CS4225/CS5425 Big Data Systems for Data Science

Assignment 1: Introduction and Hadoop

Chengxi Xue/Bingsheng He
School of Computing
National University of Singapore
xuechengxi@u.nus.edu

School of Computing

Coding Assignment Guideline for CS4225&5425: Assignment 1

- 1.Guide for installation and Configuration
- 2.A Warm-Up Example
- 3. Tasks 1&2
- 4. Information about assignment 1:
  - Submission requirement
  - What is this coding assignment about (you need to implement them into Hadoop)

Guide for Installation and Configuration

- You can choose “VirtualBox” or docker to build a Hadoop/Spark clusters
  - We recommend Docker, since it is more lightweight to build a cluster than using virtual machines.
  - **You are not requested to use Docker**, but I think it is a good solution.
- The manual provided (Installation&Configuration.docx) is for your reference.
  - It may not cover all the details and you may find some problems in your environments. → find solutions online.

Example learning—WordCount

- **WordCount** is a famous example for Hadoop.
- You should run **WordCount** example and learn:
  - 1.How to build a Hadoop project.
  - 2.The simple structure for a Hadoop application
  - 3.How to read multiple input files in MapReduce
- Notes:
  - You need to be familiar with the system.
  - Run the **WordCount** (WordCount.java) to make sure that your Hadoop setup is correct.

Task Overview

- Motivation
  - Text and documents are big data.
  - Text and document processing is fundamental for many Web applications.
- In this coding assignment, you need to implement two tasks with Hadoop.
  - Task 1: Given TWO textual files, count the number of words that are common.
  - Task 2: Recommendation System
  - Detailed guideline and specification are given in the later slides.

Task1:

- Motivation
  - We choose **CommonWords** as our first task because it is representative and it is based on **WordCount**. After reading the **WordCount** example, I think all of you can complete this task easily.

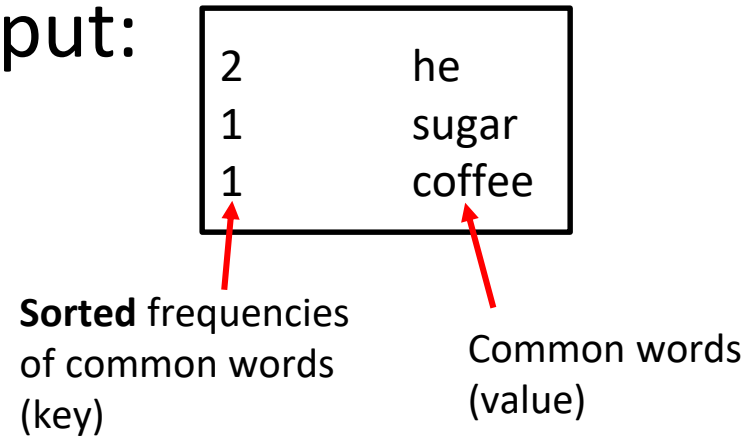
- Problem
  - Given TWO textual files, count the number of words that are common
- Goals
  - You should learn how to
    - write programs that involve multiple stages
    - perform a task that combine information from two files
- Scenarios to consider
  - Remove stop-words like “a”, “the”, “that”, “of”, …
  - Sort the output in descending order of number of common words

- Input data
  - Use the stop-word file:
    - Stopwords.txt
  - Use the following two files:
    - Task1-input1.txt
    - Task1-input2.txt
- Output
  - Wordcount for two input files
  - Top-15 output of the result using the data files listed above (you only need to extract these 15 output from the sorted output)

Running Example

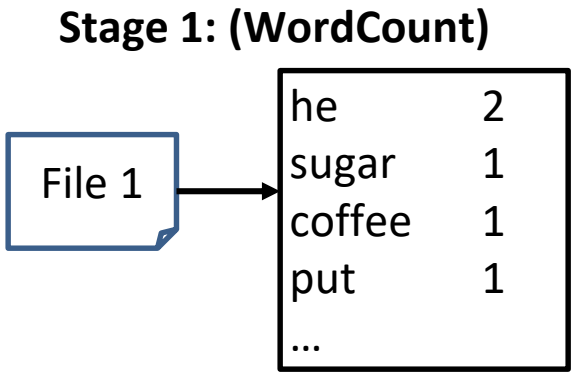
- File 1
  - He put some sugar into his coffee, as he always did.
- File 2
  - He had sugar in his coffee, though he is diabetic.

- Output:



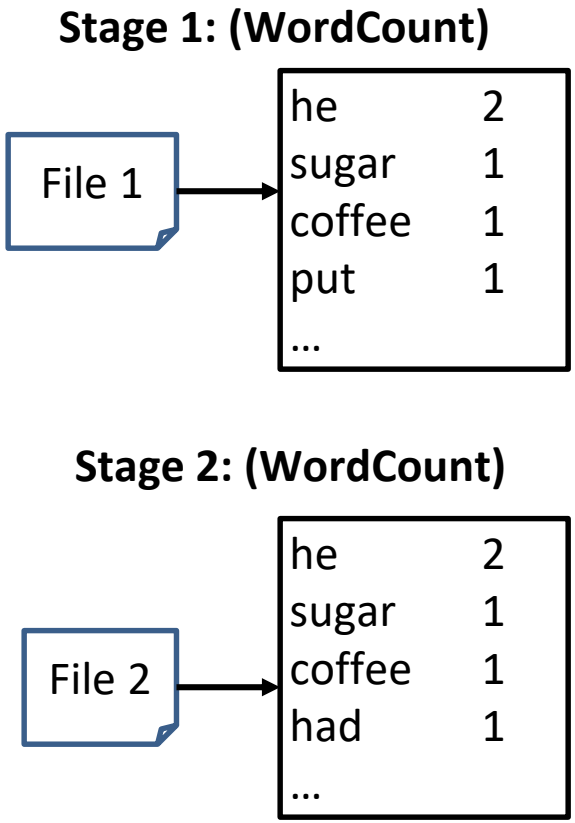
Naïve Solution

- 4 MapReduce stages



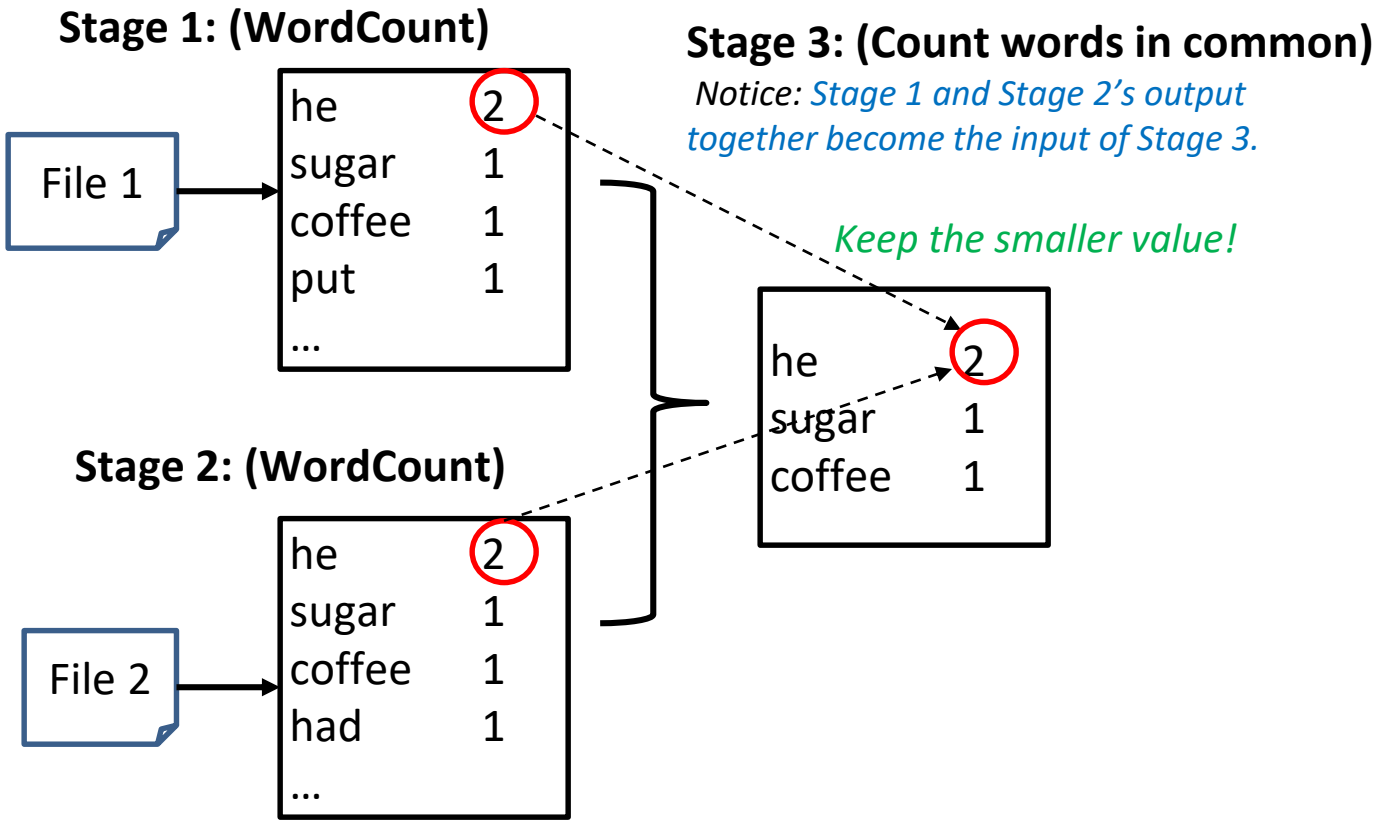
Naïve Solution

- 4 MapReduce stages



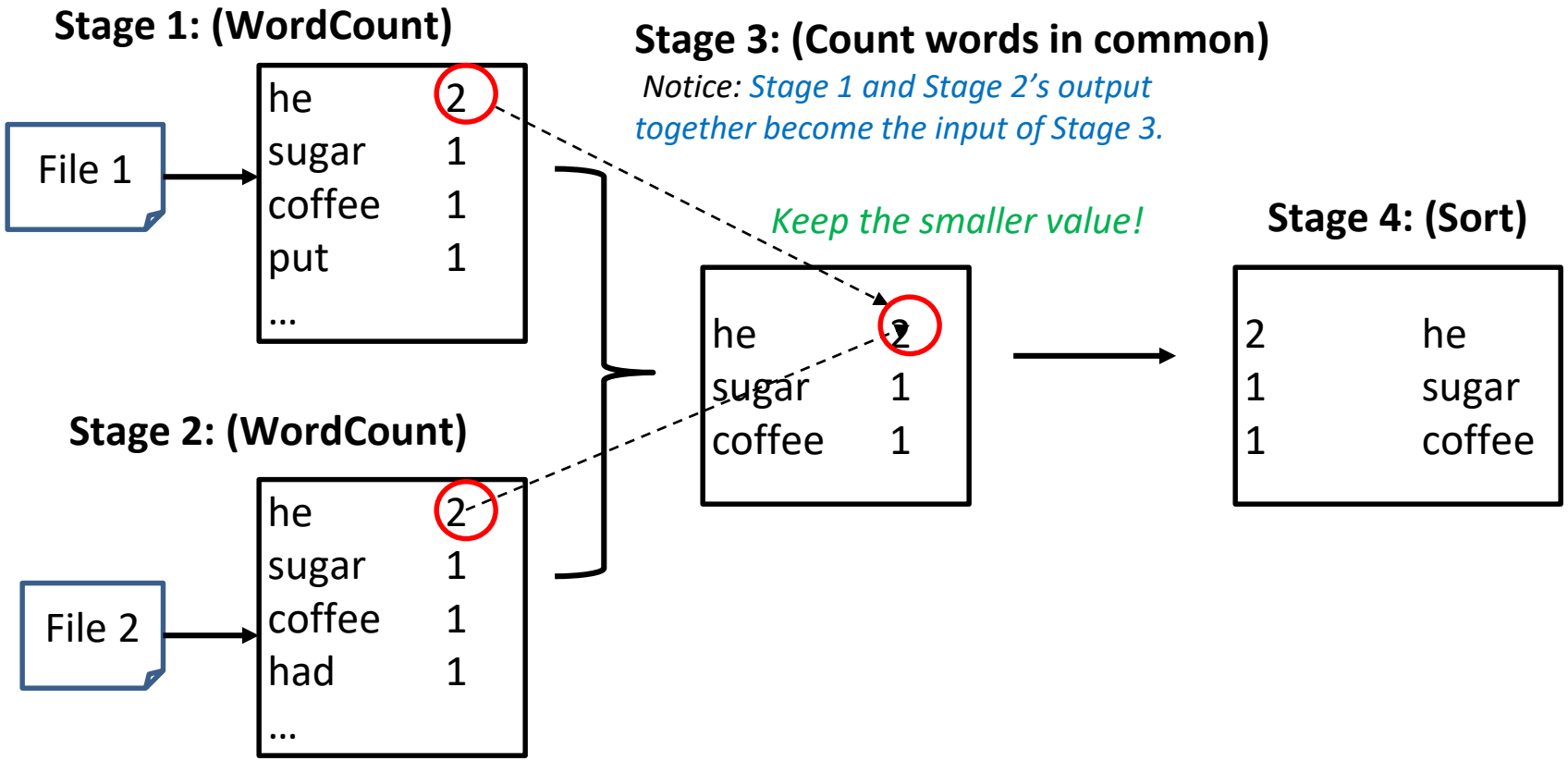
Naïve Solution

- 4 MapReduce stages

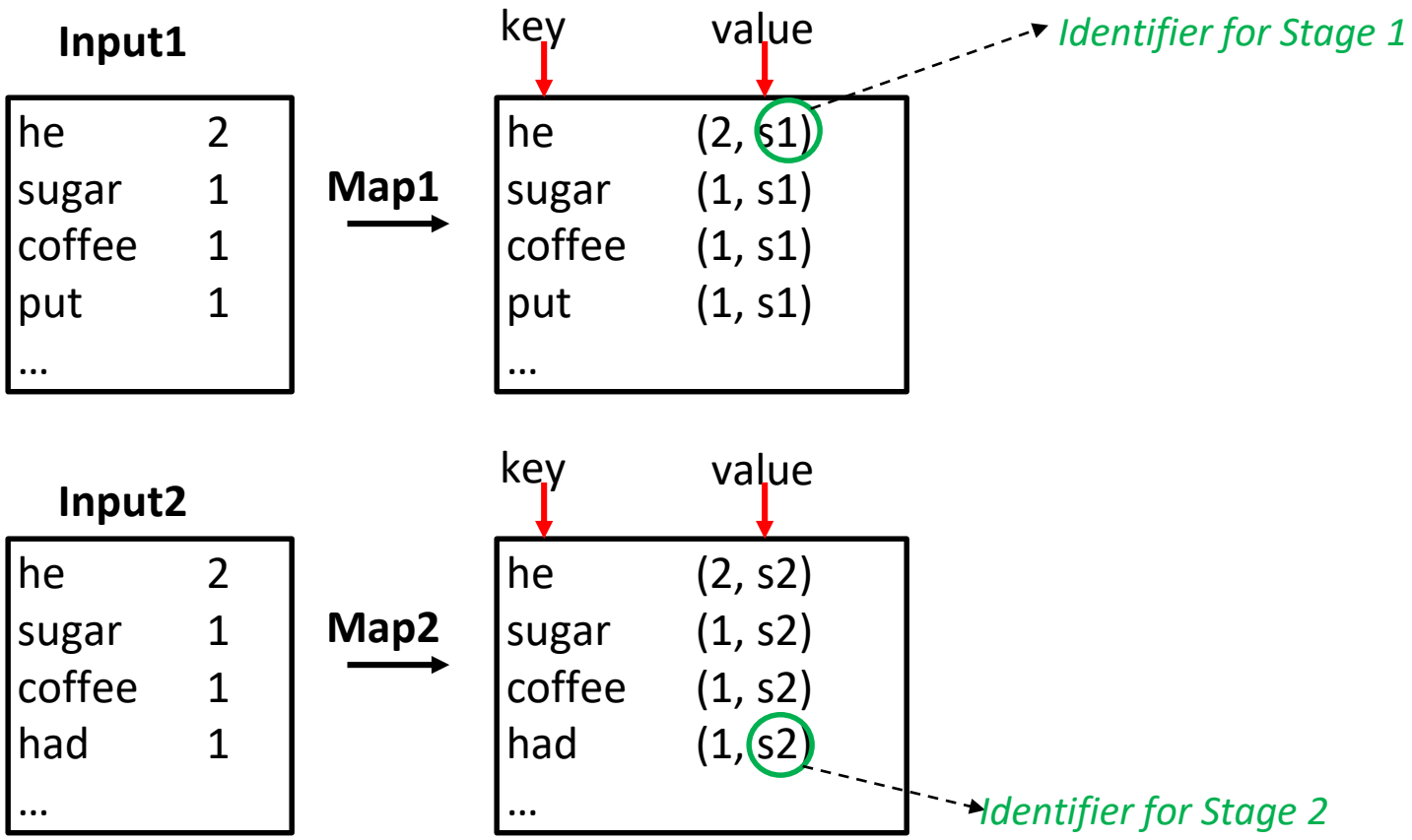


Naïve Solution

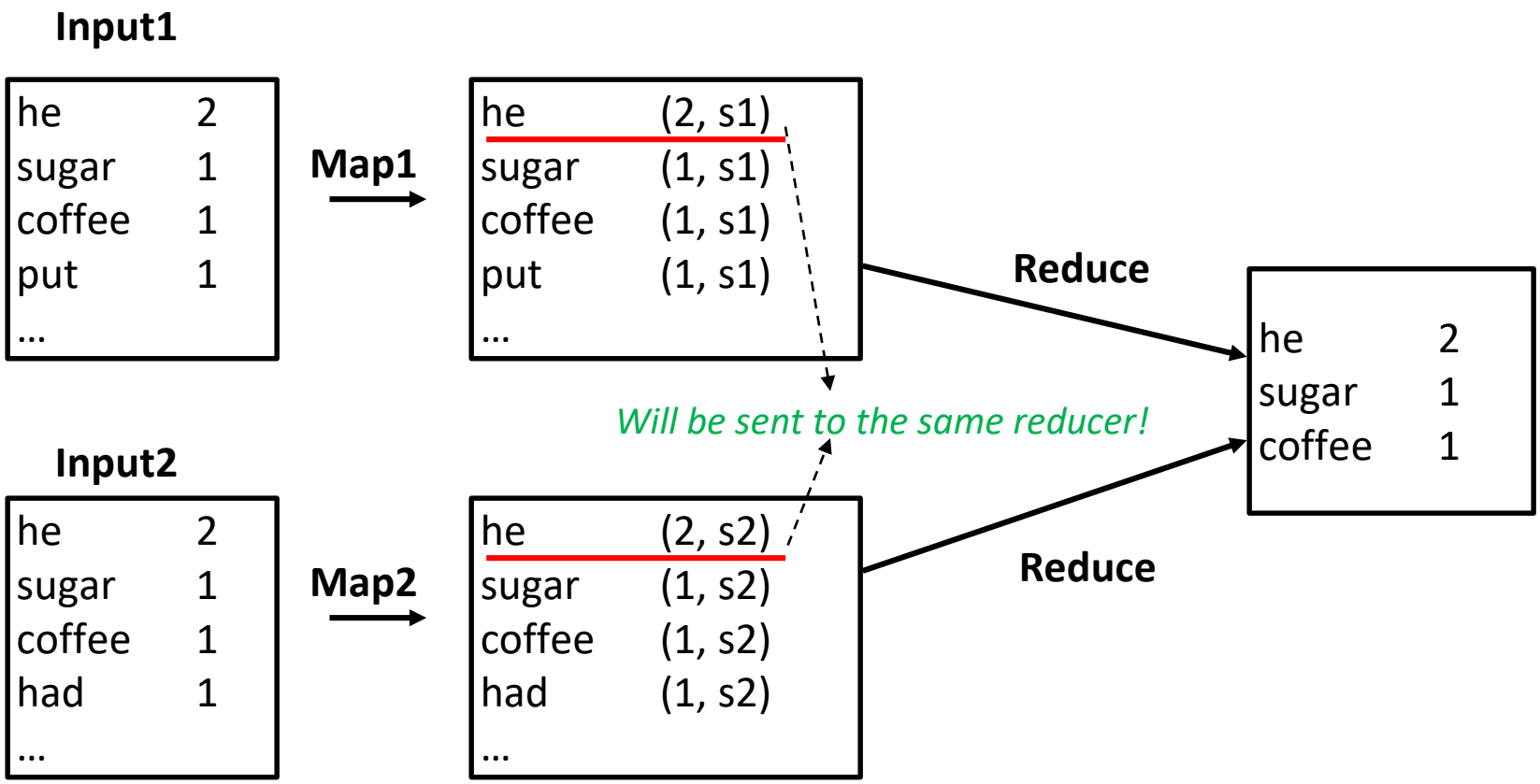
- 4 MapReduce stages



Zoom in Stage 3



Zoom in Stage 3



Implementation for Stage 3

- Input:
  - Stage 1 output files
  - Stage 2 output files

- Problem:
  - We have two different input paths.
  - We have two different map functions.

Implementation for Stage 3

- Usage of MultipleInputs

```
//in the beginning
*Import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;

// in main function, add the following sentences:
MultipleInputs.addInputPath(job, inputPath1,
                    inputFormatClass1, mapper1.class);

MultipleInputs.addInputPath(job, inputPath2,
                    inputFormatClass2, mapper2.class);
```

Input path 1

Input format of files in path 1

Map function for files in path 1

Implementation for Stage 3

- Define two types of mappers

```
//Mapper 1: (deal with word counts of file 1)
public static class Mapper1 extends Mapper<Object, Text, Text, Text>{
   public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            //read one line, parse into  (word, frequency) pair
            //output (word, frequency_s1)
   }
}

//Mapper 2: (deal with word counts of file2)
public static class Mapper2 extends Mapper<Object, Text, Text, Text>{
   public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            //read one line, parse into  (word, frequency) pair
            //output (word, frequency_s2)
   }
}
```

Implementation for Stage 3

- Define one reducer function

```
//Reducer: (get the number of common words)
public static class Reducer1 extends Reducer< Text, Text, Text, IntWritable>{
   public void reduce(Text key, Iterable<Text> value, Context context) throws
            IOException, InterruptedException {

            //parse each value (e.g., n1_s1), get frequency (n1) and stage identifier (s1)
            //if the key has two values, output (key, samller_frequency)
            //if the key has only one value, output nothing

   }
}
```

Overall Implementation

- Put all the codes into one file

```
//in the beginning, import necessary libraries
import …;

//define all the mapper classes and reducer classes
public static class WCMapper…
public static class Mapper1…
public static class Mapper2…
public static class SortMapper…
public static class Reducer1…
……

//Main function
//for Stage 1:
(1) new a job (job1),
(2) set job1 information (e.g., input/output path, etc.)
(3) job1.waitForCompletion(true)
//For stage 2
……(some procedure)
//for Stage 3
……(some procedure)
//for Stage 4
……(some procedure)
```

Remove Stopwords

- Put stop word file into HDFS
  - e.g. hadoop fs -put stw_file_path hdfs_dir

- In the beginning of WordCount.java
  - Add the following libraries

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashSet;
import java.util.Set;
```

Remove Stopwords

- In mapper, add setup function to load stopwords file from HDFS and parse contents into a set of words

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
  Set<String> stopwords = new HashSet<String>();
  @Override
  protected void setup(Context context){
    Configuration conf = context.getConfiguration();
    try {
     Path path = new Path("path.stopwords");
     FileSystem fs= FileSystem.get(new Configuration());
     BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(path)));
     String word = null;
     while ((word= br.readLine())!= null) {
     stopwords.add(word);
     }
    } catch (IOException e) {
     e.printStackTrace();
    }
  }
```

store stopwords

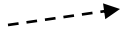
Replace path.stopwords with the real stopword file path in HDFS

-Read contents from the file
-Parse each line to get a stopword.
-Keep all the words into stopwords set

Remove Stopwords

- Modify mapper function to filter stopwords

```
public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      if(stopwords.contains(word.toString()))
       continue;
      context.write(word, one);
    }
  }
}
```



Ignore the word if it is contained in stopwords set

Task2:

- In this project, we will build a **Recommendation System** on Item Collaborative Filtering using Hadoop MapReduce.

Collaborative Filtering (CF)

- Collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple data sources.

- Motivation:
  - CF comes from the idea that the people often get the best recommendations from someone with tests similar to themselves.
  - CF encompasses techniques for matching people with similar interests and making recommendations on this basis.

CF Types: Memory Based and Model Based

- Memory based:
  - User-based: uses a similarity-based vector model to identify the k most similar users to an active user.
  - Item-based: based on the similarity between itsms calculated using people's ratings of those items.

- Model based:
  - Models are developed using different data mining, machine learning algorithms to predict users' rating of unrated items. E.g. Bayesian networks, Clustering models.

CF: Item-Based

- Item-based techniques have two major parts:
  - **1)** analyze the user-item matrix to identify relationships between different items,
  - **2)** use these relationships to indirectly compute recommendations for uses.
- So we need to implement two parts:
  - PART 1: Compute the similarities between items.
  - PART 2: Predict the recommendation scores for every user.

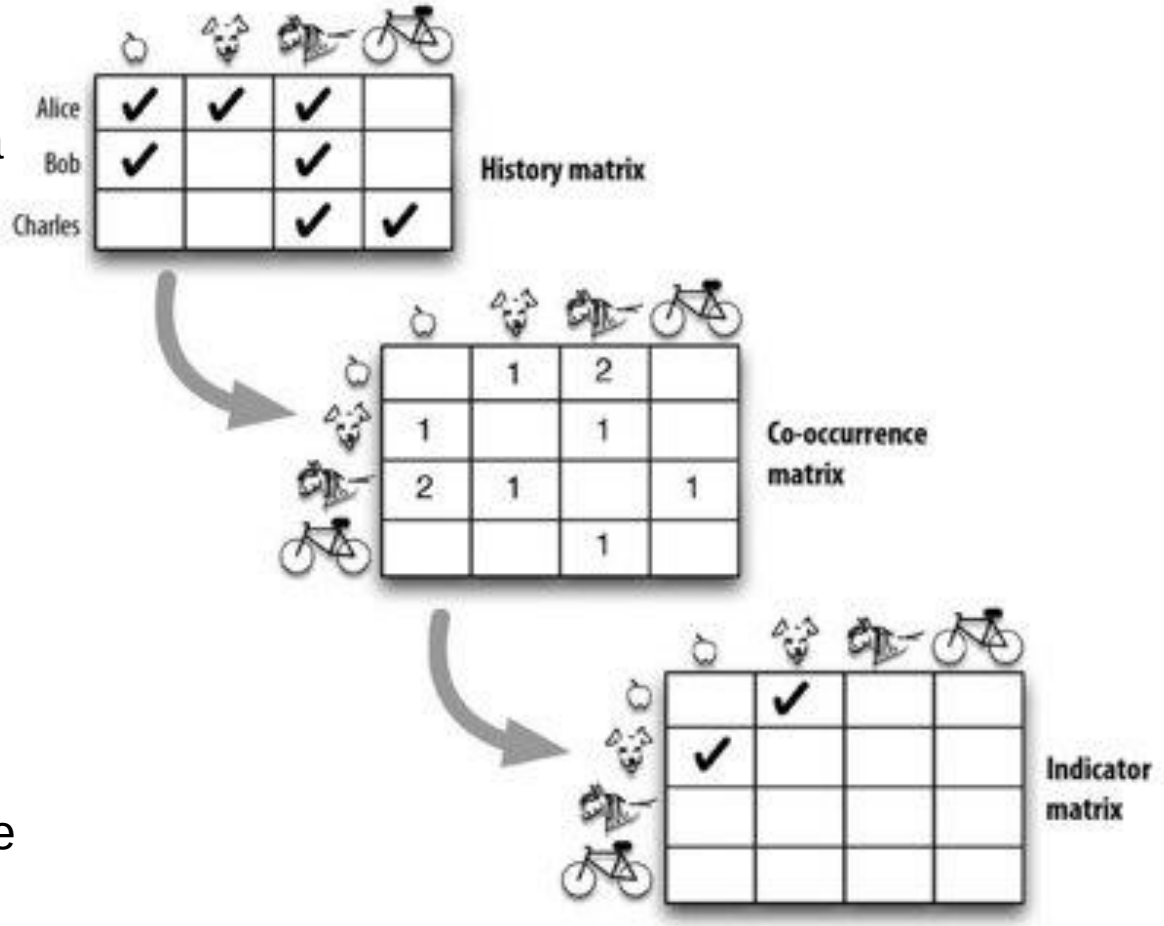
PART 1

We can build a co-occurrence matrix baed on items to represent the similarities, based on the below three steps:

1) Build history matrix
Records the interactions between users and items as a user-by-item matrix

2) Build co-occurrence matrix
An item-by-item matrix, recording which items appeared together in user histories

3) Build indicator matrix
Retains only the anomalous co-occurrences that will be the clues for recommendation



PART 2

- We have the input data like Figure 1.
- Figure 2 shows the meaning,
- Figure 3 is a score matrix from our data for a specific user.

```
1  1,1,0.5
2  1,2,4.5
3  1,3,4.5
4  1,4,2.0
5  1,5,2.0
6  1,6,1.5
7  1,7,1.5
8  1,8,1.5
9  1,9,4.5
10 2,1,3.5
11 2,4,3.5
12 3,8,1.0
13 4,1,4.5
14 4,3,4.0
15 4,9,2.0
16 5,4,0.0
17 5,5,1.0
18 5,7,0.5
```

Figure 1

USER ID	ITEM ID	SCORE
1	1	0.5
1	2	4.5
1	3	4.5
1	4	2.0
1	5	2.0
1	6	1.5
1	7	1.5

.
.
.

Figure 2

	USER 1
ITEM1	0.5
ITEM2	4.5
ITEM3	4.5
ITEM4	2.0

.
.
.

Figure 3

PART 2

- We get the score by using a matrix product of co-occurrence and score matrix.
- Co-occurrence is like similarity: the more two items occur together, the more they are probably related.

CO-OCCURRENC MATRIX

	100	101	102
100	3	3	1
101	3	3	1
102	1	1	1

SCORE MATRIX

	USER 4
100	3.0
101	0.0
102	0.0

RESULT

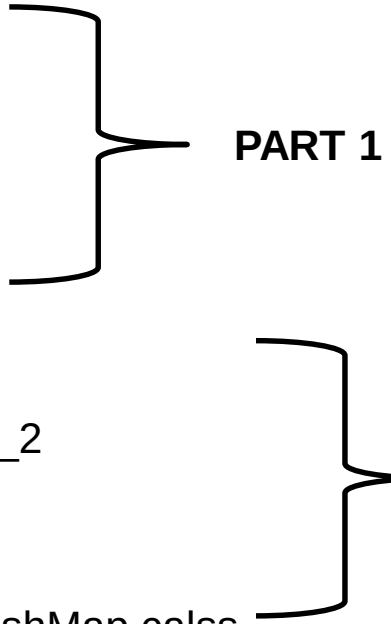
R
9
9
3

100,101,102 are ITEM ID

A higher R value means better recommendation.

Structure of The Project

- We can implement CF based on the below structure.
  - Recommend.java -- main function
  - Step1.java:get score matrix, group the users.
  - Step2.java:build co-occurrence matrix.
  - Step3.java:Further processing for matrices
  - Step4_1.java:First part of Matrix multiplication
  - Step4_2.java:Second part of Matrix multiplication_2
  - Step5.java filtering and sorting
  - HDFSAPI:DAO for HDFS; SortHashMap.java: HashMap calss



You can see the details in sample code (Task2_code), you need to follow this structure and fill all the part.

Submission requirement

- Task1:
- Input data (in Task1-data)
  - Use the following two files:task1-input1.txt & task1-input2.txt
  - Stop words are provided: Stopwords.txt

- Task2:
- Input data (in Task2-data)
  - Use data.zip

Submission requirement

- Deadline: Feb 23, 2019 11:59pm
- Submit the following:
  - Your whole project included in the MapReduce program (not just .java file)
  - Task1: Top-15 output of the result using the data files listed above.
  - Task2: The recommendation scores for the user whose ID is same as the last three number of your student ID.(E0204123)→ User ID:123

Submission requirement

- Submission
  - Report: A simple description (1-2 pages pdf) about your code (If you did not follow the stages described in the slides, then you should describe your own scheme)
  - Code: Make sure your code is self-contained, and please submit a simple README to explain how to run your project using HDFS.

- Files should be compressed in a zip file to IVLE, with the name [Your Student ID]-Assignment1.zip

Marking Schemes

- Total: 8% of final mark.
  - Task1 Code & Report: 3%
  - Task2 Code & Report: 3%
  - Writing assessment: 2%
    - The written assessment's questions depend on your submission. You need to understand your code. For example, please explain some specific lines of your code.
    - The written assessment will be conducted in tutorial session.
    - Time: Tutorial Week 7 ("Buffer Week").

Notice

- Please don't consider this homework as the same as ACM-ICPC programming contest (check by exact input-output pairs), we use this to enhance your understanding about the programming using Hadoop
- Don't need to worry about whether your result "exactly matches" final result.

Feedbacks are Welcome

- Email me: xuechengxi@u.nus.edu
- Or, post your questions in the IVLE forum (preferred).