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?

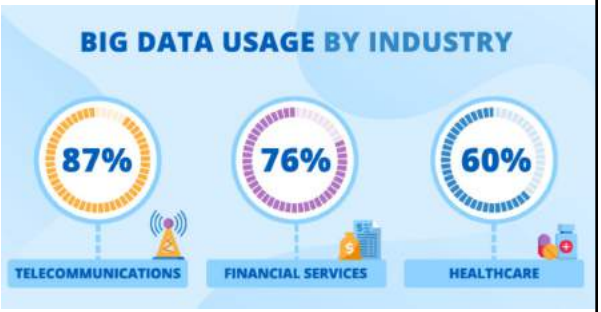
- Netflix
  - Predicts what exactly their customers will enjoy watching with Big Data
  - Recommendation engine
  - Analyzing customer's preferences for the content creating
- Government
  - Public services
  - FDA uses Big Data to detect and study patterns of food-related illnesses and diseases

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

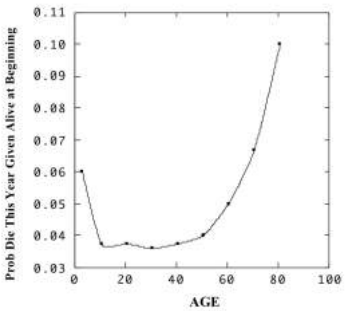


https://www.scnsoft.com/blog/big-data-use-cases-stats-and-examples

Use all of the data, or just a little? [1/5]

John Graunt

London, 17th century



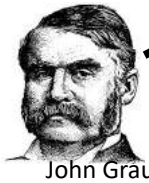
- Founder of the science of demography: *Bills of Mortality*
  - Estimate the population of London, which is then used to estimate the number of eligible 'fighting men' available to the king
  - Dispel the myth that the plague was rifer in years of monarchy changes
  - Investigate how London was expanding

http://www.theactuary.com/archive/old-articles/part-3/who-was-captain-john-graunt-3F/ 7

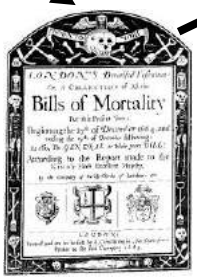
7

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

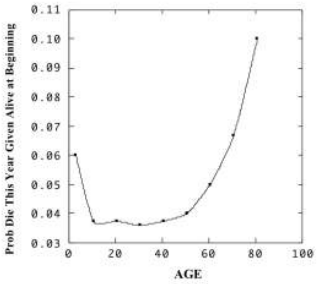
Use all of the data, or just a little? [2/5]



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

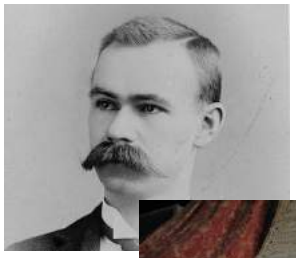
9

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

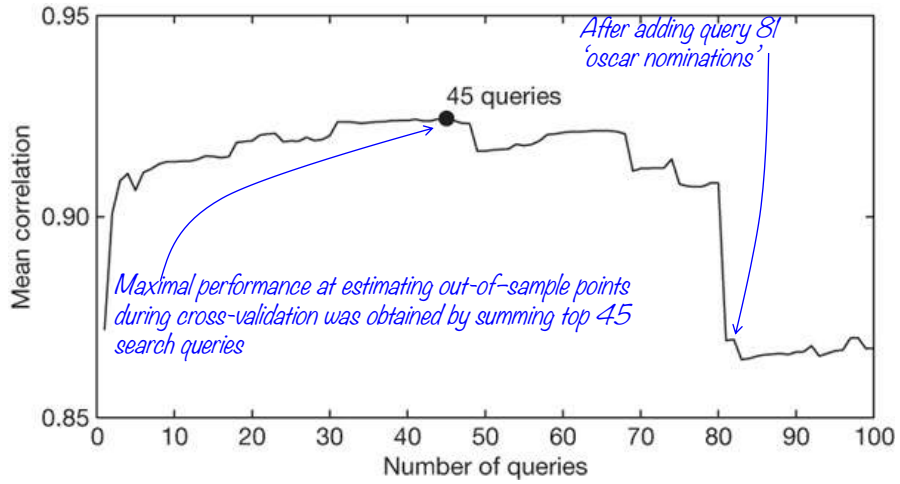
- $logit(I(t)) = \alpha logit(Q(t)) + \varepsilon$ where:
  - $I(t)$ is the percentage of ILI physician visits at time t
  - $Q(t)$ is the ILI-related query fraction at time t
  - α is the multiplicative coefficient
  - ε is the error term
  - $logit(p) = ln(p/(1-p))$

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)

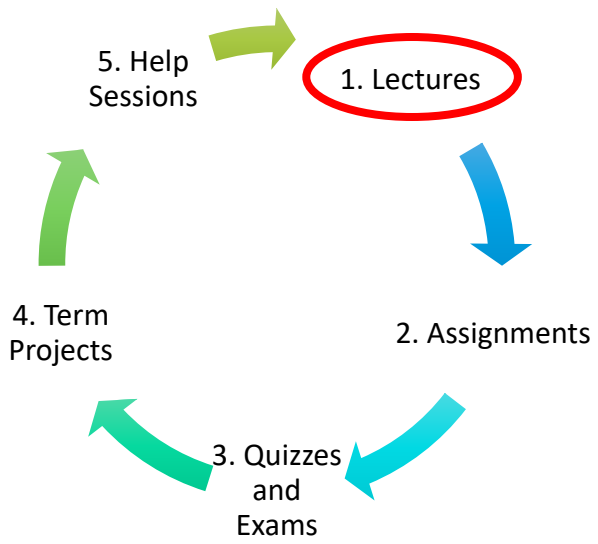
- Professor
  - sangmi@colostate.edu
  - Office hours
    - Friday 9:00AM ~ 10:00AM and by appointment
  - Office: CS374 (and via MSTeams)
  - URL: http://www.cs.colostate.edu/~sangmi

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

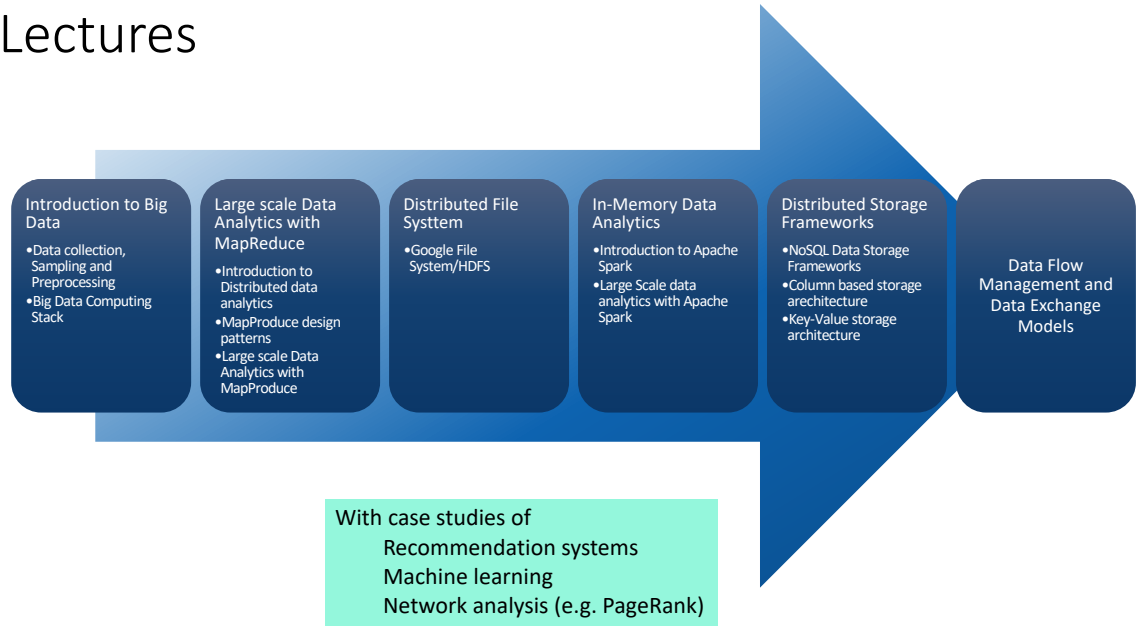


32

32

COLORADO STATE UNIVERSITY — CS435 Introduction to Big Data

Lectures

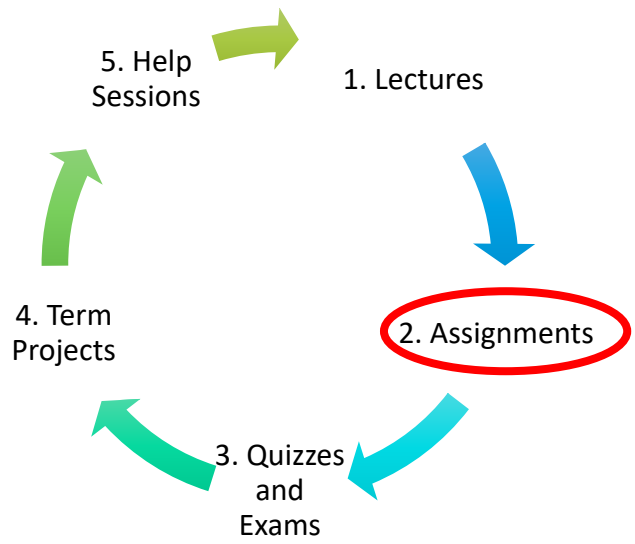


With case studies of
- Recommendation systems
- Machine learning
- Network analysis (e.g. PageRank)

33

33

Course Structure



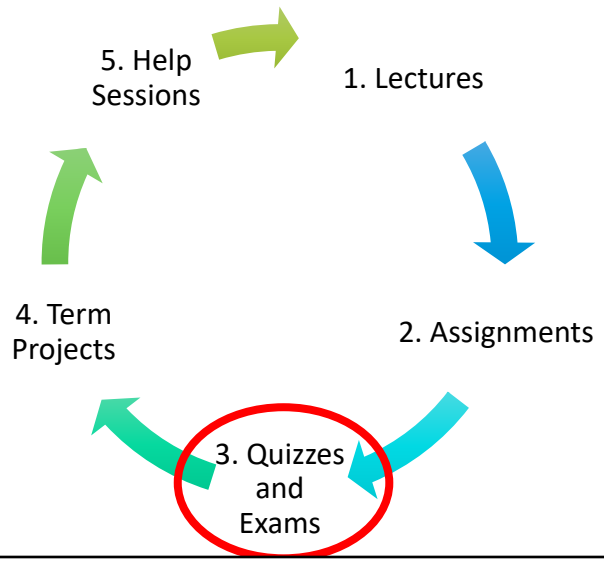
Programming Assignments (1/2)

- 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



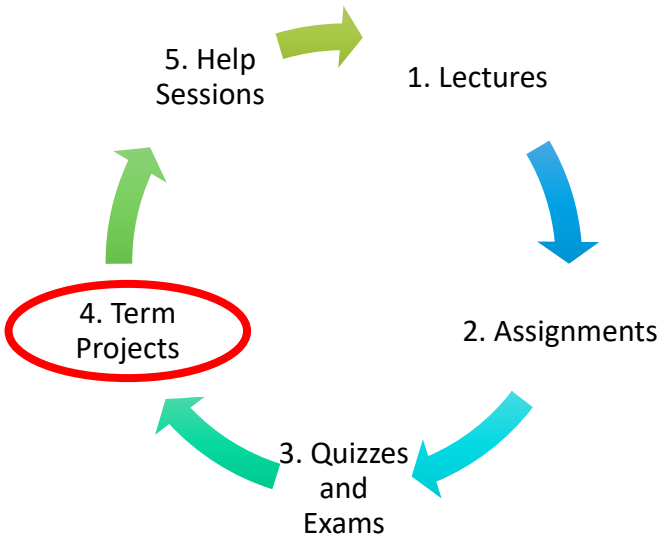
Quizzes and Exams (1/2)

- 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



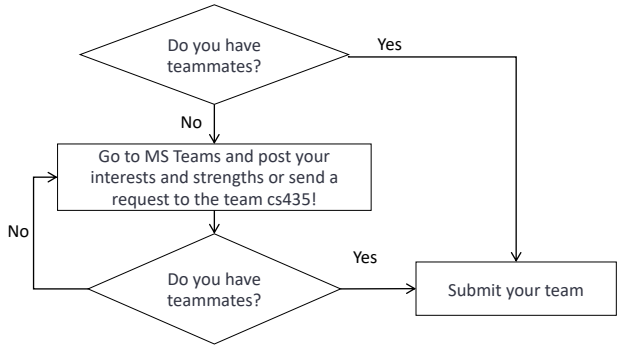
Term Project

- 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?

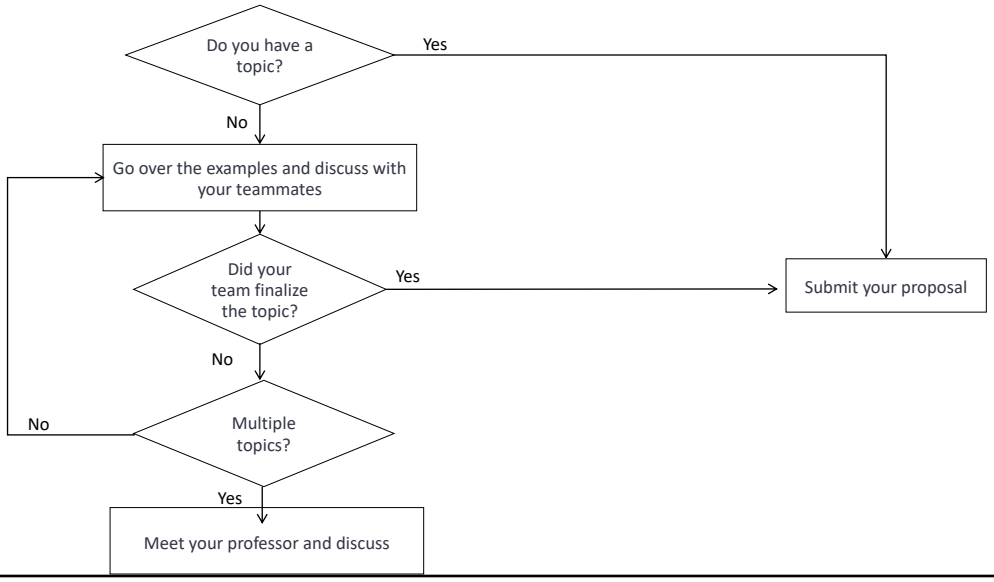


42

42

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data

How do I select my Term Project Topic?



43

43

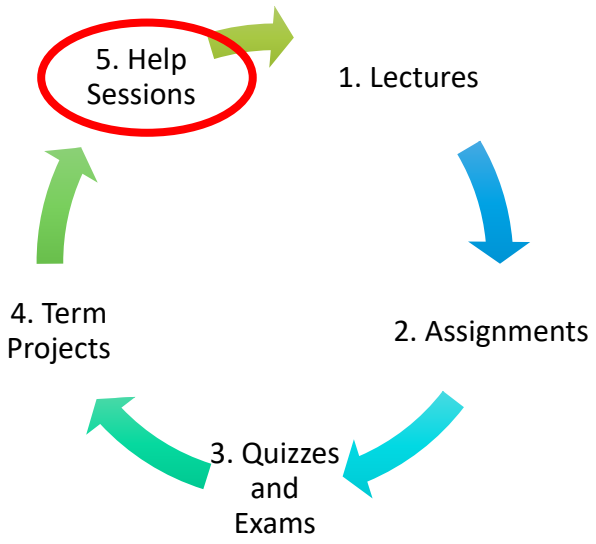
Example topics from previous courses

- 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

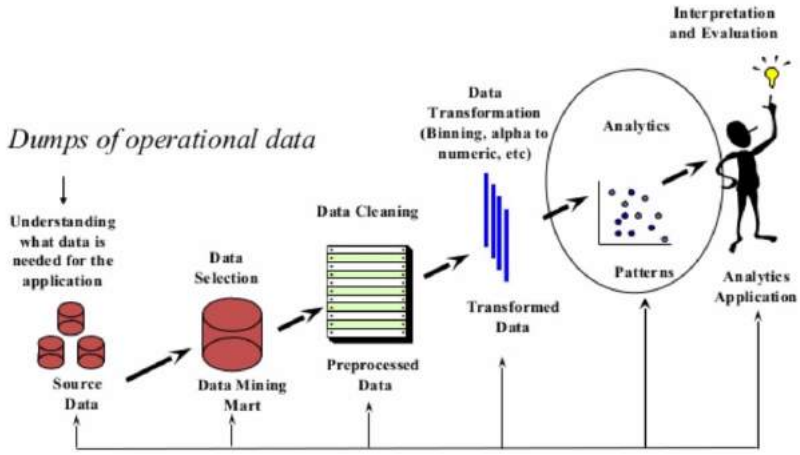
Part 1. Large Scale Data Analytics

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

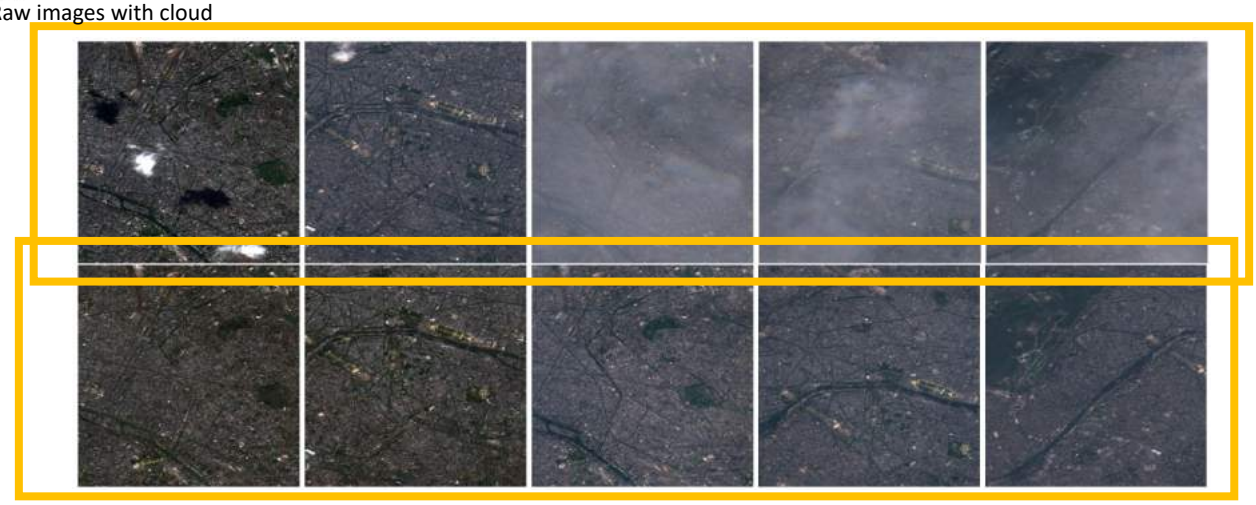
Missing Values

- **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



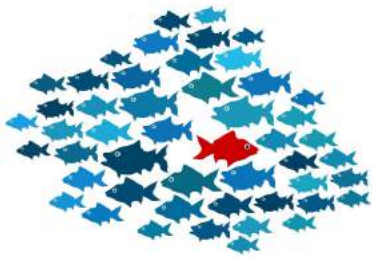
Cloud removed image using GAN

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

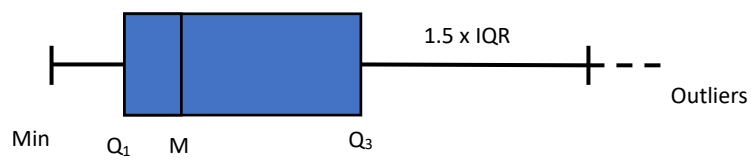
13

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

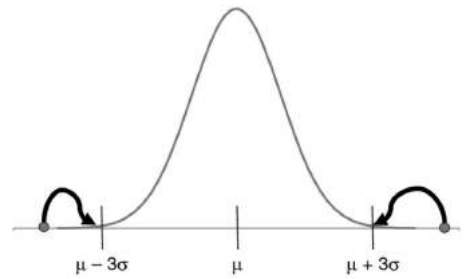
- Measuring how many standard deviations an observation is away from the mean
  - $z_i = \frac{x_i - \mu}{\sigma}$
    where μ represents the average of the variable and σ its standard deviation

- 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

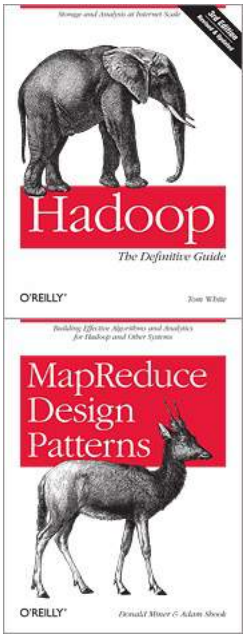
Operations and Scheduling Layer
Apache Ambari
Apache Oozie
Apache Zookeeper

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.
```

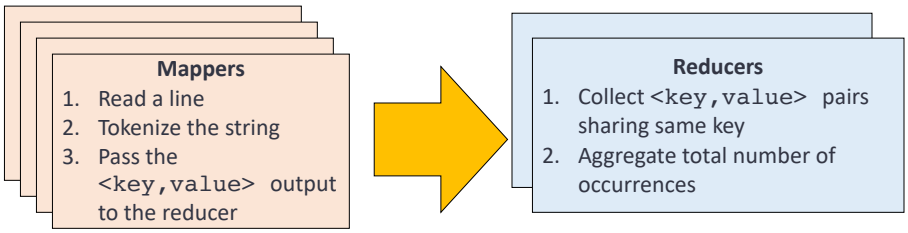

Example 1: WordCount [2/5]

- 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

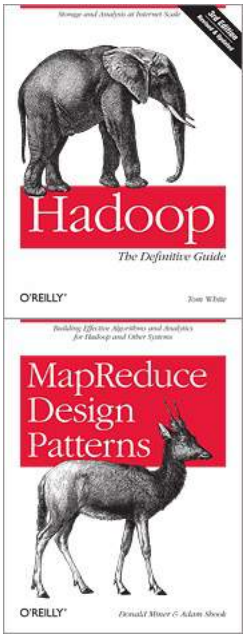
- Topic 1: Introduction to MapReduce: example 2
- Topic 2: MapReduce Design Pattern I. Summarization Patterns

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:

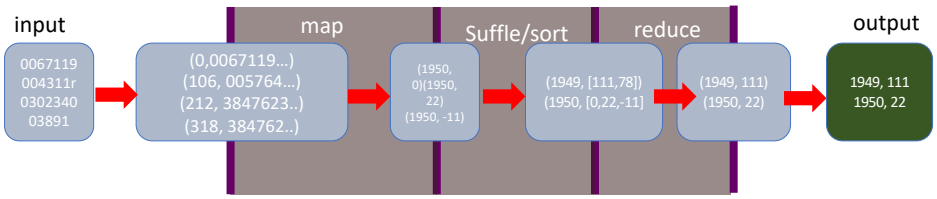
```
(1949, [111, 78])
(1950, [0, 22, –11])
```


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

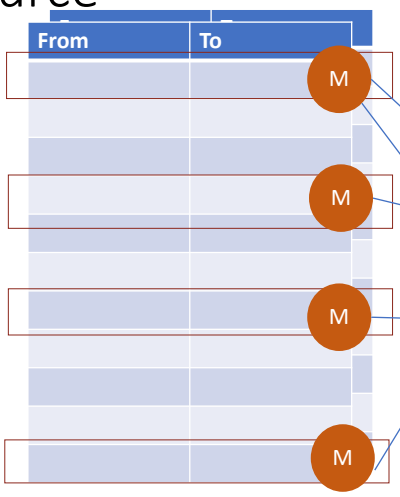
- Check each record and provide as output only those records containing “220,4,5,20” as a source IP

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

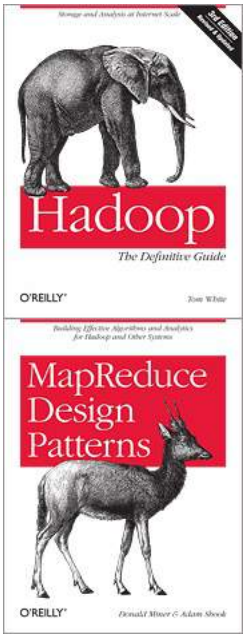
1. What should be the input for your Mapper?
2. What should be the output from your Mapper?
3. What should be the input for your Reducer?
4. What should be the output from your Reducer?

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?

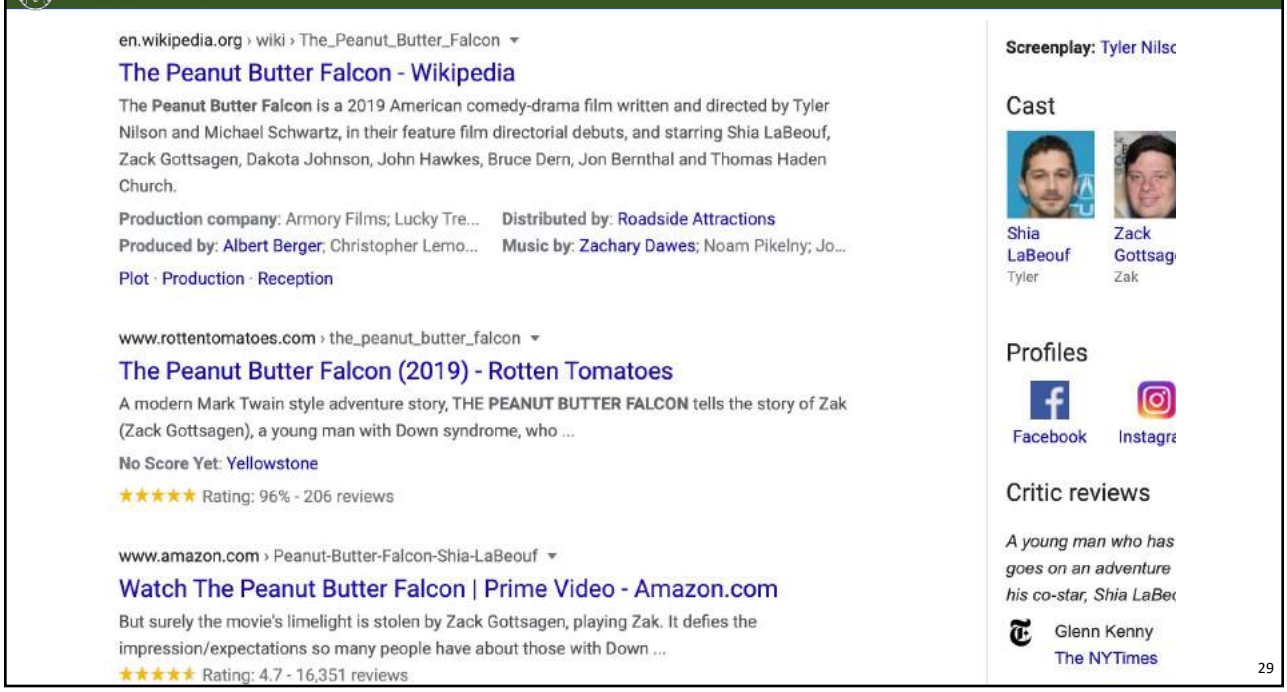
- Tools for solving problems
  - Reusable and providing a general framework
  - Developers can spend less time figuring out how she/he is going to solve the problem

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**



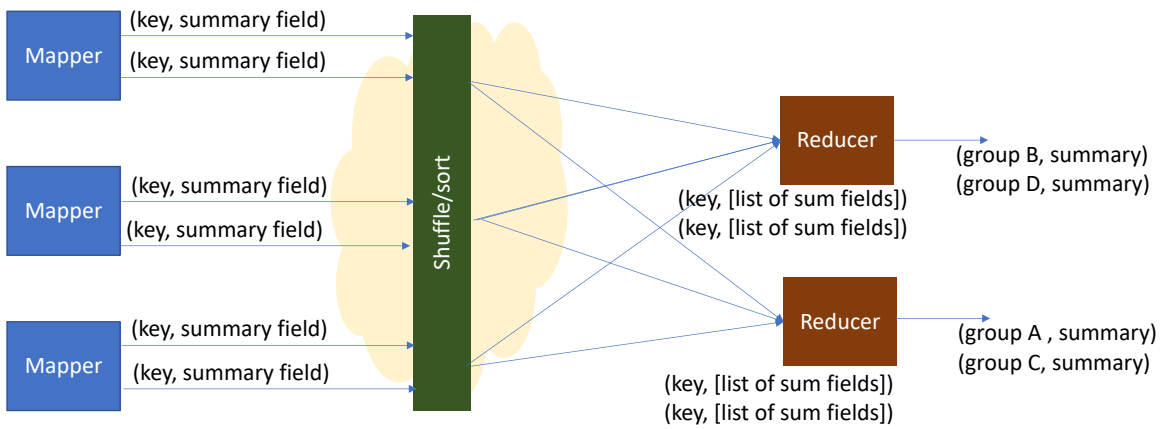
Design Pattern 1: Summarization Patterns

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**

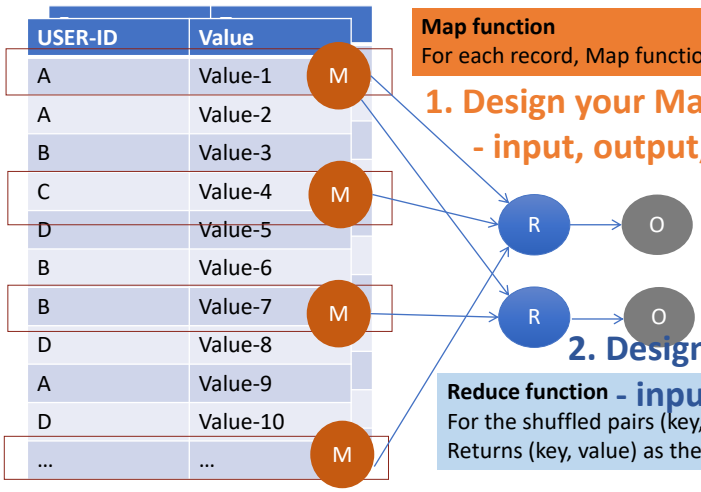
Structure



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

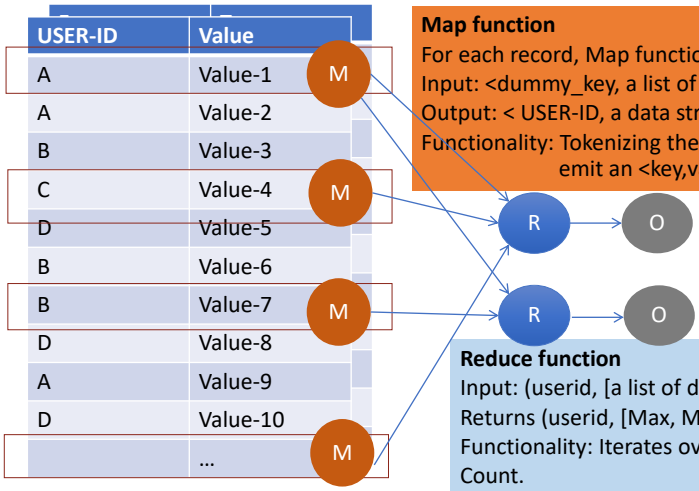
2. Design your Reduce function

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;
```

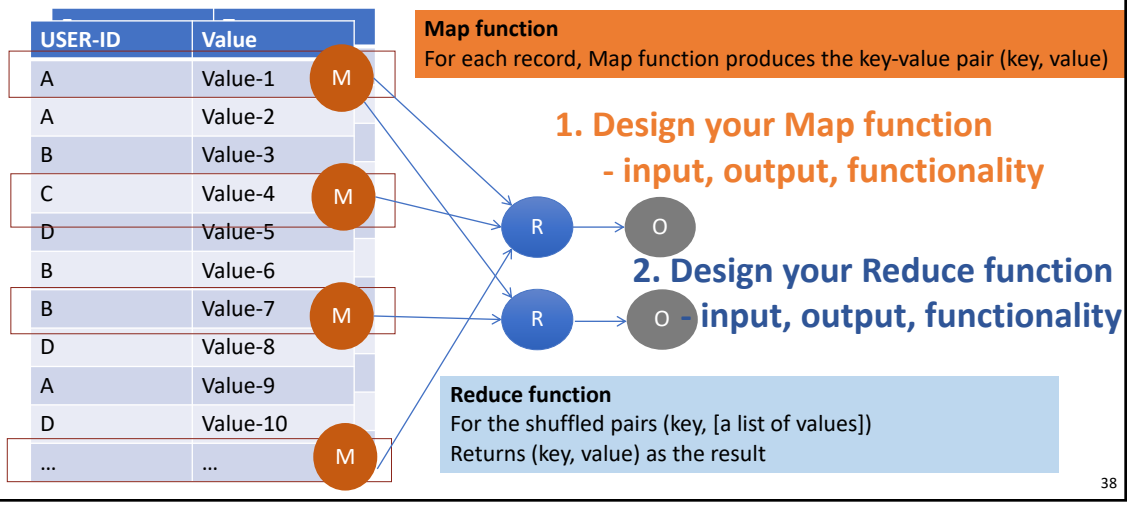

Reducer Code --continued

```
        // 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

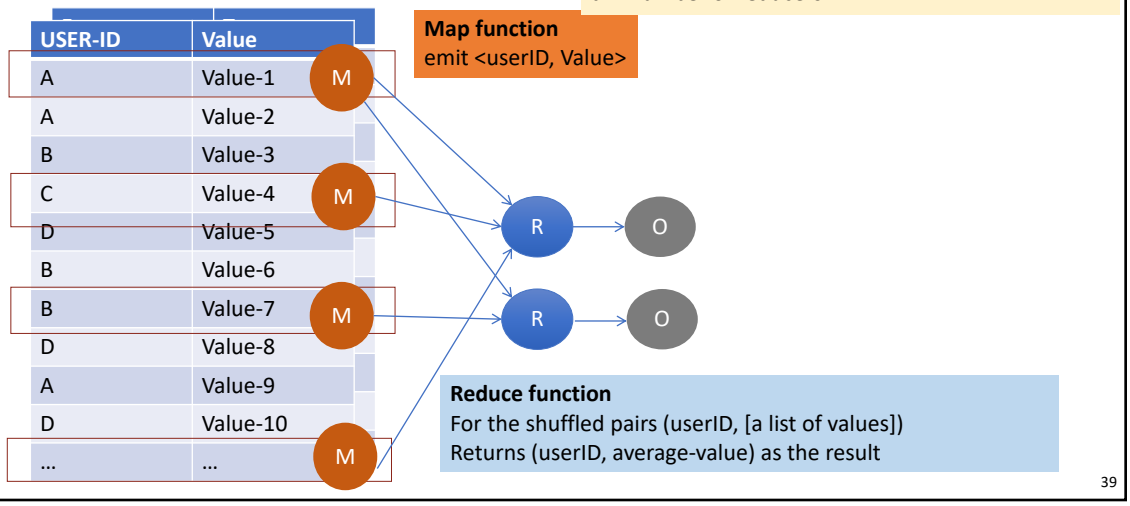
38

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)

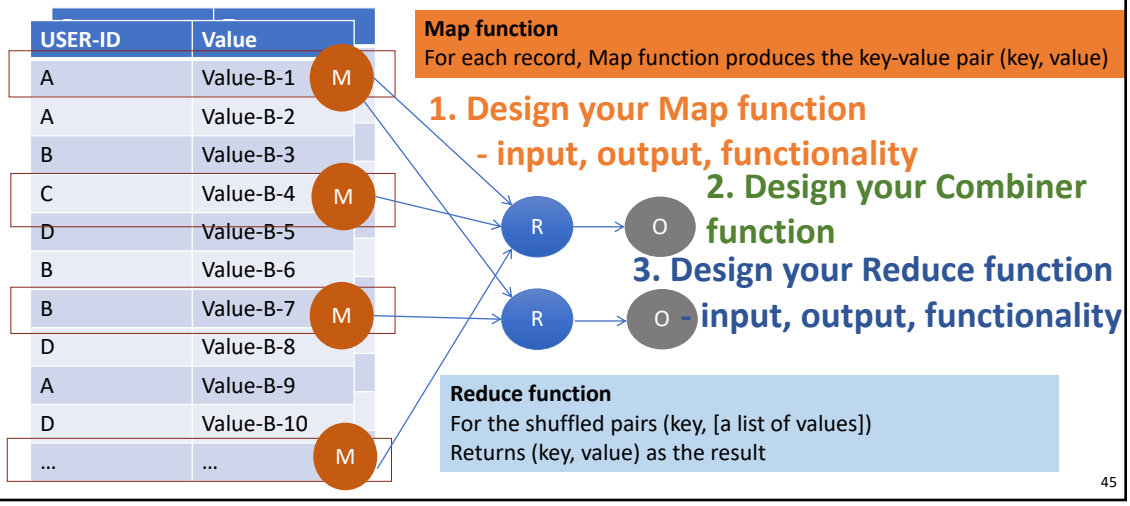
Question

- Can we use a combiner function for finding average value?
  - Yes
  - No

Numerical Summarization Example 2. Average

- Calculating the Average of ***Values*** per user

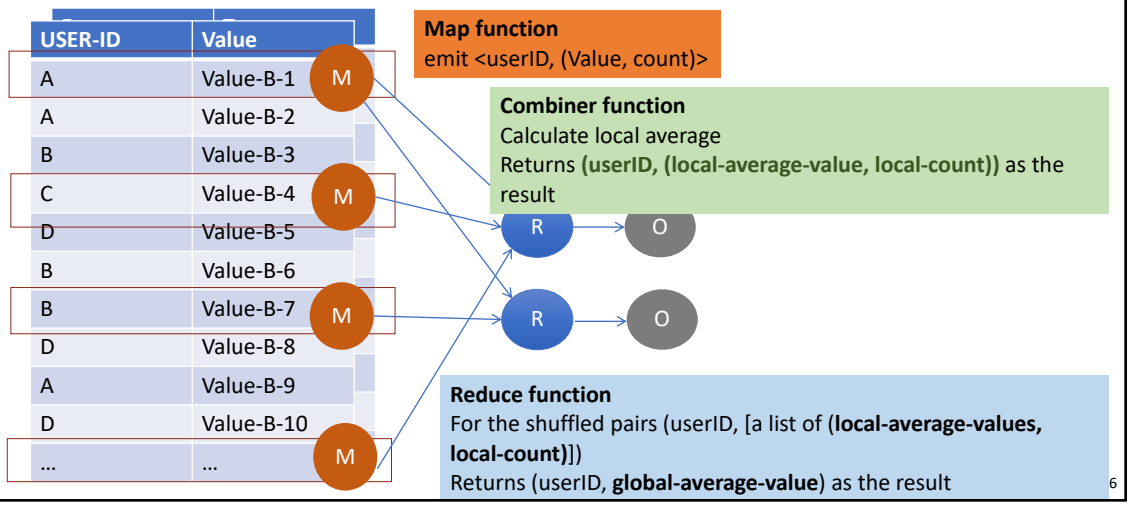
USER-ID	Value
A	Value-B-1
A	Value-B-2
B	Value-B-3
C	Value-B-4
D	Value-B-5
B	Value-B-6
B	Value-B-7
D	Value-B-8
A	Value-B-9
D	Value-B-10
…	…



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

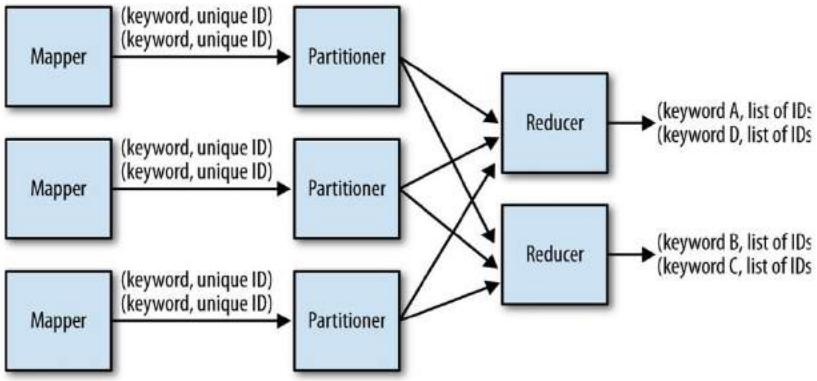
Inverted Index

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

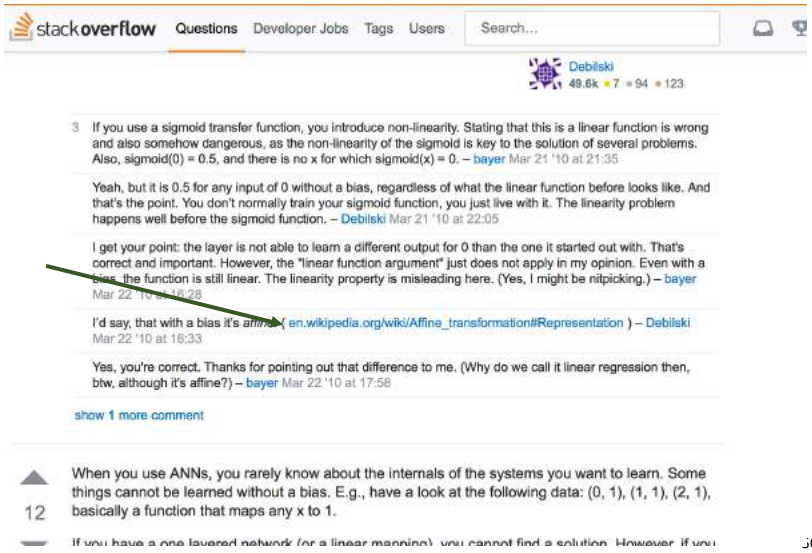
*** Example of an inverted index for document 1 and 2 (D1, and D2)**
D1= "Colorado State University"
{Colorado, State, University}
D2= "University of Colorado"
{University, of, Colorado}
Inverted Index
Colorado – {D1, D2}
State - {D1}
University – {D1, D2}
of – {D2}

Structure



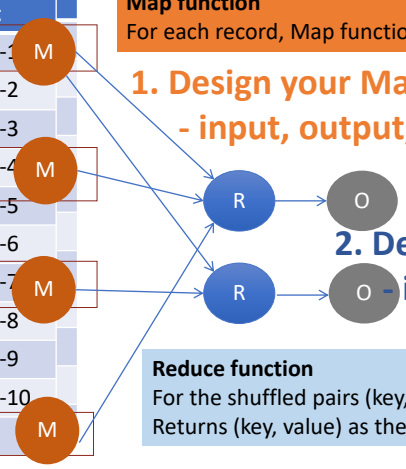
Inverted index of StackOverflow links to Wikipedia

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)

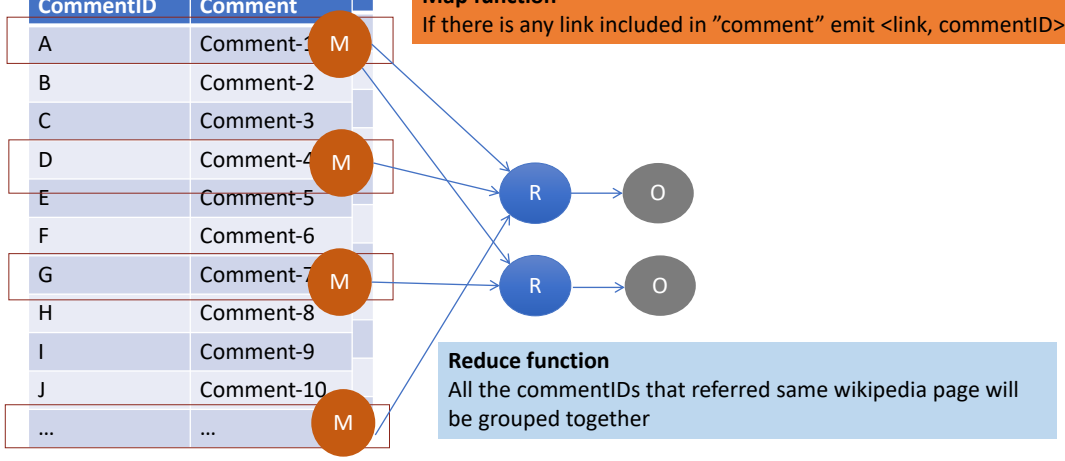
1. Design your Map function
- input, output, functionality

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

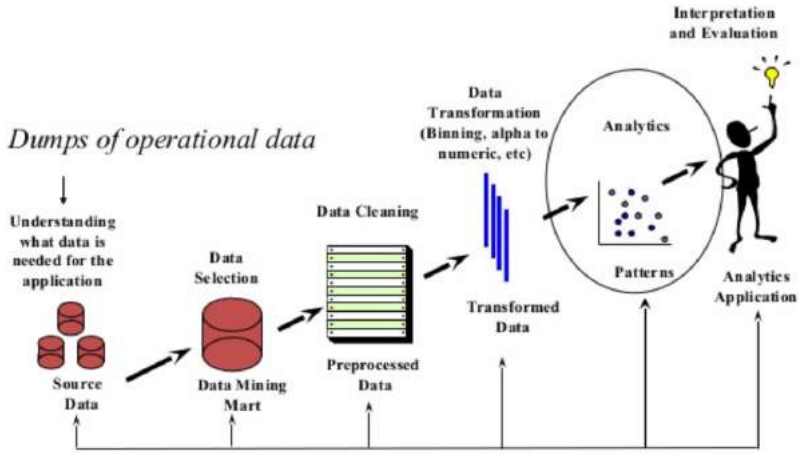
Part 1. Large Scale Data Analytics

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

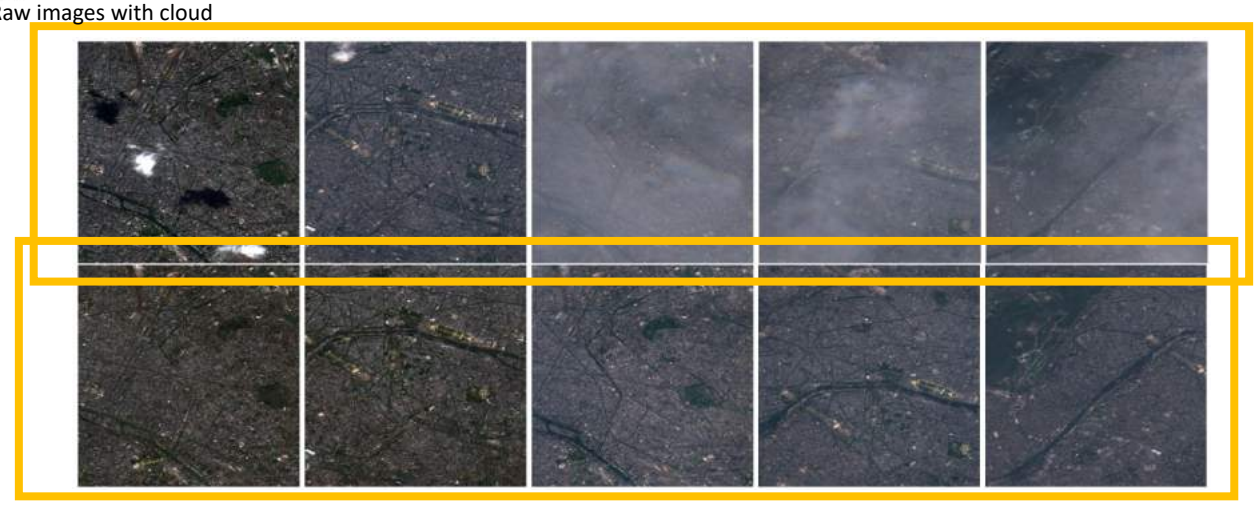
Missing Values

- **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



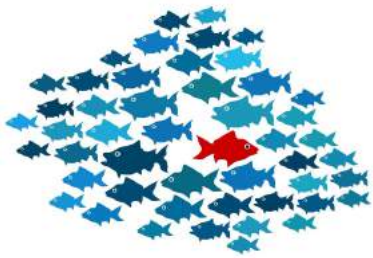
Cloud removed image using GAN

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

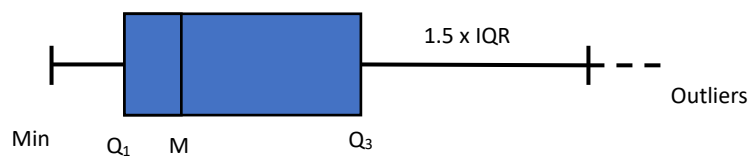
69

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

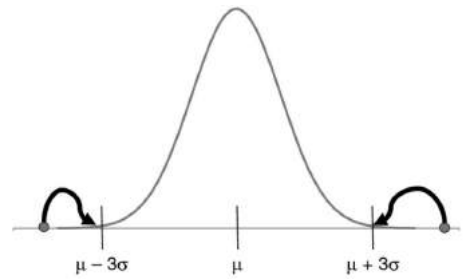
- Measuring how many standard deviations an observation is away from the mean
  - $z_i = \frac{x_i - \mu}{\sigma}$
    where μ represents the average of the variable and σ its standard deviation

- 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

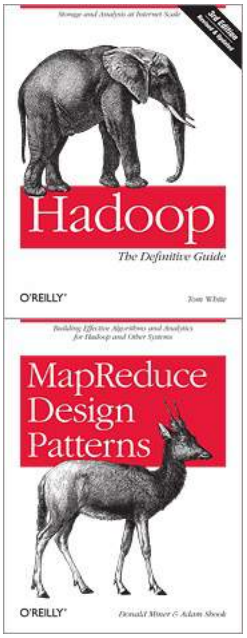
Operations and Scheduling Layer
Apache Ambari
Apache Oozie
Apache Zookeeper

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.
```

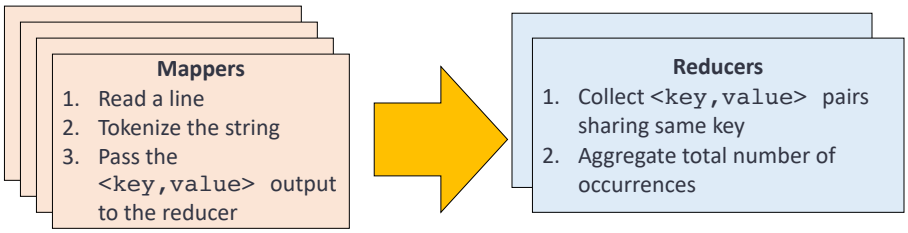

Example 1: WordCount [2/5]

- 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

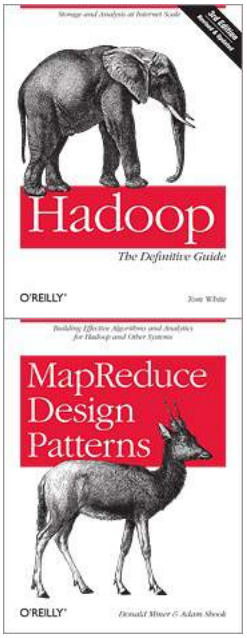
- Topic 1: MapReduce Design Pattern I. Summarization Patterns
- Topic 2: MapReduce Design Pattern II. Filter Patterns

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

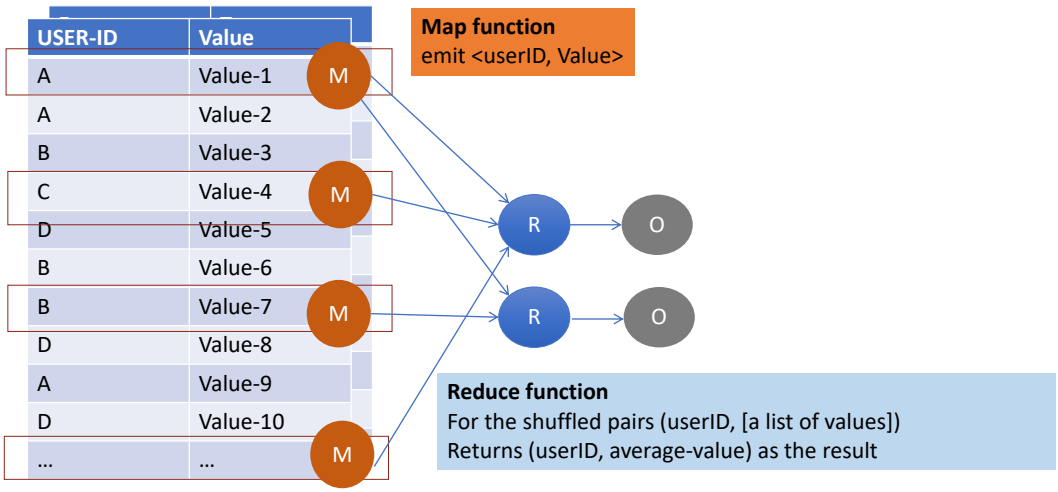
1. Numerical Summarizations

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)

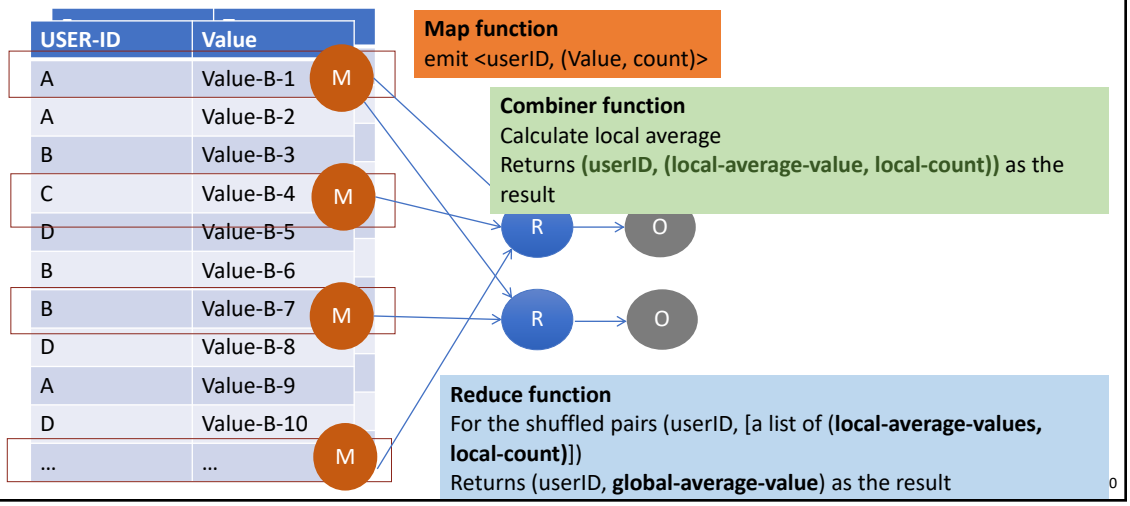
With Combiner

- 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

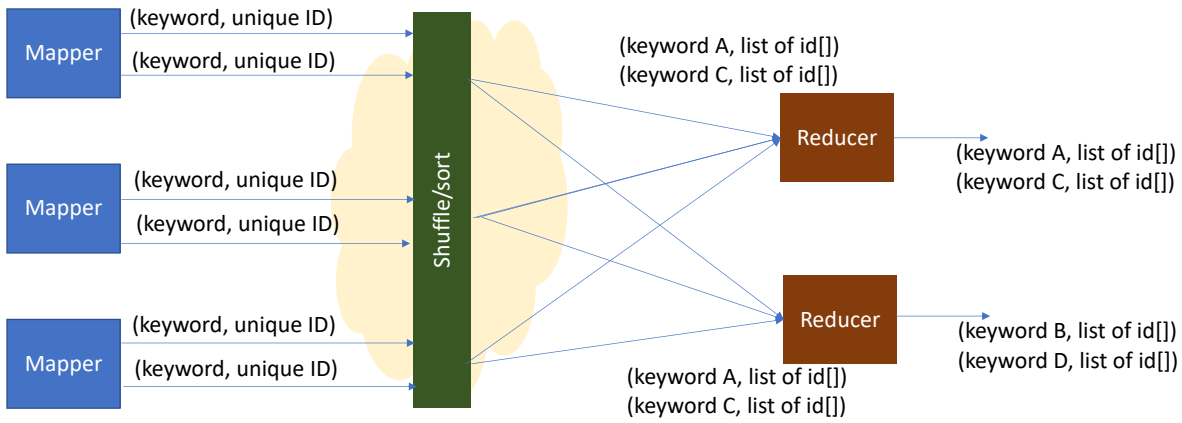
Inverted Index

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

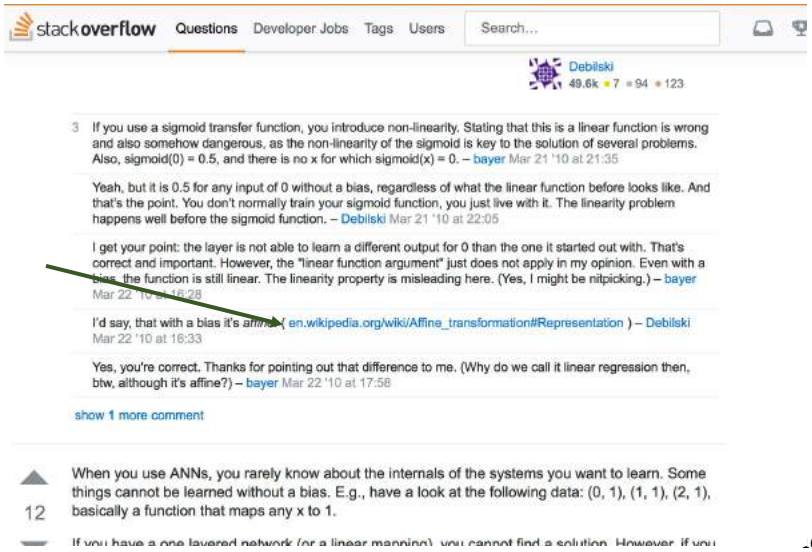
*** Example of an inverted index for document 1 and 2 (D1, and D2)**
D1= "Colorado State University"
{Colorado, State, University}
D2= "University of Colorado"
{University, of, Colorado}
Inverted Index
Colorado – {D1, D2}
State - {D1}
University – {D1, D2}
of – {D2}

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

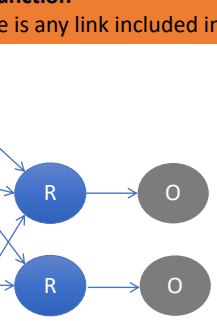
Reduce function
For the shuffled pairs (key, [a list of values])
Returns (key, value) as the result

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

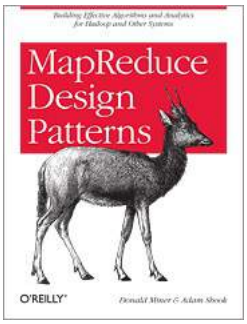
- **Advantages of using Map-Only job**
  - Between map and reduce
    - Sort and Shuffle
      - Expensive computation
  - The Map-only jobs can avoid sort and shuffle
  - Output of mapper is written to local disk
    - Your mapper can write output to the HPDF files directly

Part 1. Large Scale Data Analytics

Design Pattern 2: Filtering Patterns

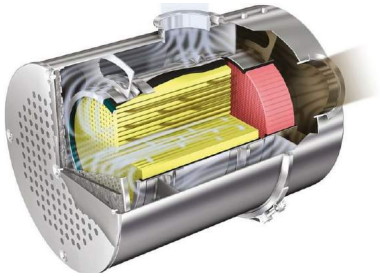
This material is built based on,

- 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

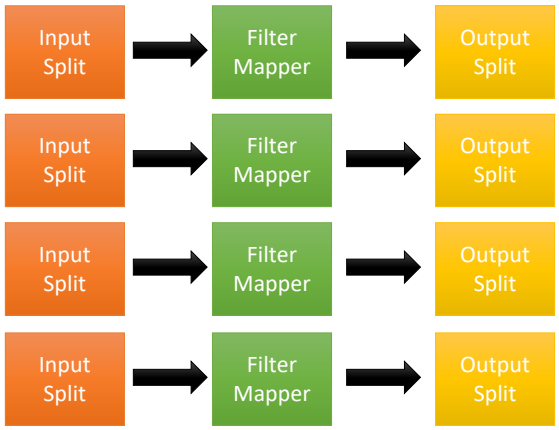
1. Simple Random Sampling

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.

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*

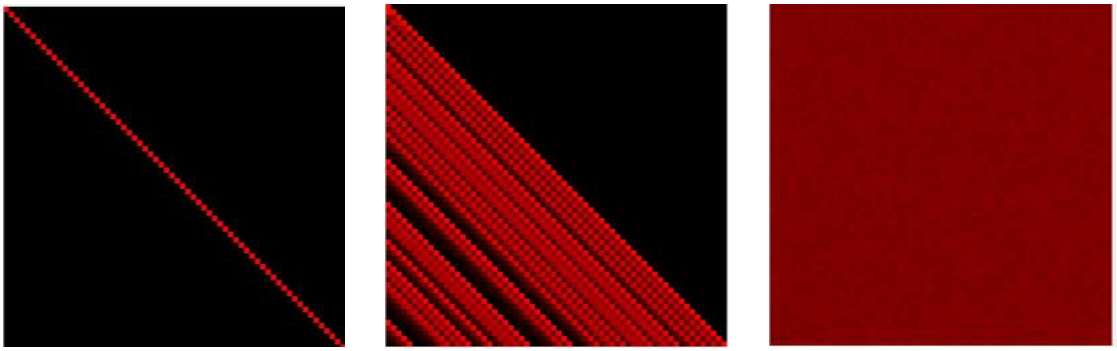
After encoding 5, 10 and 15	1	1	1	0	0	1	1	0
Check h1(7) = 7	1	1	1	0	0	1	1	0
Check h2(7) = 1	1	1	1	0	0	1	1	0
Check h3(7) = 7	1	1	1	0	0	1	1	0

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

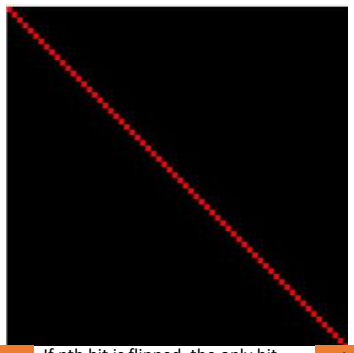
Evaluating Hash Functions: Avalanche Effect

- 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

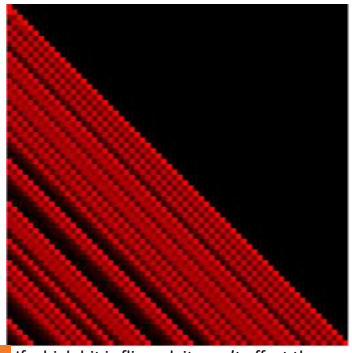


Avalanche Effect

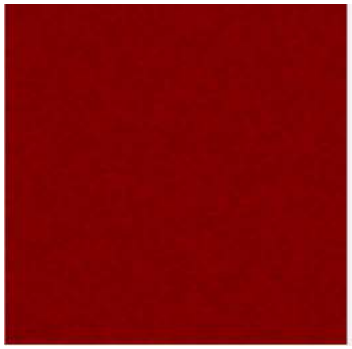
- Each bit should have **50% chances** to change if you **change 1 bit** of the input
- Avalanche Diagram
  - A grid such that the (x,y) cell's color represents the probability that flipping x'th bit of input will result of yth bit being flipped in the output



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**

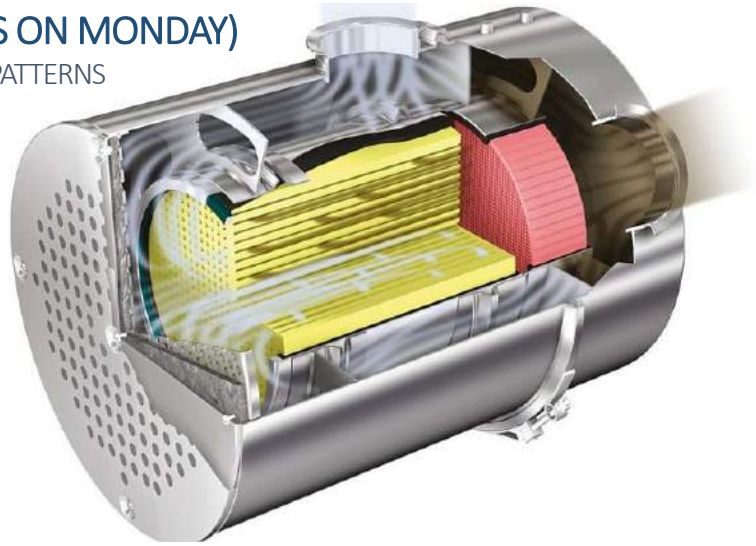
How big should I make my Bloom Filter?

- 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

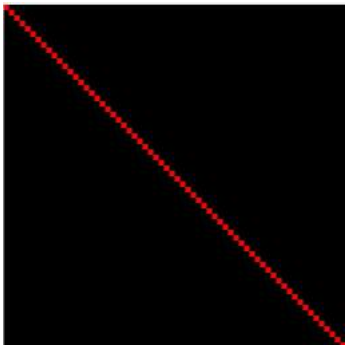
- Avalanche diagram
- Term project

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)

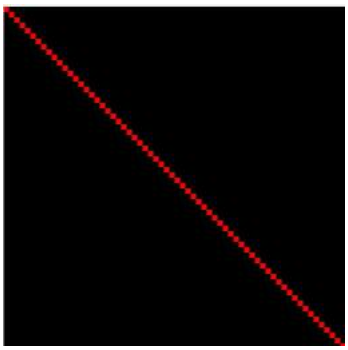


n^{th} bit

0th bit

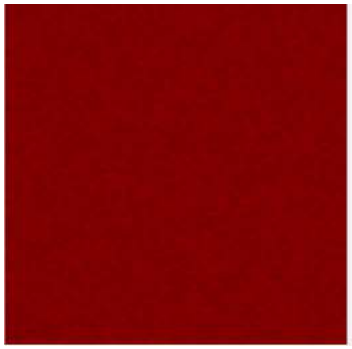
Evaluating Hash Functions: Avalanche Effect

- 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

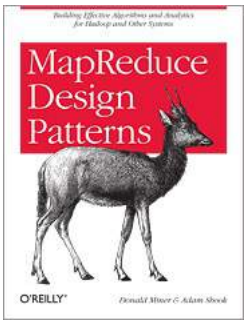
- **Your participation within your team will be assessed by your team members**
- Participation
  1. Not active for the task
  2. Actively participated
  3. Significant contribution
  4. Taking initiative
- Task implementation
  - Achieving the individual goals successfully within the time frame that 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

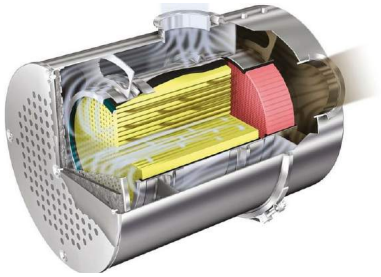
This material is built based on,

- 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

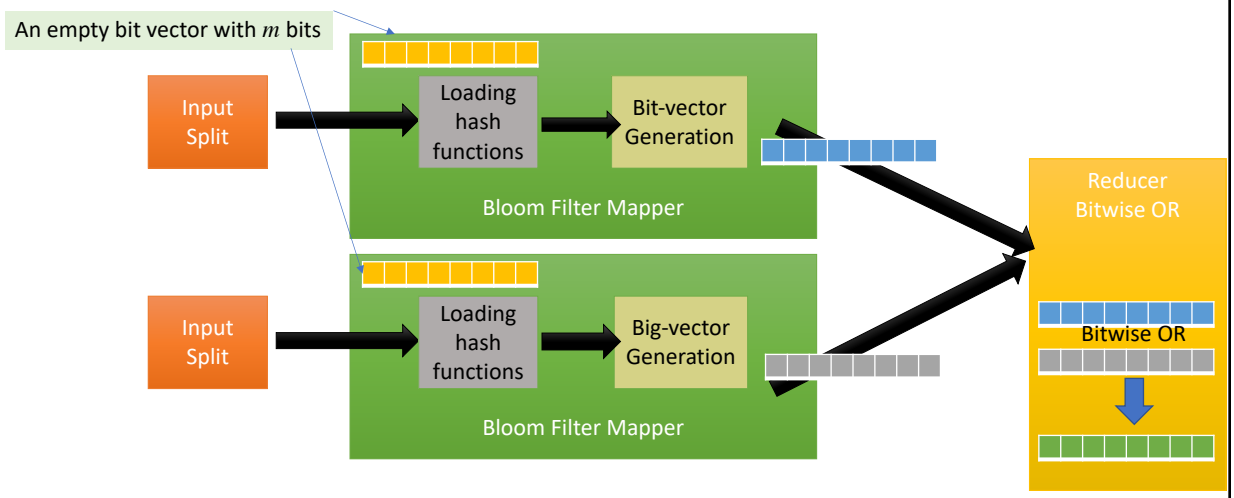
Applying a Bloom filter

- Is 8 part of set *T*?
- *h1(8), h2(8), h3(8)*
- *8 is **NOT** a part of set T*

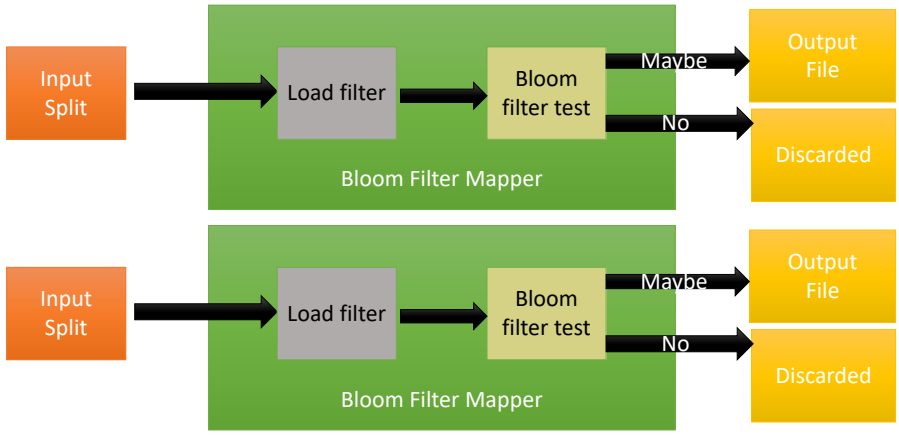
h1(x) = 3x mod 8
h2(x) = (2x +3) mod 8
h3(x) = x mod 8

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`

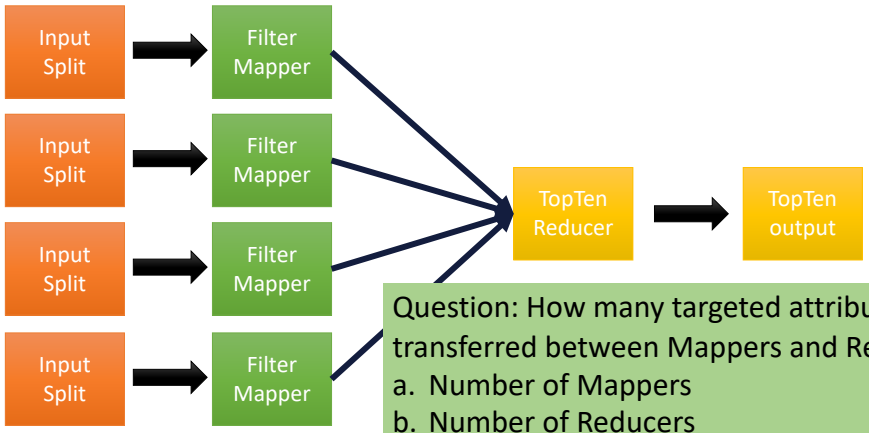
- Retrieves a **relatively small number (top K)** of records, according to a ranking scheme in your dataset, no matter how large the data

- 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?



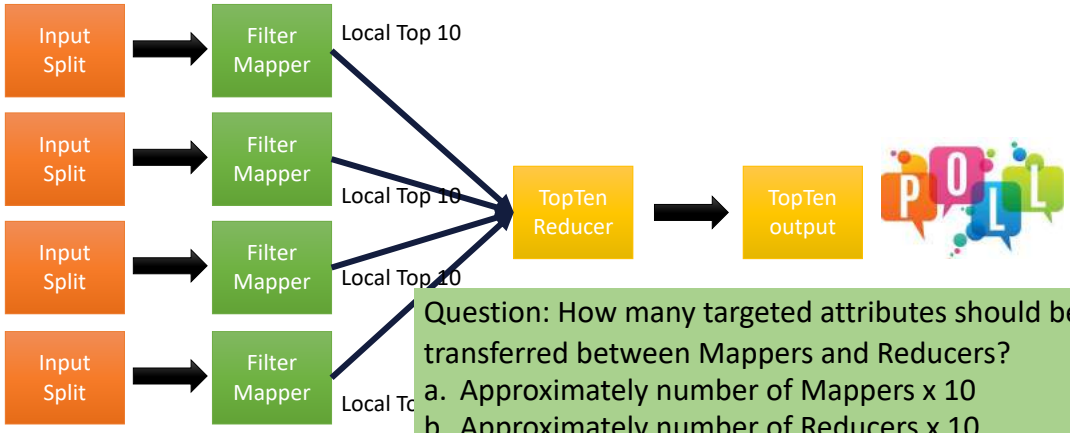
Question: How many targeted attributes should be transferred between Mappers and Reducers?

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?

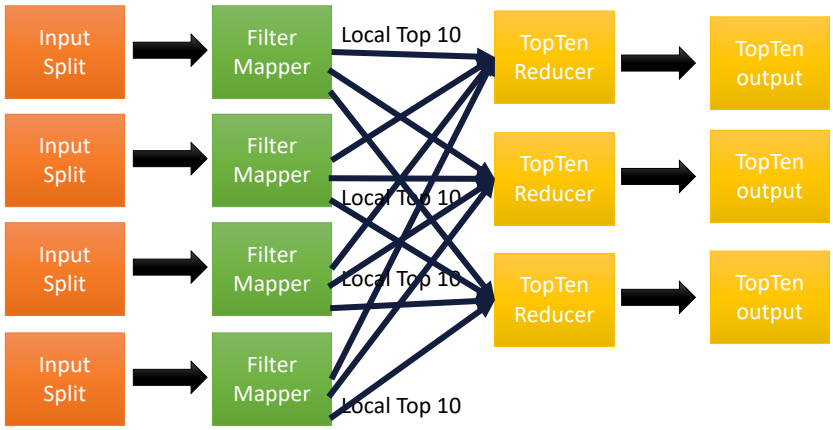


Question: How many targeted attributes should be transferred between Mappers and Reducers?

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

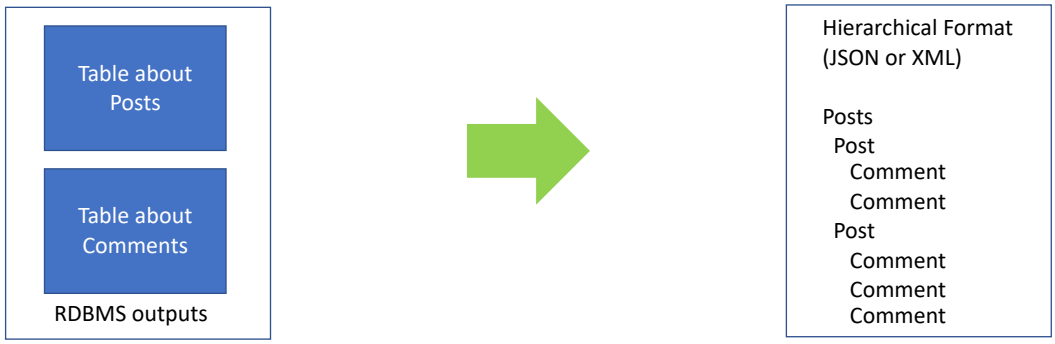
1. "Structured" to "hierarchical" pattern
2. Partitioning and binning patterns
3. Total order sorting

MapReduce Design Patterns III: Data Organization Patterns

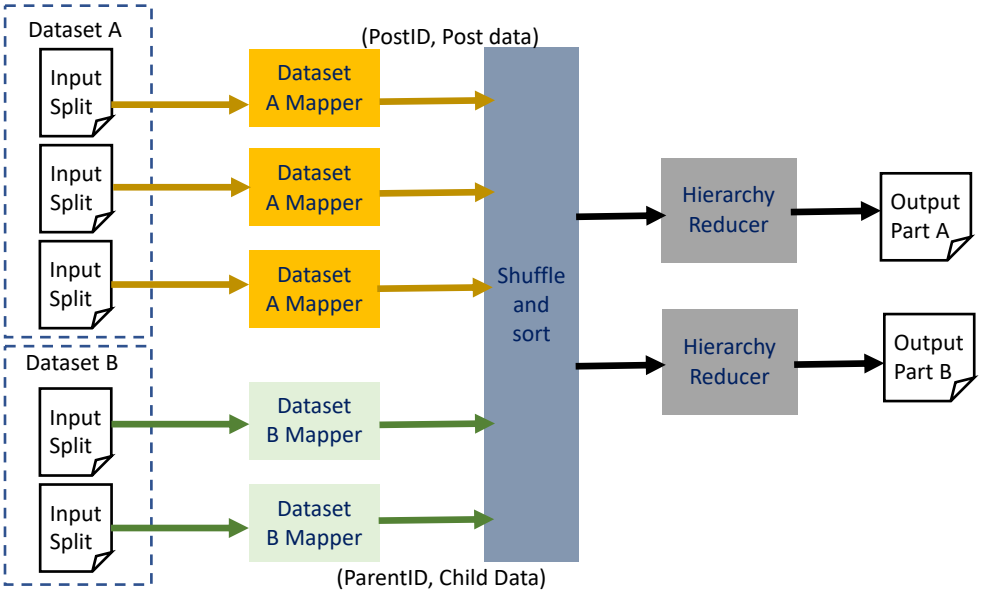
1. "Structured" to "hierarchical" pattern

"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());
      }
    } ...
```

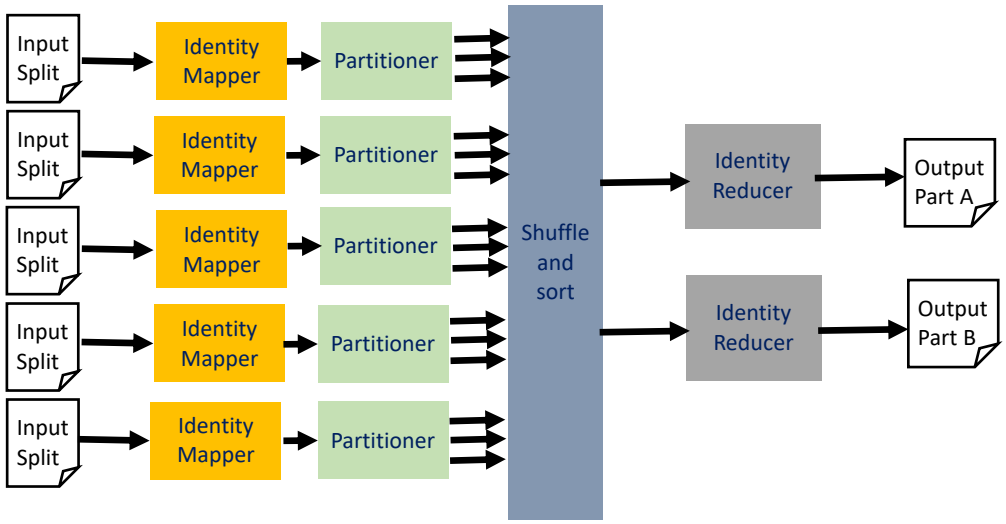
MapReduce Design Patterns III: Data Organization Patterns

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**

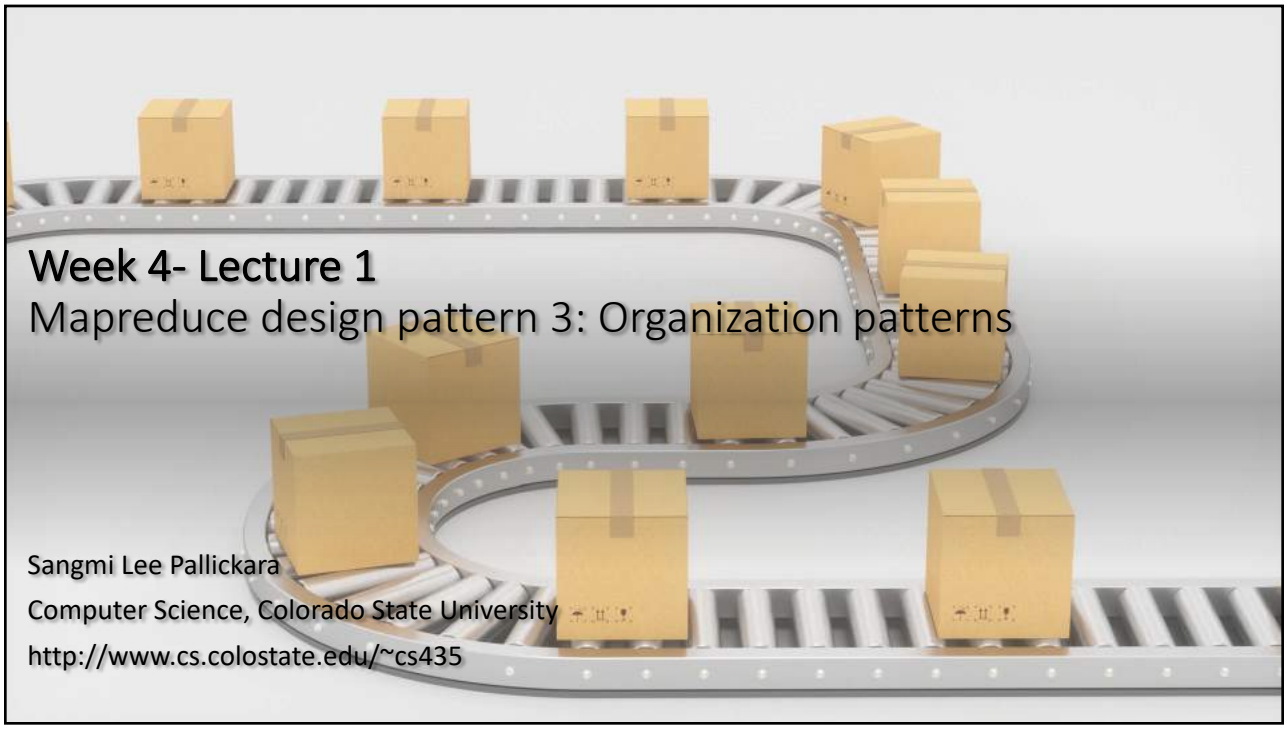
TeraSort Benchmark

- 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

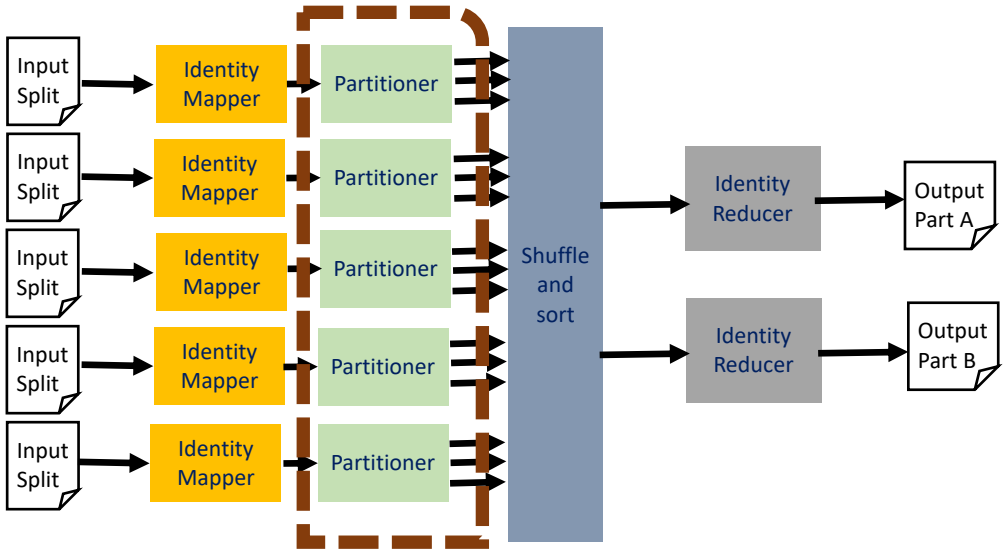
MapReduce Design Patterns III: Data Organization Patterns

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**

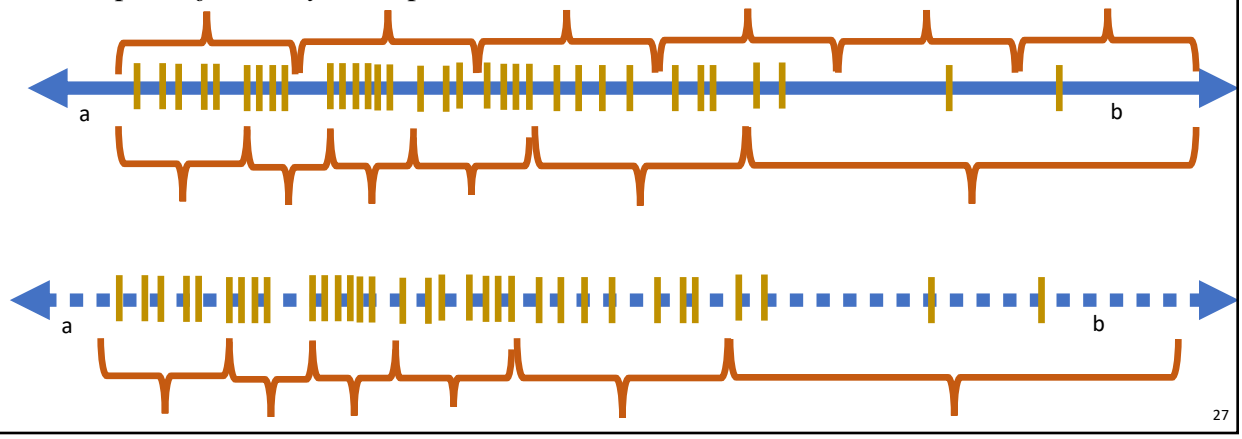
Structure of Total Order Sorting Pattern

- 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*

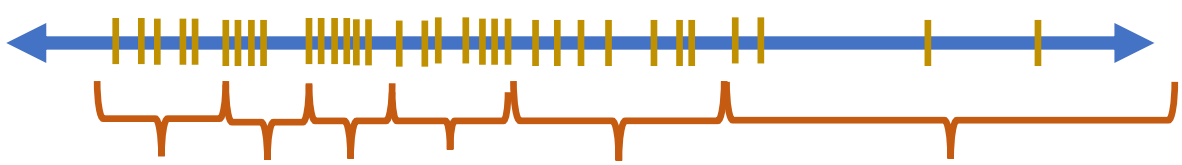


Structure of the Total Order Sorting Pattern
- Analysis phase

- 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

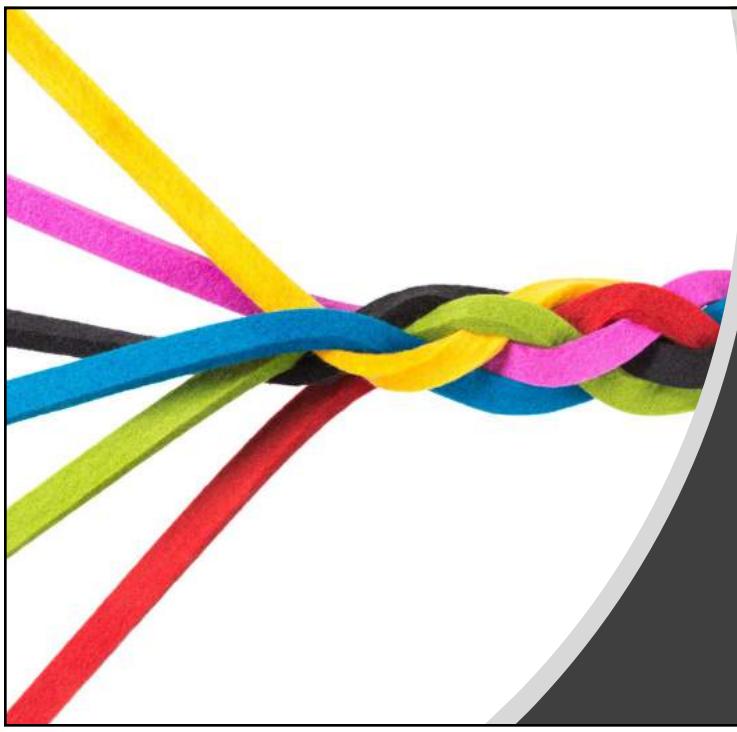
TeraSort Benchmark APIs

- 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

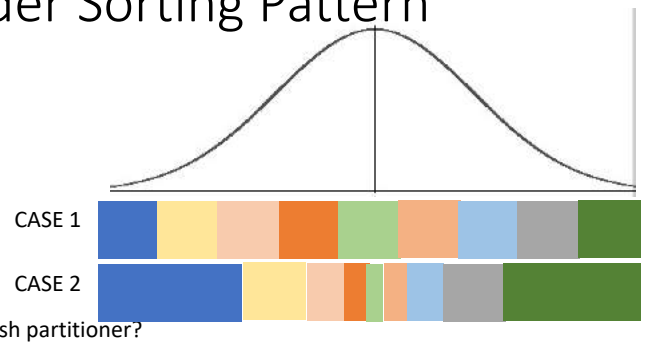
FAQs

- 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

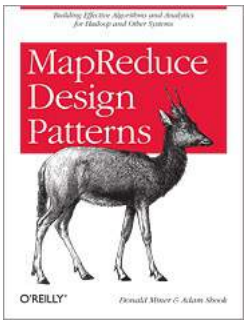
- MapReduce Design Pattern III: Organization Patterns
  - Partitioning
  - Total order sorting (TeraSort)

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

COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 4-B Fall 2021Computer Science

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

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

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

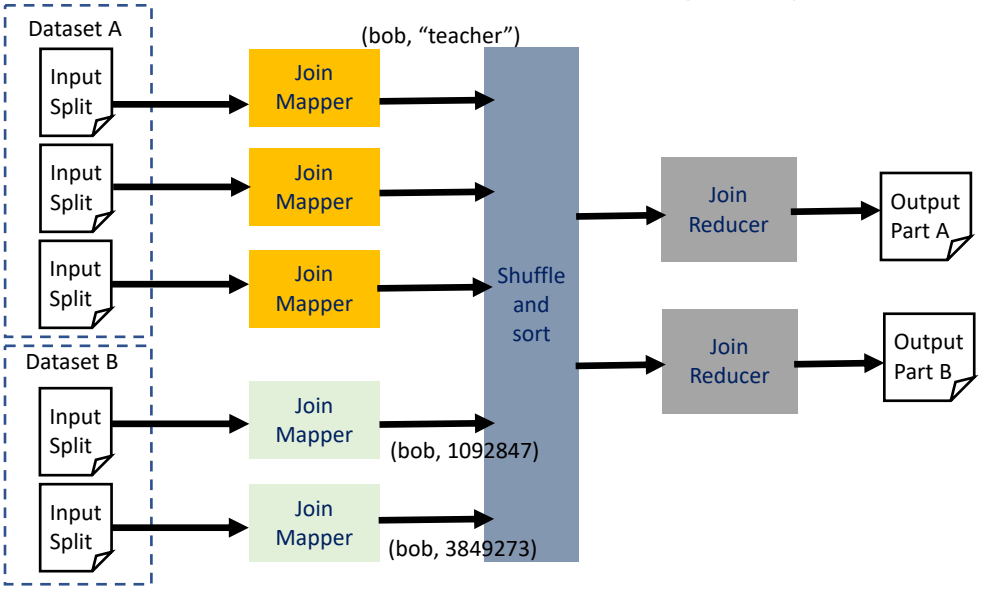
- Worksheet question 2:

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`

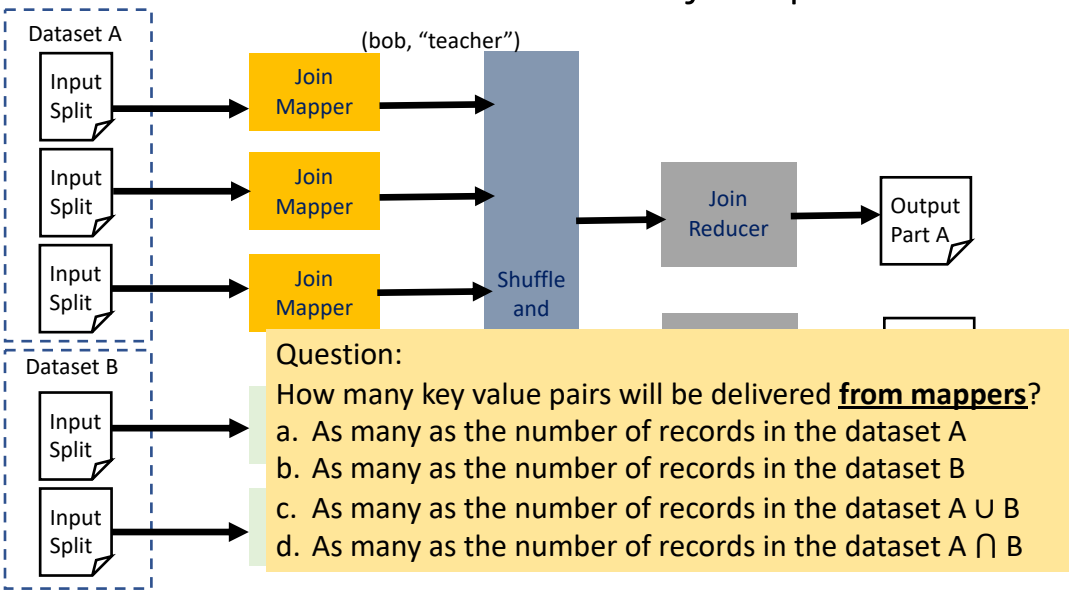
Table B contains the following attributes: `student_ID, address, phone_number, DOB`

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);
       }
    }
 } ...
```

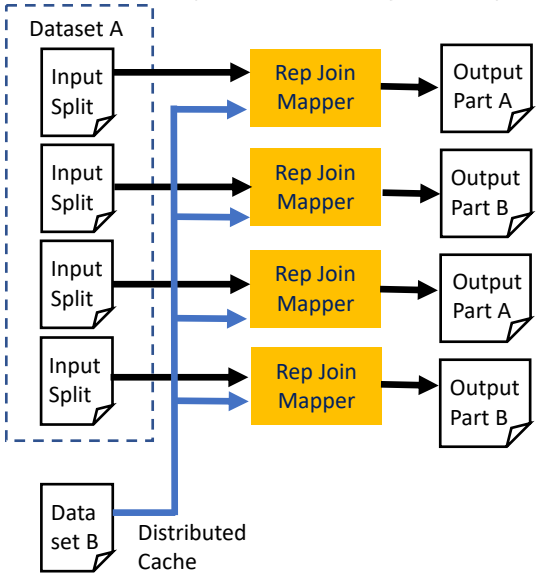

MapReduce Design Patterns IV: Join Patterns

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);
  }
 }
}
```

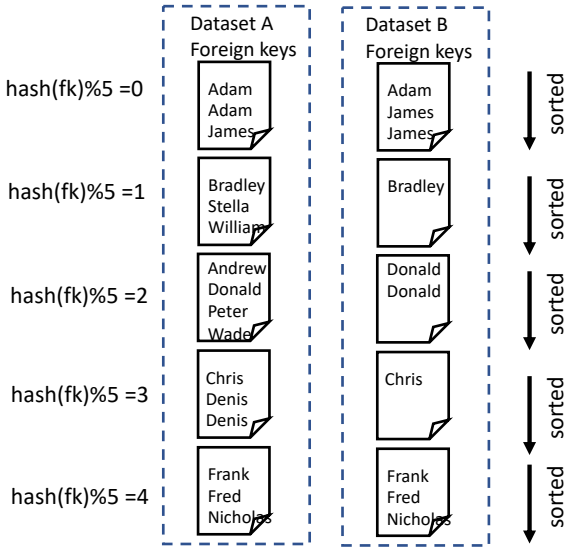

MapReduce Design Patterns IV: Join Patterns

3. Composite Join

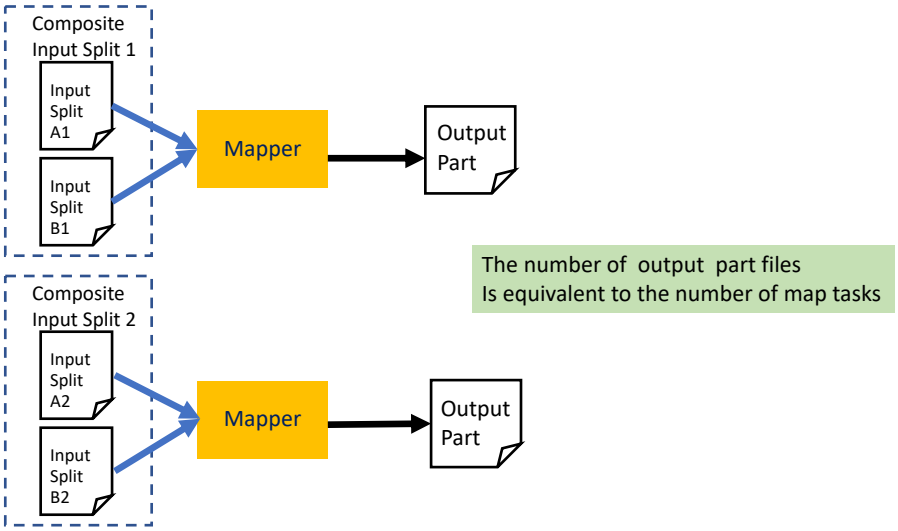
Composite Join

- 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.

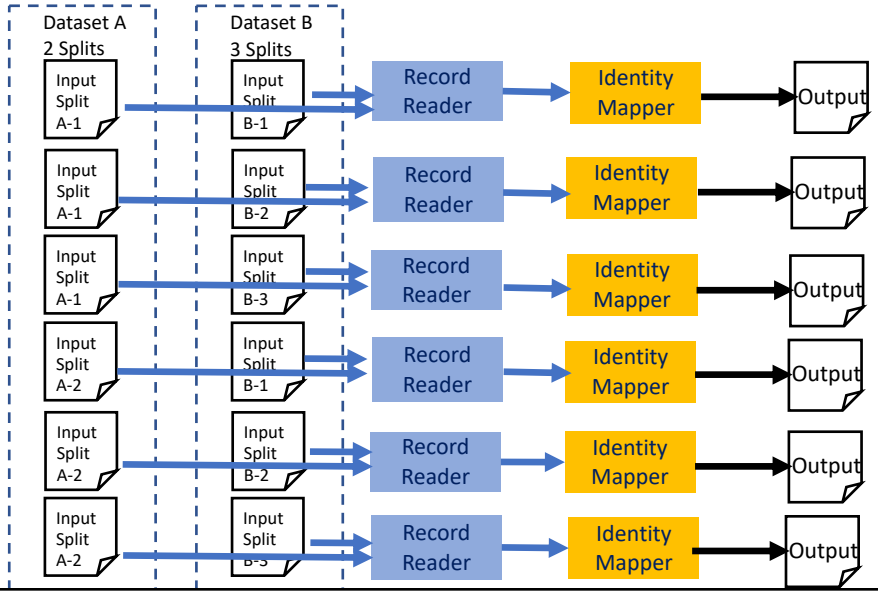
MapReduce Design Patterns IV: Join Patterns

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

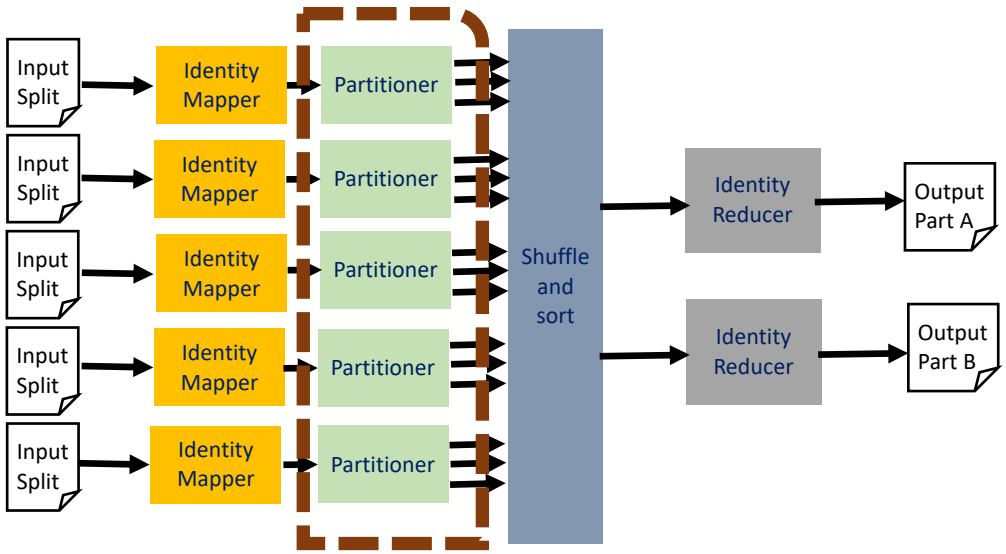
MapReduce Design Patterns III: Data Organization Patterns

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**

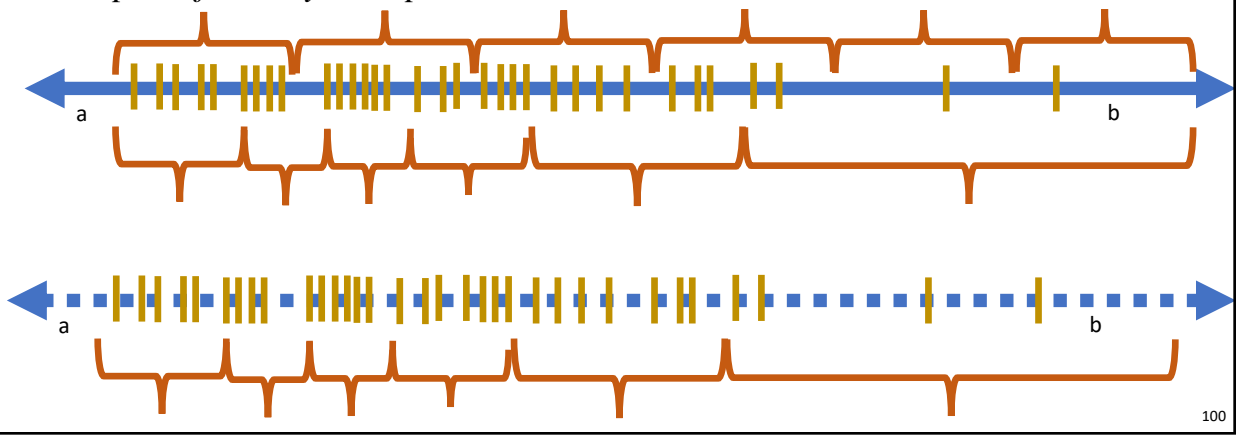
Structure of Total Order Sorting Pattern

- 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*

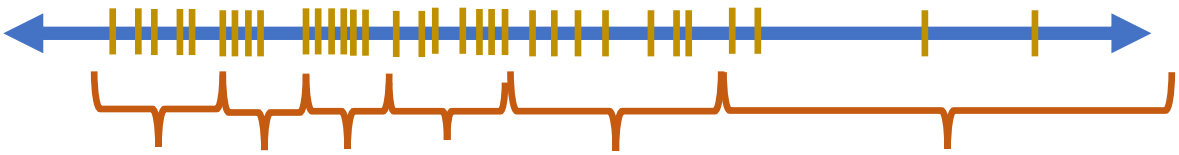


Structure of the Total Order Sorting Pattern
- Analysis phase

- 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

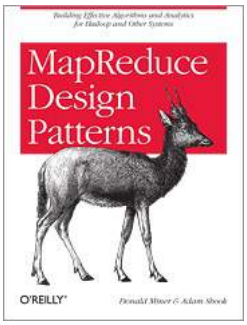
1. Reduce Side Join Pattern
2. Replicated Join Pattern

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:

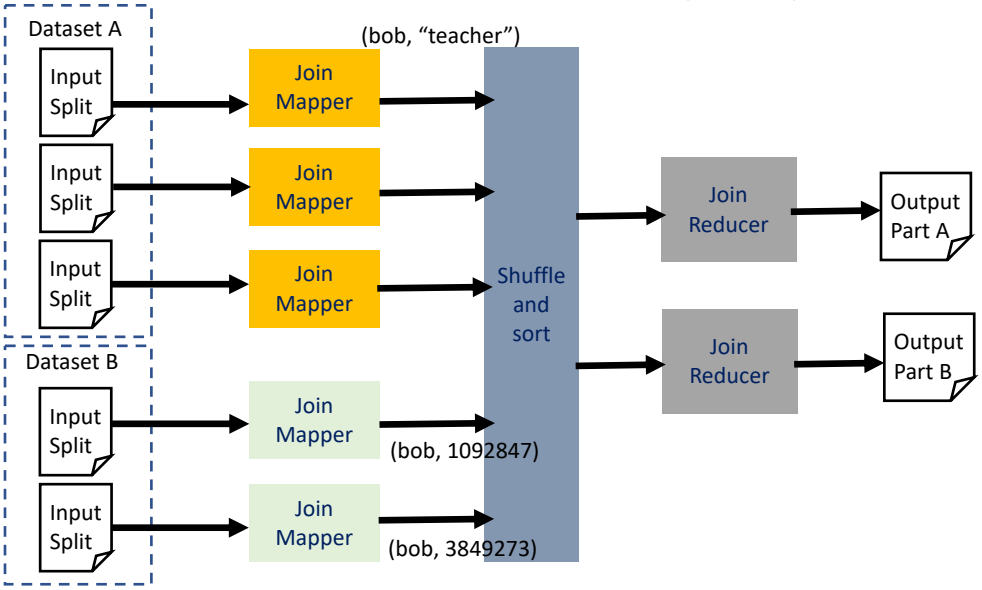
`user_ID, address, phone_number, DOB`

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.

REVIEW

Structure of the reduce side join pattern



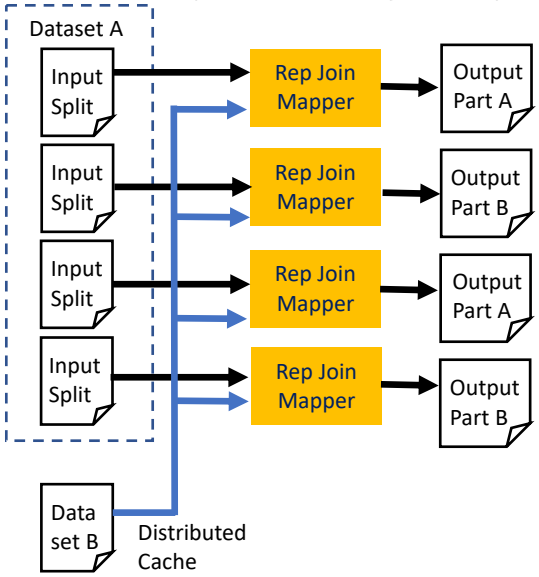
REVIEW

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



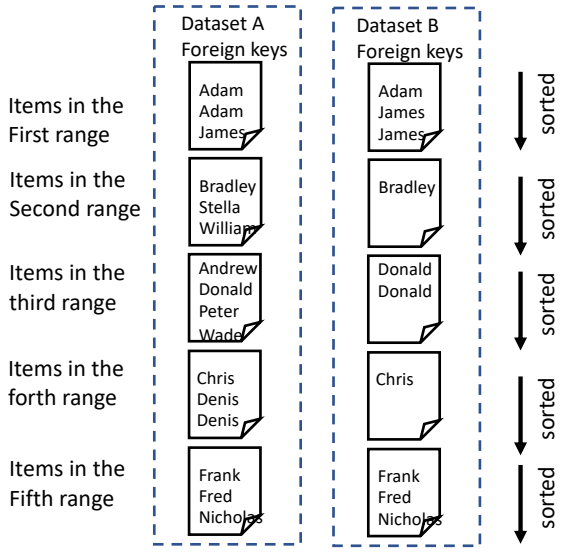
MapReduce Design Patterns IV: Join Patterns

3. Composite Join

Composite Join

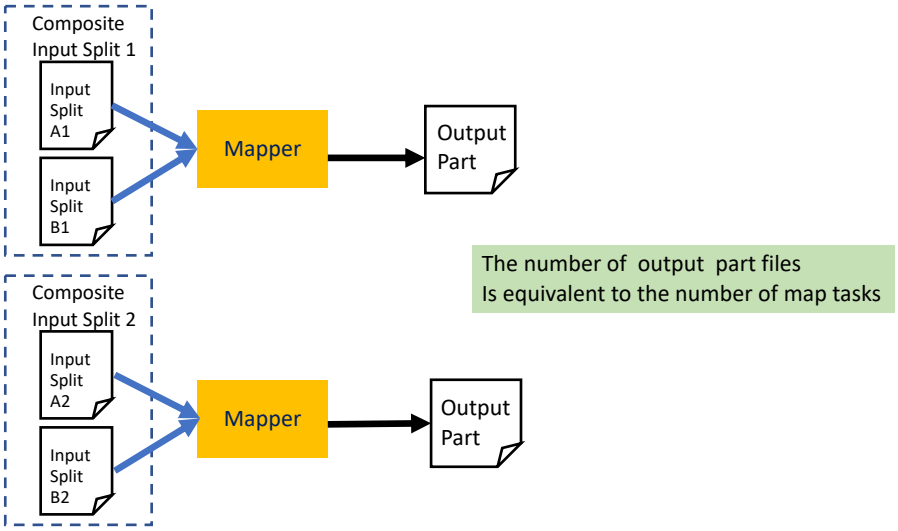
- Joins **very large datasets together**
  - And if the datasets are sorted by foreign key
- **No shuffle and sort needed**
- Each input dataset must be **partitioned and sorted in a specific way** and **divided into the same number of partitions**
  - **e.g. transactions, log files**

Structure of the composite join pattern



COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 5-A Fall 2021Computer Science

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

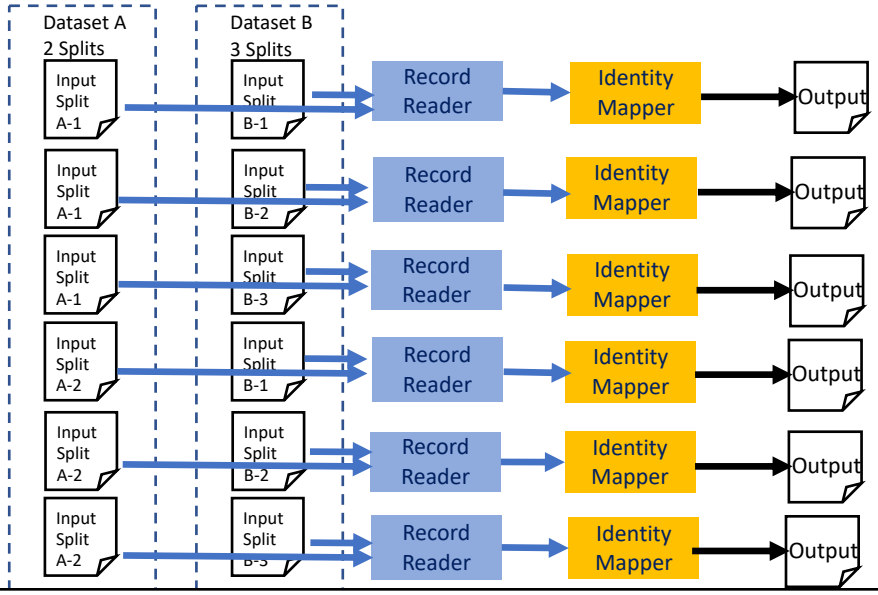
Cartesian Product

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

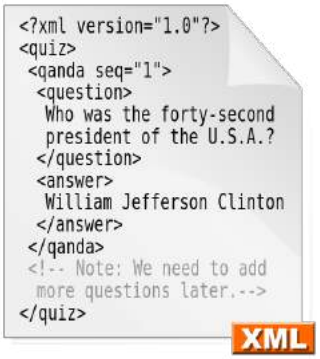
- `getSplits()`
  - retrieves the configured input using the `JobContext` object
  - returns a `List` of `InputSplit` objects
    - `getLocations()` of `InputSplit` returns the list of hostnames where the input split is located
    - This provides clue to the system to determine where to process the 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,
 );
```

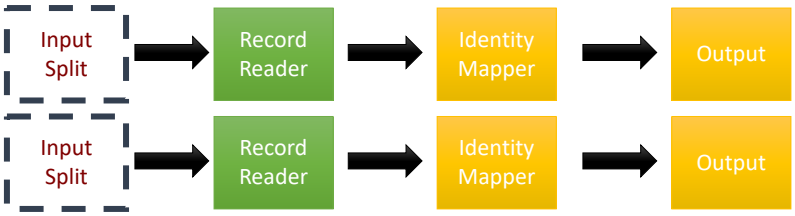
- In your class Main()
  - `job.setOutputKeyClass(DBOutputWritable.class);`
  - The DBOutputWritable should be able to manage the MyTable schema

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

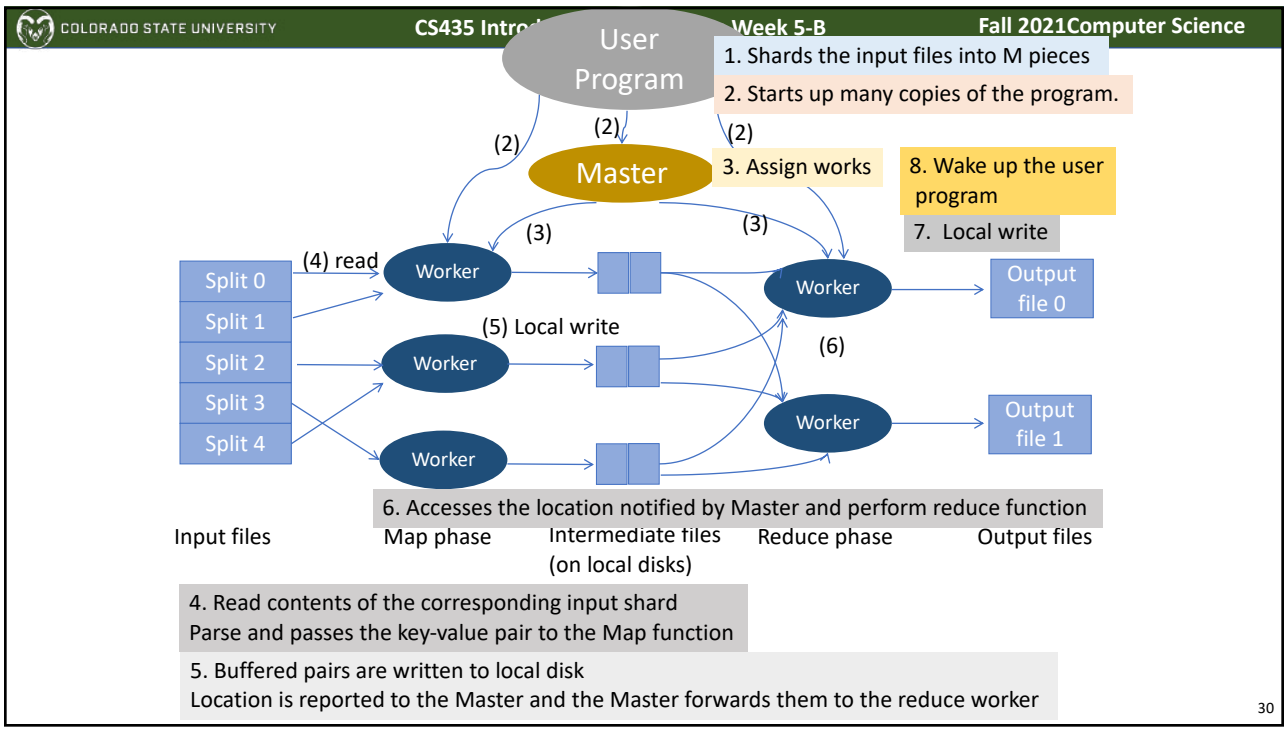
- Record reader is where the data is actually generated
- nextKeyValue
  - Uses a simple random number generator to create a random record
  - The number of words ranges from 1 to 30
  - Once all the records are generated, the record reader returns false
    - signals to the framework mapper that there is no input information

Part 1. Large Scale Data Analytics

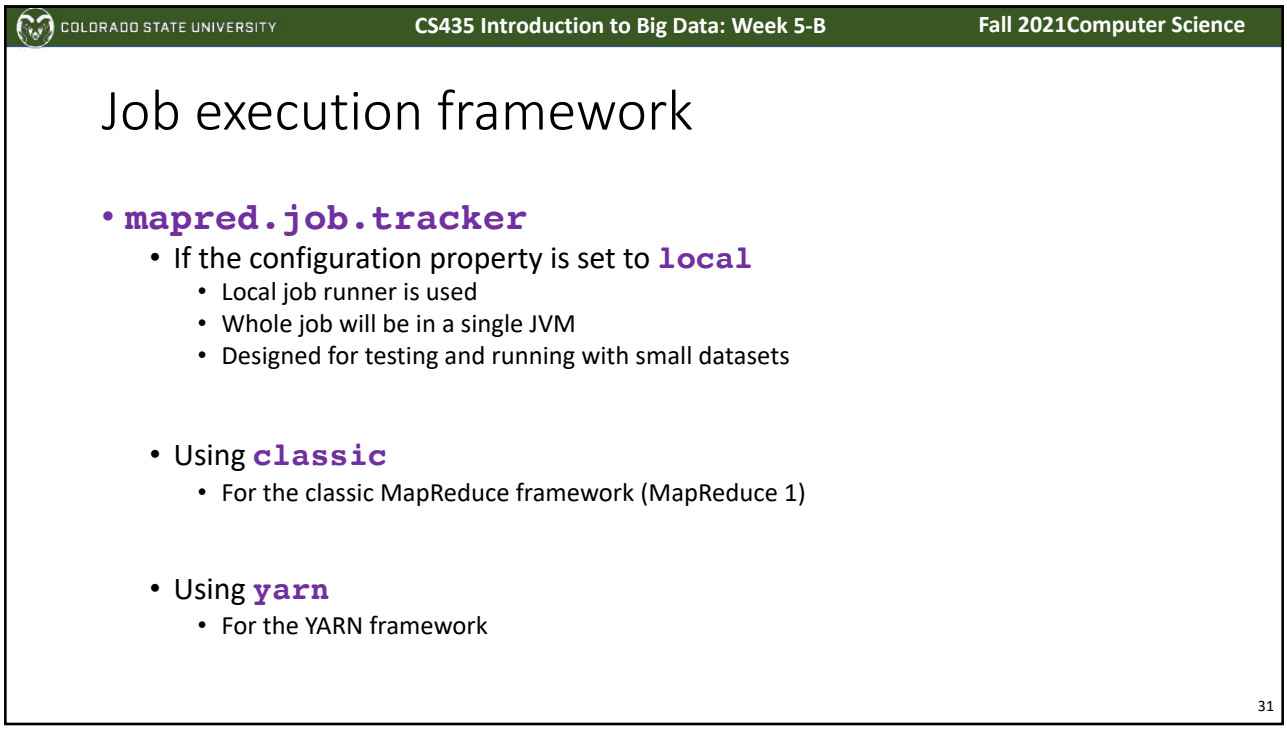
How MapReduce Works

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

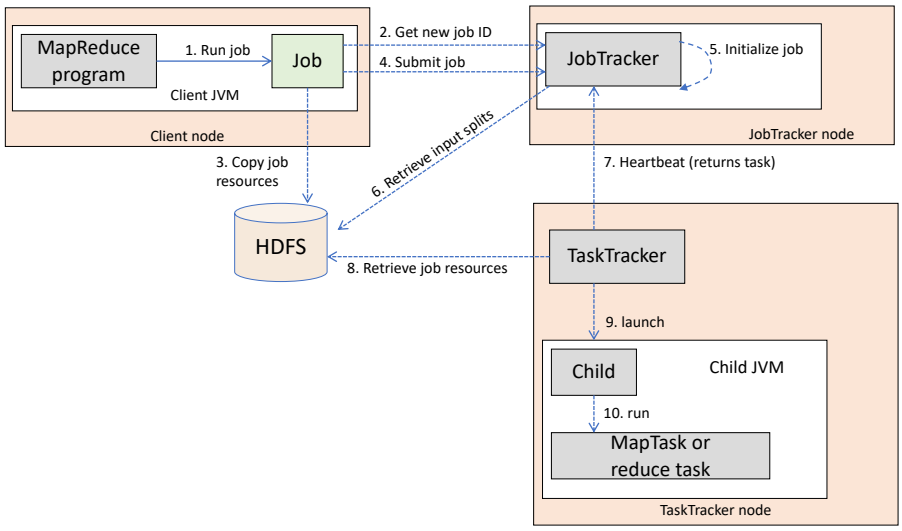
Job submission: JobTracker

- 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

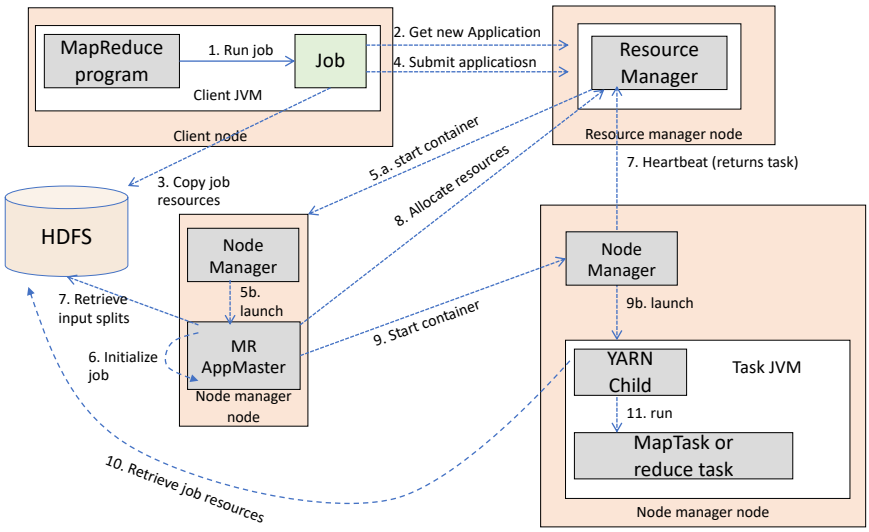
YARN (MapReduce 2)

- To provide the **scalability** to MapReduce
  - **Splitting responsibility of the jobtracker**
    - **Scheduling**
    - **Task progress monitoring**
- MapReduce is one type of YARN application

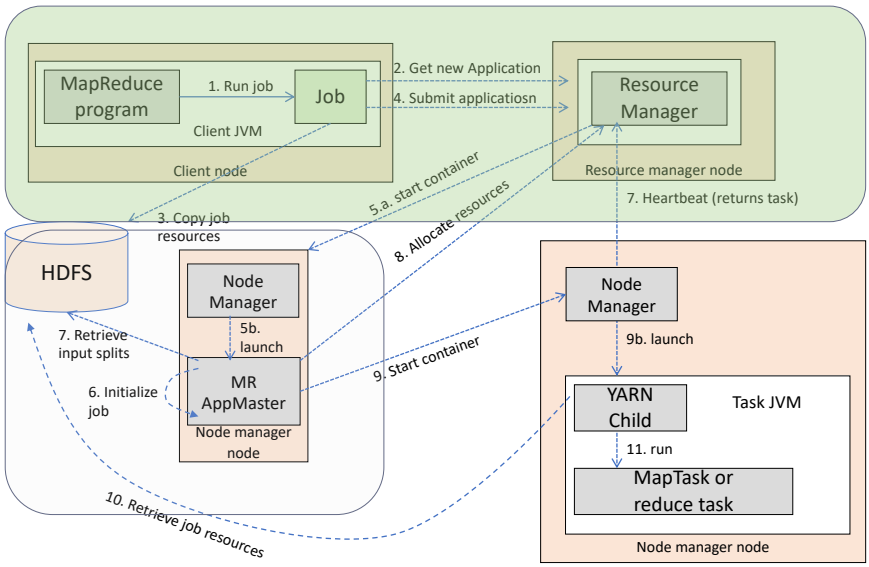
YARN (MapReduce 2)

- **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]

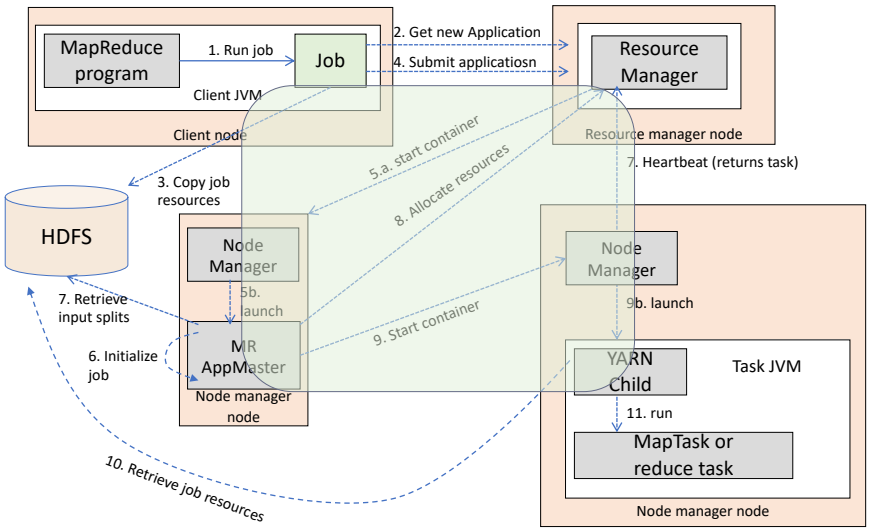
- **Application master**
  - The application master for MapReduce jobs
    - `MRAppMaster`
    - Initializes the job by creating bookeeping objects
      - To keep track of the job's progress
  - Retrieves the input splits
  - Creates a map task object for each split
  - Creates reduce task object
    - 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



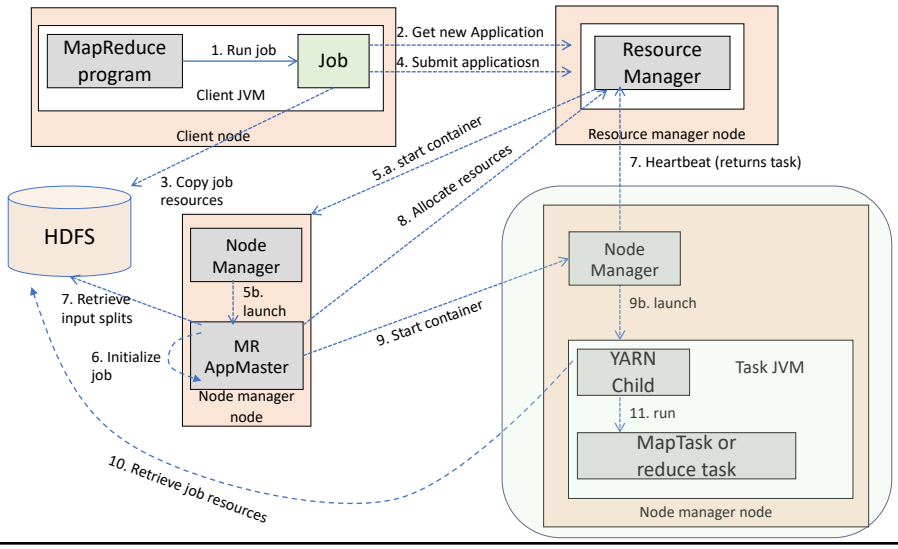
Task assignment [1/2]

- 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`

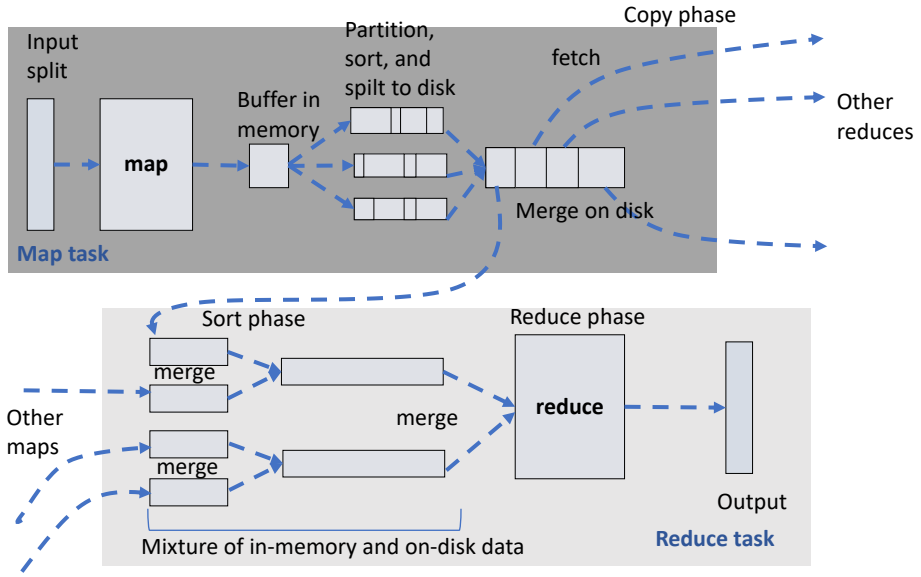
Part 1. Large Scale Data Analytics-How MapReduce Works

How Partitioner and Combiner work

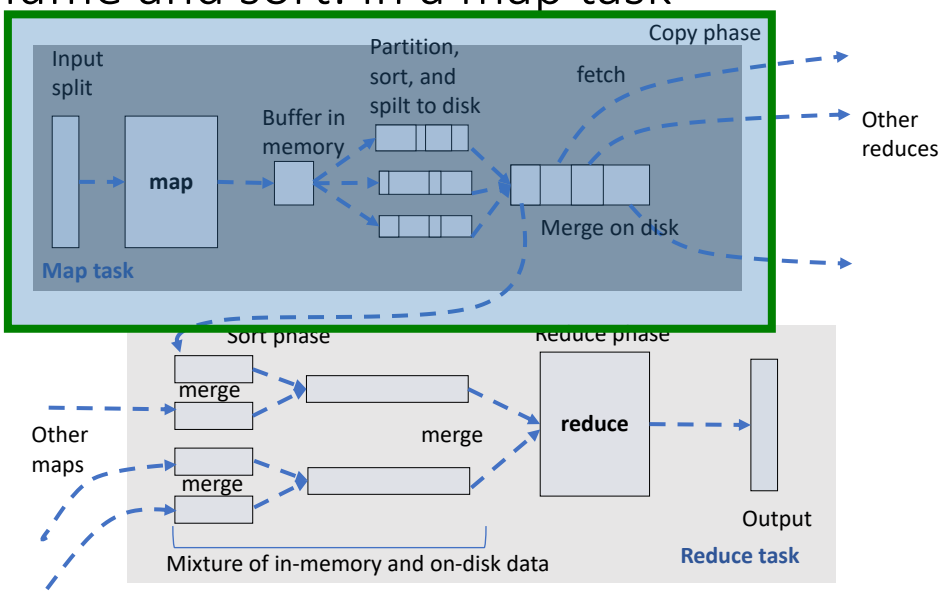
Shuffle and Sort

- 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**

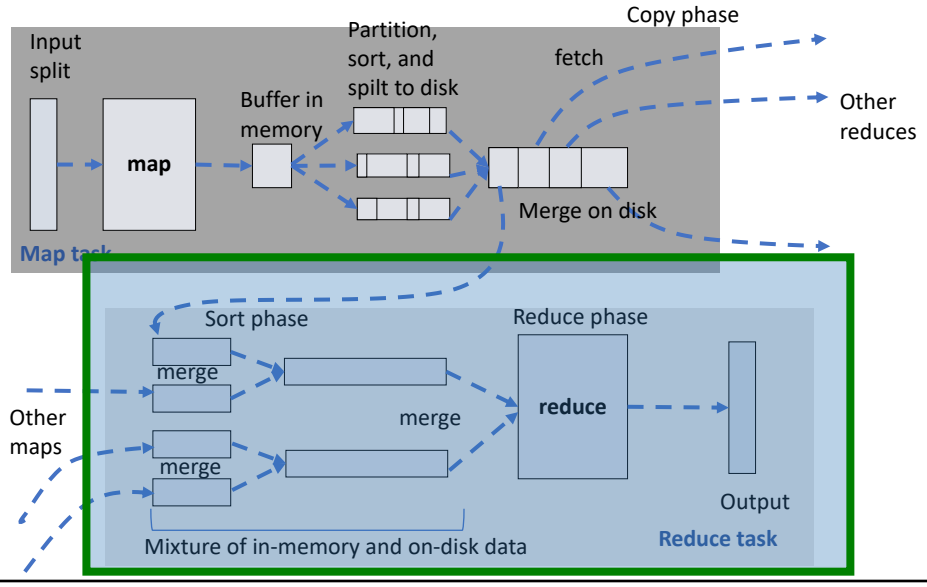
Execution of Combiner

- 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**

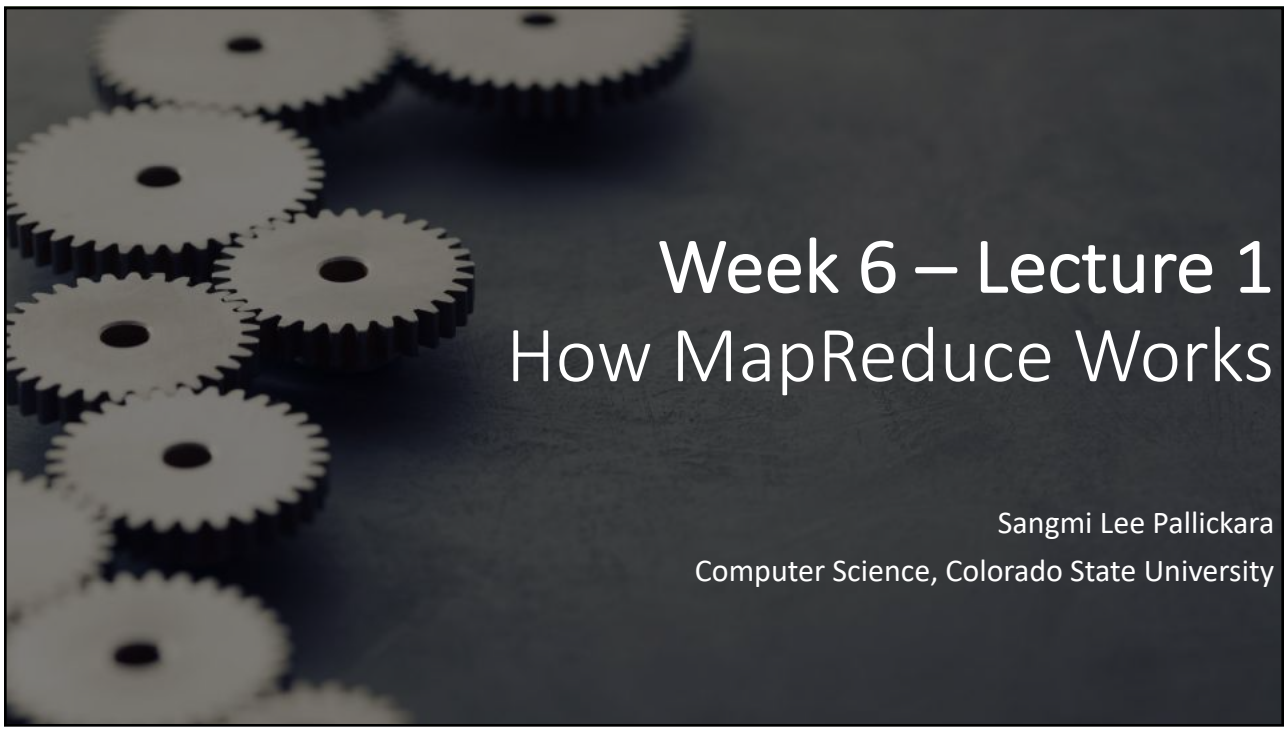
Node manager failure in YARN

- 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

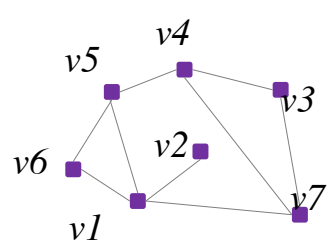
- Small Network
  - Highly clustered, like regular lattices, yet have small characteristic path lengths, like random graphs
- Duncan Watts and Steven Strogatz
  - Small-World Networks are a kind of random network
  - clustering coefficient and average node-to-node distance (A.K.A. shortest path length)

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

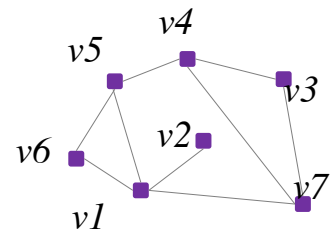
Programming Assignment 2 -How to identify the small-world networks

- **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 = $L \quad = \frac{1}{v(v-1)} \sum_{if\ n,i \neq j} d_{i,j}$



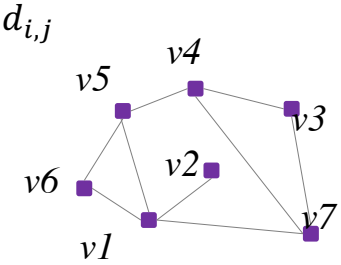
Note: your graph is not a directed graph. Therefore, (a, c) = (c, a). You should not count the same pairs of nodes twice.

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

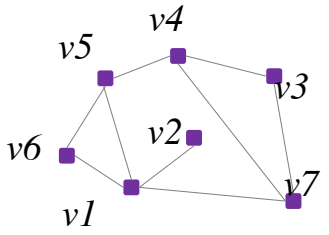
9

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

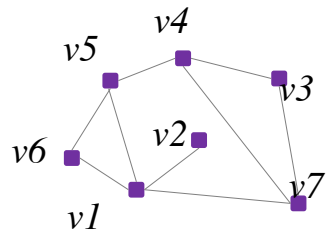
Programming Assignment 2
-How to identify the small-world networks

- **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?

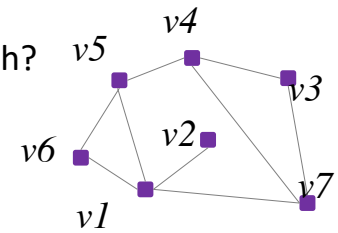


COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 6-A Fall 2021Computer Science

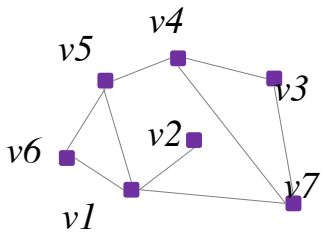
Programming Assignment 2
-How to identify the small-world networks

- **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`

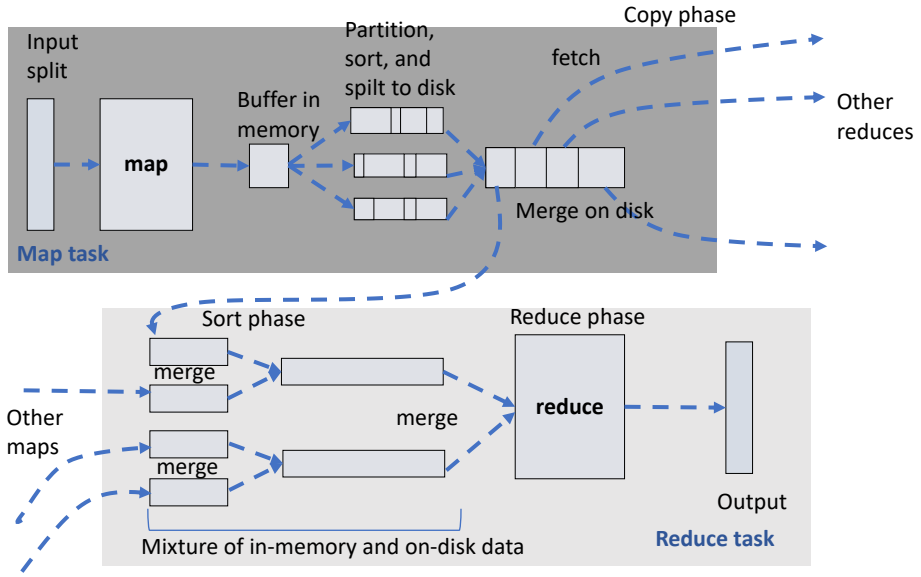
Part 1. Large Scale Data Analytics-How MapReduce Works

How Shuffle, Sort, Partitioner and Combiner work

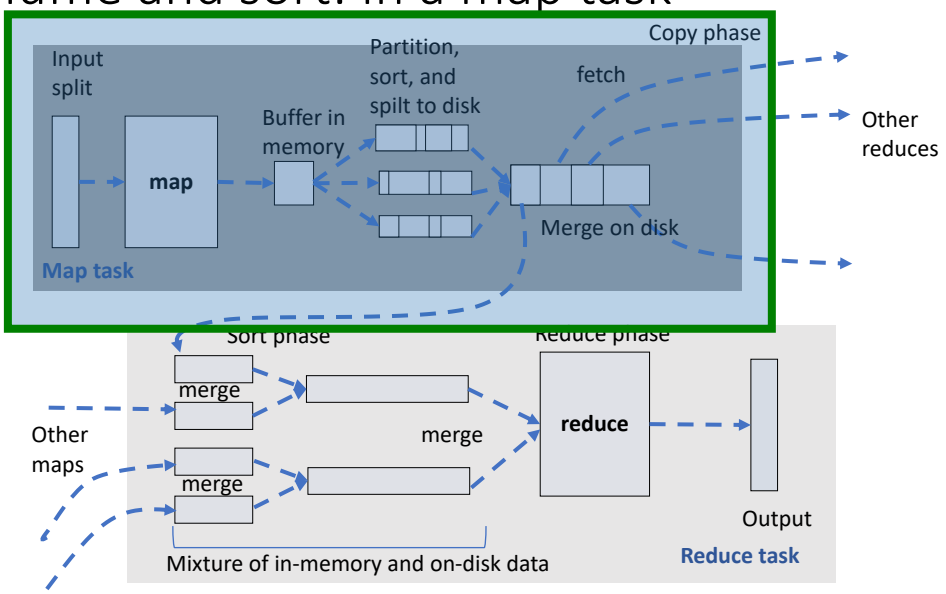
Shuffle and Sort

- 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**

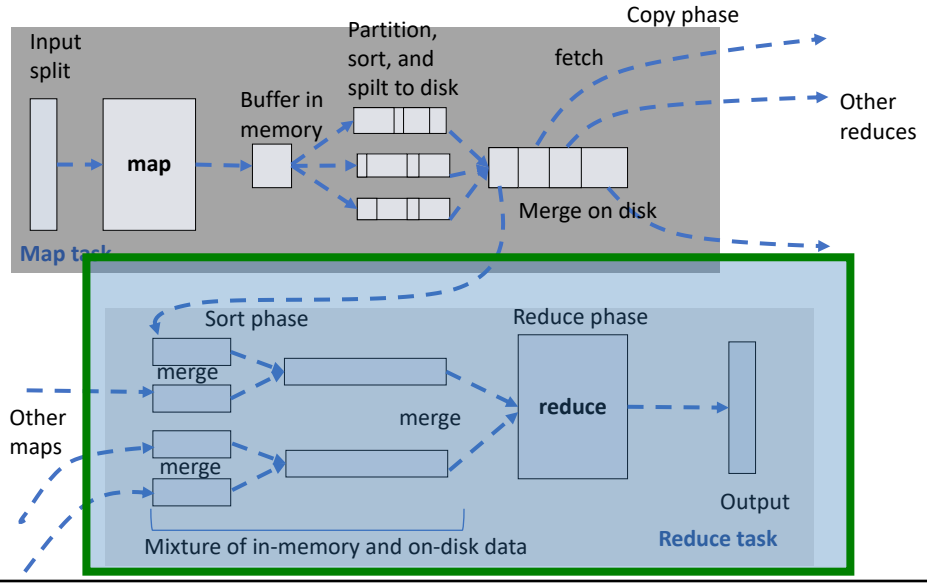
Execution of Combiner

- 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

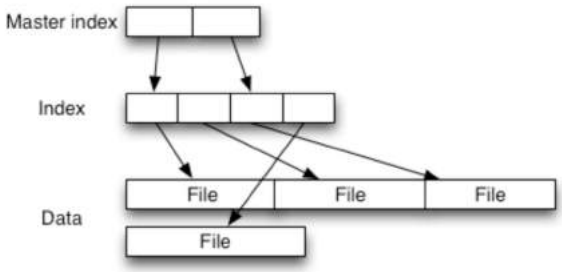
- Small files
  - Files those are significantly smaller than the HDFS block size
- HDFS cannot handle lots of files!
- Every file/directory and block in HDFS
  - Represented as an object in the namenode's memory
  - 150 bites per object (rue of thumb)
- e.g. 10 million files: each using a block: use 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

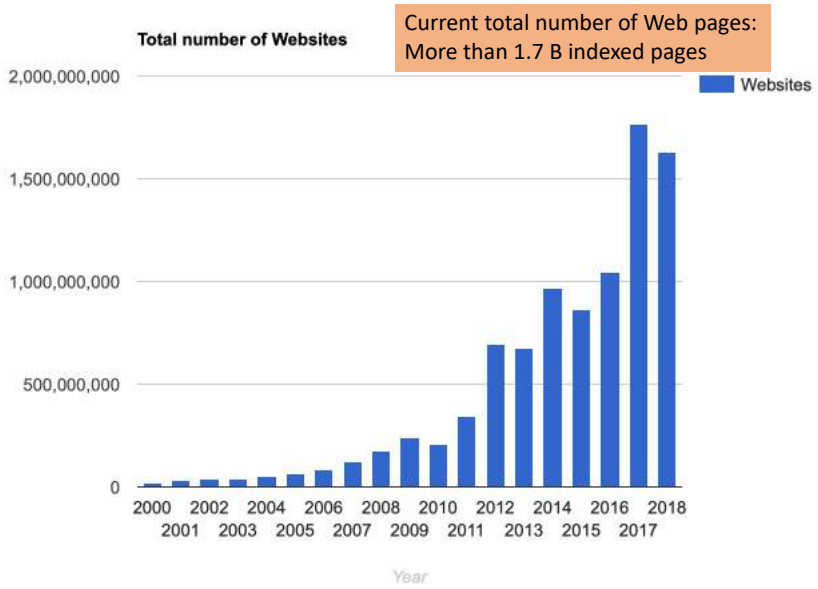
Part 1. Large Scale Data Analytics

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**

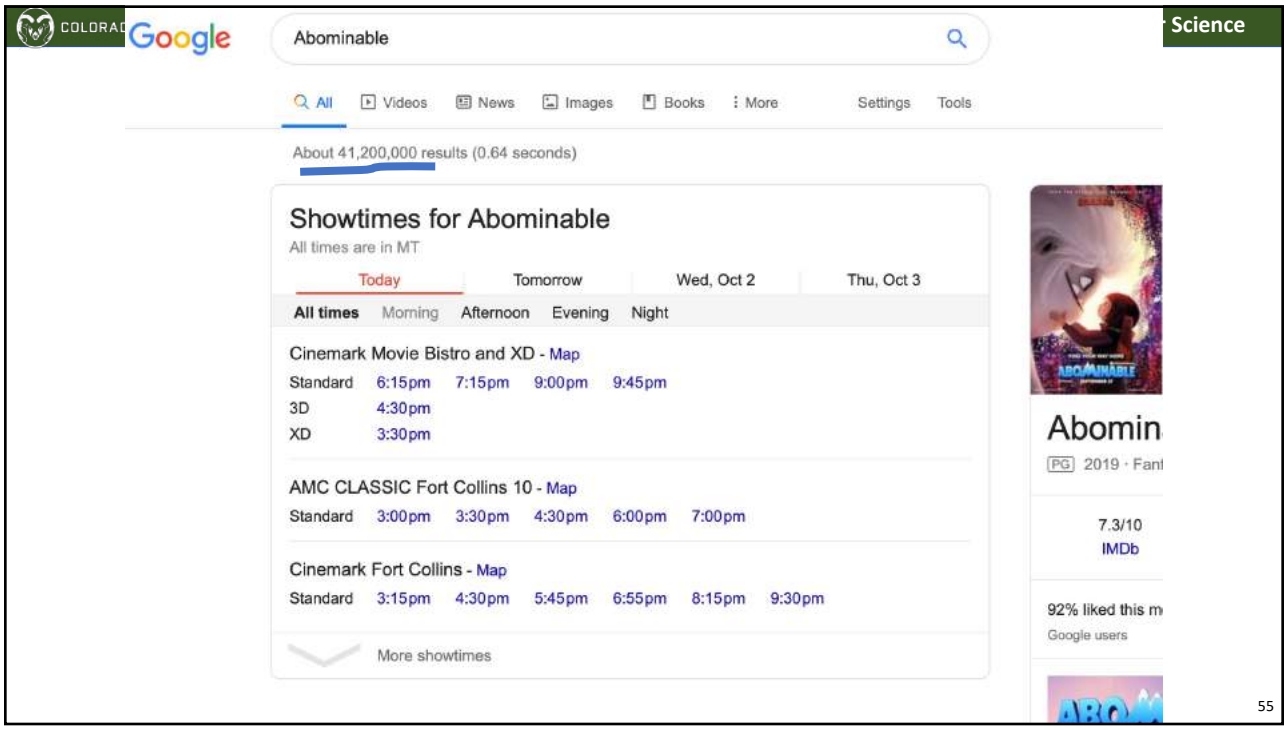
Current total number of Web pages:
More than 1.7 B indexed pages



http://www.internetlivestats.com/total-number-of-websites/

54

54



55

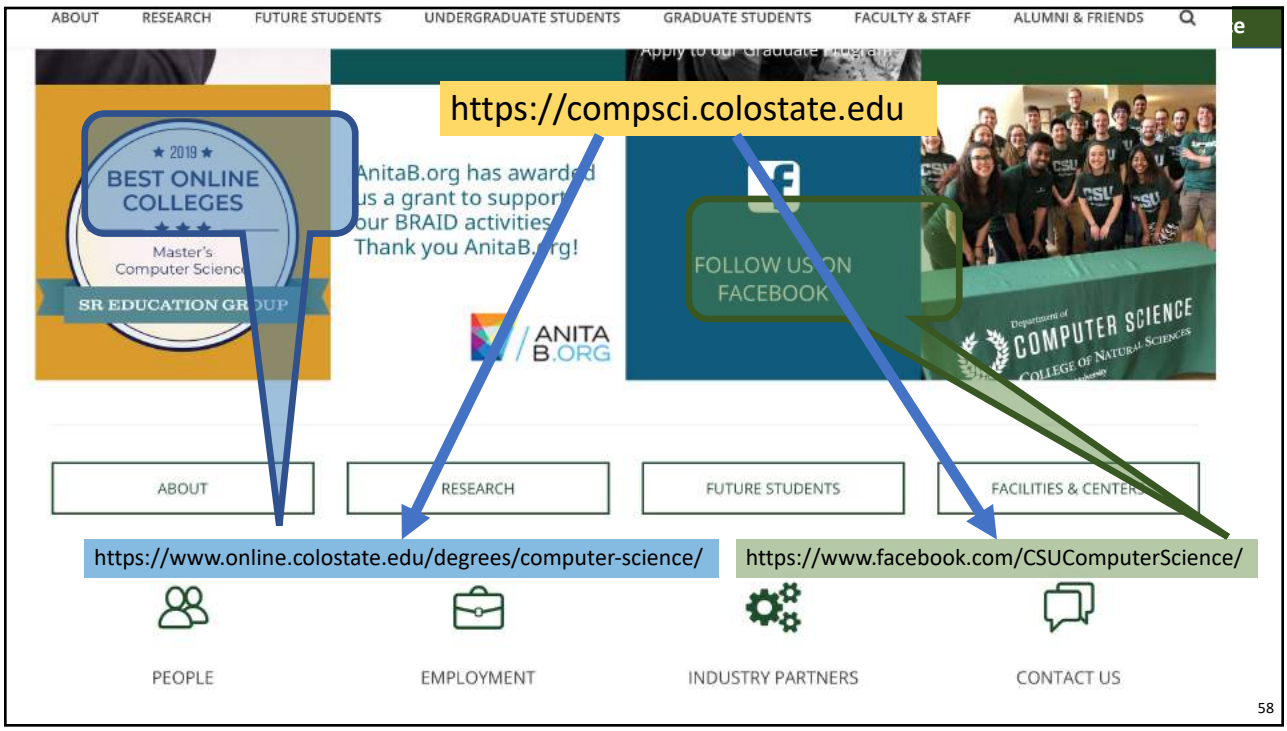
55

PageRank

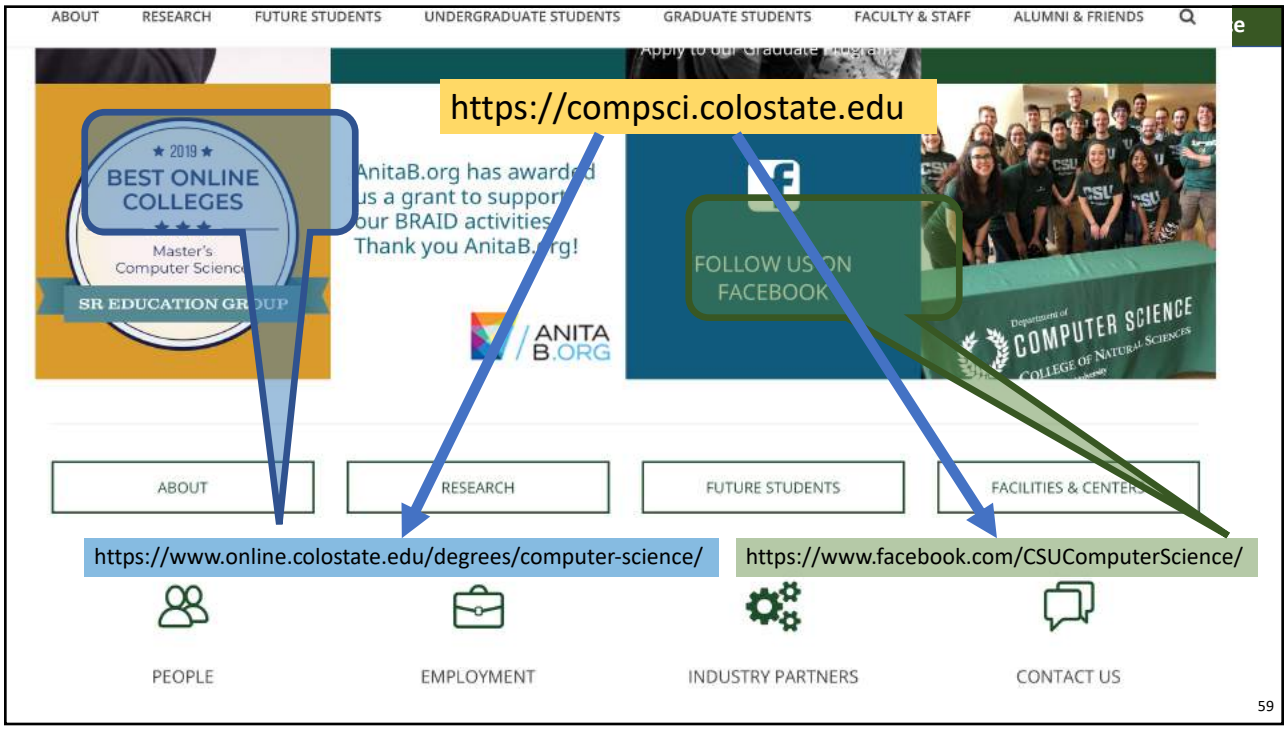
- Goals
  - Providing **effective summaries** for the search results
  - **Ordering/Ranking results**
- Simulate random Web surfers
  - Pages that would have a large number of surfers were considered more "important" than pages that would rarely be visited
- The content of a page was judged not only 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,

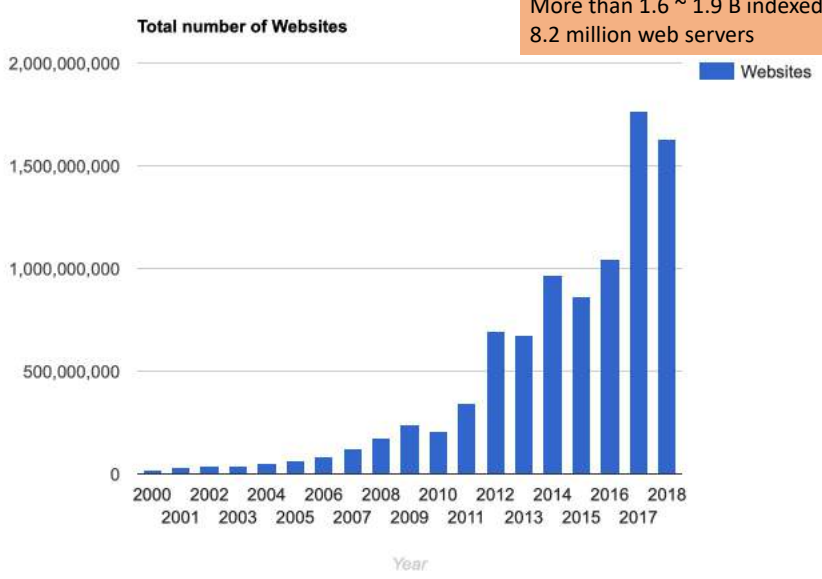
- Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman, "Mining of Massive Datasets", Cambridge University Press, 2012
  - Chapter 5
- http://infolab.stanford.edu/~ullman/mmds.html

Part 1. Large Scale Data Analytics

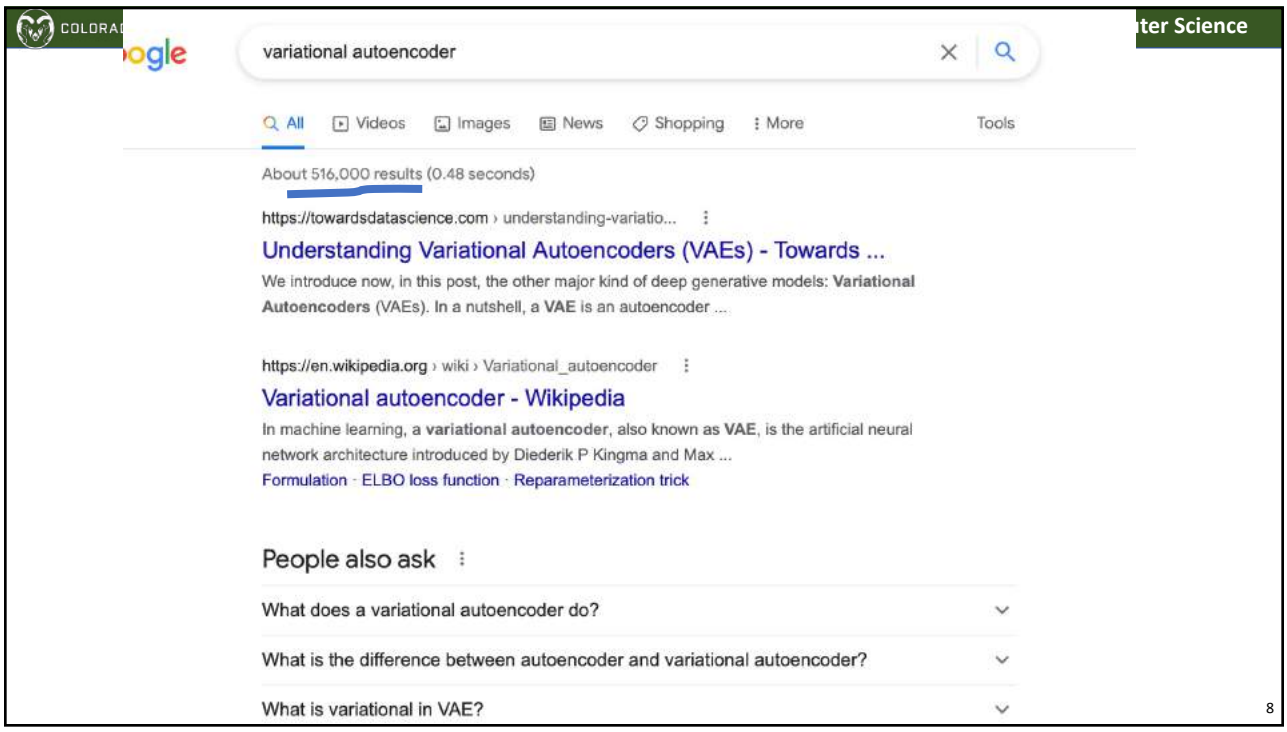
Web-Scale Link Analysis and Social Network Analysis

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

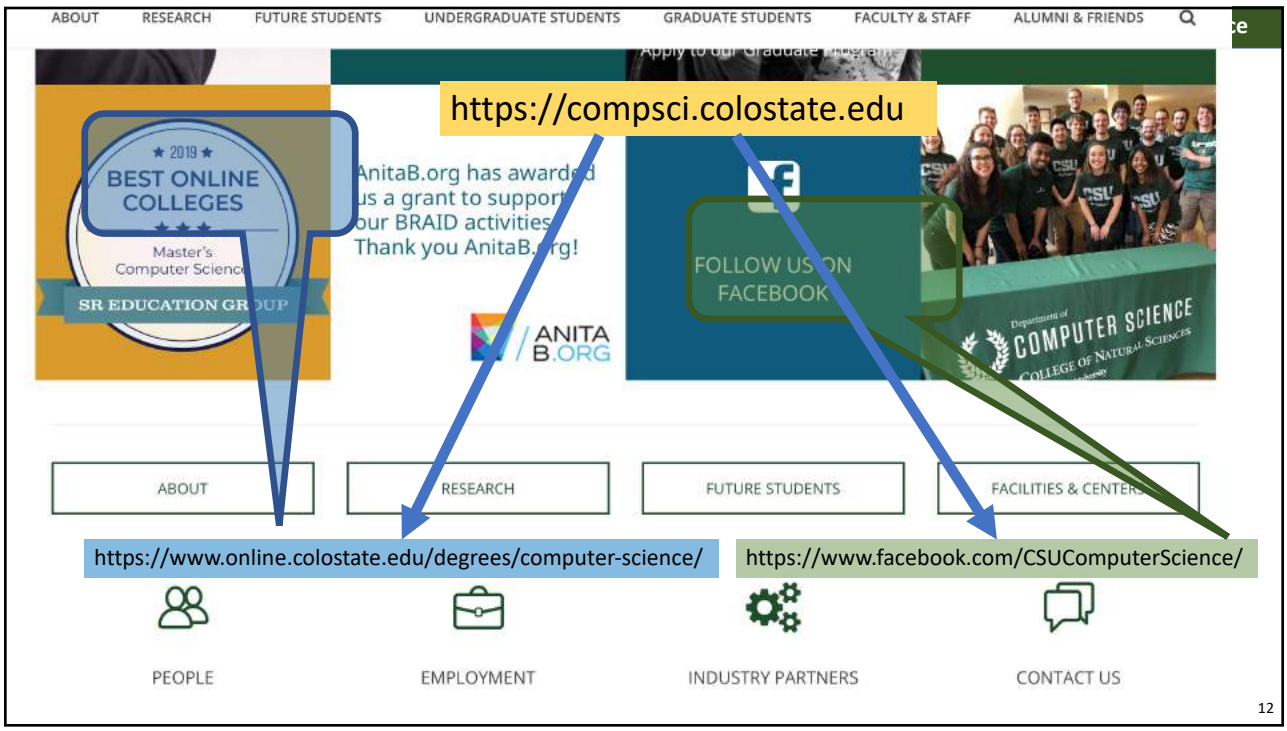
9

PageRank

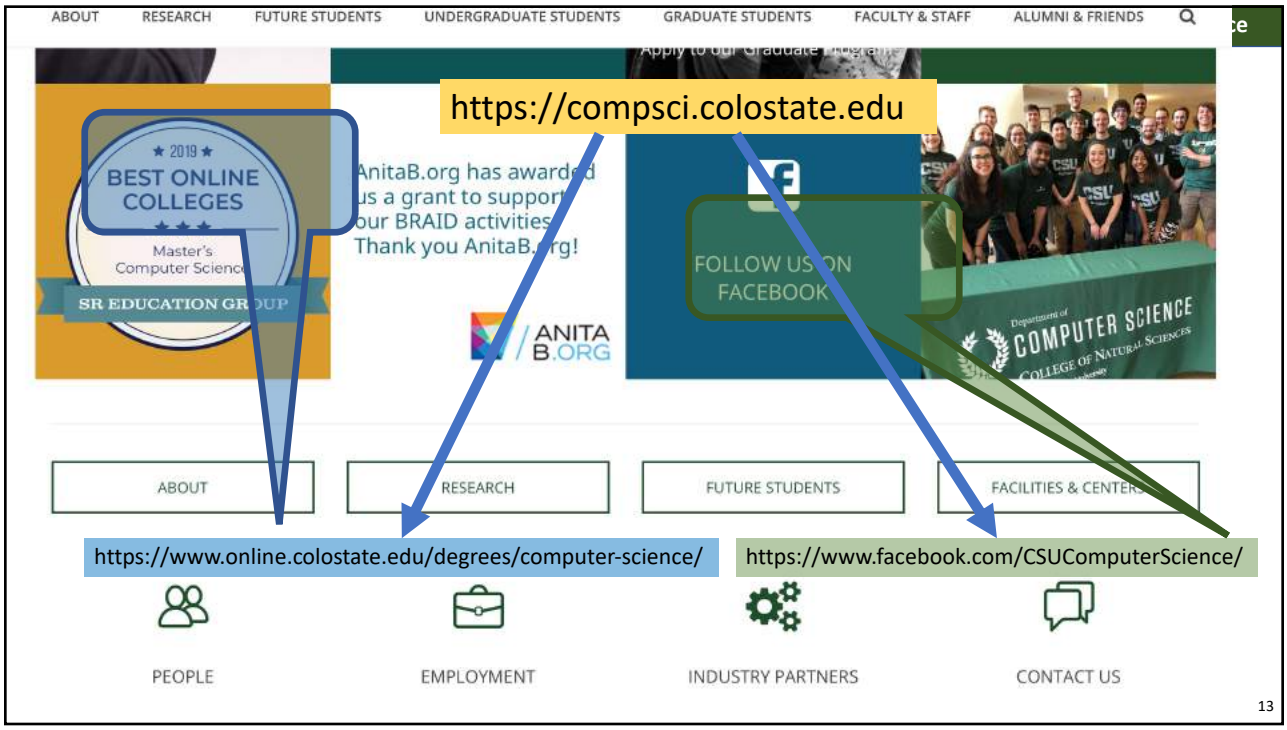
- Goals
  - Providing **effective summaries** for the search results
  - **Ordering/Ranking results**
- Simulate random Web surfers
  - Pages that would have a large number of surfers were considered more "important" than pages that would rarely be visited
- The content of a page was judged not only 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

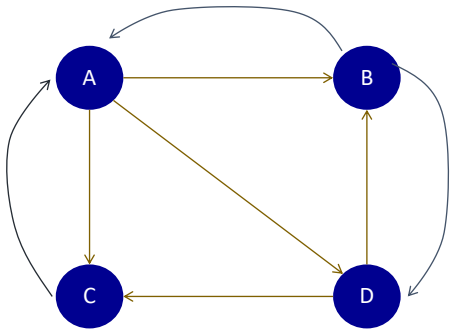
12


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]

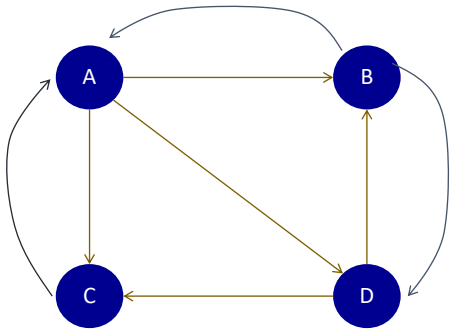
- Page A has links to B, C and D
- Page B has links to A and D
- Page C has a link to A
- Page D has links to B and C



Worksheet Question #1

- The URL of the page A is included in

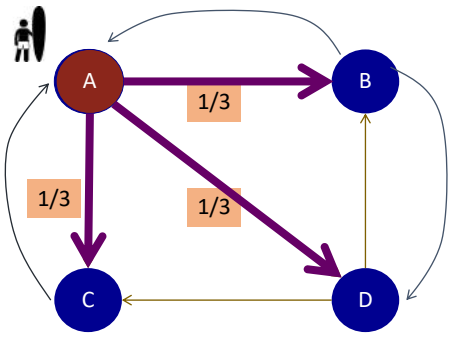
a. B, C, and D
b. A
c. B and C
d. None



Answer: c –B and C

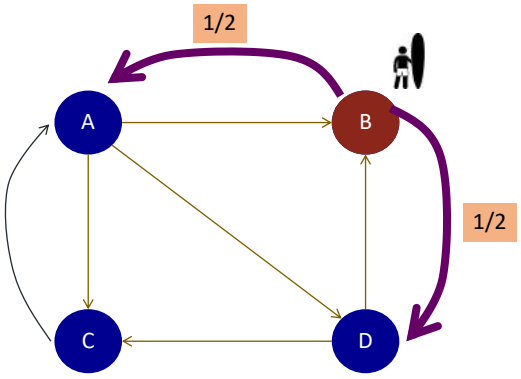
Example [2/5]

- 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]

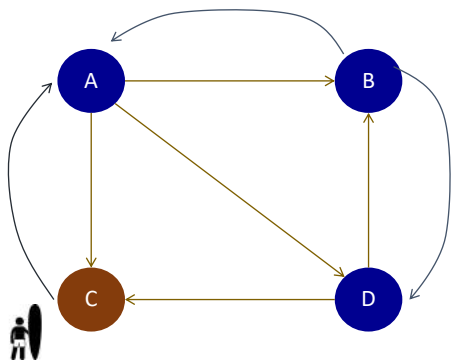
- Now suppose the random surfer at B
  - B has probability of ½ of being at A, ½ of being at D and 0 of being at B or C



Question

- Now suppose the random surfer at C
  - What is the probability that C will be at A?

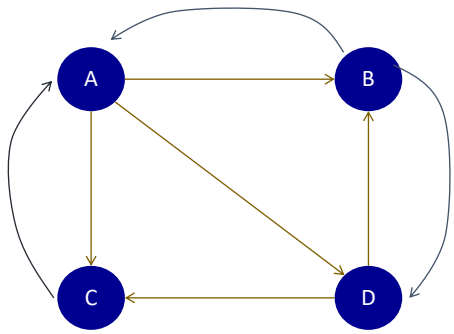
  a. 1/4 b. 1/2 c. 1 d. 0



Answer: C

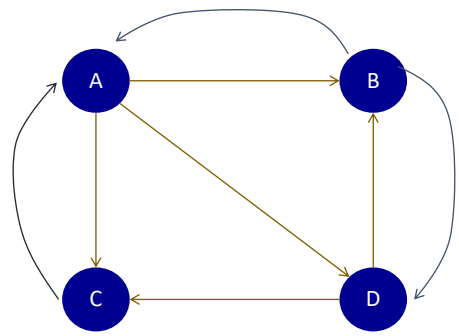
Example [4/5]

- **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]

$$M = \begin{array}{c} \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \\ \begin{array}{cccc} A & B & & \end{array} \end{array}$$

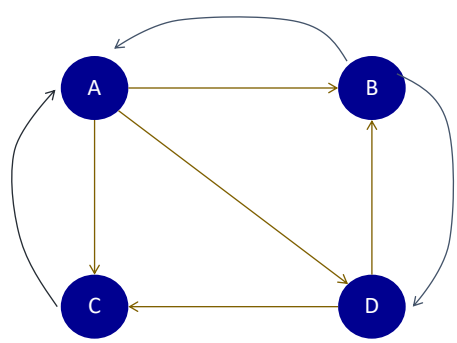


- The first column
  - a surfer at A has a 1/3 probability of next being at each of the other pages
- The second column
  - a surfer at B has a ½ probability of being next at A and the same for being at D

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

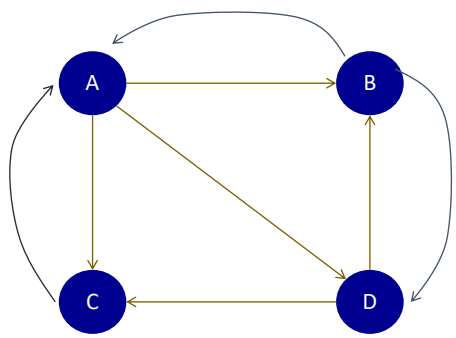
$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1.2 & 0 & 0 \end{bmatrix}$$



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

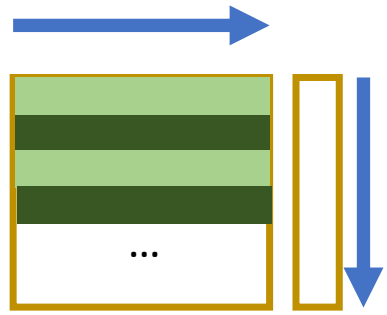
Matrix-vector Multiplication by MapReduce [3/3]

- 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

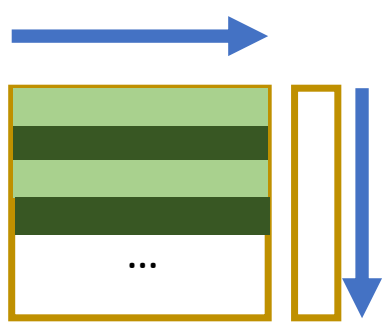
The Map function

- 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)

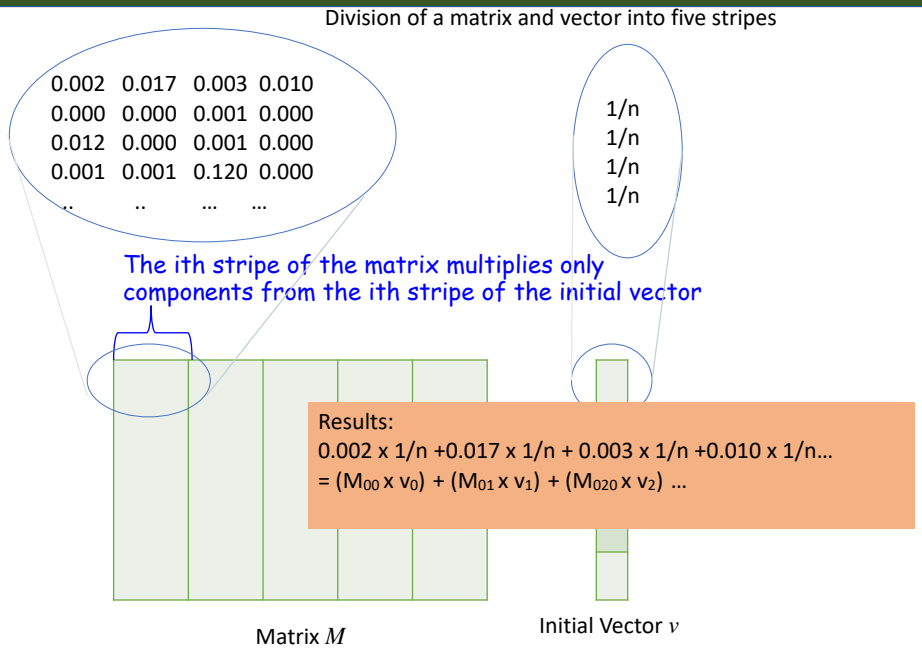


CASE 2: WITH VERY LARGE v (partitioning needed)

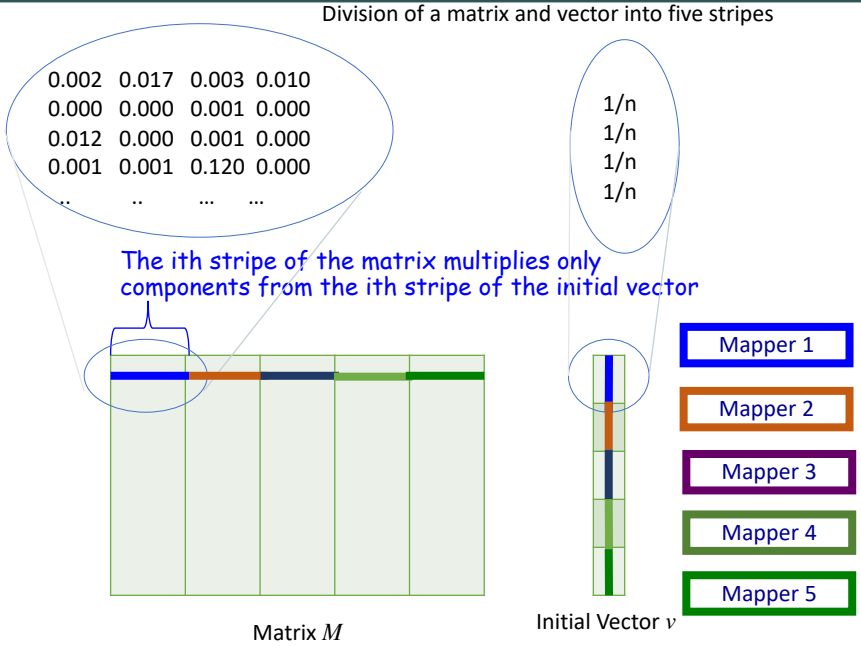
What if the vector v cannot fit in main memory?

- 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



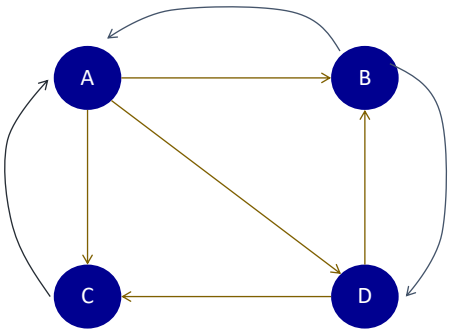
Division of a matrix and vector into five stripes



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

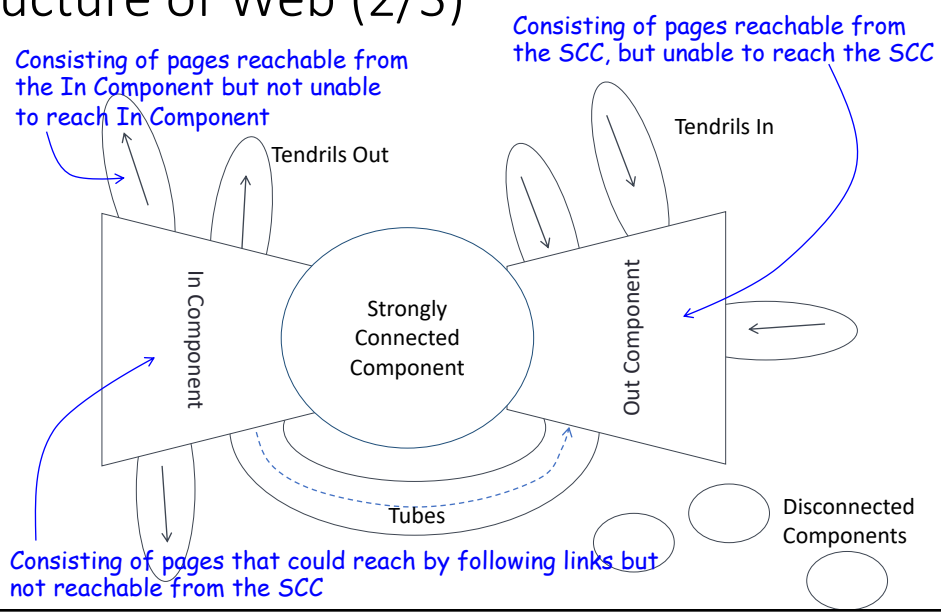
Link Analysis

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

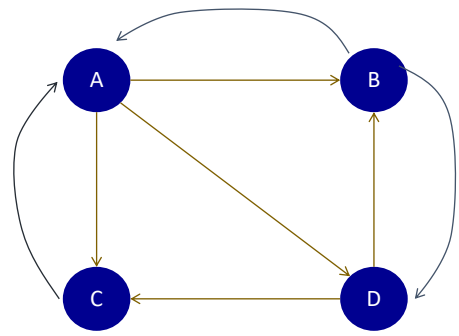
Topics to be covered

- 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

$$v_1=Mv_0=\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}=\begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

4

4

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

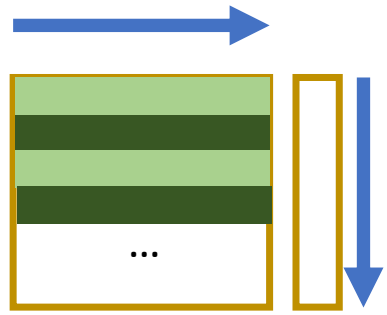
5

5

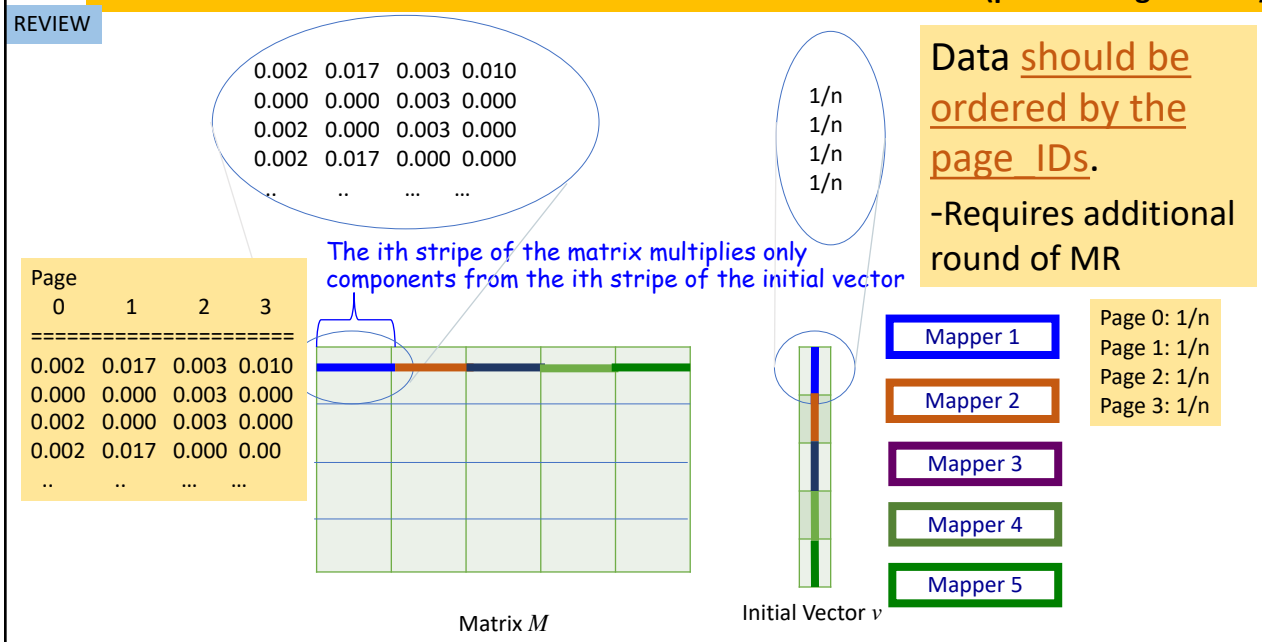
REVIEW **: MATRIX vs. VECTOR MULTIPLICATION: Without partitioning the vector**

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
..     ..     ...    ...
```

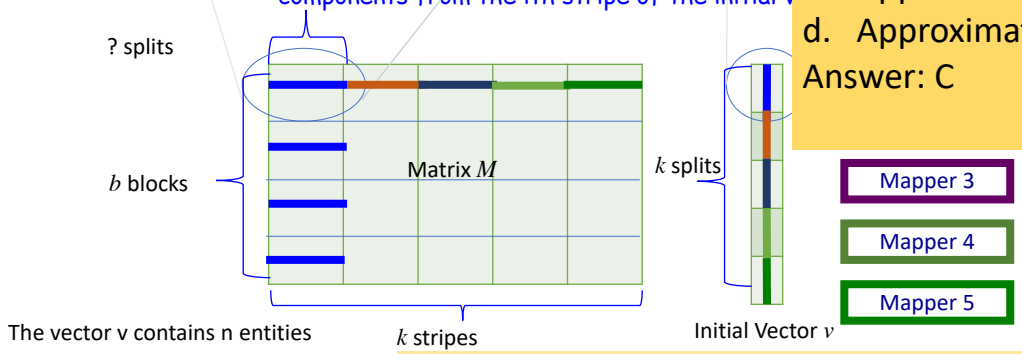
Page 0: 1/n
Page 1: 1/n
Page 2: 1/n
Page 3: 1/n

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

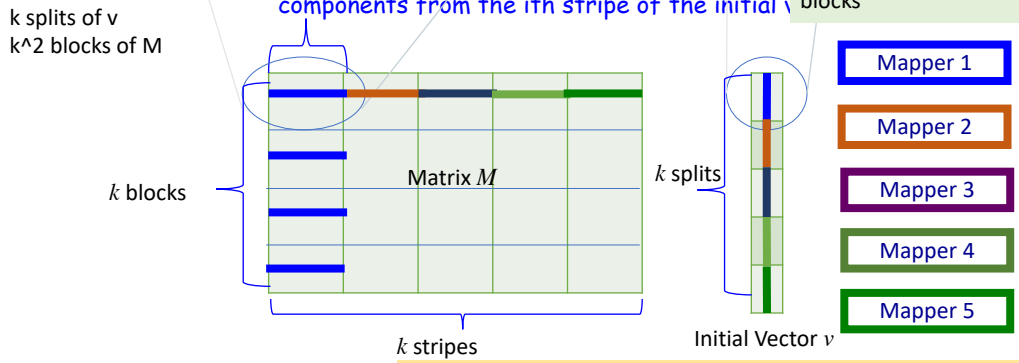
Mapping blocks within a stripe of M to the corresponding split of v

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

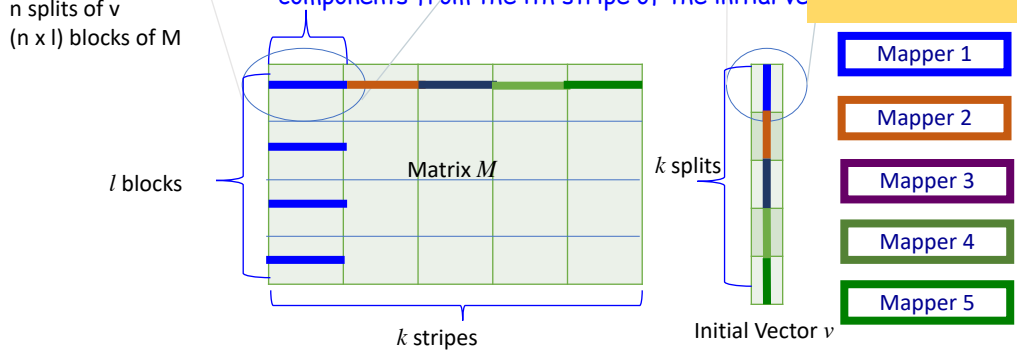
```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

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

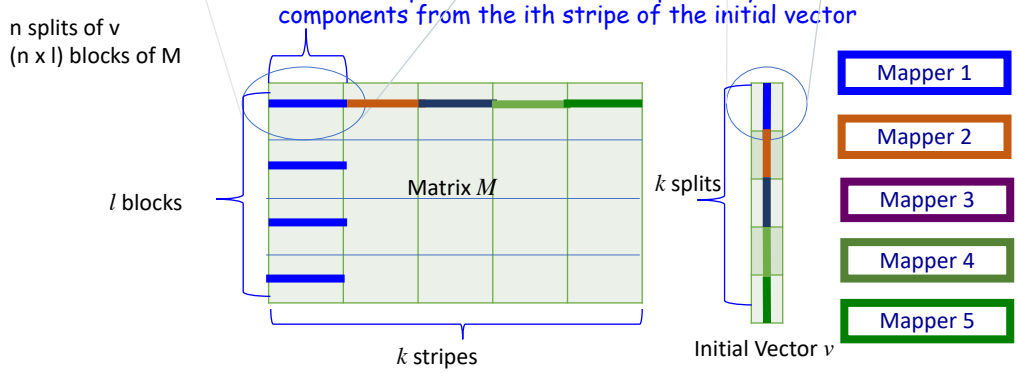
```
Page: 0   1       2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..     ..     ...    ...
```

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

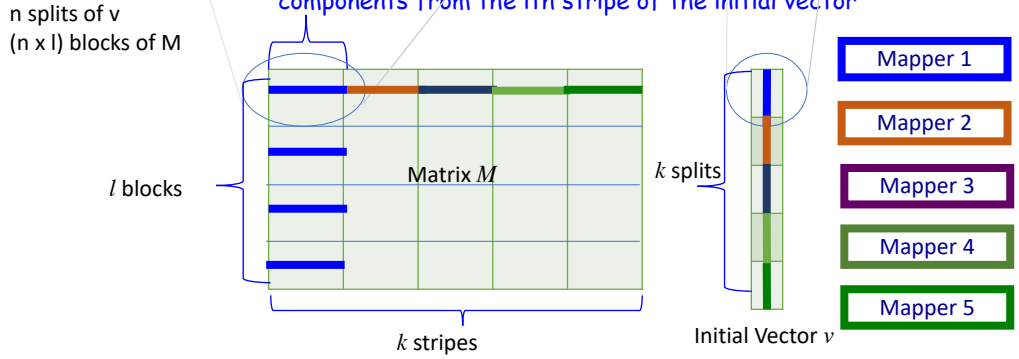
```
Page: 0    1        2       3
=====================
0.002  0.017  0.003  0.010
0.000  0.000  0.003  0.000
0.002  0.000  0.003  0.000
0.002  0.017  0.000  0.00
..       ..      ...    ...
```

```
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

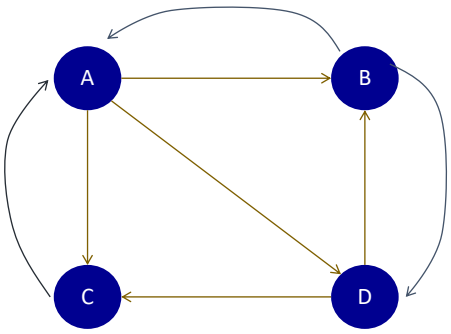
13

REVIEW

Random-walk style implementation

- **Without a transition matrix**

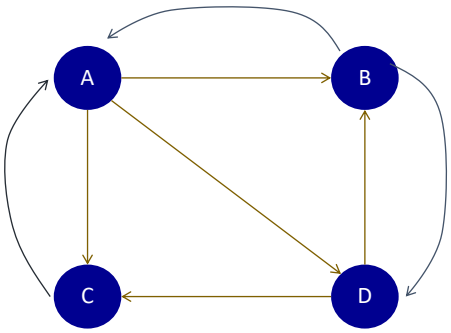
- By performing the **next visit**, the page A will propagate its current page rank value to the page B, C, or D as much as the probability to be in one of these pages
  - E.g. Suppose the current page rank value of A is PR(A). This will be distributed to the page B, C, and D as much as PR(A)*1/3



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

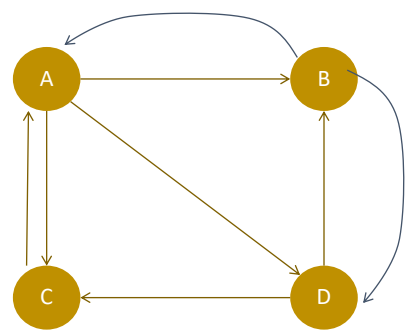
Challenges in PageRank Algorithm for the real Web: (1) Dead ends

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



$$M=\begin{bmatrix} 0 & 1/2 & \mathbf{0} & 0 \\ 1/3 & 0 & \mathbf{0} & 1/2 \\ 1/3 & 0 & \mathbf{0} & 1/2 \\ 1/3 & 1/2 & \mathbf{0} & 0 \end{bmatrix}$$

Example-continued

- Repeatedly multiplying the vector by M:

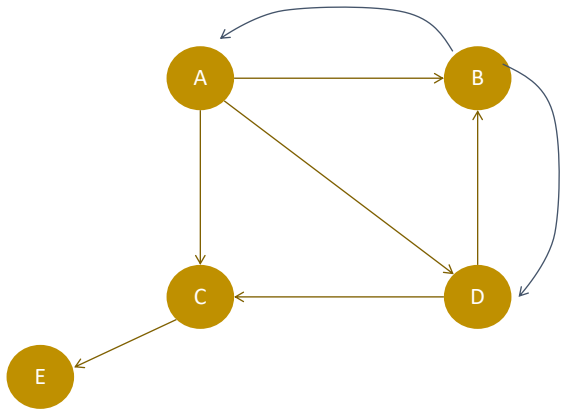
$$\begin{bmatrix} ¼ \\ ¼ \\ ¼ \\ ¼ \end{bmatrix} \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix} \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix} \ldots \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- **The probability of a surfer being anywhere goes to 0** as the number of steps increase

Approaches to dealing with dead ends

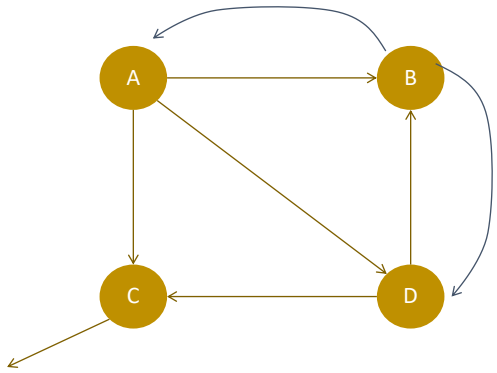
- **Recursive deletion**
  - Drop the dead ends from the graph
    - Drop their incoming arcs as well
    - Doing so **may create more dead ends**
      - Drop those new dead ends
- **Taxation**
  - Modify the process
    - Random surfers are assumed to move around on the Web without links

Example of recursive deletion (1/4)

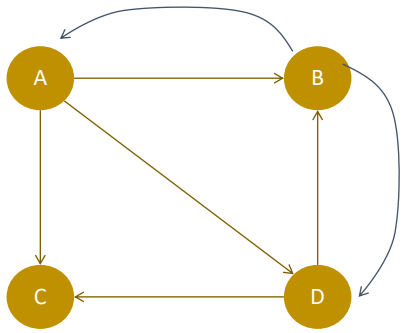


COLORADO STATE UNIVERSITY CS435 Introduction to Big Data: Week 7-A Fall 2021 Computer Science

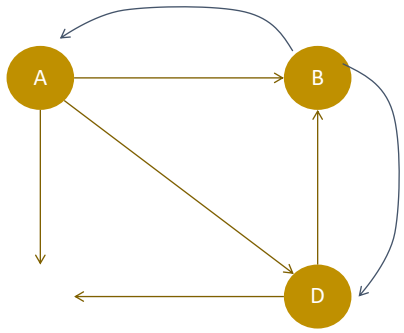
Example of recursive deletion (1/4)



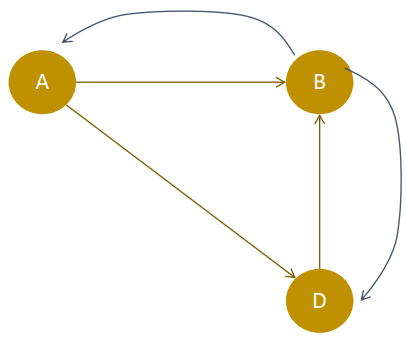
Example of recursive deletion (1/4)



Example of recursive deletion (1/4)



Example of recursive deletion (1/4)



Example of recursive deletion (2/4)

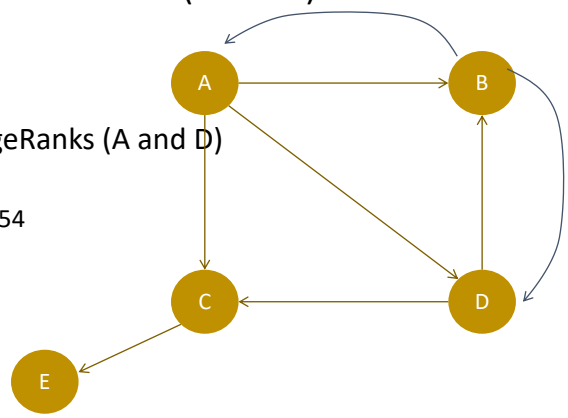
- The final matrix for the graph is

$$M = \begin{bmatrix} 0 & ½ & 0 \\ ½ & 0 & 1 \\ ½ & ½ & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \begin{bmatrix} 3/12 \\ 5/12 \\ 4/12 \end{bmatrix} \begin{bmatrix} 5/24 \\ 11/24 \\ 8/24 \end{bmatrix} \quad \ldots \quad \begin{bmatrix} 2/9 \\ 4/9 \\ 3/9 \end{bmatrix}$$

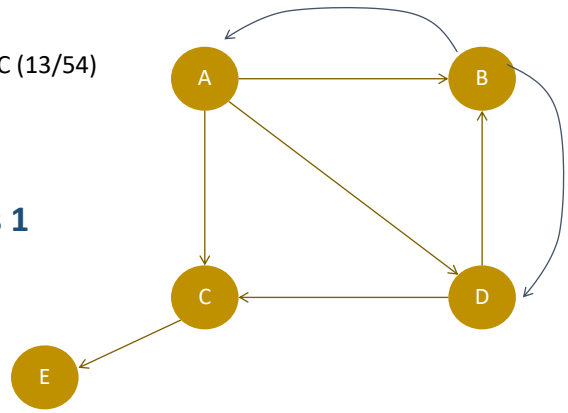
Example of recursive deletion (3/4)

- 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

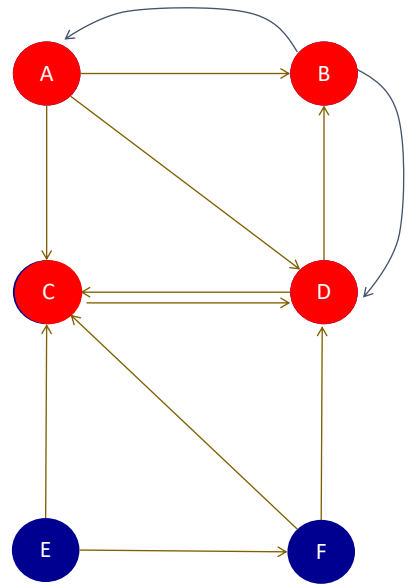
- **Spider traps**
  - A set of nodes with **no dead ends but no arcs out** (from the set of nodes to the outside of the set)
  - This can appear intentionally or unintentionally on the Web
- Spider traps causes the PageRank calculation to **place all the PageRank within the spider traps**

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?

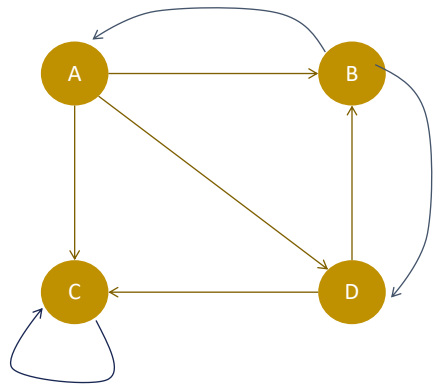


- If we perform the usual iteration to compute the PageRank of the nodes, we get

$$\begin{bmatrix} 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \end{bmatrix} \begin{bmatrix} 1/12 \\ 5/36 \\ 11/36 \\ 4/18 \\ 0 \\ 1/12 \end{bmatrix} \begin{bmatrix} 0.042 \\ 0.046 \\ 0.194 \\ 0.222 \\ 0 \\ 0 \end{bmatrix} \ldots$$

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

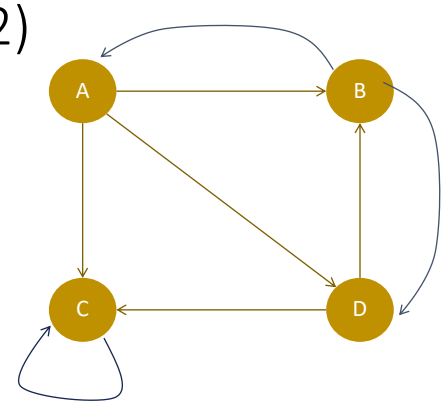
$$v' = \beta M v + (1-\beta)e/n$$

- 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 $\beta=0.8$

$$M=\begin{bmatrix} 1/3 & 1/2 & 1/2 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 0 \end{bmatrix}$$

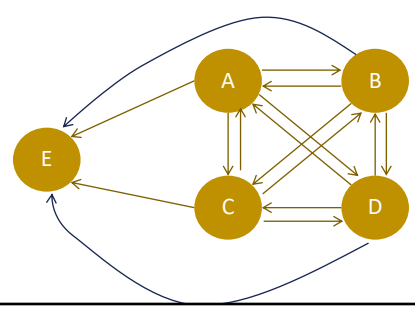
$$v_0=\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\beta\, M\, v_0 + (1-\beta)e/n$$

$$= 0.8 \times \begin{bmatrix} ⅓ & ½ & ½ \\ ⅓ & 0 & ½ \\ ⅓ & ½ & 0 \end{bmatrix} \times \begin{bmatrix} ⅓ \\ ⅓ \\ ⅓ \end{bmatrix} \quad + \quad (1-0.8)e/3$$

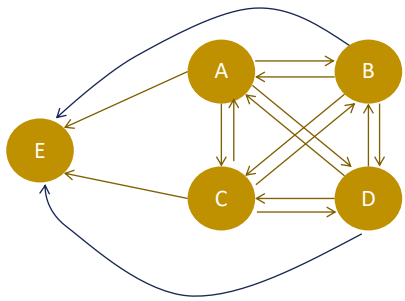
Example 2

- ***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



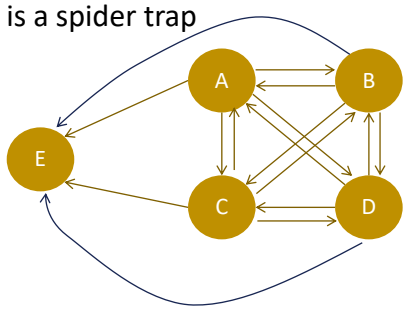
Example 2

- Determine the PageRank of each page assuming $\beta=0.8$
- Is there a dead end?
- Is there a spider trap?
- Is there a clique?

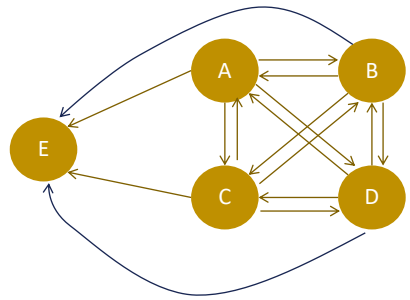


Example 2

- 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



$$M=\begin{bmatrix} 0 & ¼ & ¼ & ¼ & 0 \\ ¼ & 0 & ¼ & ¼ & 0 \\ ¼ & ¼ & 0 & ¼ & 0 \\ ¼ & ¼ & ¼ & 0 & 0 \\ ¼ & ¼ & ¼ & ¼ & 0 \end{bmatrix}$$

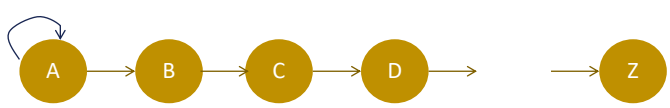
$$v_0=\begin{bmatrix} ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \end{bmatrix}$$

$$\beta M v_0 + (1-\beta)e/n$$

$$= 0.8 \times \begin{bmatrix} 0 & ¼ & ¼ & ¼ & 0 \\ ¼ & 0 & ¼ & ¼ & 0 \\ ¼ & ¼ & 0 & ¼ & 0 \\ ¼ & ¼ & ¼ & 0 & 0 \\ ¼ & ¼ & ¼ & ¼ & 0 \end{bmatrix} \begin{bmatrix} ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \\ ⅕ \end{bmatrix} + (1-0.8) \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} /5$$

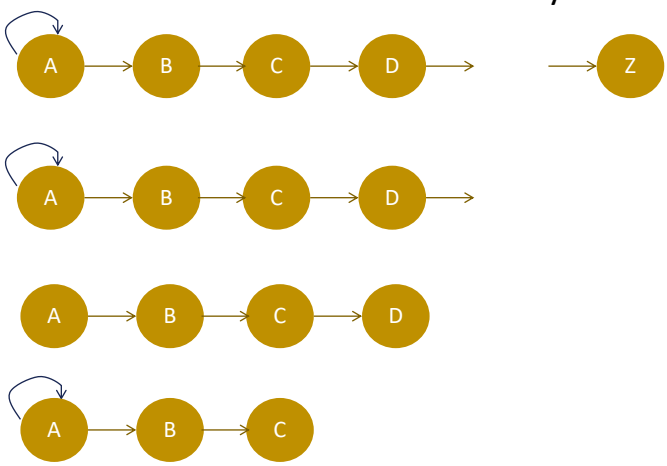
Example 3

- 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?



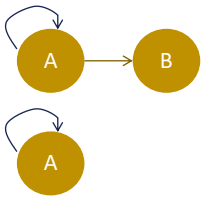
Example 3

- Remove all of the dead ends recursively



Example 3

- Remove all of the dead ends recursively

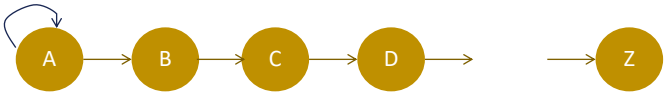


- What is v_0 and M?

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?