FAQs

- PAQ has been posted
  - Feb. 6, 5:00PM via Canvas
  - Individual submission (No team submission)
  - If you have not been assigned the "port range", please contact the GTA immediately

Topics

- MapReduce Design Pattern II. Filtering Patterns
- 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
Mapper Code

```java
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

```java
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());
    }
}
```

Combiner

• How can you improve the performance of the previous MapReduce software using a Combiner?

Data Organization Patterns

• Reorganizing data
  • Partitioning, Sharding and Sorting
  1. “Structured” to “hierarchical” pattern
  2. Partitioning and binning patterns
  3. Total order sorting

Part 1. Large Scale Data Analytics

Design Pattern 3: Data Organization Patterns

MapReduce Design Patterns II: 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

**Driver Code**

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "PostCommentBuildingDriver");
    MultipleInputs.addInputPath(job, new Path(args[0]));
    TextInputFormat.setInputFormat(job);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setReducerClass(CommentMapper.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setJarByClass(PostCommentBuildingDriver.class);
    job.waitForCompletion(true) ? 0 : 2;
}
```

**Mapper Code (for posts)**

```java
public static class PostMapper extends Mapper<Object, Text, Text, Text> {
    private Text post = null;
    private String comment = null;
    private String postid = null;
    private String parentid = null;
    private String value = null;
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Text[] values = new Text[4];
        String[] split_values = value.toString().split('"');
        for (int i = 0; i < split_values.length; i++) {
            String split_value = split_values[i].trim();
            if (split_value.charAt(0) == 'P') { postid = split_value.substring(1, split_value.length()); }
            else if (split_value.charAt(0) == 'C') { parentid = split_value.substring(1, split_value.length()); }
            else if (split_value.charAt(0) == 'T') { post = split_value.substring(1, split_value.length()); }
            else if (split_value.charAt(0) == 'M') { comment = split_value.substring(1, split_value.length()); }
        }
        context.write((new Text(postid)), value.toString());
        context.write((new Text(parentid)), value.toString());
    }
}
```

**Mapper Code (for comments)**

```java
public static class CommentMapper extends Mapper<Object, Text, Text, Text> {
    private Text comment = null;
    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").trim());
        // Flag this record for the reducer and then output
        outvalue.set("P" + value.toString());
        context.write(outkey, outvalue);
    }
}
```

**Reducer Code**

```java
public static class PostCommentHierarchyReducer extends Reducer<Text, Text, Text, Text> {
    private String value = 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.toString().equals("P")) { post = t.toString().substring(1).trim(); }
        }
        // For each comment record
        for (Text c : values) {
            // If this is the comment record, append the flag
            if (c.toString().equals("C")) { comments.add(c.toString()); }
        }
        // If there are comments
        if (!comments.isEmpty()) {
            // Add the comment(s) to the result
            context.write((new Text(post)), comments.toString());
        }
    }
}
```

**Structure of the structured to hierarchical pattern**

- Dataset A
  - Input
  - Output
- Dataset B
  - Input
  - Output
- Dataset C
  - Input
  - Output

- Structure of the structured to hierarchical pattern

  - Input Data
  - Processing
  - Output Data
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());
    }
    }

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.

MapReduce Design Patterns III: Data Organization Patterns

2. Partitioning pattern

Structure of the partitioning pattern

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 by
    gender in different age groups

Partitioner
- Map Tasks
  - Input
    - (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 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
  - Method
    - Read the age field and apply conditions
  - Output
    - The data of key-value pairs are segmented into three collections of key-value pairs

```java
int age = Integer.parseInt(str);  
if (age >= 20) {  
  return 0;  
} else if (age >= 20 && age <= 30) {  
  return 1;  
} else {  
  return 2 % numReduceTasks;  
}
```

Driver Code (Highest salaried employee example)

```java
public static class LastAccessDatePartitioner extends Partitioner {
    private static final String MIN_LAST_ACCESS_DATE_YEAR = 
    "min.last.access.date.year";
    private int minLastAccessDateYear = 0;

    public int getPartition(ImmutableKey key, Text value, int numPartitions) {
        int keyInt = key.getInt(MIN_LAST_ACCESS_DATE_YEAR);  
        int valueInt = value.getInt(MIN_LAST_ACCESS_DATE_YEAR);  
        int result = keyInt % numPartitions;
        return result;
    }
}
```

Partitioner code (Highest salaried employee example)

```java
public static class LastAccessDatePartitioner extends Partitioner < ImmutableKey, Text > {  
    private static final String MIN_LAST_ACCESS_DATE_YEAR = 
    "min.last.access.date.year";
    private int minLastAccessDateYear = 0;

    public int getPartition(ImmutableKey key, Text value, int numPartitions) {  
        int keyInt = key.getInt(MIN_LAST_ACCESS_DATE_YEAR);  
        int valueInt = value.getInt(MIN_LAST_ACCESS_DATE_YEAR);  
        int result = keyInt % numPartitions;
        return result;
    }
    
    public int setMinLastAccessDateYear(int year) {  
        minLastAccessDateYear = year;
        return 0;
    }
    
    public int getMinLastAccessDateYear() {  
        return minLastAccessDateYear;
    }
}
```

Mapper Code (Highest salaried employee example)

```java
public static class LastAccessDateMapper extends Mapper < Object, Text, IntWritable, IntWritable > {  
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  
        // Parse the input key and value into a Calendar object
        Calendar cal = Calendar.getInstance();  
        cal.set(Integer.parseInt(value.toString().split("\t")[0]), 
            Integer.parseInt(value.toString().split("\t")[1]));  
        // Get the last access date
        Calendar lastAccessDate = cal;  
        // Compare the last access date with the minimum access date
        int minLastAccessDateYear = lastAccessDate.get(Calendar.YEAR);  
        // Set the partition number based on the difference in years
        int partition = (lastAccessDate.get(Calendar.YEAR) - 
            minLastAccessDateYear) % numReduceTasks;
        context.write(key, new IntWritable(partition));
    }
}
```

Partitioner code (Highest salaried employee example)

```java
public static class LastAccessDatePartitioner extends Partitioner < ImmutableKey, Text > {
    private static final String MIN_LAST_ACCESS_DATE_YEAR = 
    "min.last.access.date.year";
    private int minLastAccessDateYear = 0;

    public int getPartition(ImmutableKey key, Text value, int numPartitions) {  
        int keyInt = key.getInt(MIN_LAST_ACCESS_DATE_YEAR);  
        int valueInt = value.getInt(MIN_LAST_ACCESS_DATE_YEAR);  
        int result = keyInt % numPartitions;
        return result;
    }
    
    public int setMinLastAccessDateYear(int year) {  
        minLastAccessDateYear = year;
        return 0;
    }
    
    public int getMinLastAccessDateYear() {  
        return minLastAccessDateYear;
    }
}
```
Reducer Code (Highest salaried employee example)

```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
- Highest salaried employee example

- Observation
  - Recent years will have more users
  - Provide finer grained segmentations to the recent years
  - e.g. Monthly partitions for recent 3 years

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

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] <= key < sample[i+1] are sent to reducer i
  - Total 1,000 lines of java code

Questions?