FAQs

• How does Hadoop mapreduce run the map instance?
• Why do we need to evaluate large data using bloom filter?
• Does mapper have to generate at least 1 key/value output pair?
• How do we calculate Z-score?
• No make-up quiz or exam
  • If you are employed or have any emergency
    • Provide a document specifying the duration and purpose clearly
      • e.g. doctor’s note, manager’s email, your coach’s letter.

Topics covered in this lecture

• MapReduce Design Pattern II. Filtering Patterns (continued)
• MapReduce Design Pattern III. Data Organization Patterns

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
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

Data Organization Patterns

• Reorganizing data
  • Partitioning, Sharding and Sorting

1. “Structured” to “hierarchical” pattern
2. Partitioning and binning patterns
3. Total order sorting
Reducer Code

public static class PostCommentHierarchyReducer extends Reducer<Text, Text, Text, Text> {
    private String post = null;
    private Text postWithCommentChildren = new Text();
    private String PostId = null;

    public void reduce(Text key, Text value, Context context) throws IOException, InterruptedException {
        if (key.toString().charAt(0) == 'P') {
            post = key.toString().substring(1, key.toString().length()).trim();
        }
    }
}

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[1]), TextInputFormat.class, CommentMapper.class);
    MultipleInputs.addInputPath(job, new Path(args[2]), TextInputFormat.class, PostMapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setOutputFormatClass(TextOutputFormat.class);
    TextOutputFormat.setOutputPath(job, new Path(args[3]));

    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 = null;

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String s = value.toString();
        String[] arr = s.split("\</\p\>");
        for (String t : arr) {
            if (t.charAt(0) == 'P') {
                context.write(t.toString(), new Text());
            }
        }
    }
}

Mapper Code (for Comments)

public static class CommentMapper extends Mapper<Object, Text, Text, Text> {
    private Text outkey = new Text();
    private Text outvalue = null;

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        PostCommentHierarchyReducer.reducer = new PostCommentHierarchyReducer();
        String s = value.toString();
        String[] arr = s.split("\</\p\>");
        for (String t : arr) {
            if (t.charAt(0) == 'P') {
                context.write(t.toString(), new Text());
            }
        }
    }
}
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

Example

- Task
  - Given a set of user information (with the last access date per user), partition the records based on the year of last access date, one partition per year
- Data

Partitioner

- 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 in different groups (e.g. gender, or age group)

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>James</td>
<td>45</td>
<td>Male</td>
<td>50000</td>
</tr>
<tr>
<td>2001</td>
<td>James</td>
<td>45</td>
<td>Male</td>
<td>51000</td>
</tr>
<tr>
<td>2002</td>
<td>Kevin</td>
<td>34</td>
<td>Male</td>
<td>45000</td>
</tr>
<tr>
<td>2003</td>
<td>Maria</td>
<td>30</td>
<td>Female</td>
<td>43000</td>
</tr>
<tr>
<td>2004</td>
<td>Julia</td>
<td>25</td>
<td>Female</td>
<td>23000</td>
</tr>
<tr>
<td>2005</td>
<td>Lavanya</td>
<td>20</td>
<td>Male</td>
<td>42000</td>
</tr>
<tr>
<td>2006</td>
<td>Joseph</td>
<td>19</td>
<td>Male</td>
<td>19000</td>
</tr>
<tr>
<td>2007</td>
<td>Steve</td>
<td>22</td>
<td>Male</td>
<td>32000</td>
</tr>
<tr>
<td>2008</td>
<td>Max</td>
<td>24</td>
<td>Male</td>
<td>28000</td>
</tr>
<tr>
<td>2009</td>
<td>Jenifer</td>
<td>28</td>
<td>Female</td>
<td>24000</td>
</tr>
<tr>
<td>2010</td>
<td>Nick</td>
<td>33</td>
<td>Male</td>
<td>39000</td>
</tr>
<tr>
<td>2011</td>
<td>MaryAnn</td>
<td>18</td>
<td>Female</td>
<td>56000</td>
</tr>
<tr>
<td>2012</td>
<td>Kelvin</td>
<td>39</td>
<td>Male</td>
<td>65000</td>
</tr>
</tbody>
</table>

Partitioner

- Map Tasks
  - Input
    - (Dummy_key, *1201 t James t 45 t male t 50000")
  - Method
    - Read the value and extract gender information
      ```java
      String[] str = value.toString().split("\t", -3);
      String gender = str[3];
      context.write(new Text(gender), value);
      ```
  - Output
    - Gender data and value
      ```java
      context.write(new Text(gender), value);
      ```
Partitioner [3/5]

- 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
  
```java
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 [4/5]

- How many partitions will be generated?
- What will be the partitions delivered to the reducers?

Example (continued)

- Task
  - Given a set of user information (with the last access date per user), partition the records based on the year of last access date, one partition per year
- Data

Driver Code

```java
// Set custom partitioner and min last access date
job.setPartitionerClass(LastAccessDataPartitioner.class);
LastAccessDataPartitioner.setMinMaxLastAccessData(job, 2008);

// Last access dates span between 2008-2011, or 4 years
job.setNumReduceTasks(4);
```
Mapper Code

```java
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

```java
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);
    }
}
```

Reducer Code

```java
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

- Performed 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
- Uses a sorted list of N-1 sampled keys that define the key range for each reduce
- Dynamic and load balanced

Reducer

- The number of reducers needs to be equal to the total 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
  - 1TB, it took 62 seconds

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+1] are sent to reducer i
  - Total 1,000 lines of Java code
Questions?