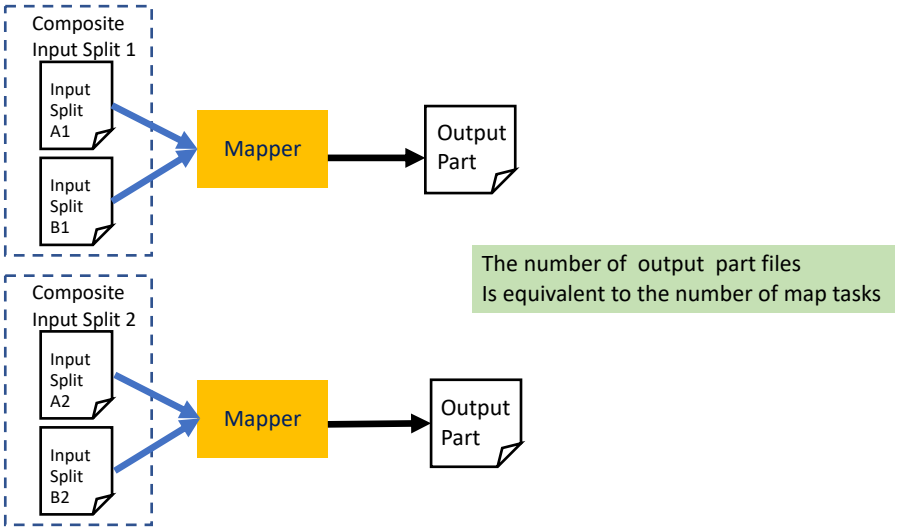
Composite Join

- Joins **very large datasets together**
  - And if the datasets are sorted by foreign key
- **No shuffle and sort needed**
- Each input dataset must be **partitioned and sorted in a specific way** and **divided into the same number of partitions**
  - **e.g. transactions, log files**

Structure of the composite join pattern



Structure of the composite join pattern



Joining process (Inner Join)- Within a group

UserID	Credit Card
12	Master
13	Master
14	Visa
15	Visa
15	Discover
17	Master
17	Visa
17	Discover
20	Visa
20	American Express

UserID	Payments
11	FC Utility
12	FC Utility
13	FC Utility
13	Coloradoan
13	Netflix
13	Xcel
13	Honda Finance
14	Xcel
19	FC Utility
20	Xcel

(UserID-A, Credit card, UserID-B, Payments)

(12, Master, 12, FC Utility)
(13, Master, 13, FC Utility)
(13, Master, 13, Coloradoan)
(13, Master, 13, Netflix)
(13, Master, 13, Xcel)
(13, Master, 13, Honda Finance)
(14, Visa, 14, Xcel)
(20, Visa, 14, Xcel)
(20, American Express, 14, Xcel)

```
A.userID = A.0, B.userID = B.0
While (A.userID <= A.max && B.userID<=B.max) {.
   If A.userID == B.userID write a joined tuple
   Else If  (A.userID >= B.userID) B.userID++
    else if (A.userID < B.userID) A.userID++
}
```


Joining process (Inner Join)

For N items in the left dataset and M items in the right dataset
Naïve approach (without sorting and exact matching partitioning) takes O(NM)
With the composite join, it takes O(N)

UserID	Credit Card
12	Master
13	Master
14	Visa
15	Visa
15	Discover
17	Master
17	Visa
17	Discover
20	Visa
20	American Express

UserID	Payments
11	FC Utility
12	FC Utility
13	FC Utility
13	Coloradoan
13	Netflix
13	Xcel
13	Honda Finance
14	Xcel
19	FC Utility
20	Xcel

Joining process (Inner Join)

Worksheet 5-A-Q1

Assume that there are N records in the credit card dataset and M records in the payment dataset.
If these datasets are not sorted, what will be the computational complexity of this process?

a. O(N) b. O(M) c. O(NM) d. O(1)

UserID	Credit Card
12	Master
13	Master
14	Visa
15	Visa
15	Discover
17	Master
17	Visa
17	Discover
20	Visa
20	American Express

UserID	Payments
11	FC Utility
12	FC Utility
13	FC Utility
13	Coloradoan
13	Netflix
13	Xcel
13	Honda Finance
14	Xcel
19	FC Utility
20	Xcel

Joining process (Inner Join)

Worksheet 5-A- Q2

Assume that there are N records in the credit card dataset and M records in the payment dataset.
If these datasets are sorted by the foreign key and we implement the algorithm that we discussed in the previous slide, what will be the computational complexity of this process?

a. O(N) b. O(M) c. O(NM) d. O(1)

UserID	Credit Card
12	Master
13	Master
14	Visa
15	Visa
15	Discover
17	Master
17	Visa
17	Discover
20	Visa
20	American Express

UserID	Payments
11	FC Utility
12	FC Utility
13	FC Utility
13	Coloradoan
13	Netflix
13	Xcel
13	Honda Finance
14	Xcel
19	FC Utility
20	Xcel

Joining process (Inner Join)-Between groups

Worksheet 4-A: Assume that your MapReduce involves k different groups within the sorted datasets A and B. There is no overlapping ranges of the keys between groups. If your MapReduce applies the Composite Join pattern, what would be the expected communication
between the groups?

a. If the matching groups from dataset A and B cover the same range, no communication between groups will be needed.
b. Only final tuples will need to interact with other groups.
c. There will be no communication required.
d. Only redundant foreign keys will cause interactions between the groups.

UserID	Credit Card
12	Master
…	…
2000	American Express

UserID	Payments
11	FC Utility
…	…
1998	FC Utility

UserID	Credit Card
2021	Visa
…	…
2999	American Express

UserID	Payments
2022	FC Utility
…	…
2987	FC Utility

Joining process (Inner Join)-Between groups

UserID	Credit Card
12	Master
…	…
2000	American Express

UserID	Payments
11	FC Utility
…	…
1998	FC Utility

UserID	Credit Card
2021	Visa
…	…
2999	American Express

UserID	Payments
2022	FC Utility
…	…
2987	FC Utility

UserID	Credit Card
3002	Master
…	…
3909	Visa

UserID	Payments
3002	FC Utility
…	…
4000	FC Utility

Hash function h(fk)= fk mod 10
There is no overlapping range
Between groups.
No computation across the
sub-tables will be needed

Example: Composite user comment join

- Preconditions
  - User and comment datasets should be **preprocessed by MapReduce**
  - **`CompositeInputFormat`** will be used
  - Output should use the **`TextOutputFormat`**
    - Key: userID
    - Value: userXML or comment XML
    - Hadoops `KeyValueTextOutputFormat` can parse these information
- Note: Since **`CompositeInputFormat`** uses `Text` objects as the key for comparisons
  - **Align your keys as `Text` objects** (rather than `LongWritable`)
  - e.g. "12345" comes before "2"

Driver Code [1/2]

```
public static void main( String[] args) throws Exception {
  Path userPath = new Path(args[0]);
  Path commentPath = new Path(args[1]);
  Path outputDir = new Path(args[2]);
  String joinType = args[3];
  JobConf conf = new JobConf("CompositeJoin");
  conf.setJarByClass(CompositeJoinDriver.class)
  conf.setMapperClass(CompositeMapper.class);
  conf.setNumReduceTasks(0);
  // Set the input format class to a CompositeInputFormat  ass.
  // The CompositeInputFormat will parse all of our input  les
and output
  // records to our mapper.
  conf.setInputFormat(CompositeInputFormat.class);
```

This input format scans through the left dataset and Identifies matching in the right dataset. It generates input value, (item from The left dataset, item from the right Dataset)

Driver Code [2/2]

```
  // The composite input format join expression
  // will set how the records
  // are going to be read in, and in what input format.
  conf.set("mapred.join.expr",
          CompositeInputFormat.compose(joinType,
          KeyValueTextInputFormat.class, userPath, commentPath));
  TextOutputFormat.setOutputPath(conf, outputDir);
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);
  RunningJob job = JobClient.runJob(conf);
  while (!job.isComplete()) {
     Thread.sleep(1000);
  }
  System.exit(job.isSuccessful() ? 0 : 1);
}
```


Mapper Code

```
public static class CompositeMapper extends
MapReduceBase implements Mapper < Text, TupleWritable,
Text, Text > {
  public void map(Text key, TupleWritable value,
OutputCollector < Text, Text > output, Reporter
reporter) throws IOException {
    // Get the first two elements in the tuple and
output them
    output.collect((Text)value.get(0),
                   (Text)value.get(1));
  }
}
```

All of the matching has been done in CompositeInputFormat already.

MapReduce Design Patterns IV: Join Patterns

4. Cartesian Product

Cartesian Product

- Pairs **every record** from multiple inputs
- Applicability
  - **You want to analyze relationships between all pairs of records**
  - e.g. Correlation analysis

Cartesian Product

Dataset A 1 2 3 4 5 6 7 8 9 10

Dataset B A B C D E F G H I J

(1,A)	(2,A)	(3,A)	(4,A)	(5,A)	(6,A)	(7,A)	(8,A)	(9,A)	(10,A)
(1,B)	(2,B)	(3,B)	(4,B)	(5,B)	(6,B)	(7,B)	(8,B)	(9,B)	(10,B)
(1,C)	(2,C)	(3,C)	(4,C)	(5,C)	(6,C)	(7,C)	(8,C)	(9,C)	(10,C)
(1,D)	(2,D)	(3,D)	(4,D)	(5,D)	(6,D)	(7,D)	(8,D)	(9,D)	(10,D)
(1,E)	(2,E)	(3,E)	(4,E)	(5,E)	(6,E)	(7,E)	(8,E)	(9,E)	(10,E)
(1,F)	(2,F)	(3,F)	(4,F)	(5,F)	(6,F)	(7,F)	(8,F)	(9,F)	(10,F)
(1,G)	(2,G)	(3,G)	(4,G)	(5,G)	(6,G)	(7,G)	(8,G)	(9,G)	(10,G)
(1,H)	(2,H)	(3,H)	(4,H)	(5,H)	(6,H)	(7,H)	(8,H)	(9,H)	(10,H)
(1, I)	(2, I)	(3, I)	(4, I)	(5, I)	(6, I)	(7, I)	(8, I)	(9, I)	(I0, I)
(1, J)	(2, J)	(3, J)	(4, J)	(5, J)	(6, J)	(7, J)	(8, J)	(9, J)	(I0, J)

A B C D E F G H I J

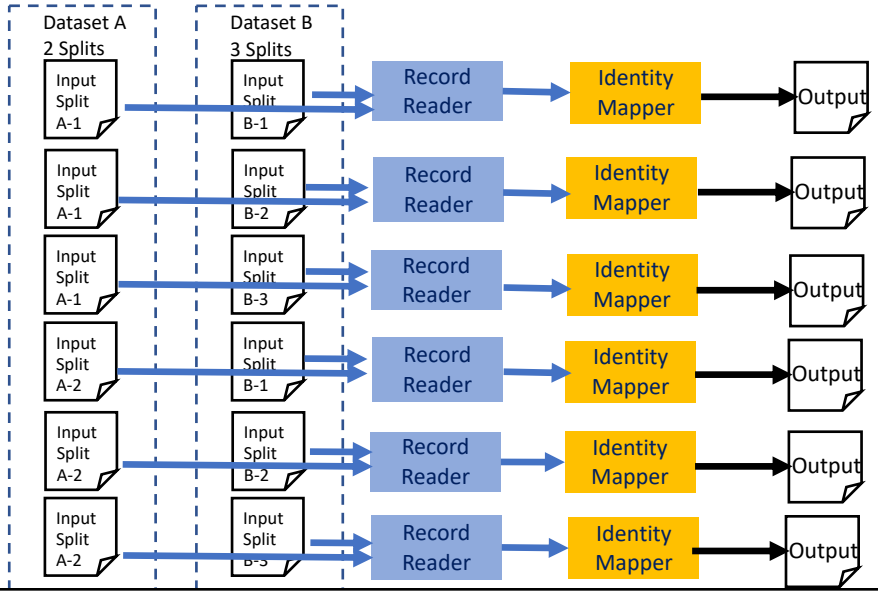
Cartesian Product

1 2 3 4 5 6 7 8 9 10

A B C D E F G H I J

(1,A)	(2,A)	(3,A)	(4,A)	(5,A)	(6,A)	(7,A)	(8,A)	(9,A)	(10,A)
(1,B)	(2,B)	(3,B)	(4,B)	(5,B)	(6,B)	(7,B)	(8,B)	(9,B)	(10,B)
(1,C)	(2,C)	(3,C)	(4,C)	(5,C)	(6,C)	(7,C)	(8,C)	(9,C)	(10,C)
(1,D)	(2,D)	(3,D)	(4,D)	(5,D)	(6,D)	(7,D)	(8,D)	(9,D)	(10,D)
(1,E)	(2,E)	(3,E)	(4,E)	(5,E)	(6,E)	(7,E)	(8,E)	(9,E)	(10,E)
(1,F)	(2,F)	(3,F)	(4,F)	(5,F)	(6,F)	(7,F)	(8,F)	(9,F)	(10,F)
(1,G)	(2,G)	(3,G)	(4,G)	(5,G)	(6,G)	(7,G)	(8,G)	(9,G)	(10,G)
(1,H)	(2,H)	(3,H)	(4,H)	(5,H)	(6,H)	(7,H)	(8,H)	(9,H)	(10,H)
(1, I)	(2, I)	(3, I)	(4, I)	(5, I)	(6, I)	(7, I)	(8, I)	(9, I)	(I0, I)
(1, J)	(2, J)	(3, J)	(4, J)	(5, J)	(6, J)	(7, J)	(8, J)	(9, J)	(I0, J)

Structure of the Cartesian Product



Performance Analysis

- Cartesian product requires a massive data space
  - To compute a Cartesian product between N records and M records
  - N x M data points are created
    - Requires many Map slots and a very long time

Example: Cartesian Product

- Comments comparison
  - Inspects a pair of comments and determines how similar they are
  - If they are similar enough, the pair is stored in the output file
- During the setup phase
  - `getInputSplits`
    - Creates the **cross-product of the input splits** and sets into a list of **CompositeInputSplits**

Cartesian Product

1 2 3 4 5 6 7 8 9 10 -> Q | X

A B C D E F G H I J -> Y | Z

Q X

Y

Z

(1,A)	(2,A)	(3,A)	(4,A)	(5,A)	(6,A)	(7,A)	(8,A)	(9,A)	(10,A)
(1,B)	(2,B)	(3,B)	(4,B)	(5,B)	(6,B)	(7,B)	(8,B)	(9,B)	(10,B)
(1,C)	(2,C)	(3,C)	(4,C)	(5,C)	(6,C)	(7,C)	(8,C)	(9,C)	(10,C)
(1,D)	(2,D)	(3,D)	(4,D)	(5,D)	(6,D)	(7,D)	(8,D)	(9,D)	(10,D)
(1,E)	(2,E)	(3,E)	(4,E)	(5,E)	(6,E)	(7,E)	(8,E)	(9,E)	(10,E)
(1,F)	(2,F)	(3,F)	(4,F)	(5,F)	(6,F)	(7,F)	(8,F)	(9,F)	(10,F)
(1,G)	(2,G)	(3,G)	(4,G)	(5,G)	(6,G)	(7,G)	(8,G)	(9,G)	(10,G)
(1,H)	(2,H)	(3,H)	(4,H)	(5,H)	(6,H)	(7,H)	(8,H)	(9,H)	(10,H)
(1, I)	(2, I)	(3, I)	(4, I)	(5, I)	(6, I)	(7, I)	(8, I)	(9, I)	(I0, I)
(1, J)	(2, J)	(3, J)	(4, J)	(5, J)	(6, J)	(7, J)	(8, J)	(9, J)	(I0, J)

QY XY

QZ XZ

Then, how do we manage the input splits?

HDFS Block Size vs. Hadoop Input Split

- Block Size
  - Physical Location where the data been stored (default size of the HDFS block is 128MB)
  - All blocks of the file are of the same size except the last block
  - HDFS distributes these blocks (chucks) across the cluster
  - Achieves more parallelism and fault tolerance (with replication)
- Input Split
  - Logical representation of block
    - More or less than a Block Size
  - Used during data processing in MapReduce program or other processing techniques
  - InputSplit does not contain actual data but a reference to the data
  - Split acts as a broker between the block and mapper
    - During MapReduce execution, Hadoop scans through the blocks and create InputSplits and assign them to individual mapper

Example of Using Input Split [1/2]

- 1.28GB file
  - Divided into 10 blocks (128MB)
  - InputFormat.getSplits() is responsible for generating the input splits
    - Default: creates one input split for each HDFS block
- Case 1. If input split is not specified
  - HDFS block size will be the split size
  - 10 mappers are initialized to load the file
  - Each mapper loads one block
- Case 2. If the start and end positions of the records are not in the same block?
  - InputSplit provides the Start and End positions (offsets)
  - Mapper is going to load the block of data according to start and end offset values

Example of Using Input Split [2/2]

- 1.28GB file
  - Divided into 10 blocks (128MB)
  - InputFormat.getSplits() is responsible for generating the input splits
    - Default: creates one input split for each HDFS block
- Case 3. If the specified split size was not correct
  - The whole file will form one input split and processed by one map
  - Requires processing very large file
- Case 4. If your resource is limited and you want to limit the number of maps?
  - Specify the Split size as 256MB
  - Only 5 maps will be executed

From Input Splits to Mapper tasks

- Hadoop's Interface `InputFormat<K,V>`
  - `RecordReader<K,V>`
  - `InputSplit[]`
- Validate the input-specification of the job
- Split-up the input file(s) into logical `InputSplits`
  - Assign to an individual Mapper
- Provide the `RecordReader` implementation to the user to glean input records from the logical `InputSplit` for processing Mapper

RecordReader<K,V>

Modifier and Type	Method and Description
void	close()Close this InputSplit to future operations.
K	createKey()Create an object of the appropriate type to be used as a key.
V	createValue()Create an object of the appropriate type to be used as a value.
long	getPos()Returns the current position in the input.
float	getProgress()How much of the input has the RecordReader consumed i.e.
boolean	next(K key, V value)Reads the next key/value pair from the input for processing.

InputSplit

Modifier and Type	Method and Description
long	getLength()Get the total number of bytes in the data of the InputSplit.
String[]	getLocations()Get the list of hostnames where the input split is located.

Input Format Code

```
public static class CartesianInputFormat extends FileInputFormat {
  public static final String LEFT_INPUT_FORMAT = "cart.left.inputformat";
  public static final String LEFT_INPUT_PATH = "cart.left.path";
  public static final String RIGHT_INPUT_FORMAT = "cart.right.inputformat";
  public static final String RIGHT_INPUT_PATH = "cart.right.path";

  public static void setLeftInputInfo(JobConf job, Class <? extends
FileInputFormat > inputFormat, String inputPath) {
    job.set(LEFT_INPUT_FORMAT, inputFormat.getCanonicalName());
    job.set(LEFT_INPUT_PATH, inputPath);
  }
  public static void setRightInputInfo(JobConf job, Class <? extends
FileInputFormat > inputFormat, String inputPath) {
    job.set(RIGHT_INPUT_FORMAT, inputFormat.getCanonicalName());
    job.set(RIGHT_INPUT_PATH, inputPath);
  }
```



```
  public InputSplit[] getSplits(JobConf conf, int numSplits) throws IOException
{
    // Get the input splits from both the left and right datasets
    InputSplit[] leftSplits = getInputSplits(conf, conf.get(LEFT_INPUT_FORMAT),
    conf.get(LEFT_INPUT_PATH), numSplits);
    InputSplit[] rightSplits = getInputSplits(conf,
conf.get(RIGHT_INPUT_FORMAT), conf.get( RIGHT_INPUT_PATH), numSplits);
    // Create our CompositeInputSplits, size equal to
    // left.length * right.length
    CompositeInputSplit[] returnSplits =
              new CompositeInputSplit[leftSplits.length * rightSplits.length];
```

```
int i = 0;
    // For each of the left input splits
    for (InputSplit left : leftSplits) {
      // For each of the right input splits
      for (InputSplit right : rightSplits) {
        // Create a new composite input split composing of the two
        returnSplits[i] = new CompositeInputSplit(2);
        returnSplits[i].add(left);
        returnSplits[i].add(right);
        ++i;
} }
```



```
  // Return the composite splits
  LOG.info("Total splits to process: " + returnSplits.length);
  return returnSplits;
}

public RecordReader getRecordReader(InputSplit split, JobConf conf,
Reporter reporter) throws IOException {
  // Create a new instance of the Cartesian record reader
  return new CartesianRecordReader((CompositeInputSplit)split, conf,
reporter);
}
```

```
private InputSplit[] getInputSplits(JobConf conf, String
inputFormatClass, String inputPath, int numSplits) throws
ClassNotFoundException, IOException {
  // Create a new instance of the input format
  FileInputFormat inputFormat =
              (FileInputFormat)ReflectionUtils
                 .newInstance(Class.forName(inputFormatClass), conf);
  // Set the input path for the left data set
  inputFormat.setInputPaths(conf, inputPath);
  // Get the left input splits
  return inputFormat.getSplits(conf, numSplits);
  }
}
```


Record reader

```
public static class CartesianRecordReader < K1, V1, K2, V2 > implements
RecordReader < Text, Text > {
  // Record readers to get key value pairs
  private RecordReader leftRR = null, rightRR = null;
  // Store configuration to re-create the right record reader
  private FileInputFormat rightFIF;
  private JobConf rightConf;
  private InputSplit rightIS;
  private Reporter rightReporter;
  // Helper variables
  private K1 lkey;
  private V1 lvalue;
  private K2 rkey;
  private V2 rvalue;
  private boolean goToNextLeft = true, alldone = false;
```

```
public CartesianRecordReader(CompositeInputSplit split, JobConf conf,
Reporter reporter) throws IOException {
  this.rightConf = conf;
  this.rightIS = split.get(1);
  this.rightReporter = reporter;
  // Create left record reader
  FileInputFormat leftFIF = (FileInputFormat)ReflectionUtils
                .newInstance(Class.forName(conf.get(CartesianInputFormat
                .LEFT_INPUT_FORMAT)),conf);

  leftRR = leftFIF.getRecordReader(split.get(0),conf, reporter);
```



```
// Create right record reader
  rightFIF = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(conf
.get(CartesianInputFormat.RIGHT_INPUT_FORMAT)), conf);
  rightRR = rightFIF.getRecordReader(rightIS, rightConf, rightReporter);
  // Create key value pairs for parsing
  lkey = (K1)this.leftRR.createKey();
  lvalue = (V1)this.leftRR.createValue();
  rkey = (K2)this.rightRR.createKey();
  rvalue = (V2)this.rightRR.createValue();
}
```

```
public boolean next(Text key, Text value) throws IOException {
  do {
  // If we are to go to the next left key/value pair
  if (goToNextLeft) {
    // Read the next key value pair, false means no more pairs
    if (!leftRR.next(lkey, lvalue)) {
      // If no more, then this task is nearly finished
      alldone = true; break;
    }else {
      // If we aren't done, set the value to the key and set
      // our flags
      key.set(lvalue.toString());
      goToNextLeft = alldone = false;
      // Reset the right record reader
      this.rightRR = this.rightFIF.getRecordReader(this.rightIS, this.rightConf,
this.rightReporter);
     }
}
```



```
  // Read the next key value pair from the right data set
  if (rightRR.next(rkey, rvalue)) {
  // If success, set the value
  value.set(rvalue.toString());
  } else {
    // Otherwise, this right data set is complete
    // and we should go to the next left pair
    goToNextLeft = true;
  }
   // This loop will continue if we finished reading key/value
   // pairs from the right data set
  } while (goToNextLeft);
  // Return true if a key/value pair was read, false otherwise
  return !alldone;
 }
}
```

Driver Code

```
public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {
  // Configure the join type
  JobConf conf = new JobConf("Cartesian Product");
  conf.setJarByClass(CartesianProduct.class);
  conf.setMapperClass(CartesianMapper.class);
  conf.setNumReduceTasks(0);
  conf.setInputFormat(CartesianInputFormat.class);
  // Configure the input format
  CartesianInputFormat.setLeftInputInfo(conf, TextInputFormat.class,
args[0]);
  CartesianInputFormat.setRightInputInfo(conf, TextInputFormat.class,
args[0]);
  TextOutputFormat.setOutputPath(conf, new Path(args[1]));
```


Driver Code

```
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);
  RunningJob job = JobClient.runJob(conf);
  while (!job.isComplete()) {
    Thread.sleep(1000);
  }
  System.exit(job.isSuccessful() ? 0 : 1);
}
```

Mapper Code (comparing comments)

```
public static class CartesianMapper extends MapReduceBase implements
Mapper < Text, Text, Text, Text > {
  private Text outkey = new Text();
  public void map(Text key, Text value, OutputCollector < Text, Text >
output, Reporter reporter) throws IOException {
    // If the two comments are not equal
    if (!key.toString().equals(value.toString())) {
      String[] leftTokens = key.toString().split("\\s");
      String[] rightTokens = value.toString().split("\\s");
      HashSet < String > leftSet =
             new HashSet < String >(Arrays.asList(leftTokens));
      HashSet < String > rightSet =
             new HashSet < String >(Arrays.asList(rightTokens));
      int sameWordCount = 0;
      StringBuilder words = new StringBuilder();
```



```
    for (String s : leftSet) {
        if (rightSet.contains(s)) {
           words.append(s + ",");
           ++sameWordCount;
         }
    }
    // If there are at least three words, output
    if (sameWordCount > 2) {
      outkey.set(words + "\t" + key);
      output.collect(outkey, value);
    }
  }
 }
}
```

Questions?

Week 5 – Lecture 2
MapReduce Design Pattern V: Input/Output Patterns

Sangmi Lee Pallickara

Computer Science, Colorado State University

0

FAQs

- Vote for Jacob! (Magna-Shox)
  - https://www.denverstartupweek.org/pitch_contest/entries
  - Semi-Final: pitch competition to win $34,000 for Denver Startup Week
  - So far, #1!
- Multiple MapReduce Jobs?
  - e.g. TeraSort (analysis → sorting phase)
  - e.g. Filtering → Joining

1

1

Chaining MapReduce Jobs

```
public static void main(String [] args) throws Exception {
        Configuration conf = new Configuration();
        Job job1 = Job.getInstance(conf, "degree");
        /*Your set up here for the job 1*/
        job1.waitForCompletion(true);

        Configuration conf2 = new Configuration();
        Job job2 = Job.getInstance(conf2, "frequency");
        /*Your set up here for the job 2*/

        FileOutputFormat.setOutputPath(job1, new Path(args[1]));
        FileInputFormat.addInputPath(job2, new Path(args[1]));
}
```


Topics discussed in the previous class

- MapReduce Design Pattern IV. Join Patterns
  - Composite Join Pattern
  - Cartesian Product Pattern

Topics to be covered

- MapReduce Design Pattern V. Input/Output Patterns
- How MapReduce Works?

Part 1. Large Scale Data Analytics

Design Pattern 5: Input/Output Patterns

Modify the way data is loaded from disk

- Approach 1: Configuring how contiguous chucks of input are generated from blocks in HDFS
  - `InputFormat`
- Approach 2: Configuring how records appear in the map phase
  - `RecordReader`

Modify the way data is stored on disk

- Approach 1: Configuring how contiguous chucks of output are generated from blocks in HDFS
  - `OutputFormat`
- Approach 2: Configuring how records are stored after the map/reduce phase
  - `RecordWriter`

Roles of `InputFormat` in Hadoop

1. **Validate** the input configuration for the job (i.e., checking that the data is there).
2. **Split the input blocks and files into logical chunks** of type `InputSplit`, each of which is assigned to a map task for processing.
3. Create the **`RecordReader` implementation** to be used to create key/ value pairs from the raw `InputSplit`. These pairs are sent one by one to their mapper

Methods of the `InputFormat` abstract

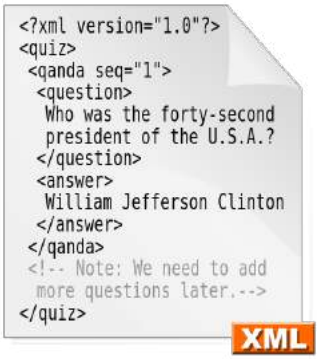
- `getSplits()`
  - retrieves the configured input using the `JobContext` object
  - returns a `List` of `InputSplit` objects
    - `getLocations()` of `InputSplit` returns the list of hostnames where the input split is located
    - This provides clue to the system to determine where to process the map task
    - Good place to throw any necessary exceptions
- `createRecordReader()`
  - Called by framework and generates `RecordReader`

RecordReader (1/2)

- **Generates key/value pairs**
- **Fixing boundaries**
  - Input split boundary might not exactly match the record boundary
  - e.g. `TextInputFormat` reads text files using a `LineRecordReader` to create key/value pairs
    - Will the chunk of bytes for each input split be lined up with a new line character, to mark the line for the `LineRecordReader`?
  - Those bits that are stored on a different node are streamed from a DataNode hosting the block
    - Handled by the `FSDataInputStream` class

RecordReader (2/2)

- Reads Bytes from the input source
- Generates `WritableComparable` key and Writable value
  - An object-oriented way to present information to a mapper
- Example
  - `TextInputFormat` grabs each line
    - "<?xml version="1.0"?>" and "<quiz>" will be injected to the different Mappers
    - Customized `RecordReader` can read lines after the input split boundary
    - Each `RecordReader` should starts at the beginning of an XML element



Methods of the `RecordReader` (abstract)

- `initialize()`
- `getCurrentKey() and getCurrentValue()`
- `next(key, value)`
- `getProgress()`
- `close()`

Schema on read

- `InputSplit` represents a **byte-oriented** view of the split
- `RecordReader` prepares data (record) for a mapper
  - **Only the `RecordReader` maintains the schema**

OutputFormat

- Similar to an input format
- Tasks
  - Validate the output configuration for the job
  - Create the `RecordWriter` implementation that will write the output of the job
- FileOutputFormat
  - File based output
    - Most output from MapReduce job is written to HDFS
  - TextOutputFormat (extended FileOutputFormat)
    - Stores key/value pairs to HDFS at a configured output directory with a tab delimiter
    - Validates the output file directory

Storing data in an External DB

- MapReduce job is not restricted to storing data to HDFS
- MapReduce can do a parallel bulk write
  - Your storage should be able to handle the large number of connections from the many tasks
- e.g. `DBOutputFormat<K DBWritable, V>`
  - Objects that read from/written to a database should implement DBWritable
  - If we have the following table in the database:

```
CREATE TABLE MyTable (
   counter        INTEGER NOT NULL,
   timestamp      BIGINT  NOT NULL,
 );
```

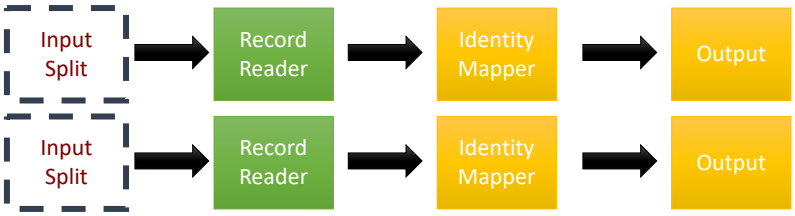
- In your class Main()
  - `job.setOutputKeyClass(DBOutputWritable.class);`
  - The DBOutputWritable should be able to manage the MyTable schema

I/O Pattern Example: `Generating Data`

- Generates a lot of data from scratch
  - This pattern does not load data
- Use cases:
  - Generating random data
  - Generating artificial data as part of a benchmark
    - TeraGen/TeraSort and DFSIO
- This pattern is map-only

Structure

- The `InputFormat` **creates the fake splits** from **nothing**
- The `RecordReader` takes its fake split and **generates random records**
- The `IdentifyMapper` is used to just **write the data out** as it comes in



Structure

- The `InputFormat` **creates the fake splits** from **nothing**
  - **Worksheet question 1: Which component should create the fake splits?**
  - **a. InputFormat**
  - **b. RecordReader**
  - **c. Mapper**
  - **d. Reducer**
- The `RecordReader` takes its fake split and **generates random records**
- The `IdentifyMapper` is used to just **write the data out** as it comes in

Structure

- The `InputFormat` **creates the fake splits** from nothing
- The `RecordReader` takes its fake split and **generates random records**
- The `IdentifyMapper` is used to just **write the data out** as it comes in

Worksheet Question 2:
How many mapper instances will be launched?
a. 1 b. 0 c. number of the fake splits d. random

`Identity` Mapper

- Implements `Mapper<K,V, K,V>`
  - `conf.setMapperClass(IdentityMapper.class);`
- Identity Mapper takes input key/value pair and returns without any processing
- Other implementations of Mapper
  - `InverseMapper, TokenCountMapper, ChainMapper`,.. Etc.

I/O Pattern Example : `Generating Data`

- Goal
  - Generates random StackOverflow data
  - Take a list of 1,000 words and make random blurbs

Driver code

```
public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   int numMapTasks = Integer.parseInt(args[0]);
   int numRecordsPerTask = Integer.parseInt(args[1]);
   Path wordList = new Path(args[2]);
   Path outputDir = new Path(args[3]);
   Job job = new Job(conf, "RandomDataGenerationDriver");
   job.setJarByClass(RandomDataGenerationDriver.class);
   job.setNumReduceTasks(0);
   job.setInputFormatClass(RandomStackOverflowInputFormat.class);
   RandomStackOverflowInputFormat.setNumMapTasks(job, numMapTasks);
   RandomStackOverflowInputFormat
         .setNumRecordPerTask(job, numRecordsPerTask);
   RandomStackOverflowInputFormat.setRandomWordList(job, wordList);
```


continued

```
      TextOutputFormat.setOutputPath( job, outputDir);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(NullWritable.class);
      System.exit(job.waitForCompletion(true) ? 0 : 2);
}
```

InputSplit Code

```
public static class FakeInputSplit extends InputSplit implements Writable {
   public void readFields(DataInput arg0) throws IOException {
   }
   public void write(DataOutput arg0) throws IOException {
   }
   public long getLength() throws IOException, InterruptedException {
     return 0;
   }
   public String[] getLocations() throws IOException, InterruptedException
{
     return new String[0];
   }
}
```


InputFormat code

```
public static class RandomStackOverflowInputFormat extends InputFormat < Text,
NullWritable > {
  public static final String NUM_MAP_TASKS = "random.generator.map.tasks";
  public static final String NUM_RECORDS_PER_TASK =
"random.generator.num.records.per.map.task";
  public static final String RANDOM_WORD_LIST = "random.generator.random.word.file";
  public List < InputSplit > getSplits(JobContext job) throws IOException {
    // Get the number of map tasks configured for int
    numSplits = job.getConfiguration().getInt(NUM_MAP_TASKS, -1);
    // Create a number of input splits equivalent to the number of tasks
    ArrayList < InputSplit > splits = new ArrayList < InputSplit >();
    for (int i = 0; i < numSplits; + + i) {
       splits.add(new FakeInputSplit());
    }
    return splits;
  }
```

continued

```
  public RecordReader < Text, NullWritable > createRecordReader(InputSplit
split, TaskAttemptContext context) throws IOException, InterruptedException {
    // Create a new RandomStackOverflowRecordReader and initialize it
    RandomStackOverflowRecordReader rr = new RandomStackOverflowRecordReader();
    rr.initialize( split, context);
    return rr;
  }
  public static void setNumMapTasks(Job job, int i) {
    job.getConfiguration().setInt( NUM_MAP_TASKS, i);
  }
  public static void setNumRecordPerTask(Job job, int i) {
    job.getConfiguration(). setInt( NUM_RECORDS_PER_TASK, i);
  }
  public static void setRandomWordList( Job job, Path file) {
    DistributedCache.addCacheFile( file.toUri(), job.getConfiguration());
  }
}
```


RecordReader code

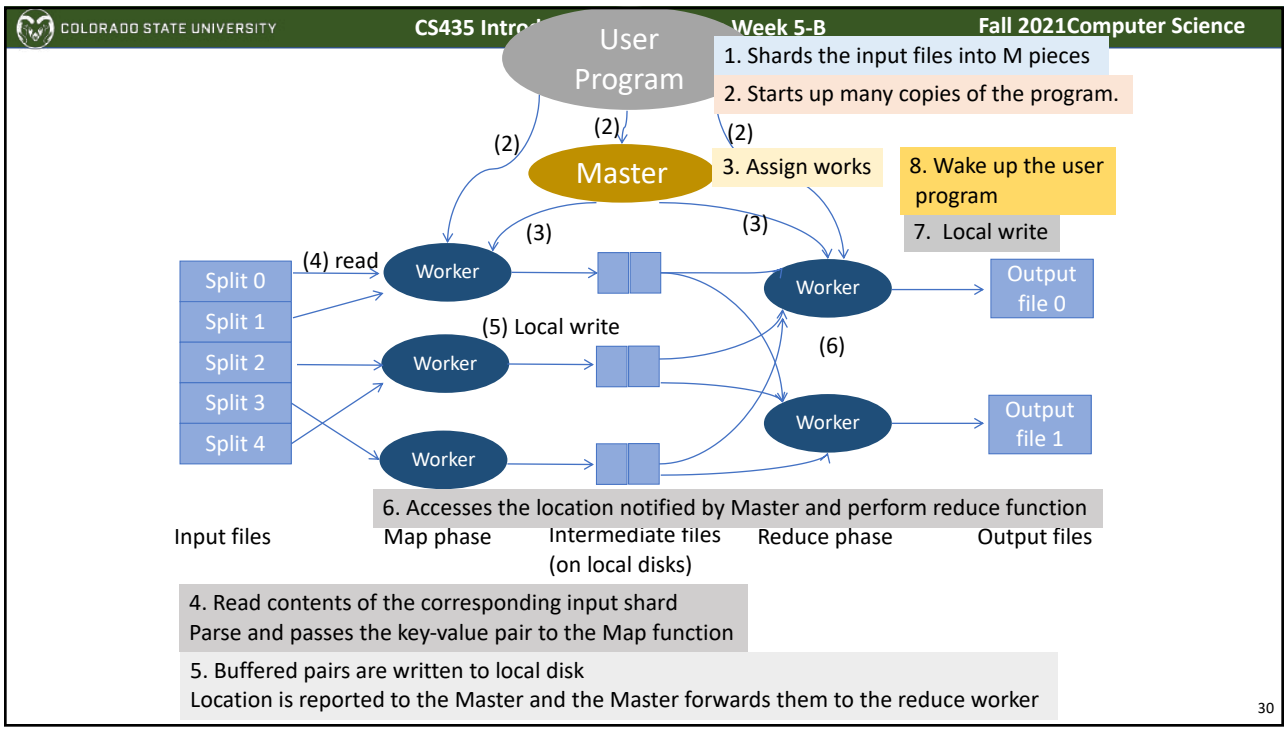
- Record reader is where the data is actually generated
- nextKeyValue
  - Uses a simple random number generator to create a random record
  - The number of words ranges from 1 to 30
  - Once all the records are generated, the record reader returns false
    - signals to the framework mapper that there is no input information

Part 1. Large Scale Data Analytics

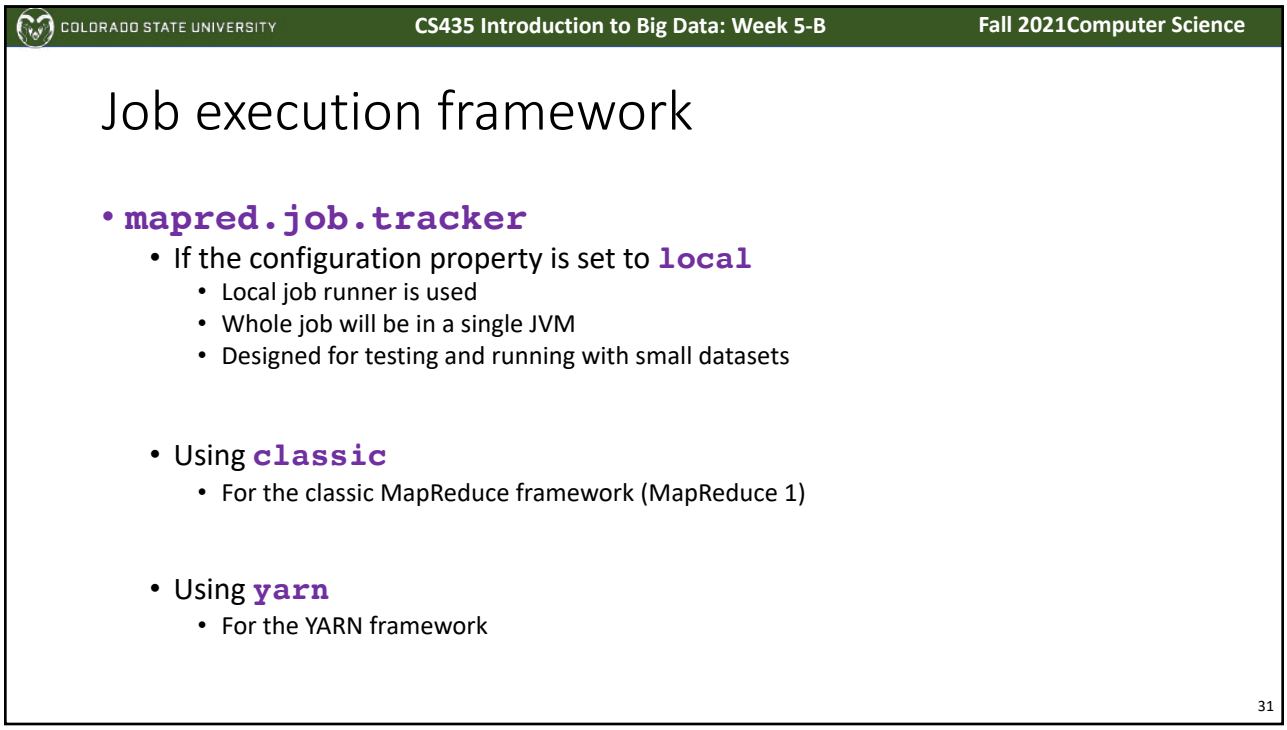
How MapReduce Works

Programming components of MapReduce

- Driver
- Mapper
- Reducer
- InputFormat
- Combiner
- Partitioner
- OutputFormat


COLORADO STATE UNIVERSITY
CS435 Intro
Week 5-B
Fall 2021Computer Science
User Program
1. Shards the input files into M pieces
2. Starts up many copies of the program.
(2)
(2)
(2)
Master
3. Assign works
8. Wake up the user program
7. Local write
(3)
(3)
(4) read
Split 0
Split 1
Split 2
Split 3
Split 4
Worker
Worker
Worker
Worker
Worker
(5) Local write
(6)
Output file 0
Output file 1
6. Accesses the location notified by Master and perform reduce function
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files
4. Read contents of the corresponding input shard
Parse and passes the key-value pair to the Map function
5. Buffered pairs are written to local disk
Location is reported to the Master and the Master forwards them to the reduce worker
30

30


COLORADO STATE UNIVERSITY
CS435 Introduction to Big Data: Week 5-B
Fall 2021Computer Science
Job execution framework
• mapred.job.tracker
• If the configuration property is set to local
• Local job runner is used
• Whole job will be in a single JVM
• Designed for testing and running with small datasets
• Using classic
• For the classic MapReduce framework (MapReduce 1)
• Using yarn
• For the YARN framework
31

31

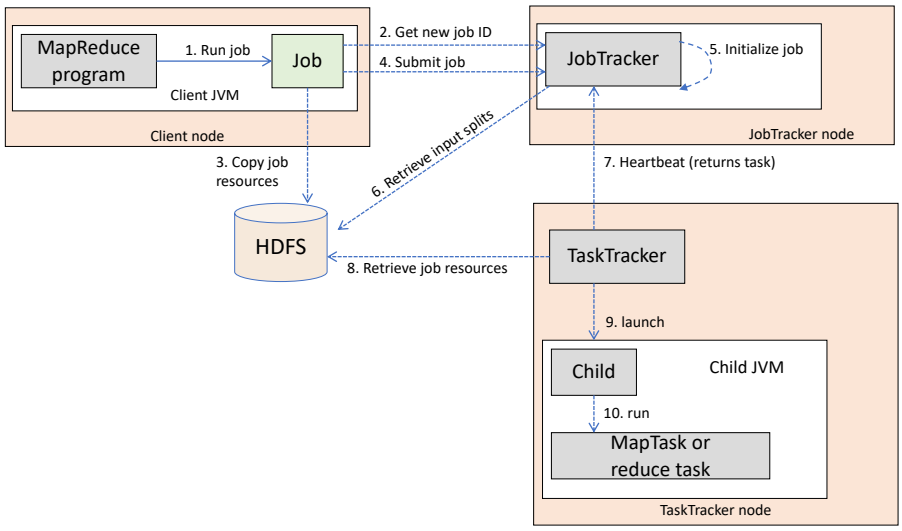
Job submission: JobTracker

- Single method call
  - `submit()` on a `Job` object
    - You can also call `waitForCompletion()`
- Creates an internal `JobSubmitter` instance
  - Calls `submitJobInternal()` on it
- `waitForCompletion()`
  - Polls the job's progress once per second
  - Reports the progress to the console if it has changed since the last report
- Job completes successfully
  - The job counters are displayed
  - Otherwise, the error is logged to the console

Part 1. Large Scale Data Analytics-How MapReduce Works

Classic Framework

MapReduce job in the Classic framework



Job submission process in Classic framework

- Asks the `jobtracker` for a new `jobID`
  - Step 2
- Checks the output specification of the job
- Computes the input splits for the job
- Copies the resources to the `jobtracker`'s filesystem in a directory named after the `jobID`
  - The job JAR file, the configuration file, and the computed input splits
- Tells the `jobtracker` that the job is ready for execution by calling `sumitJob()` on `JobTracker`
  - Step 4

Job Initialization in the Classic mode

- `JobTracker` puts the job into an internal queue
  - Creating an object to represent the job being run
  - Bookkeeping information to keep track of the status and progress of its tasks
- Creating the list of tasks
  - The job scheduler retrieves the input splits
  - The job scheduler creates a map task for each split
  - The number of reduce tasks to create
    - mapred.reduce.tasks property in the Job
      - setNumReduceTasks()
  - A job setup task and a job cleanup task are created

Task assignment

- `Jobtracker` select a job to run
  - Default algorithm
    - Based on the priority list of jobs
- `Tasktrackers` have **a fixed number of slots** for map tasks and reduce tasks
  - These are set independently
  - These are selected based on the number of cores and the memory
- The default scheduler fills empty map task slots first
  - Before it fills the reduce task slots
  - Scheduling the **reduce task does not need to consider data locality**
- `Tasktrackers` run a simple loop that periodically **sends heartbeat** method calls to the `Jobtracker`

Part 1. Large Scale Data Analytics-How MapReduce Works

Classic Framework: Progress and status updates

Progress and status updates [1/3]

- A **Job and each of its tasks have a status**
  - State of the job or task
    - e.g. running, successfully complete, failed
  - The **progress** of maps and reduces
  - The values of **the job's counters**
  - A status message or description

Progress and status updates [2/3]

- `Tasks`
  - have a set of counters
  - Count various events at the task run
    - e.g. the number of map output records written
  - If a task reports progress
    - it sets a flag to indicate that the status change should be sent to the `Tasktracker`
      - Checked every 3 seconds
- `Tasktracker`
  - Tasks notify the current task status to the `Tasktracker`
    - if the flag is set
  - `Tasktracker` sends heartbeats to the `Jobtracker` every 5 seconds (minimum)

Progress of Map task and Reduce task

- Map task
  - Tracks the percentage of InputSplit consumed by mapper
- Reduce task
  - Sort-shuffle-reduce (1/3 each)
  - Reduce
    - Tracks percentage of Reduce tasks consumed by Reducer(s)

Progress and status updates [3/3]

- `JobTracker`
  - Combines updates to produce a global view
- `Job`
  - Receives the latest status by polling the `JobTracker` every second
  - Prints job statistics and counters to the console

Part 1. Large Scale Data Analytics-How MapReduce Works

YARN Framework

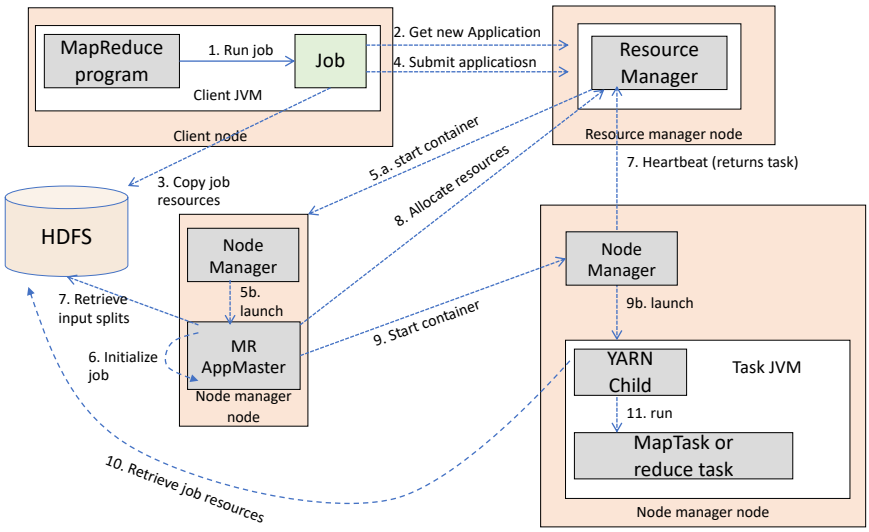
YARN (MapReduce 2)

- To provide the **scalability** to MapReduce
  - **Splitting responsibility of the jobtracker**
    - **Scheduling**
    - **Task progress monitoring**
- MapReduce is one type of YARN application

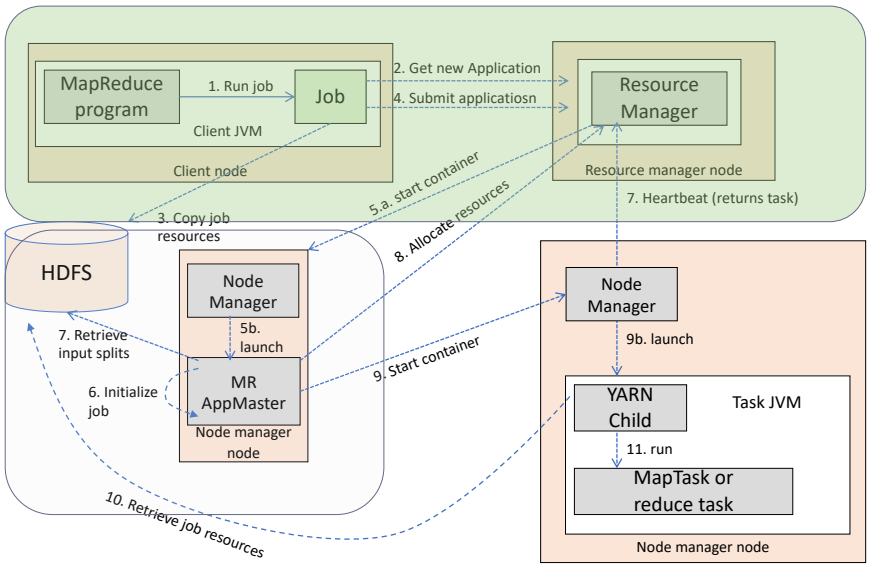
YARN (MapReduce 2)

- **Resource manager**
  - Manages the use of resources across the cluster
- **Node manager**
  - Launches and monitors the compute containers on machines in the cluster
- **Application master**
  - Manages the lifecycle of applications running on the cluster
  - Application master negotiates with the resource manager for cluster resources
    - Number of container and certain memory limit
    - Node managers oversee containers not to use more resources than allocated

A MapReduce job using YARN



Job Initialization



Job Initialization [1/3]

- `submitApplication()`
  - **Resource manager** will **hands off the request to the scheduler**
  - The scheduler allocates a **container**
    - The resource manager then launches the application master's process in the container

Job Initialization [2/3]

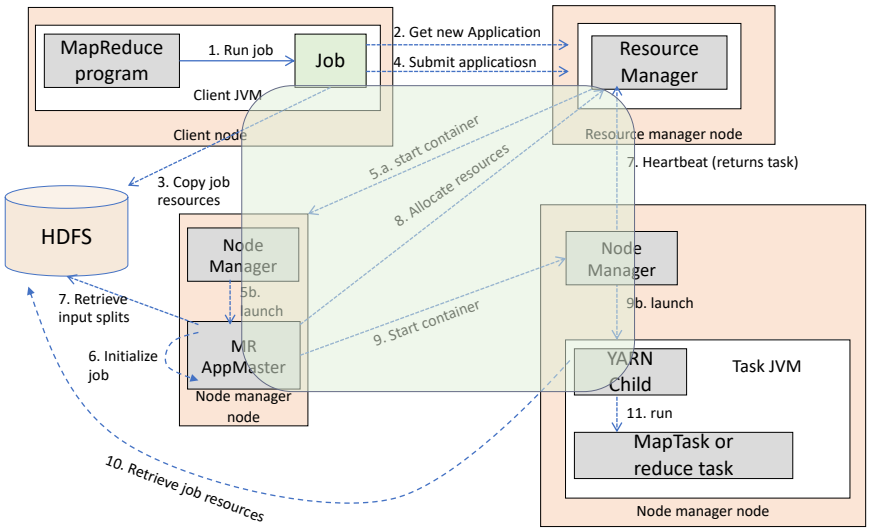
- **Application master**
  - The application master for MapReduce jobs
    - `MRAppMaster`
    - Initializes the job by creating bookeeping objects
      - To keep track of the job's progress
  - Retrieves the input splits
  - Creates a map task object for each split
  - Creates reduce task object
    - Mapreduce.job.reduces property

Job Initialization [3/3]

- Application master
  - **Plans job execution**
    - If the jobs is **small**, Application Master will run the tasks **in the same JVM as itself**
  - **Uber task**
    - The overhead of allocating and running tasks in new container outweighs the gain to be had it running them in parallel, compared to running them sequentially on one node

What is a small job?
A small job is one that has less than 10 mappers, only one reducer, and an input size that is less than the size of one HDFS block
See the lecture note of Week 5-A

Task assignment



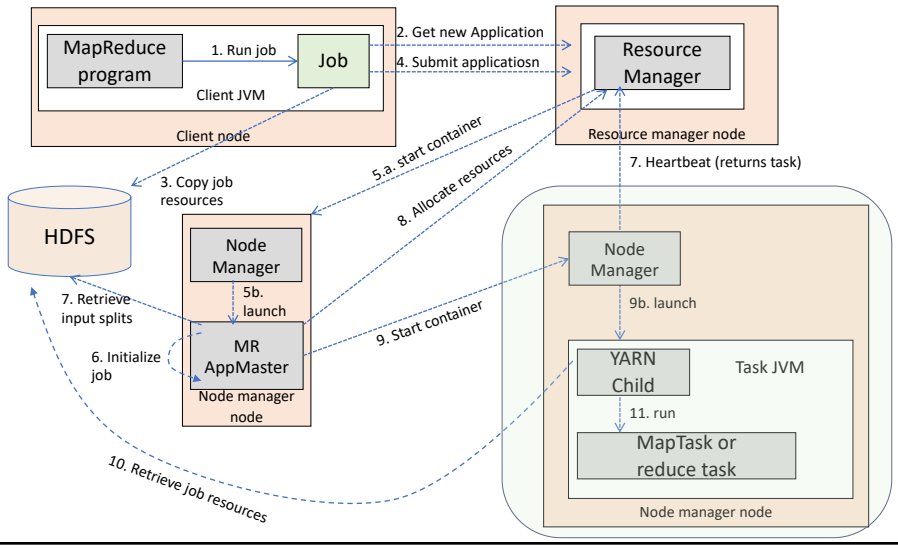
Task assignment [1/2]

- Application Master requests container for all the map and reduce tasks in the job
  - From the resource manager (Step 8)
- All the requests includes information about each map tasks' locality
  - Host and corresponding racks that the input split resides on
- Scheduler attempts to place tasks on data-local nodes in the ideal case
  - If it is not possible, the scheduler prefers rack-local placement
    - Job is running on a node in the same rack

Task assignment [2/2]

- Requests specify required memory
  - 1024MB (by default)
  - This is configurable
    - `mapreduce.map.memory.mb`
    - `mapreduce.reduce.memory.mb`
- In YARN, resources are managed more fine-grained
  - Applications may request a memory capability that is anywhere between the minimum allocation and a maximum allocation
  - `yarn.scheduler.capacity.minimum-allocation-mb`
    - Default minimum: 1024MB
  - `yarn.scheduler.capacity.maximum-allocation-mb`
    - Default maximum: 10240 MB
  - Tasks can request any memory allocation between 1 and 10GB(default) in mulitple of 1GB
    - `mapreduce.map.memory.mb` and `mapreduce.reduce.memory.mb`

Task execution



Task execution

- Application master starts the container by contacting node manager
  - The task is executed by YarnChild
  - YarnChild runs in a dedicated JVM

Progress and status updates

- Task reports its progress and status back to its application master
  - Every 3 seconds over the umbilical interface
- The client polls the application master every second
  - `mapreduce.client.progressmonitor.pollinterval`

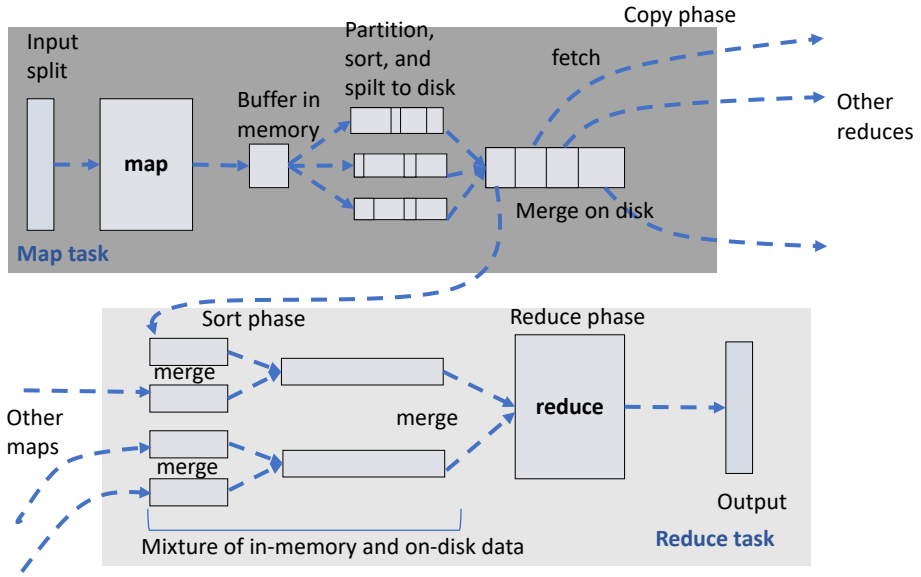
Part 1. Large Scale Data Analytics-How MapReduce Works

How Partitioner and Combiner work

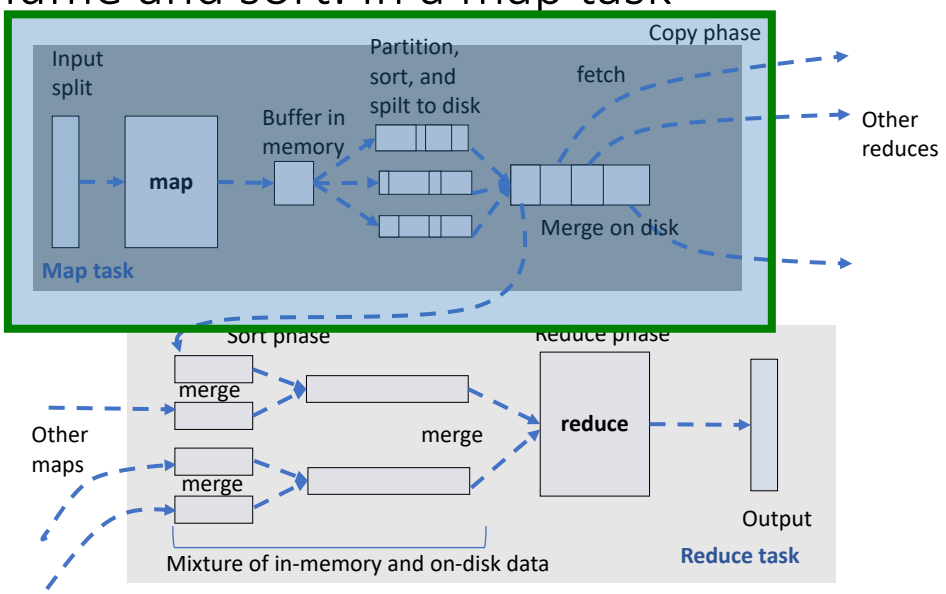
Shuffle and Sort

- Sort
  - MapReduce makes the guarantee that the input to every reducer is sorted by key
- Shuffle
  - MapReduce transfers the map outputs to the reducers as inputs

Shuffle and sort



Shuffle and sort: In a map task



In a map task

- Each map task has a circular memory buffer
  - For output
  - 100MB by default
  - `io.sort.mb`
- When the contents of the buffer reaches the threshold, a background thread starts spill the contents to disk
  - Default 0.80
  - Spills are written in round-robin-fashion to the local directory
    - `mapred.local.dir`

Partitioning

- Before the data is written to the local disk, data is divided into partitions corresponding to the reducers
- The background thread performs an in-memory sort by key
  - Within each partition
- Each time the memory buffer reaches the spill threshold, a new spill file is created
  - There can be several spill files after the last output record is written
  - The spill files are merged into a single partitioned and sorted output file

Output file

- **Combiner is run**
- Output files from map can be **compressed**
- The output file's partitions are made available to the **reducers over HTTP**

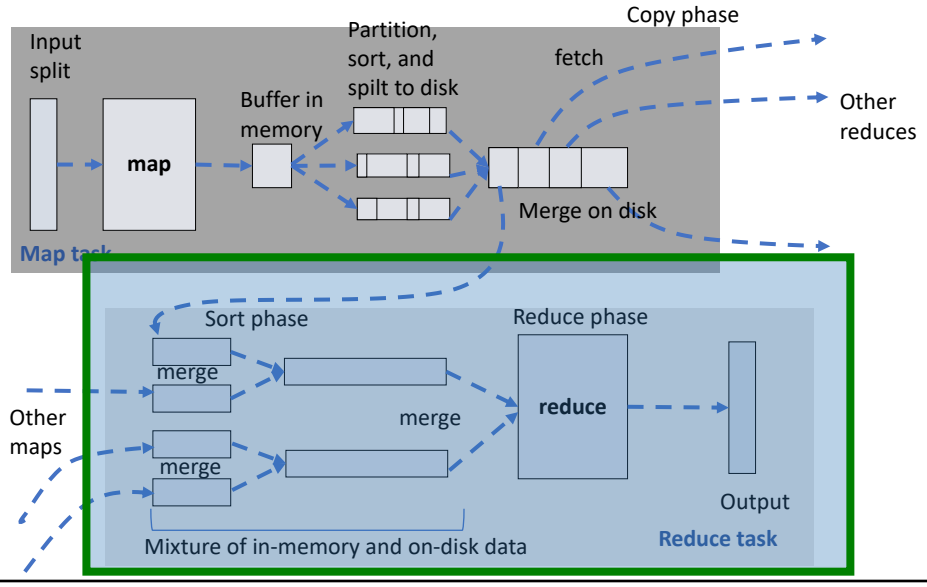
Execution of Combiner

- Hadoop **does not** guarantee on how many times a combiner will be called for each output key

Properties of a Combiner
Combiner's functionality must be
- Commutative
- Associative

Input types (key/value pair) and the output types must be the same
- Same as output of mapper as well

Shuffle and sort: In a reduce task



Copy phase

- The reduce task needs the map output from several map tasks across the cluster

- Copy phase
  - The reduce task starts copying their outputs as soon as each completes
    - The map tasks may finish at different times
    - **Merges them into larger and sorted files**
      - Decompresses the compressed files

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 5-B Fall 2021Computer Science

Sort phase

- Sort phase
  - All of the map outputs should be moved and copied to the reduce task
  - Merging and sorting the map outputs
  - Sorting is done in rounds
    - If there are 50 map outputs and the **merge factor** was 10
      - 5 intermediate files
      - Merging intermediate files: additional round
      - 6 rounds of merging will be required
    - Final round
      - A mixture of in-memory and on-disk segments
      - Directly feeds the reduce function
        - Without writes a single sorted file to disk

Part 1. Large Scale Data Analytics-How MapReduce Works

Failures

Failures in Classic MapReduce

- The **child task** fails
  - Runtime exception from the user code
  - The child JVM reports the error back to its parent before it exits
    - Written in the user logs
  - Tasktracker marks the task attempt as failed
    - Frees a slot to run another task
  - Sudden exit of the child JVM
    - Tasktracker notices that the process has exited and mars the attempt as failed
- **Hanging tasks**
  - If there is no progress update for a while
    - Mark the task as failed
    - Timeout period is normally 10 minutes
      - `mapred.task.timeout`

Tasktracker failure in Classic MR

- **Tasktracker stops sending heartbeats**
  - Jobtracker will notice if it hasn't received one for 10 minutes (configurable)
  - Remove it from the pool of tasktracker
- Jobtracker arranges tasks including the completed jobs
  - Because the output may not be accessible
- Tasktracker can also be blacklisted if more than four tasks from the same job fail (set by `mapred.max.tracker.failures`)
  - Blocklisted tasktrackers are **not assigned tasks**
    - Until faults expire

Jobtracker failure in Classic MR

- The **most serious failure** mode
- Hadoop has **no mechanism** for dealing with jobtracker failure
  - **Single point of failure**
- All running jobs fail
- After restarting a jobtracker
  - **Job should be resubmitted**
- This is improved with YARN

Task failure in YARN

- Failure of the running task is similar to the classic case
  - Runtime exception and sudden exit of the JVM are propagated back to the application master
  - **The task attemp is marked as failed**
  - **Hanging tasks are noticed by the application manager** by the absence of a ping over the umbilical channel

Application master failure in YARN

- No heartbeats to the resource manager from the application master
  - **The resource manager will detect the failure and start a new instance of the master running in a new container**
  - All tasks will be **rerun** (default)
    - Recovery can be enabled
- **Client will access resource manager to get the new address of the application master**

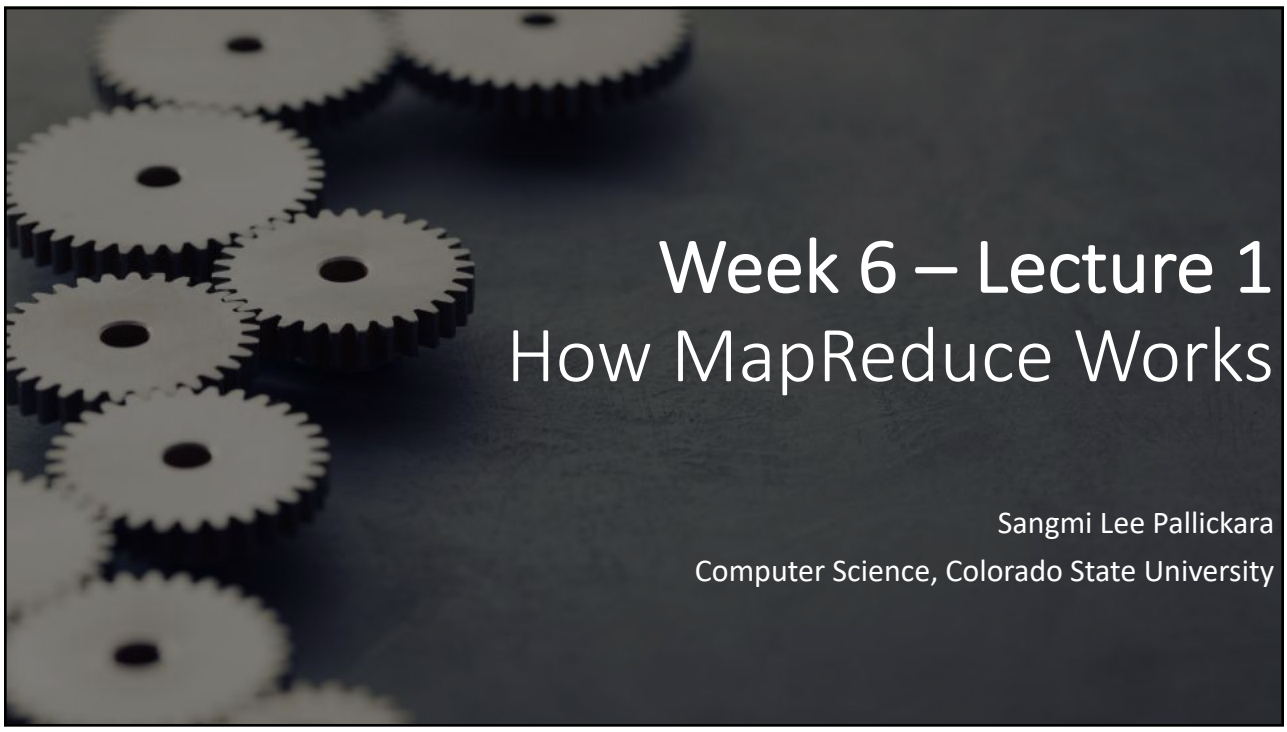
Node manager failure in YARN

- Resource manager will stop getting heartbeats
  - Remove the failed node manager from the pool of available nodes
- **Any task or application master running on the failed node manager will be recovered**

Resource manager failure

- After a crash and a new resource manager instance is brought up (by administrator)
  - **It recovers from the saved state**
    - **Check points saved in the persistent storage**
    - Non-completed jobs are included

Questions?



0

FAQs

- Please sign up for the software demo – PA1
- PA2: October 14
  - Start early, finish early, and get your grade early!

1

Topics discussed in the previous class

- MapReduce Design Pattern V. Input/Output Patterns
- How MapReduce Works? (This will be continued)

Topics to be covered

- Programming Assignment #2
- How MapReduce works - Continued

Programming Assignment 2

- Objective
  - Small-World Networks - Analyzing Large-scale Social Networks using MapReduce with the Google+ friend networks data
- 6 degrees of separation (Stanley Milgram, 1969)
- Image Credit: https://en.wikipedia.org/wiki/Small-world_experiment

Programming Assignment 2

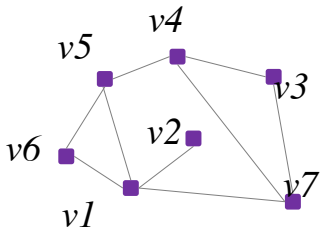
- Small Network
  - Highly clustered, like regular lattices, yet have small characteristic path lengths, like random graphs
- Duncan Watts and Steven Strogatz
  - Small-World Networks are a kind of random network
  - clustering coefficient and average node-to-node distance (A.K.A. shortest path length)

Programming Assignment 2

- Small World Networks
  - A type of mathematical graph in which most nodes are not neighbors of one another but the neighbors of any given node are likely to be neighbors of each other and most nodes can be reached from every other node by a small number of hops or steps
- How to identify the small-world networks
  - The Average Geodesic Path Length
  - Clustering Coefficient: Large degree of Transitivity

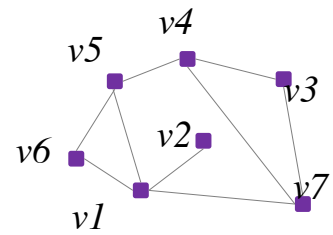
Programming Assignment 2 -How to identify the small-world networks

- **The Average Geodesic Path Length**
- Path between V1 and V3
- Path_1 = v1-v5-v4-v3
- Path_2 = v1-v5-v4-v7-v3
- Path_3 = v1-v6-v5-v4-v3
- L(Path_1)=3
- L(Path_2)=4
- L(Path_3)=4
- Shortest path between node v1 and v3 is 3



Programming Assignment 2
-How to identify the small-world networks

- **The Average Geodesic Path Length**
- $d_{1,3}$ = Shortest path length between node v1 and v3 = 3
- $d_{1,2}$ = Shortest path length between node v1 and v2 = 1
- **$d_{1,4}$ = Shortest path length between node v1 and v4 = 2 (there are two "shortest" paths. However we are considering only "distance". Therefore there are only 1 $d_{1,4}$.)**
- $d_{1,7}$ = Shortest path length between node v1 and v7 = 1
- $d_{5,7}$ = Shortest path length between node v5 and v7 = 2
- …
- Average geodesic path length = $L \quad = \frac{1}{v(v-1)} \sum_{if\ n,i \neq j} d_{i,j}$



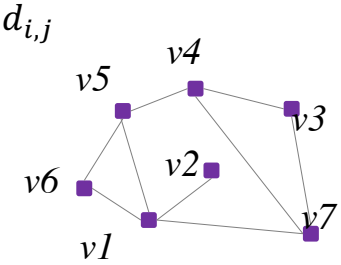
Note: your graph is not a directed graph. Therefore, (a, c) = (c, a). You should not count the same pairs of nodes twice.

8

8

Programming Assignment 2
-How to identify the small-world networks

- **The Average Geodesic Path Length**
- Average geodesic path length = $L \quad = \frac{1}{v(v-1)} \sum_{if\ n,i \neq j} d_{i,j}$
- How to calculate AGP?
  - Dijkstra's algorithm?
  - BFS?
  - Note that we do not collect the paths. We need ONLY the length!
- $\sum_{if\ n,i \neq j} d_{i,j} = \sum_{for\ d_{i,j}=1} d_{i,j} + \sum_{for\ d_{i,j}=2} d_{i,j} + \sum_{for\ d_{i,j}=3} d_{i,j} + \cdots + \sum_{for\ d_{i,j}=\max_length} d_{i,j}$
- Your max_length in PA2 is 8
- How do you make sure a path is the shortest path?



9

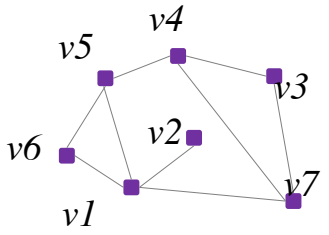
9

Programming Assignment 2 -How to identify the small-world networks

- **The Average Geodesic Path Length**
- You should eliminate duplicated pairs (and select only the shortest one)
- A path a-b-c and a-r-t-h-g-c should not be counted together. Only 2 (from a-b-c) should be considered as the shortest pathlength between node a and c

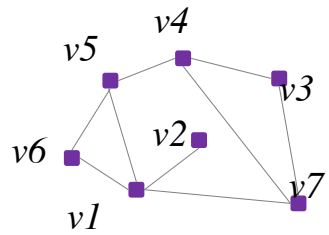
Programming Assignment 2 -How to identify the small-world networks

- **Cluster Coefficient: Large degree of Transitivity**
- Probability that one's friends are also friends of each other
- How many triples do involve triangles?
- triples:
  - v1-v6-v5
  - v6-v5-v4
  - v6-v5-v1
  - v4-v3-v5
  - …
- Triangles
  - v1-v6-v5
  - v7-v3-v4



Programming Assignment 2
-How to identify the small-world networks

- **Cluster Coefficient: Large degree of Transitivity**
- $C = \frac{3 \times number\ of\ triangles}{number\ of\ connected\ triples}$
- Do not include the redundant triples
  - This is NOT a directed graph
  - E.g. a-b-c vs. c-b-a
- Why 3 x?
- A single triangle (v1-v6-v5) involves three triples
  - v1-v6-v5
  - v6-v5-v1
  - v5-v1-v6
  - Do you find any duplicate triples?

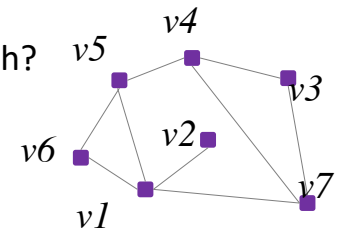


COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 6-A Fall 2021Computer Science

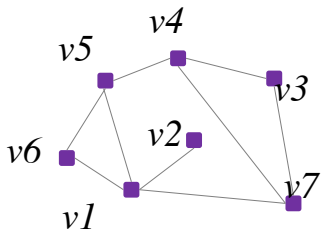
Programming Assignment 2
-How to identify the small-world networks

- **Cluster Coefficient: Large degree of Transitivity**
- $C = \frac{3 \times number\ of\ triangles}{number\ of\ connected\ triples}$
- Question
- What is the cluster coefficient of the example graph?

a. 6/17
b. 2/17
c. 6/34
d. 2/34



Answer



Programming Assignment 2
- How to identify the small-world networks

	L-gfn	*L-random*	*C-gfn*	*C-random*
Google+ Friend Network				

Table 3. The submission requirement 3

- Do you have all the numbers in Table 3?
- To prepare for the discussion, please read Watts and Strogatz's paper
- Use the equation listed in the paper to justify your answer

Watts, D.J. and Strogatz, S.H., 1998. Collective dynamics of 'small-world networks. *nature*, *393*(6684), pp.440-442

Part 1. Large Scale Data Analytics

How MapReduce Works

YARN: Task execution

- Application master starts the container by contacting node manager
  - The task is executed by YarnChild
  - YarnChild runs in a dedicated JVM

YARN: Progress and status updates

- Task reports its progress and status back to its application master
  - Every 3 seconds over the umbilical interface
- The client polls the application master every second
  - `mapreduce.client.progressmonitor.pollinterval`

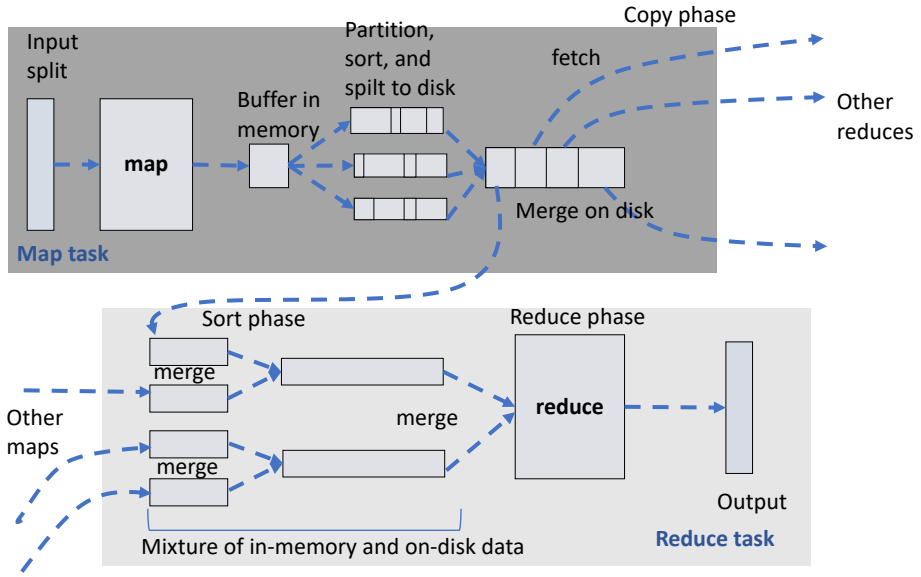
Part 1. Large Scale Data Analytics-How MapReduce Works

How Shuffle, Sort, Partitioner and Combiner work

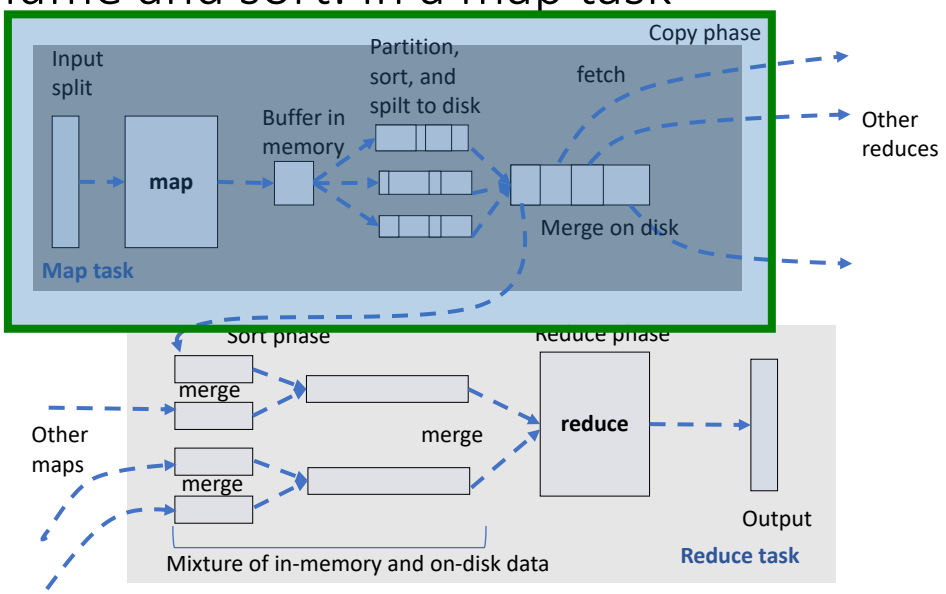
Shuffle and Sort

- Sort
  - MapReduce makes the guarantee that the input to every reducer is sorted by key
- Shuffle
  - MapReduce transfers the map outputs to the reducers as inputs

Shuffle and sort



Shuffle and sort: In a map task



In a map task

- Each map task has a circular memory buffer
  - For output
  - 100MB by default
  - `io.sort.mb`
- When the contents of the buffer reaches the threshold, a background thread starts spill the contents to disk
  - Default threshold is 0.80
  - Spills are written in round-robin-fashion to the local directory
    - `mapred.local.dir`

Partitioning

- Before the data is written to the local disk, data is divided into partitions corresponding to the reducers
- The background thread performs an in-memory sort by key
  - Within each partition
- Each time the memory buffer reaches the spill threshold, a new spill file is created
  - There can be several spill files after the last output record is written
  - The spill files are merged into a single partitioned and sorted output file

Output file

- **Combiner is run (see the next slide)**
- Output files from map can be **compressed**
- The output file's partitions are made available to the **reducers over HTTP**

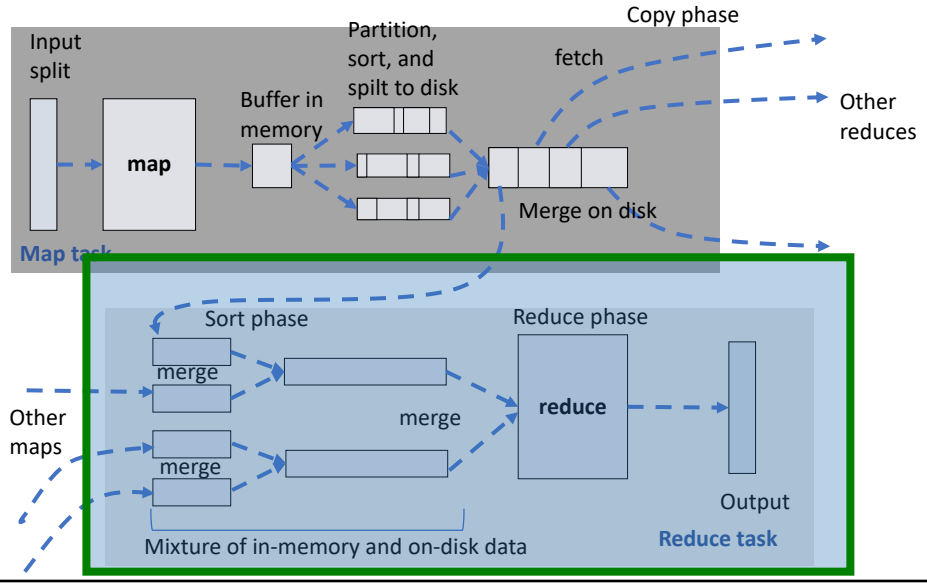
Execution of Combiner

- Hadoop **does not** guarantee on how many times a combiner will be called for each output key

Properties of a Combiner
Combiner's functionality must be
- Commutative
- Associative

Input types (key/value pair) and the output types must be the same
- Same as output of mapper as well

Shuffle and sort: In a reduce task



Copy phase

- The reduce task needs the map output from several map tasks across the cluster

- Copy phase
  - The reduce task starts copying their outputs as soon as each completes
    - The map tasks may finish at different times
    - **Merges them into larger and sorted files**
      - Decompresses the compressed files

Sort phase

- Sort phase
  - All of the map outputs should be moved and copied to the reduce task - done
  - Merging and sorting the map outputs
  - Sorting is done in rounds
    - If there are 50 map outputs and the **merge factor** was 10
      - 5 intermediate files, 1 final output
      - Merging intermediate files: additional round
      - 6 rounds of merging will be required
    - Final round
      - A mixture of in-memory and on-disk segments
      - Directly feeds the reduce function

Part 1. Large Scale Data Analytics-How MapReduce Works

Failures

Failures in Classic MapReduce

- The **child task** fails
  - Runtime exception from the user code
  - The child JVM reports the error back to its parent before it exits
    - Written in the user logs
  - Tasktracker marks the task attempt as failed
    - Frees a slot to run another task
  - Sudden exit of the child JVM
    - Tasktracker notices that the process has exited and mars the attempt as failed
- **Hanging tasks**
  - If there is no progress update for a while
    - Mark the task as failed
    - Timeout period is normally 10 minutes
      - `mapred.task.timeout`

Tasktracker failure in Classic MR

- **Tasktracker stops sending heartbeats**
  - Jobtracker will notice if it hasn't received one for 10 minutes (configurable)
  - Remove it from the pool of tasktracker
- Jobtracker arranges tasks including the completed jobs
  - Because the output may not be accessible
- Tasktracker can also be blacklisted if more than four tasks from the same job fail (set by `mapred.max.tracker.failures`)
  - Blocklisted tasktrackers are **not assigned tasks**
    - Until faults expire

Jobtracker failure in Classic MR

- The **most serious failure** mode
- Hadoop has **no mechanism** for dealing with jobtracker failure
  - **Single point of failure**
- All running jobs fail
- After restarting a jobtracker
  - **Job should be resubmitted**
- This is improved with YARN

Task failure in YARN

- Failure of the running task is similar to the classic case
  - Runtime exception and sudden exit of the JVM are propagated back to the application master
  - **The task attemp is marked as failed**
  - **Hanging tasks are noticed by the application manager** by the absence of a ping over the umbilical channel

Application master failure in YARN

- No heartbeats to the resource manager from the application master
  - **The resource manager will detect the failure and start a new instance of the master running in a new container**
  - All tasks will be **rerun** (default)
    - Recovery can be enabled
- **Client will access resource manager to get the new address of the application master**

Node manager failure in YARN

- Resource manager will stop getting heartbeats
  - Remove the failed node manager from the pool of available nodes
- **Any task or application master running on the failed node manager will be recovered**

Resource manager failure

- After a crash and a new resource manager instance is brought up (by administrator)
  - **It recovers from the saved state**
    - **Check points saved in the persistent storage**
    - Non-completed jobs are included

Part 1. Large Scale Data Analytics-How MapReduce Works

Working with Small Files

Hadoop Small Files Problem

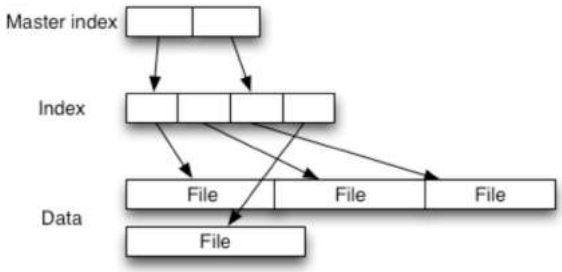
- Small files
  - Files those are significantly smaller than the HDFS block size
- HDFS cannot handle lots of files!
- Every file/directory and block in HDFS
  - Represented as an object in the namenode's memory
  - 150 bites per object (rue of thumb)
- e.g. 10 million files: each using a block: use about 3 gigabytes of memory
- Scaling up much beyond this level is a problem with current hardware
- HDFS is designed for streaming access of large files

Hadoop Small Files Problem

- MapReduce cannot handle lots of files!
- Map tasks usually process a block of input at a time
  - Due to the small files
    - Each map task will take small input
    - Large number of map tasks will be created
- Extra bookkeeping overhead
  - JVM startup overhead

Hadoop Small Files Problem

- Help 1: Reusing JVM
  - mapred.job.reuse.jvm.num.tasks
  - MultiFileInputSplit
    - Runs more than one split per map
- Help 2: HAR files
  - Hadoop Archives (Use hadoop archive command)
  - A layered file system on top of HDFS
  - Users can still see all the files
    - HDFS manages smaller number of files



Hadoop Small Files Problem

- Help 3: Sequence files
  - User the file name as a key and the file content as the value

SequenceFile File Layout

Data	Key	Value	Key	Value	Key	Value	Key	Value

- Help 4: User external storage system
  - HBASE

Part 1. Large Scale Data Analytics-How MapReduce Works

Chaining Multiple MapReduce Jobs

Chaining jobs (1/2)

- Write multiple driver methods (One for each job)
- Call the first driver method
  - `JobClient.runJob()`
- When the first job has completed
  - Call the next driver method
  - Creates a new `JobConf` object

```
JobClient.runJob(job1)
JobClient.runJob(job2)
…
```


Chaining jobs (2/2)

- Use org.apache.Hadoop.mapred.jobcontrol.Job objects
  - A `Job` takes a `JobConf` object as its constructor argument
  - Jobs can depend on one another using
    - `x.addDependingJob(y)`
    - Job `x` cannot start until y has successfully completed

```
Job job1=new Job(jobconf1);
Job job2=new Job(jobconf2);

JobControl jbcntrl = new JobControl("jbcntrl");
jbcntrl.addJob(job1);
jbcntrl.addJob(job2);
job2.addDependingJob(job1);
jbcntrl.run();
```

Part 1. Large Scale Data Analytics

. Web-Scale Link Analysis and Social Network Analysis

Web-Scale Link Analysis

This material is built based on,

- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012
  - Chapter 5
- http://infolab.stanford.edu/~ullman/mmds.html

What are these?

- Archie
- Veronica
- Infoseek
- Snap
- Direct Hit
- Lycos
- AltaVista
- Excite
- Yahoo
- Google

Early Search Engines

- They worked by crawling the Web and listing the terms
  - Words or other strings of characters other than white space
  - In an inverted index
- An ***inverted index*** is a data structure that makes it easy, given a term, to find (pointer to) all the places where that term occurs

Inverted index (1/2)

- Inverted index
  - For given texts,

```
T[0] = "Colorado State University"
T[1] = "Colorado water source"
T[2] = "University of Colorado"
```

  - We have the following inverted file index

```
"Colorado": {0,1,2}
"of":{2}
"source":{1}
"State": {0}
"University": {0,2}
"water":{1}
```


Inverted index (2/2)

```
"Colorado": {0,1,2}
"of":{2}
"source":{1}
"State": {0}
"University": {0,2}
"water":{1}
```

- A term search for the terms, "Colorado", "State", and "University" would give the set

$$\{0,1,2\} \cap \{0\} \cap \{0,2\} = \{0\}$$

Term spam

- If you were selling **shirts** on the Web
  - All you care about was that **people would see your page**
- You could add a term like **"movie"** to your page
  - Add thousands of times
    - It does not even need to show
      - Give the same color as background to the letters
  - A search engine would think this page is very important one about "movie"
- You could go to the search engine and search "movie" and see the first listed page
  - Copy that page with the same color as background

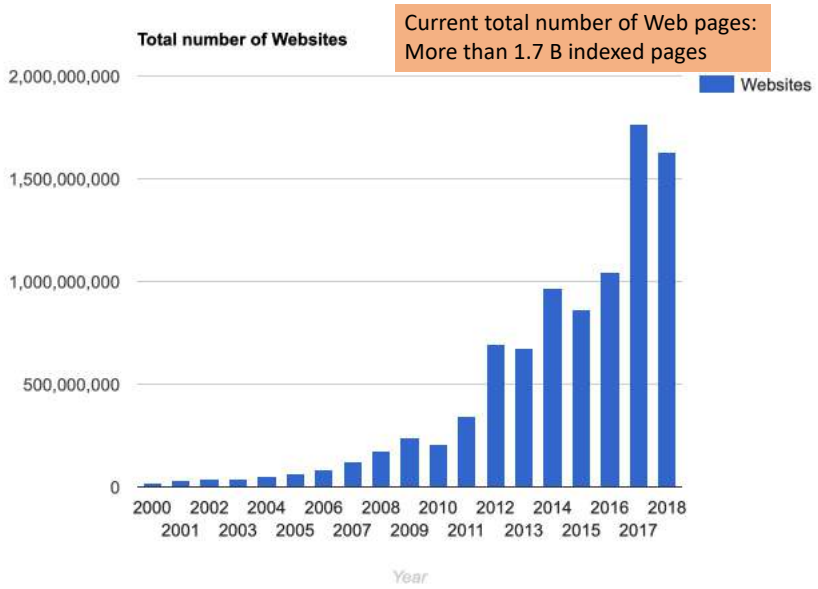
Part 1. Large Scale Data Analytics

Web-Scale Link Analysis and Social Network Analysis

Web-Scale Link Analysis: PageRank Algorithm

COLORADO STATE UNIVERSITY **CS435 Introduction to Big Data: Week 6-A** **Fall 2021Computer Science**

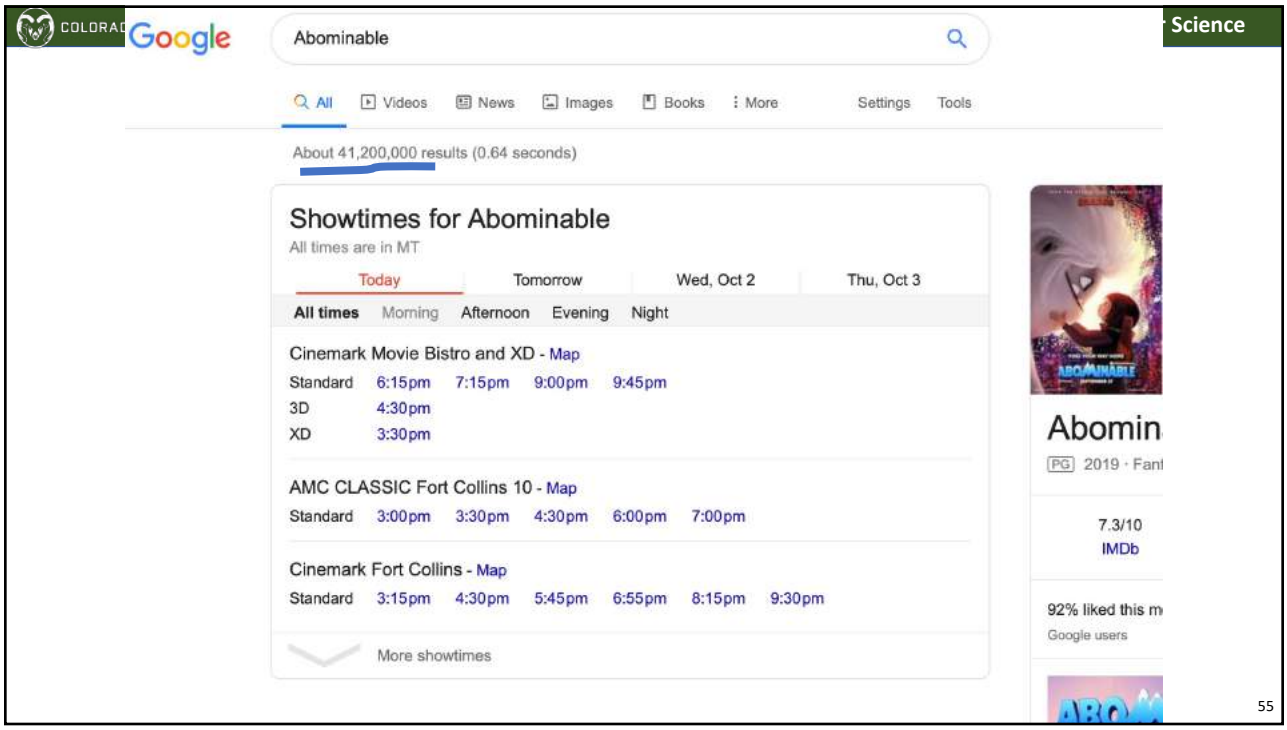
Current total number of Web pages:
More than 1.7 B indexed pages



http://www.internetlivestats.com/total-number-of-websites/

54

54



55

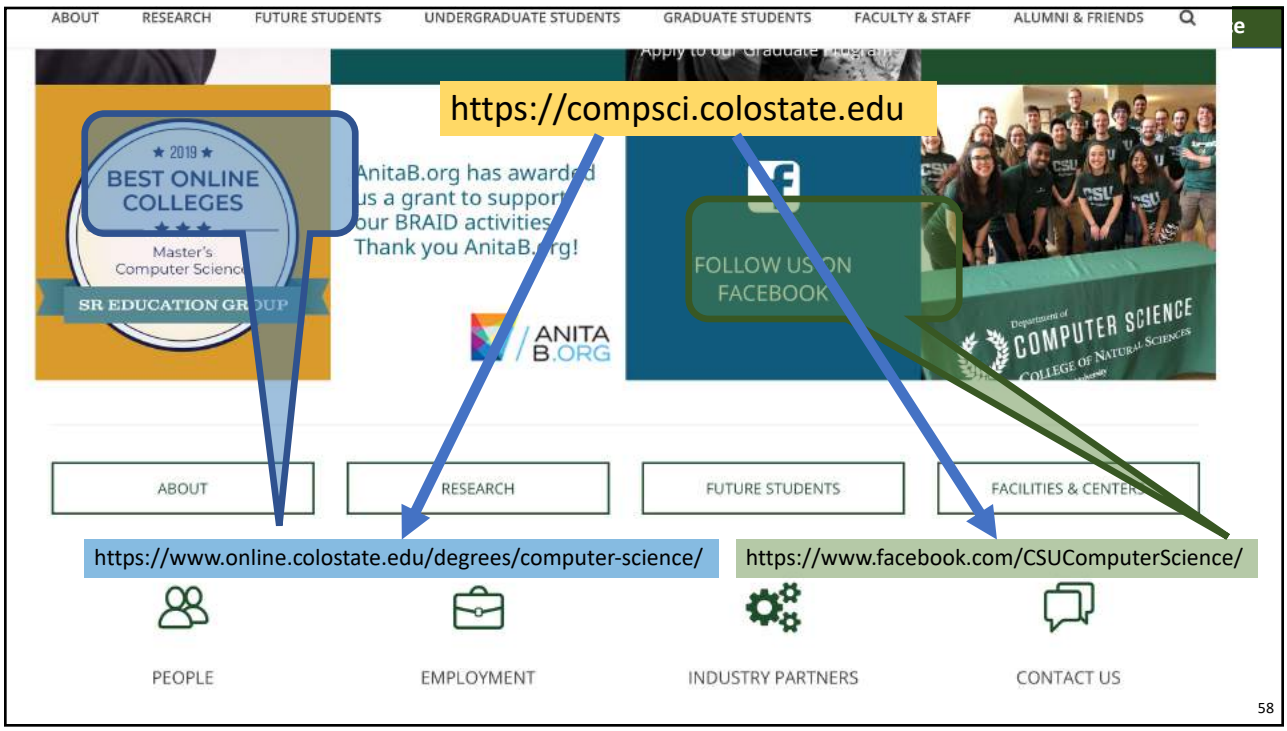
55

PageRank

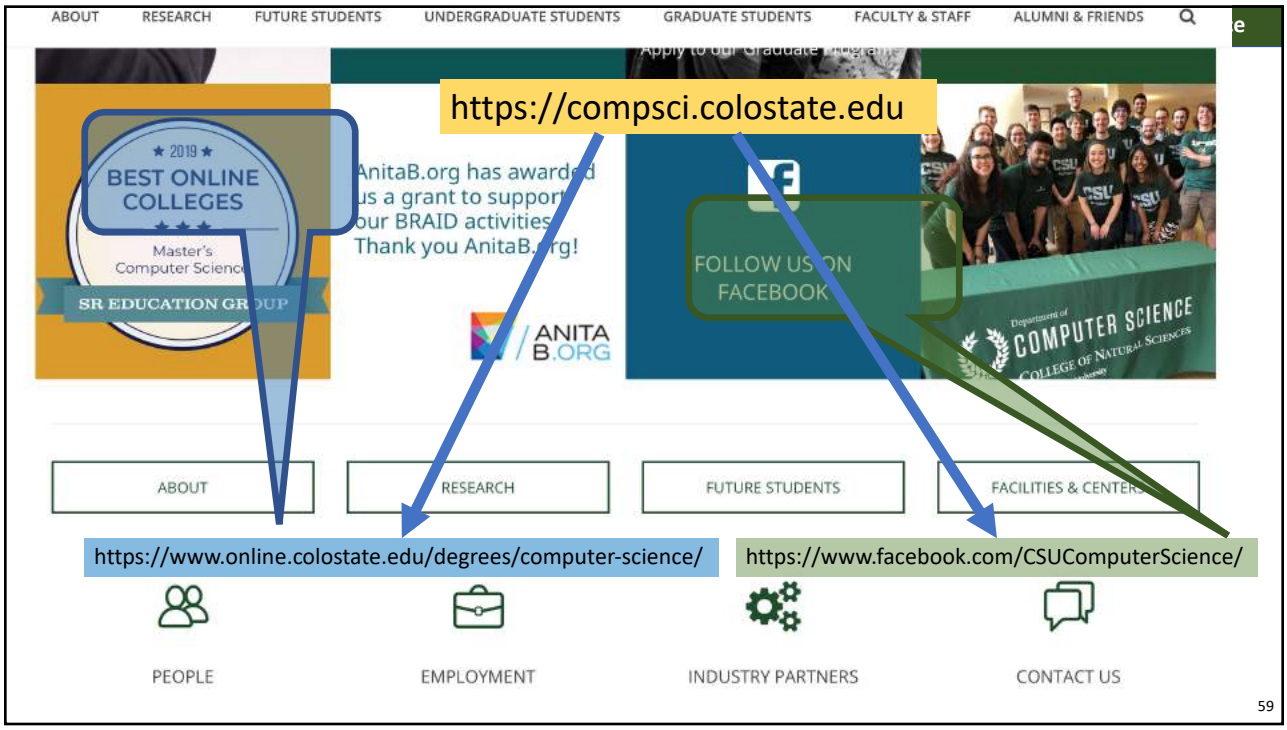
- Goals
  - Providing **effective summaries** for the search results
  - **Ordering/Ranking results**
- Simulate random Web surfers
  - Pages that would have a large number of surfers were considered more "important" than pages that would rarely be visited
- The content of a page was judged not only by the terms appearing on that page
  - But by the terms used in or near the links to that page

Definition of PageRank

- A function that assigns a real number to each page in the Web
- The **higher the PageRank** of a page, the **more "important"** it is
- There is NOT one fixed algorithm for assignment of PageRank


ABOUT
RESEARCH
FUTURE STUDENTS
UNDERGRADUATE STUDENTS
GRADUATE STUDENTS
FACULTY & STAFF
ALUMNI & FRIENDS
https://compsci.colostate.edu
★ 2019 ★
BEST ONLINE COLLEGES
Master's Computer Science
SR EDUCATION GROUP
AnitaB.org has awarded us a grant to support our BRAID activities. Thank you AnitaB.org!
ANITA B.ORG
FOLLOW US ON FACEBOOK
COMPUTER SCIENCE
ABOUT
RESEARCH
FUTURE STUDENTS
FACILITIES & CENTERS
https://www.online.colostate.edu/degrees/computer-science/
https://www.facebook.com/CSUComputerScience/
PEOPLE
EMPLOYMENT
INDUSTRY PARTNERS
CONTACT US
58

58


ABOUT
RESEARCH
FUTURE STUDENTS
UNDERGRADUATE STUDENTS
GRADUATE STUDENTS
FACULTY & STAFF
ALUMNI & FRIENDS
https://compsci.colostate.edu
★ 2019 ★
BEST ONLINE COLLEGES
Master's Computer Science
SR EDUCATION GROUP
AnitaB.org has awarded us a grant to support our BRAID activities. Thank you AnitaB.org!
ANITA B.ORG
FOLLOW US ON FACEBOOK
COMPUTER SCIENCE
ABOUT
RESEARCH
FUTURE STUDENTS
FACILITIES & CENTERS
https://www.online.colostate.edu/degrees/computer-science/
https://www.facebook.com/CSUComputerScience/
PEOPLE
EMPLOYMENT
INDUSTRY PARTNERS
CONTACT US
59

59

Questions?

Week 6 – Lecture 2
Web-Scale Link Analysis

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- Please sign up for the software demo – PA1
- PA2: October 14
  - Start early, finish early, and get your grade early!

1

Topics discussed in the previous class

- How MapReduce Works?
- Term Spam problem

Topics to be covered

- Web-Scale Link Analysis

Part 1. Large Scale Data Analytics

. Web-Scale Link Analysis and Social Network Analysis

Web-Scale Link Analysis

This material is built based on,

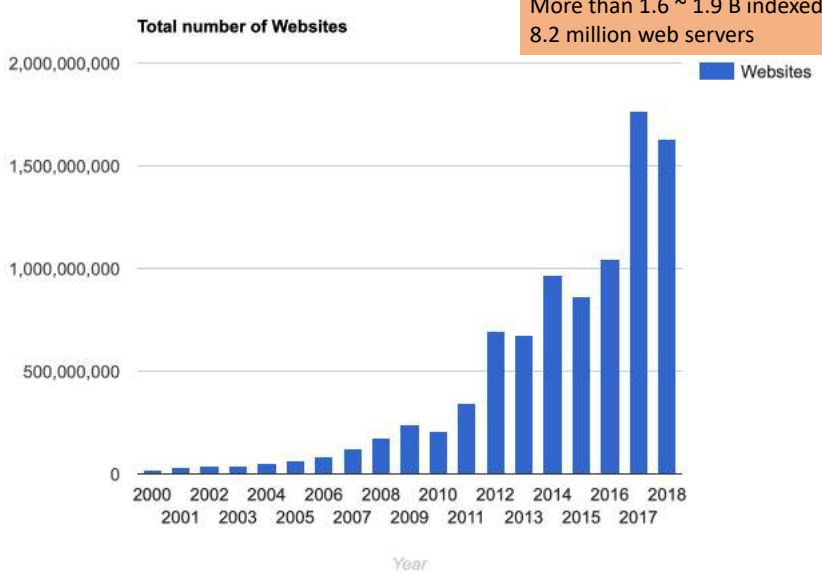
- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012
  - Chapter 5
- http://infolab.stanford.edu/~ullman/mmds.html

Part 1. Large Scale Data Analytics

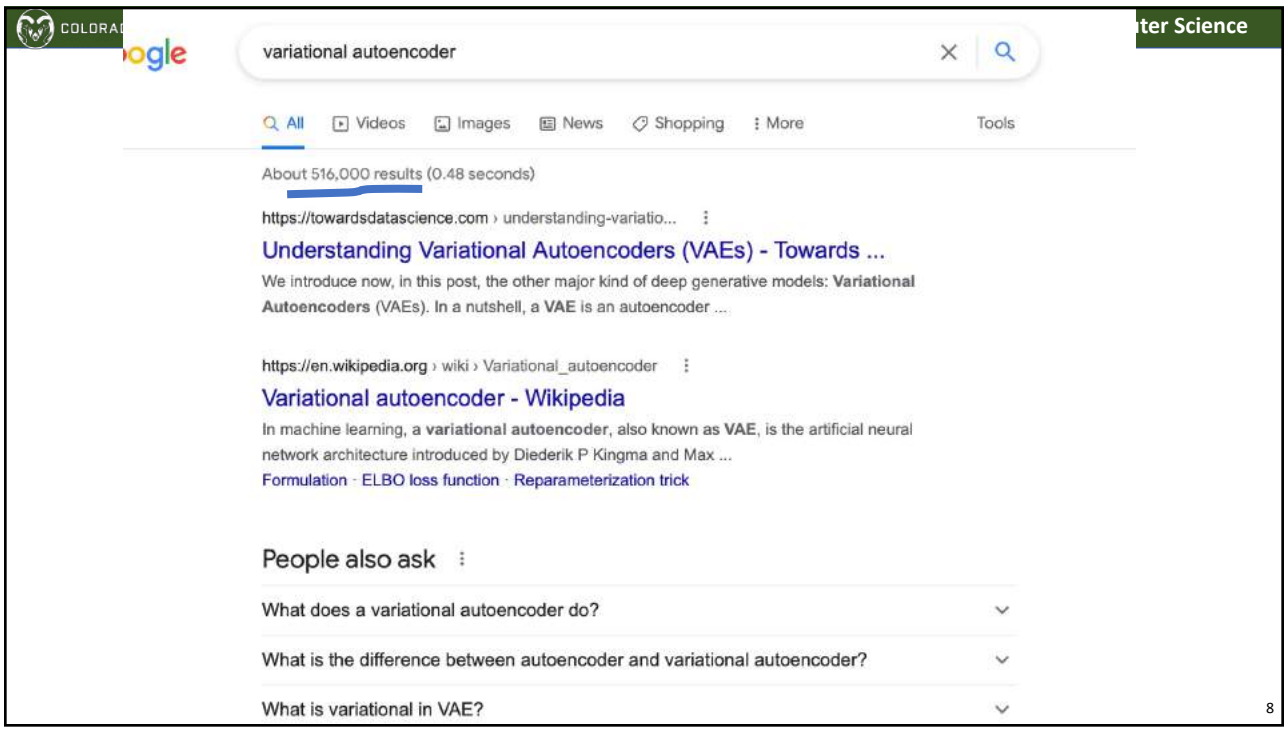
Web-Scale Link Analysis and Social Network Analysis

Web-Scale Link Analysis: PageRank Algorithm

Current total number of Web pages:
More than 1.6 ~ 1.9 B indexed pages across 8.2 million web servers



http://www.internetlivestats.com/total-number-of-websites/



8

8

Do people read all?

- Page one results enjoys 95% of all search traffic
- Traffic drops
  - 140% from 10th to 11th
  - 86% from 20th to 21st
- Traffic drops
  - 95% from page 1 to 2 of results
  - 78% from page 2 to 3
  - 58% from page 3 to 4
- In August 2012, Google launched 7 result page (instead of 10 result page)

9

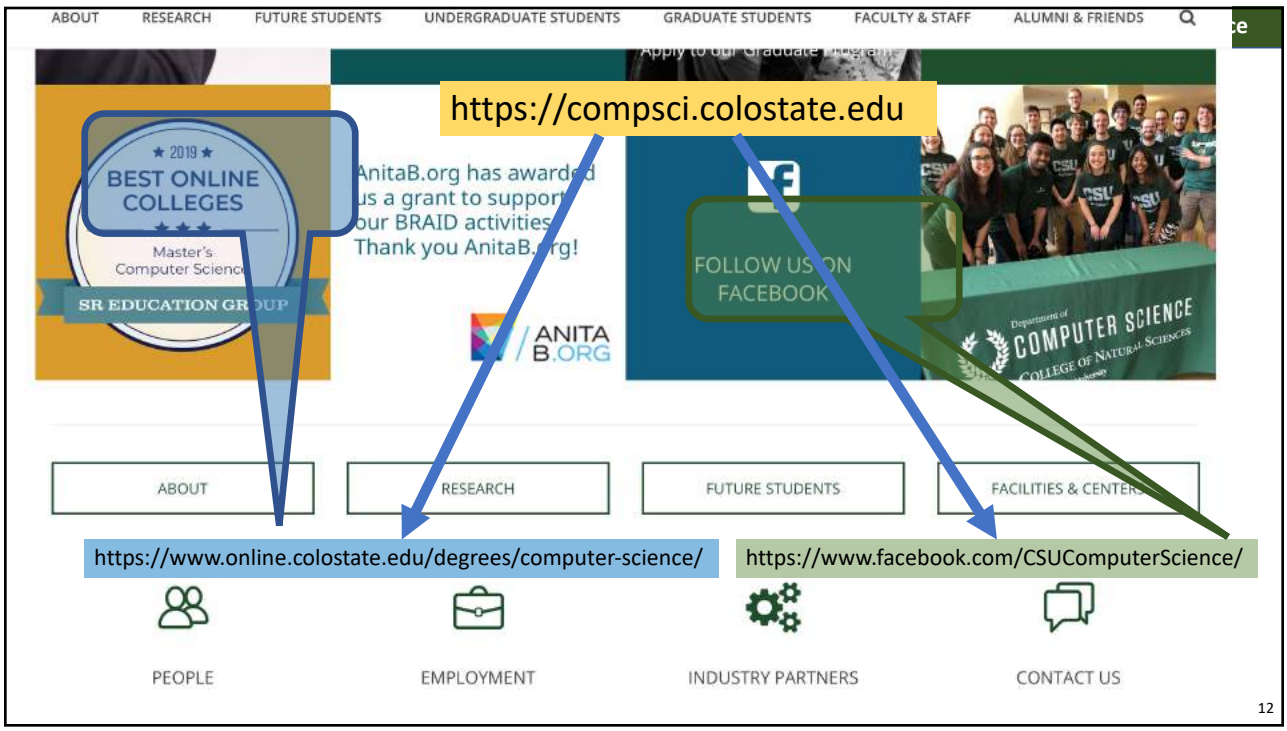
9

PageRank

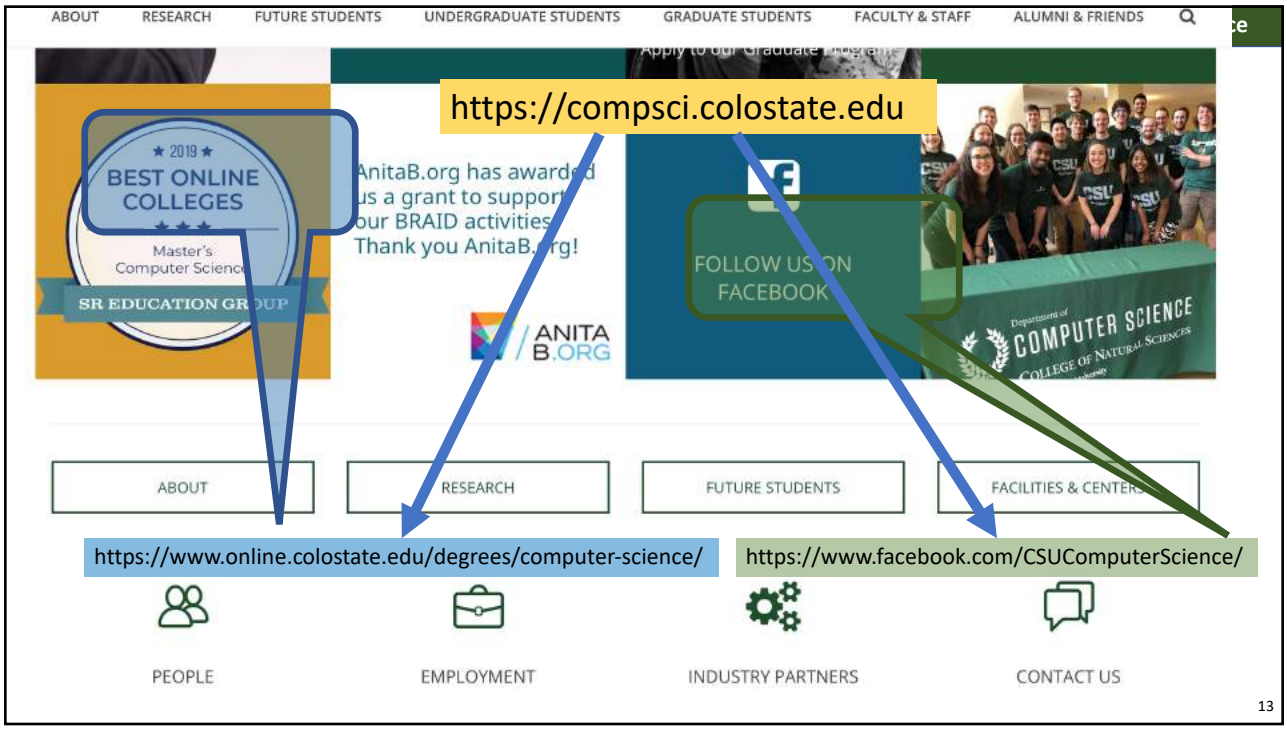
- Goals
  - Providing **effective summaries** for the search results
  - **Ordering/Ranking results**
- Simulate random Web surfers
  - Pages that would have a large number of surfers were considered more "important" than pages that would rarely be visited
- The content of a page was judged not only by the terms appearing on that page
  - But by the terms used in or near the links to that page

Definition of PageRank

- A function that assigns a real number to each page in the Web
- The **higher the PageRank** of a page, the **more "important"** it is
- There is NOT one fixed algorithm for assignment of PageRank


ABOUT
RESEARCH
FUTURE STUDENTS
UNDERGRADUATE STUDENTS
GRADUATE STUDENTS
FACULTY & STAFF
ALUMNI & FRIENDS
https://compsci.colostate.edu
★ 2019 ★
BEST ONLINE COLLEGES
Master's Computer Science
SR EDUCATION GROUP
AnitaB.org has awarded us a grant to support our BRAID activities. Thank you AnitaB.org!
ANITA B.ORG
FOLLOW US ON FACEBOOK
COMPUTER SCIENCE
ABOUT
RESEARCH
FUTURE STUDENTS
FACILITIES & CENTERS
https://www.online.colostate.edu/degrees/computer-science/
https://www.facebook.com/CSUComputerScience/
PEOPLE
EMPLOYMENT
INDUSTRY PARTNERS
CONTACT US
12

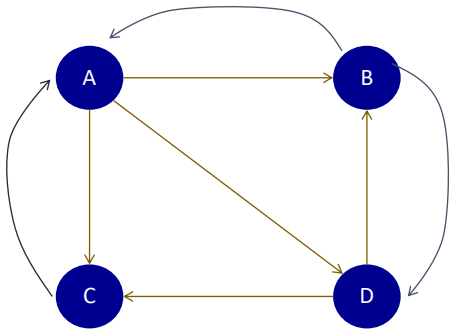
12


ABOUT
RESEARCH
FUTURE STUDENTS
UNDERGRADUATE STUDENTS
GRADUATE STUDENTS
FACULTY & STAFF
ALUMNI & FRIENDS
https://compsci.colostate.edu
★ 2019 ★
BEST ONLINE COLLEGES
Master's Computer Science
SR EDUCATION GROUP
AnitaB.org has awarded us a grant to support our BRAID activities. Thank you AnitaB.org!
ANITA B.ORG
FOLLOW US ON FACEBOOK
COMPUTER SCIENCE
ABOUT
RESEARCH
FUTURE STUDENTS
FACILITIES & CENTERS
https://www.online.colostate.edu/degrees/computer-science/
https://www.facebook.com/CSUComputerScience/
PEOPLE
EMPLOYMENT
INDUSTRY PARTNERS
CONTACT US
13

13

Example [1/5]

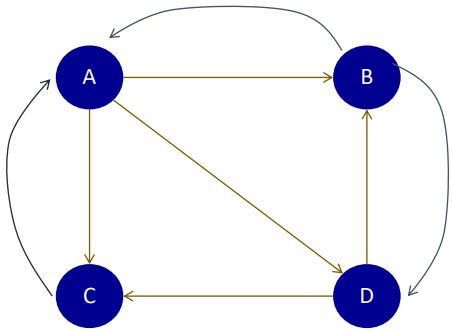
- Page A has links to B, C and D
- Page B has links to A and D
- Page C has a link to A
- Page D has links to B and C



Worksheet Question #1

- The URL of the page A is included in

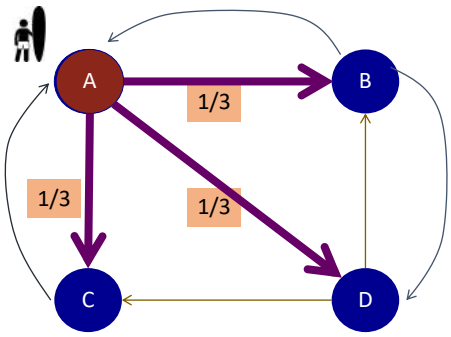
a. B, C, and D
b. A
c. B and C
d. None



Answer: c –B and C

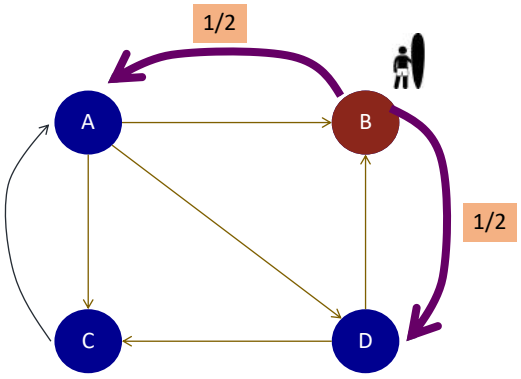
Example [2/5]

- Suppose that a **random surfer** starts at page A
- Page B, C and D will be the next with probability 1/3
  - 0 probability of being at A



Example [3/5]

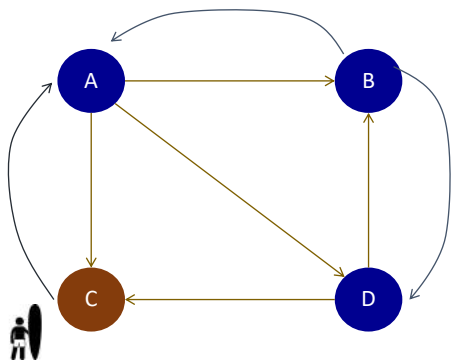
- Now suppose the random surfer at B
  - B has probability of ½ of being at A, ½ of being at D and 0 of being at B or C



Question

- Now suppose the random surfer at C
  - What is the probability that C will be at A?

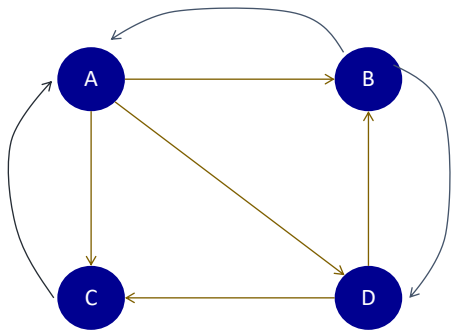
  a. 1/4 b. 1/2 c. 1 d. 0



Answer: C

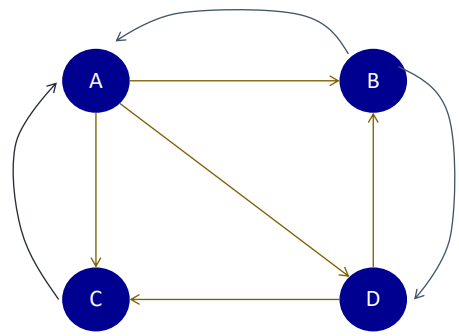
Example [4/5]

- **Transition matrix M**
  - What happens to random surfers after one step
  - M has n rows and columns (n pages)
  - What is the transition matrix for this example?



Example [5/5]

$$M = \begin{array}{c} \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \\ \begin{array}{cccc} A & B & & \end{array} \end{array}$$

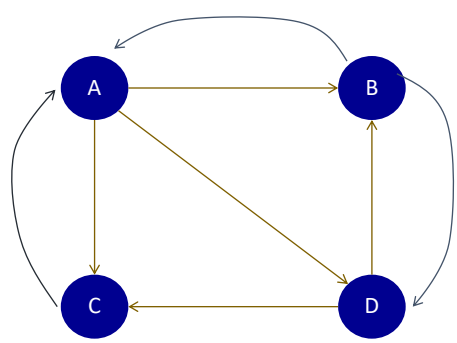


- The first column
  - a surfer at A has a 1/3 probability of next being at each of the other pages
- The second column
  - a surfer at B has a ½ probability of being next at A and the same for being at D

What does this matrix mean? [1/6]

- The **probability distribution** for the location of a random surfer
  - A column vector whose jth component is the probability that the surfer is at page j

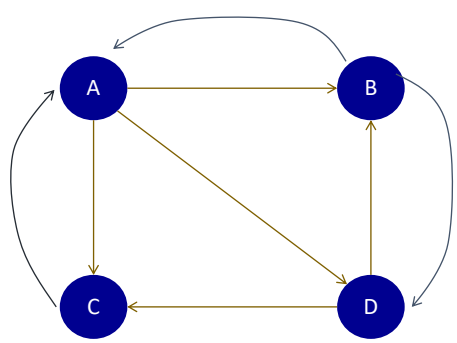
$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1.2 & 0 & 0 \end{bmatrix}$$



What does this matrix mean? [2/6]

- If we surf at any of the n pages of the Web with equal probability
  - **The initial vector** v_o will have $1/n$ for each component
  - If M is the transition matrix of the Web
    - After the first one step, the distribution of the surfer will be Mv_o
    - After two steps, $M(Mv_o) = M^2 v_0$ and so on

$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$



What does this matrix mean? [3/6]

- Multiplying the initial vector v_0 by M a total of i times
  - The distribution of the surfer **after i steps**

The probability for the next step from the current location

The probability for being in the current location

$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1.2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} =$$

The distribution of the surfer after the first step
- Probability for being in the next possible location

What does this matrix mean? [4/6]

- The probability x_i that a random surfer will be at node i at the next step

$$x_i = \sum_j m_{ij} v_j$$

- m_{ij} is the **probability that a surfer at node j will move to node i** at the next step
- v_j is the **probability that the surfer was at node j at the previous step**

What does this matrix mean? [5/6]

- The distribution of the surfer approaches **a limiting distribution v** that **satisfies $v = Mv$** provided two conditions are met:

1. The graph is **strongly connected**
   - It is possible to get from any node to any other node
2. There are **no dead ends**
   - Nodes that have no arcs out

What does this matrix mean? [6/6]

- This is a type of stochastic event models
- The limit is reached when multiplying the distribution by M another time does not change the distribution
  - The limiting v is an eigenvector of M
  - Since M is stochastic (its columns each add up to 1), v is the principle eigenvector
    - Its associated eigenvalue is the largest of all eigenvalues
- **The principle eigenvector of M**
  - **Where the surfer is most likely to be after a long time**
- Iterative computation
  - $Mv_0=v_1$, $Mv_1=v_2$, $Mv_2=v_3$.....
  - Until we find, $V_n == V_{n-1}$

For the Web, 50-75 iterations are sufficient to converge to within the error limits of double-precision arithmetic

Example

$$M=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- Suppose we apply this process to the matrix M
- The initial vector v0 and v1 after multiplying M

$$v_1=Mv_0=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix} = \quad ?$$

Example

$$M=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- Suppose we apply this process to the matrix M
- The initial vector v0 and v1 after multiplying M

$$v_1=Mv_0=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix}=\begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

Worksheet Question

$$M=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- Suppose we apply this process to the matrix M
- The initial vector v0 and v1 after multiplying M

$$v_1=Mv_0=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix}=\begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

What is v_2?

Example continued

- The sequence of approximations to the limit
  - We get by multiplying at each step by M is

$$\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix} \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix} \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix} \quad \ldots \quad \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

- This difference in probability is not noticeable
- In the real Web, there are billions of nodes of greatly varying importance
  - The probability of being at a node like www.amazon.com is orders of magnitude greater than others

Part 1. Large Scale Data Analytics

Web-Scale Link Analysis and Social Network Analysis

Web-Scale Link Analysis: Implementing the PageRank Algorithm using MapReduce

Overview

- Matrix-vector multiplication
  - Case 1: without partitioning the vector
  - Case 2: with partitioning the vector
- Random-walk style

Matrix-vector Multiplication by MapReduce [1/3]

- Suppose we have an n x n matrix M, whose element in **row** i and **column** j will be denoted M_{ij}
- Then the matrix-vector product is the vector x of length n, whose i^{th} element x_i is given by,

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Matrix-vector Multiplication by MapReduce [2/3]

- If n = 100, we do **NOT** need DFS or MapReduce
- However, if this calculation is a part of ranking Web pages (n is 10M) that goes on at search engine? The vector v cannot fit in main memory
  - More than 1.9B pages

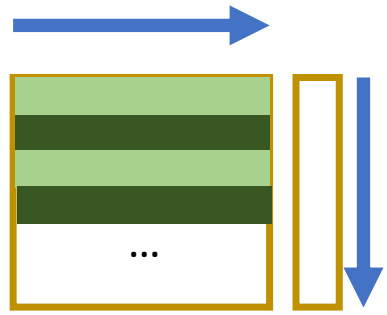
Matrix-vector Multiplication by MapReduce [3/3]

- The **matrix** M and the **vector** v each will be stored in a file of the **DFS(HDFS)**
- Assume that row-column coordinates of each matrix element will be **discoverable**
  - Either from its position in the file or explicit coordinates

CASE 1: FUNDAMENTA MATRIX vs. VECTOR MULTIPLICATION: Without partitioning the vector

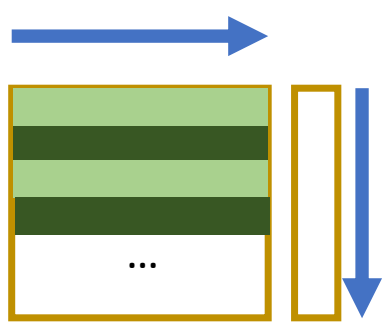
The Map function

- The Map function is written to apply to one element of M
- Each Map task will operate on a chunk of the matrix M
- From each matrix element m_{ij}, it produces the key-value pair $(i, m_{ij}v_j)$
  - i is the row number and j is the column number
- All terms of the sum that make up the component x_i of the matrix-vector product will get the same key, i (row #)



The Reduce function

- Sums all the values associated with a given key i
- The result will be a pair (i, x_i)

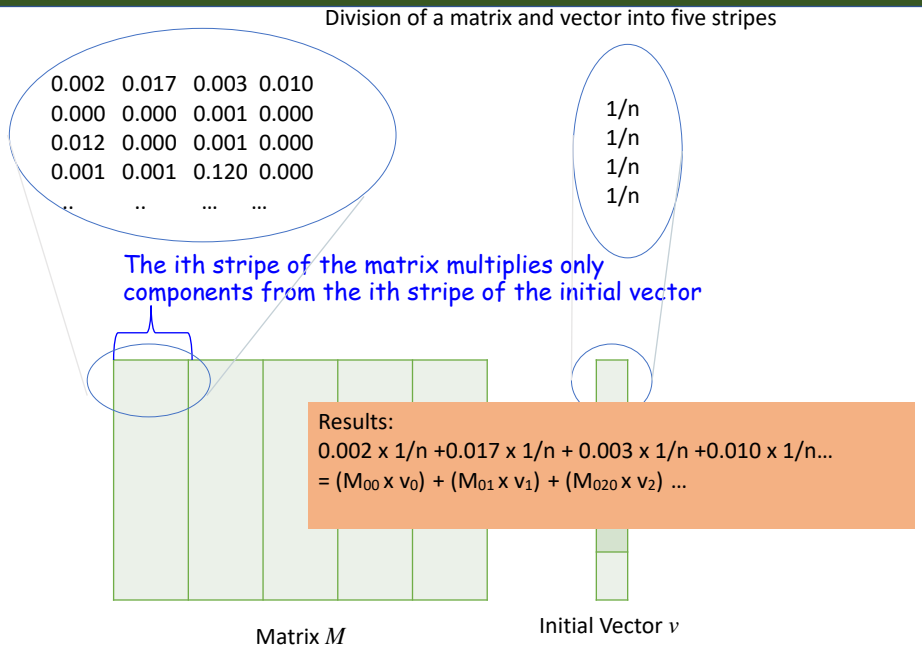


CASE 2: WITH VERY LARGE v (partitioning needed)

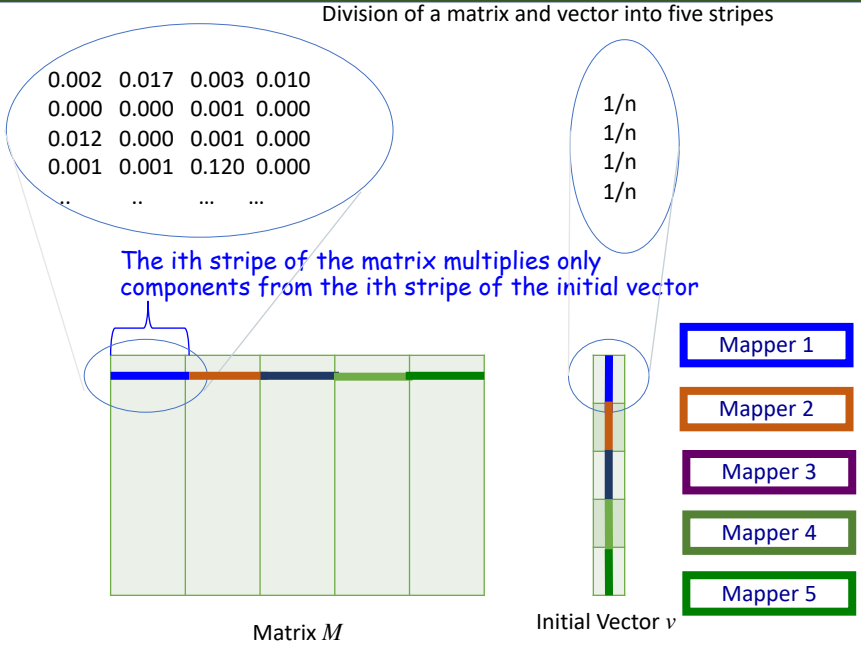
What if the vector v cannot fit in main memory?

- It is possible that the vector v is so large that it will not fit in its main memory entirely
- We can divide the matrix into vertical stripes of equal width and divide the vector into an equal number of horizontal stripes of the same height
  - The goal is to use enough stripes so that the portion of the vector in one stripe can fit into main memory

Division of a matrix and vector into five stripes



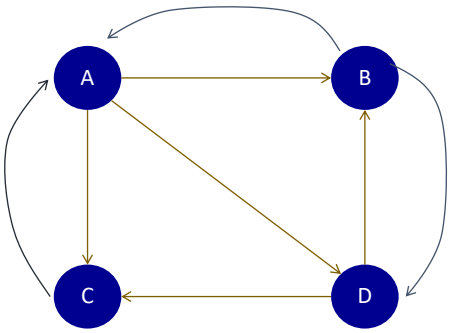
Division of a matrix and vector into five stripes



Random-walk style

Random-walk style implementation

- **Without a transition matrix**
- By performing the **next visit**, the page A will share its current page rank value with the page B, C, or D as much as the probability to be in one of these pages
  - E.g. Suppose the current page rank value of A is PR(A). This will be distributed to the page B, C, and D as much as PR(A)*1/3
- As the result of the **next visit**, each page will update the page rank value as the sum of all the passed page rank values
  - E.g. C's new_PR(C)=
    current_PR(A)*1/3 + current_PR(D)*1/2



Random-walk style implementation with MapReduce

- Mapper
  - Input: (Page-ID, a list of destinations in the out-links, current_PR(Page-ID))
  - Output: (a destination, (current_PR $_{(Page\text{-}ID)}$ / length of the list of destinations))
  - Mapper should emit (Page-ID, a list of destinations in the out-links) once also
    - To generate a final output from Reducer
    - Otherwise, your mapreduce should join the information
- Reducer
  - Reducer will have input <Page-ID, [values]>
    - There will be two types of values: partial PR values or the original input tuple
  - For the partial PR: Adding all of partial PR values
  - Generate the final output using the original input tuple
  - Emit a new (**Page-ID, a list of destinations in the out-links**, **new_PR(Page-ID**)) for the next iteration

Random-walk style implementation with MapReduce

- Initial page rank value
  - 1/(# of pages)
- Iterate this computation
  - Until the per-set number of iterations
    - More practical
  - Until the PR values converge

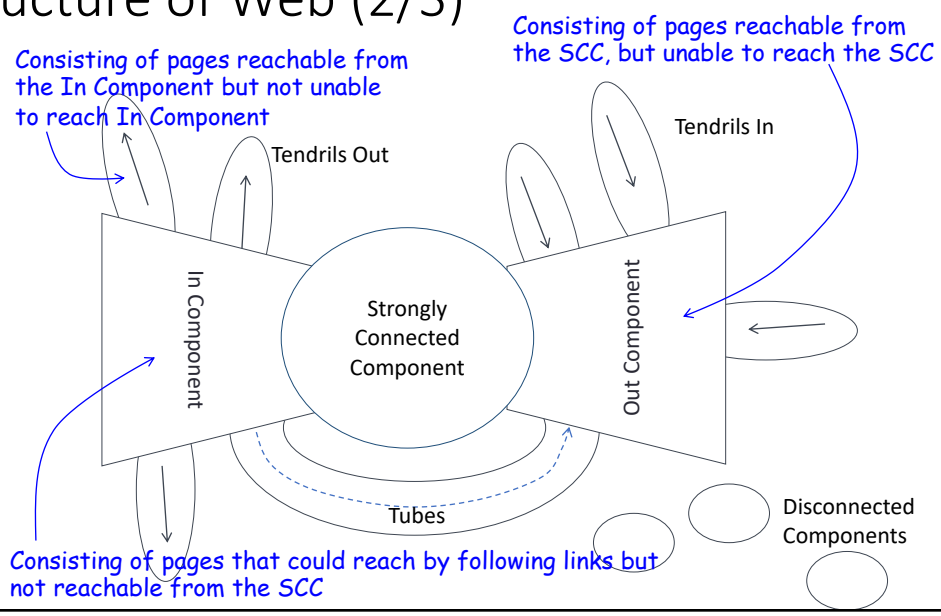
Link Analysis

PageRank Algorithm for the real Web

Structure of Web (1/3)

- Let's revisit our big assumptions
- **Is Web strongly connected?**
- **Does Web contain NO dead end?**

Structure of Web (2/3)



Structure of Web (3/3)

- Tubes
  - Pages reachable from the in-component and able to reach the output-component, but unable to reach the SCC or be reached from the SCC
- Isolated components
  - Unreachable from the large components

Anomalies from the Web structure

- These structures violate the assumptions needed for the Markov process iteration to converge to a limit
  - When a random surfer enters the out-component, they can never leave
  - Surfers starting in either the SCC or in-component are going to wind up in either the out-component or a tendril off the in-component
  - No page in the SCC or in-component winds up with any probability of a surfer being there
- Nothing in the SCC or in-component will be of any importance

Link Analysis

Challenges in PageRank Algorithm for the real Web

Problems we need to avoid

- Dead end
  - A page that has no links out
  - Surfers reaching such a page will disappear
  - In the limit, no page that can reach a dead end can have any PageRank at all
- Spider traps
  - Groups of pages that all have outlinks but they never link to any other pages

Questions?

Week 7 – Lecture 1
Web-Scale Link Analysis

Sangmi Lee Pallickara
Computer Science, Colorado State University

0

FAQs

- MapReduce version of PageRank calculation
- Random walk style

1

Topics discussed in the previous class

- Web-Scale Link Analysis: PageRank Algorithm

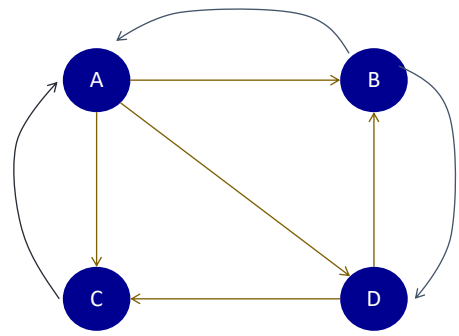
Topics to be covered

- Web-Scale Link Analysis – Dead end and Spider Trap
- Overview of Midterm (Last 10 minutes)

REVIEW

Example

$$M=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$



- Suppose we apply this process to the matrix M
- The initial vector v0 and v1 after multiplying M

$$v_1=Mv_0=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}=\begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

REVIEW

Example continued

- The sequence of approximations to the limit
  - We get by multiplying at each step by M is

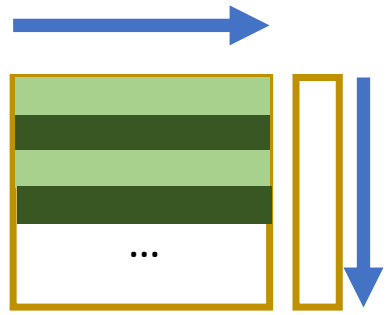
$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}\begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}\begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}\begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix}\ \ldots\ \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

- This difference in probability is not noticeable
- In the real Web, there are billions of nodes of greatly varying importance
  - The probability of being at a node like www.amazon.com is orders of magnitude greater than others

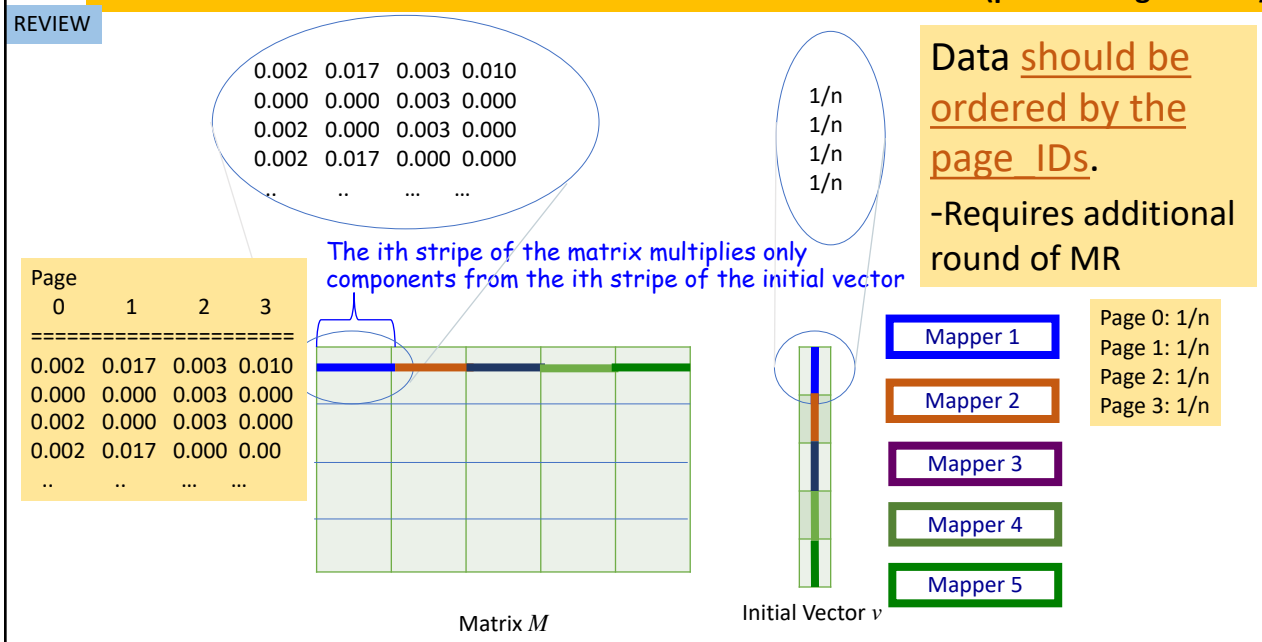
REVIEW **: MATRIX vs. VECTOR MULTIPLICATION: Without partitioning the vector**

The Map function

- The Map function is written to apply to one element of M
- Each Map task will operate on a chunk of the matrix M
- From each matrix element m_{ij}, it produces the key-value pair $(i, m_{ij}v_j)$
  - i is the row number and j is the column number
- All terms of the sum that make up the component x_i of the matrix-vector product will get the same key, i (which is row #)



REVIEW **CASE 2: MATRIX vs. VECTOR MULTIPLICATION: WITH VERY LARGE v (partitioning needed)**



CASE 2: MATRIX vs. VECTOR MULTIPLICATION: WITH VERY LARGE v (partitioning needed)

REVIEW

```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

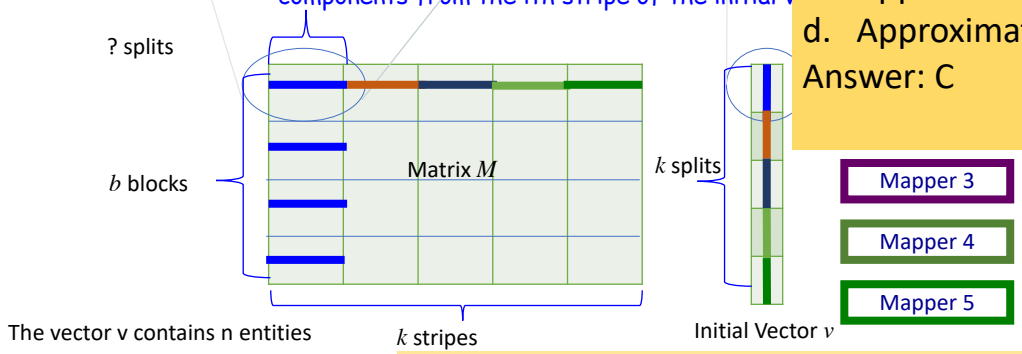
Page 0: 1/n
Page 1: 1/n
Page 2: 1/n
Page 3: 1/n

Q. What will be the number of Input Splits for M?

a. Approximately k
b. Approximately n/k
c. Approximately k x b
d. Approximately n

Answer: C

The ith stripe of the matrix multiplies only components from the ith stripe of the initial v



See the `InputFormat()` for the Cartesian Product (in the Join design pattern)

8

CASE 2: MATRIX vs. VECTOR MULTIPLICATION: WITH VERY LARGE v (partitioning needed)

REVIEW

```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

Page 0: 1/n
Page 1: 1/n
Page 2: 1/n
Page 3: 1/n

Input split

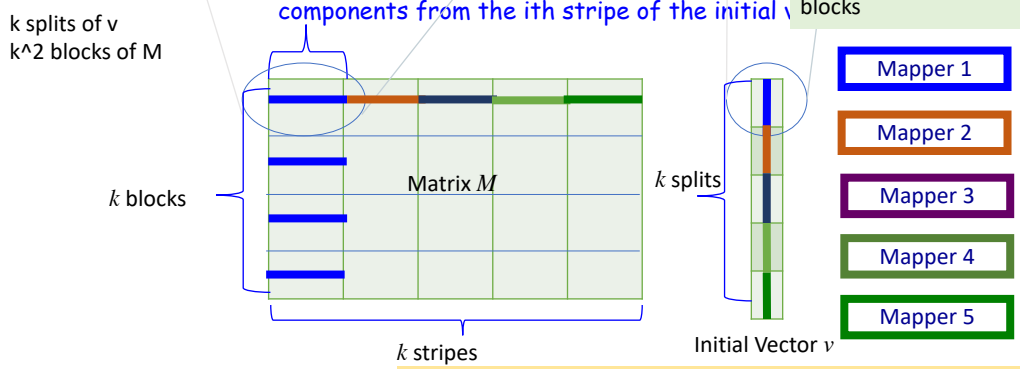
Mapping blocks within a stripe of M to the corresponding split of v

e.g.

x^{th} block in the ith strip of M should be paired with the ith split of v

A split of v will be paired with k different blocks

The ith stripe of the matrix multiplies only components from the ith stripe of the initial v



See the `InputFormat()` for the Cartesian Product (in the Join design pattern)

9

CASE 2: MATRIX vs. VECTOR MULTIPLICATION: WITH VERY LARGE v (partitioning needed)

REVIEW

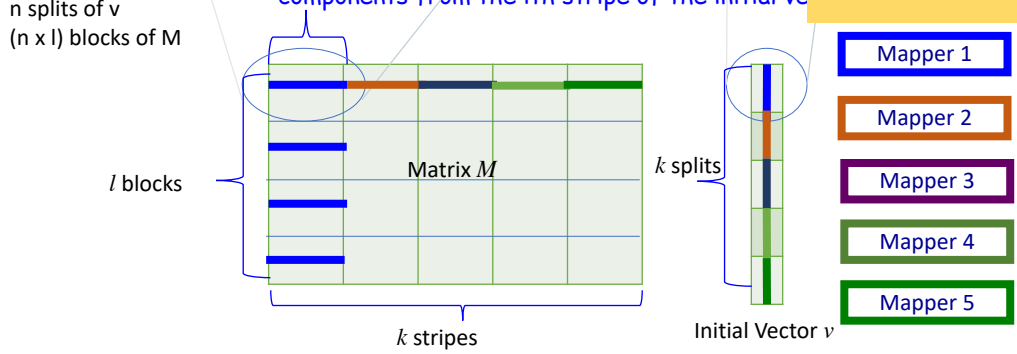
```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

Page 0: 1/n
Page 1: 1/n
Page 2: 1/n
Page 3: 1/n

Q. What is the Mapper output?

a. (null, (col#, row#, $M_{row, col}$ value))
b. (col#, $M_{row, col}$ value)
c. (row # of M, v_{row} value)
d. (row# of v, local sum of page rank)

The ith stripe of the matrix multiplies only components from the ith stripe of the initial ve



10

10

CASE 2: MATRIX vs. VECTOR MULTIPLICATION: WITH VERY LARGE v (partitioning needed)

REVIEW

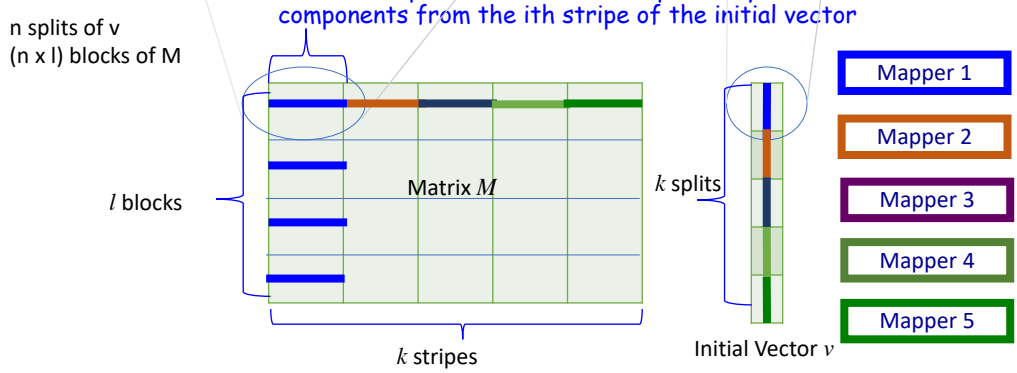
```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

Page 0: 1/n
Page 1: 1/n
Page 2: 1/n
Page 3: 1/n

Map functionality

Emit (row #, local sum of $M_{i,j}$ x v_i)

The ith stripe of the matrix multiplies only components from the ith stripe of the initial vector



11

11

CASE 2: MATRIX vs. VECTOR MULTIPLICATION: WITH VERY LARGE v (partitioning needed)

REVIEW

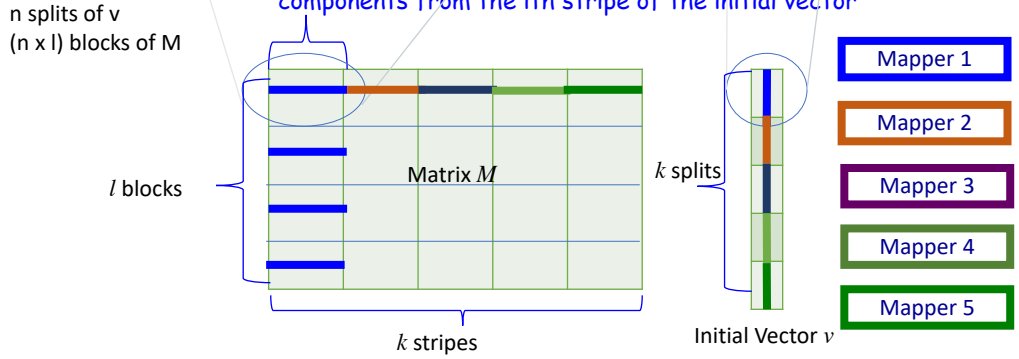
```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

```
Page 0: 1/n
Page 1: 1/n
Page 2: 1/n
Page 3: 1/n
```

What does the reduce function do?

Add all the local sum of pagerank values of the row i, and store it in the ith element of v

The ith stripe of the matrix multiplies only components from the ith stripe of the initial vector



12

12

COLORADO STATE UNIVERSITY — CS435 Introduction to Big Data: Week 7-A — Fall 2021 Computer Science

REVIEW

Question

- To implement previous Matrix-vector multiplication, which design pattern can be used?

a. Organization pattern
b. Cartesian product join pattern
c. Input/output pattern
d. All of the above

13

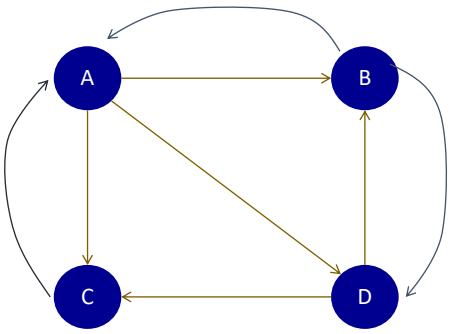
13

REVIEW

Random-walk style implementation

- **Without a transition matrix**

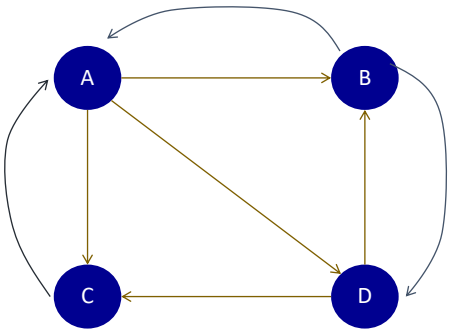
- By performing the **next visit**, the page A will propagate its current page rank value to the page B, C, or D as much as the probability to be in one of these pages
  - E.g. Suppose the current page rank value of A is PR(A). This will be distributed to the page B, C, and D as much as PR(A)*1/3



REVIEW

Random-walk style implementation

- As the result of the **next visit**, each page will update its page rank value as the sum of all the passed page rank values
  - E.g. C's new_PR(C)=
    current_PR(A)*1/3 + current_PR(D)*1/2



Part 1. Large Scale Data Analytics

Web-Scale Link Analysis and Social Network Analysis

PageRank Algorithm for the real Web

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 7-A Fall 2021 Computer Science

Part 1. Large Scale Data Analytics

Web-Scale Link Analysis and Social Network Analysis

Challenges in PageRank Algorithm for the real Web

Problems we need to avoid

- **Dead end**
  - A page that has no links out
  - Surfers reaching such a page will disappear
  - In the limit, no page that can reach a dead end can have any PageRank at all
- **Spider traps**
  - Groups of pages that all have out links but they never link to any other pages

Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

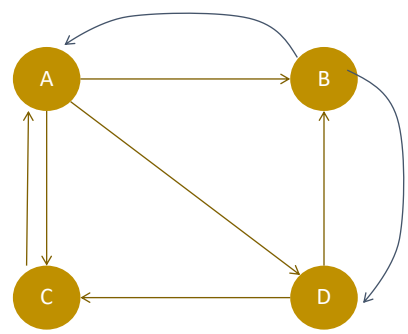
Challenges in PageRank Algorithm for the real Web: (1) Dead ends

Avoiding Dead Ends

- **What if we allow dead ends?**
  - The transition matrix of the Web is **no longer stochastic**
    - Some of the columns will sum to 0 rather than 1
- If we compute $M^i v$ for increasing powers of a substochastic matrix
  - Some of all of the components of the vector go to 0
  - substochastic matrix
    - A matrix whose column sums are **at most** 1
  - **Importance "drains out" of the Web**
    - No information about the relative importance of pages

Example

- Remove the arc from C to A
  →C becomes a **dead end**
  If a random surfer reaches C, they disappear at the next round



$$M=\begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

Example-continued

- Repeatedly multiplying the vector by M:

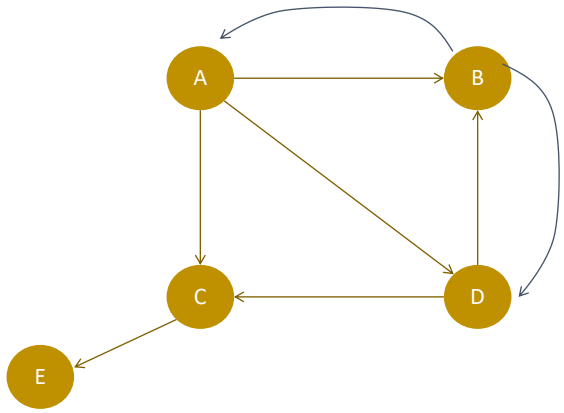
$$\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix} \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix} \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix} \ldots \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- **The probability of a surfer being anywhere goes to 0** as the number of steps increase

Approaches to dealing with dead ends

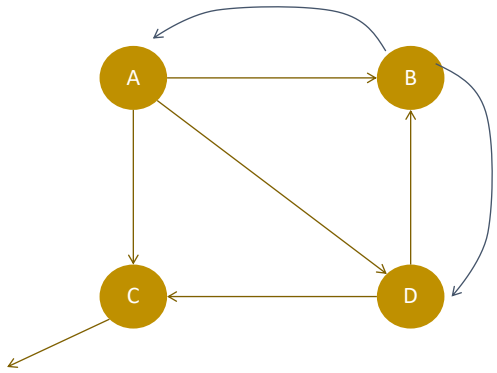
- **Recursive deletion**
  - Drop the dead ends from the graph
    - Drop their incoming arcs as well
    - Doing so **may create more dead ends**
      - Drop those new dead ends
- **Taxation**
  - Modify the process
    - Random surfers are assumed to move around on the Web without links

Example of recursive deletion (1/4)

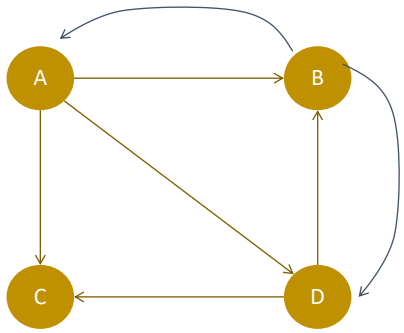


COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 7-A Fall 2021 Computer Science

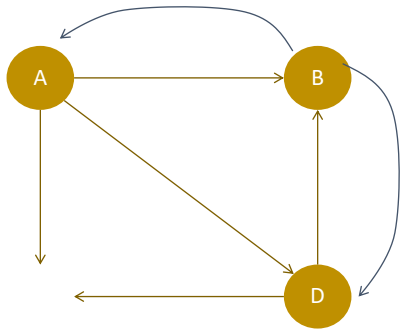
Example of recursive deletion (1/4)



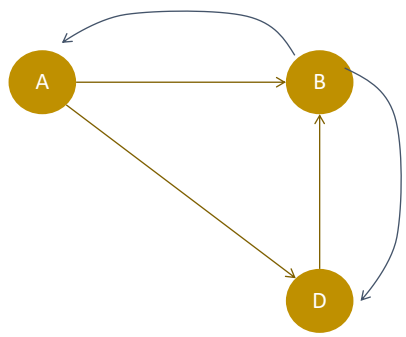
Example of recursive deletion (1/4)



Example of recursive deletion (1/4)



Example of recursive deletion (1/4)



Example of recursive deletion (2/4)

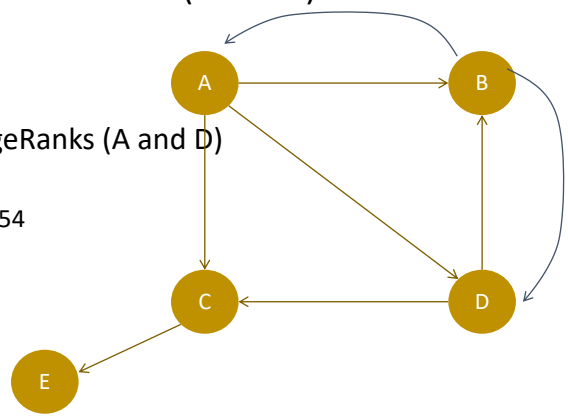
- The final matrix for the graph is

$$M = \begin{bmatrix} 0 & ½ & 0 \\ ½ & 0 & 1 \\ ½ & ½ & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \begin{bmatrix} 3/12 \\ 5/12 \\ 4/12 \end{bmatrix} \begin{bmatrix} 5/24 \\ 11/24 \\ 8/24 \end{bmatrix} \quad \ldots \quad \begin{bmatrix} 2/9 \\ 4/9 \\ 3/9 \end{bmatrix}$$

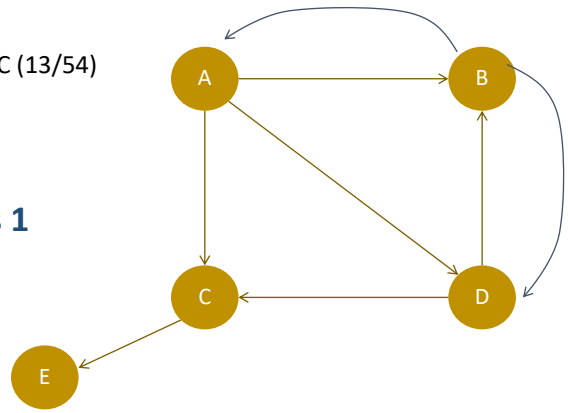
Example of recursive deletion (3/4)

- We still need to compute for the deleted nodes (C and E)
- C was the last one deleted
  - We know all its predecessors have PageRanks (A and D)
  - Therefore,
    - PageRank of C = 1/3 x 2/9 + ½ x 3/9 = 13/54



Example of recursive deletion (4/4)

- Now, we can compute the PageRank value for E
  - Only one predecessor, C
    - The PageRank of E is the same as that of C (13/54)
- The sums of the PageRanks **exceeds 1**
  - Still, it provides a good estimate



Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

Challenges in PageRank Algorithm for the real Web: (2) Spider Traps

Spider Traps and Taxation

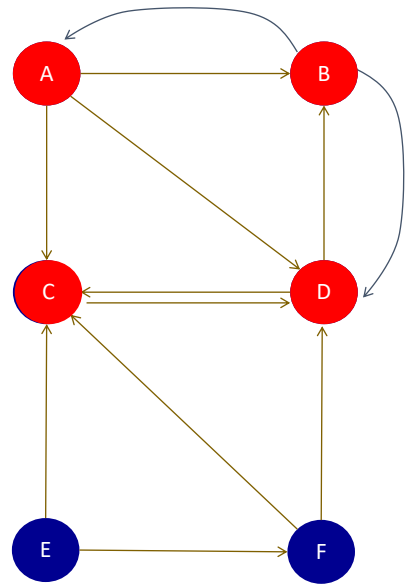
- **Spider traps**
  - A set of nodes with **no dead ends but no arcs out** (from the set of nodes to the outside of the set)
  - This can appear intentionally or unintentionally on the Web
- Spider traps causes the PageRank calculation to **place all the PageRank within the spider traps**

Example 1

- There is a simple spider trap of 4 nodes

$$M=\begin{bmatrix} 0 & 1/2 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 1/2 & 1/2 \\ 1/3 & 1/2 & 1 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$

Is there a Spider Trap in this graph?

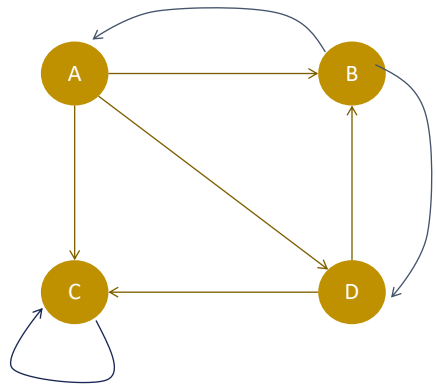


- If we perform the usual iteration to compute the PageRank of the nodes, we get

$$\begin{bmatrix} 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \end{bmatrix} \begin{bmatrix} 1/12 \\ 5/36 \\ 11/36 \\ 4/18 \\ 0 \\ 1/12 \end{bmatrix} \begin{bmatrix} 0.042 \\ 0.046 \\ 0.194 \\ 0.222 \\ 0 \\ 0 \end{bmatrix} \ldots$$

Example 2

- The arc out of C changed to point to C itself
- C is a simple spider trap of one node
  - It still has outgoing edge to itself
  - It is not a dead end



$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- If we perform the usual iteration to compute the PageRank of the nodes, we get

$$\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix} \begin{bmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{bmatrix} \begin{bmatrix} 21/288 \\ 31/288 \\ 205/288 \\ 31/288 \end{bmatrix} \quad \ldots \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- **All the PageRank is at C**
  - Once there, a random surfer can never leave

To avoid Spider traps,

- We modify the calculation of PageRank
  - Allowing each random surfer a small probability of ***teleporting*** **to a random page**
  - Rather than following an out-link from their current page

- The iterative step, where we compute a new vector estimate of PageRanks v' from the current PageRank estimate v and the transition matrix M is

$$v' = \beta M v + (1-\beta) e / n$$

- Where **β is a chosen constant**
  - Usually in the range 0.8 to 0.9

- **e is a vector for all 1's** with the appropriate number of components
- **n is the number of nodes** in the Web graph

$$v' = \beta M v + (1-\beta)e/n$$

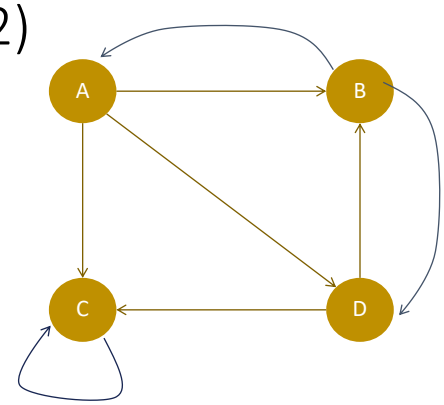
- The **term βMv** represents the case where,
  - With probability β, the random surfer decides to **follow an out-link** from their present page
- The **term $(1-\beta)e/n$** is a vector
  - Each of whose components has value $(1-\beta)/n$
  - Represents the introduction with probability $1-\beta$ of a **new random surfer at a random page**

- **If the graph has no dead ends:**
  - The probability of introducing a new random surfer is exactly equal to the probability that the random surfer will decide to follow a link from their current page
  - Surfer decides either to follow a link or teleport to a random page

- **If the graph has dead ends:**
  - The surfer goes nowhere
  - The term $(1-\beta)e/n$ does not depend on the sum of the components of the vector v, there will be some fraction of a surfer operating on Web
  - When there are dead ends, **the sum of the components of v may be less than 1**
    - But it will never reach 0

Example of Taxation (1/2)

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$



$$v' = \beta M v + (1-\beta)e/n$$

$$\beta = 0.8$$

$$v' = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} v + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

Example of Taxation (2/2)

- First few iterations:

$$\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix} \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix} \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix} \begin{bmatrix} 543/4500 \\ 707/4500 \\ 2543/4500 \\ 707/4500 \end{bmatrix} \quad \ldots \quad \begin{bmatrix} 15/148 \\ 19/148 \\ 95/148 \\ 19/148 \end{bmatrix}$$

- By being a spider trap, **C gets more than ½ of the PageRank** for itself
  - Effect is limited
  - Each of the nodes gets some of the PageRank

Part 1. Large Scale Data Analytics

Web-Scale Link Analysis and Social Network Analysis

Challenges in PageRank Algorithm for the real Web: Examples

Example 1

- Compute the PageRank of each page assuming $\beta=0.8$

$$M=\begin{bmatrix} 1/3 & 1/2 & 1/2 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 0 \end{bmatrix}$$

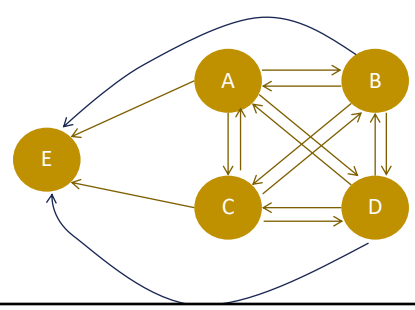
$$v_0=\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\beta\, M\, v_0 + (1-\beta)e/n$$

$$= 0.8 \times \begin{bmatrix} ⅓ & ½ & ½ \\ ⅓ & 0 & ½ \\ ⅓ & ½ & 0 \end{bmatrix} \times \begin{bmatrix} ⅓ \\ ⅓ \\ ⅓ \end{bmatrix} \quad + \quad (1-0.8)e/3$$

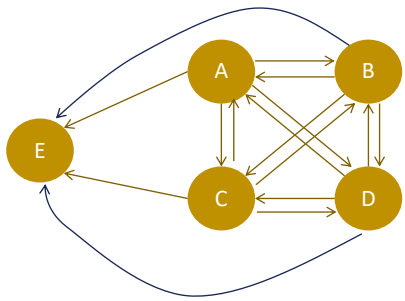
Example 2

- ***clique***
  - Set of nodes with all possible arcs from one to another
- Suppose the Web consists of a clique of n nodes and a single additional node that is the successor of each of the n nodes in the clique



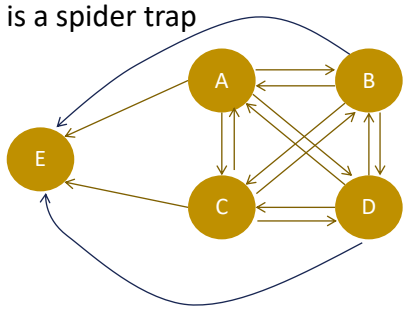
Example 2

- Determine the PageRank of each page assuming $\beta=0.8$
- Is there a dead end?
- Is there a spider trap?
- Is there a clique?

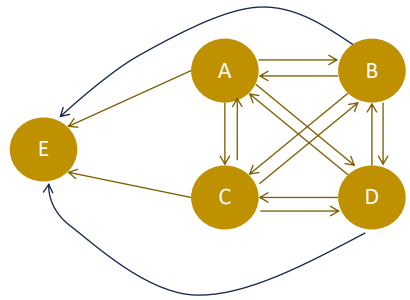


Example 2

- Determine the PageRank of each page assuming $\beta=0.8$
- Is there a dead end?
  - Yes. E
- Is the clique a spider trap? (updated)
  - No. Having clique does not mean that there is a spider trap



Example 2



$$M=\begin{bmatrix} 0 & ¼ & ¼ & ¼ & 0 \\ ¼ & 0 & ¼ & ¼ & 0 \\ ¼ & ¼ & 0 & ¼ & 0 \\ ¼ & ¼ & ¼ & 0 & 0 \\ ¼ & ¼ & ¼ & ¼ & 0 \end{bmatrix}$$

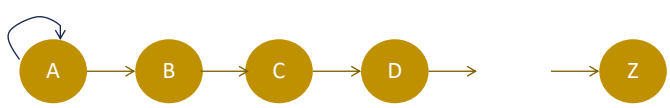
$$v_0=\begin{bmatrix} ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \end{bmatrix}$$

$$\beta M v_0 + (1-\beta)e/n$$

$$= 0.8 \times \begin{bmatrix} 0 & ¼ & ¼ & ¼ & 0 \\ ¼ & 0 & ¼ & ¼ & 0 \\ ¼ & ¼ & 0 & ¼ & 0 \\ ¼ & ¼ & ¼ & 0 & 0 \\ ¼ & ¼ & ¼ & ¼ & 0 \end{bmatrix} \begin{bmatrix} ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \end{bmatrix} + (1-0.8) \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} /5$$

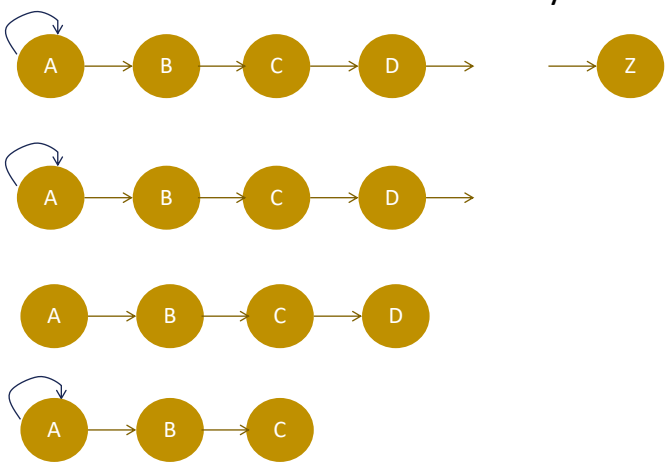
Example 3

- Suppose that we recursively eliminate dead ends from the Web graph to solve the remaining graph
- Suppose that the graph is a chain of dead ends, headed by a node with a self-loop
- What would be the PageRank assigned to each of the nodes?



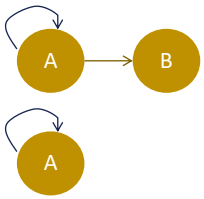
Example 3

- Remove all of the dead ends recursively



Example 3

- Remove all of the dead ends recursively

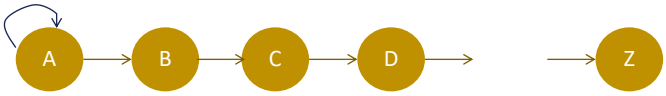


- What is v_0 and M?

Example 3

- What is v_0 and M?

- $v_0 = [1], M = [1], PageRank\ of\ A = 1$
- $PageRank\ of\ B = ½ \times 1$
- $PageRank\ of\ C = PageRank\ of\ B = ½$
- $PageRank\ of\ D = PageRank\ of\ C = ½$
- …



Questions?