PART 1. LARGE SCALE DATA ANALYTICS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

• Q2 (MapReduce)
  • Your score has been posted in canvas
  • combiner

• How to improve your MapReduce programing skills
  • Programming assignment
  • https://github.com/adamjshook/mapreducepatterns
  • Download, read, and run the code!
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**

---

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)
  \[\rightarrow (1950, 20)\]

- Second map produces
  - (1950, 25)
  - (1950, 15)
  \[\rightarrow (1950, 25)\]

- Input to the reduce function
  - (1950, [0, 20, 10, 25, 15])
  \[\rightarrow (1950, [20, 25])\]

- Output
  - (1950, 25) \[\rightarrow (1950, 25)\]
Topics

- MapReduce Design Pattern III. Data Organization Patterns
- MapReduce Design Pattern IV. Join Patterns

Part 1. Large Scale Data Analytics
Design Pattern 3: Data Organization Patterns
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1201</td>
<td>James</td>
<td>45</td>
<td>Male</td>
</tr>
<tr>
<td>1202</td>
<td>Matthew</td>
<td>40</td>
<td>Male</td>
</tr>
<tr>
<td>1203</td>
<td>Kevin</td>
<td>34</td>
<td>Male</td>
</tr>
<tr>
<td>1204</td>
<td>Maria</td>
<td>30</td>
<td>Female</td>
</tr>
<tr>
<td>1205</td>
<td>Julia</td>
<td>20</td>
<td>Female</td>
</tr>
<tr>
<td>1206</td>
<td>Lavanya</td>
<td>25</td>
<td>Female</td>
</tr>
<tr>
<td>1207</td>
<td>Joseph</td>
<td>19</td>
<td>Male</td>
</tr>
<tr>
<td>1208</td>
<td>Steve</td>
<td>22</td>
<td>Male</td>
</tr>
<tr>
<td>1209</td>
<td>Max</td>
<td>24</td>
<td>Male</td>
</tr>
<tr>
<td>1210</td>
<td>Jennifer</td>
<td>28</td>
<td>Female</td>
</tr>
<tr>
<td>1211</td>
<td>Nick</td>
<td>18</td>
<td>Male</td>
</tr>
<tr>
<td>1212</td>
<td>MaryAnn</td>
<td>33</td>
<td>Female</td>
</tr>
<tr>
<td>1213</td>
<td>Kelvin</td>
<td>39</td>
<td>Male</td>
</tr>
</tbody>
</table>

Partitioner

- Map Tasks
  - Input
    - Dummy key, data
    - (Dummy_key, “1201 \t James \t 45 \t male \t 50000”)
  - Functionality
    - Read the value and extract gender information

```java
String[] str = value.toString().split(“\t”, -3);
String gender = str[3];
```

- Output
  - Gender data and value

```java
context.write(new Text(gender), value);
```
Partitioner

- **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*

```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;
}
```

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)

Sort the user activities based on the last access year using a partitioner original data contains the timestamps of the last access (e.g. YYYY:HH:MM:SS)

... 
// 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 fmt =
            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(fmt.parse(strDate));
        outkey.set(cal.get(Calendar.YEAR));
        context.write(outkey, value);
    }
}
Partitioner code (last access date partitioner)

```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);
    }
}
```

Partitioner code (last access date partitioner)

```java
public static void setMinLastAccessDate(Job job, int minLastAccessDateYear) {
    job.getConfiguration()
        .setInt(MIN_LAST_ACCESS_DATE_YEAR, minLastAccessDateYear);
}
```
Reducer Code  (last access date partitioner)

```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

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

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

A Refresher on Joins

- A Join is an operation that combines records from two or more datasets based on a field or set of fields
  - Foreign key
- The foreign key is the field in a relational table that matches the column of another table
  - Used as a means to cross-reference between tables
### Example

<table>
<thead>
<tr>
<th>UserID</th>
<th>Reputation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UserID</th>
<th>PostID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
<tr>
<td>8</td>
<td>48678</td>
<td>HTML is not a subset of XML!</td>
</tr>
</tbody>
</table>

### Inner Join

**Dataset A**

<table>
<thead>
<tr>
<th>UserID</th>
<th>Reputation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
</tr>
</tbody>
</table>

**Dataset B**

<table>
<thead>
<tr>
<th>UserID</th>
<th>PostID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
<tr>
<td>8</td>
<td>48678</td>
<td>HTML is not a subset of XML!</td>
</tr>
</tbody>
</table>

**Default join style**

Records from both A and B that contain **identical values for the given foreign key** are brought together.

<table>
<thead>
<tr>
<th>A.UserID</th>
<th>A.Reputation</th>
<th>A.Location</th>
<th>B.UserID</th>
<th>B.PostID</th>
<th>B.Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted</td>
</tr>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
</tbody>
</table>
Outer Join

<table>
<thead>
<tr>
<th>UserID</th>
<th>Reputation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UserID</th>
<th>PostID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
<tr>
<td>8</td>
<td>48678</td>
<td>HTML is not a subset of XML!</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Reputation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UserID</th>
<th>PostID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
<tr>
<td>8</td>
<td>48678</td>
<td>HTML is not a subset of XML!</td>
</tr>
</tbody>
</table>

A. UserID | A.Reputation | A.Location | B.UserID | B.PostID | B.Text |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted</td>
</tr>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Anti Join

Full outer join minus the inner join

<table>
<thead>
<tr>
<th>User ID</th>
<th>Reputation</th>
<th>Location</th>
<th>User ID</th>
<th>Post ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
<td>3</td>
<td>48002</td>
<td>Hey, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44921</td>
<td>Please see my post below</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>8</td>
<td>48678</td>
<td>HTML is not a subset of XML!</td>
</tr>
</tbody>
</table>

9/16/2019
CS435 Introduction to Big Data – Fall 2019

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

Structure of the reduce side join pattern

- Input Split
- Join Mapper
- Shuffle and sort
- Join Reducer
- Output Part A
- Input Split
- Join Mapper
- (bob, 1092847)
- Join Reducer
- Output Part B
- Input Split
- Join Mapper
- (bob, 3849273)
- Join Reducer
- Output Part A
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

```java
...
// 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

```java
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

```java
public static class CommentJoinMapper 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 = transformXmlToMap(value.toString());
        String userId = parsed.get("UserId");
        // 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

```java
public static class UserJoinReducer extends Reducer<Text, Text, Text, Text> {
    private static final Text EMPTY_TEXT = Text.valueOf("" ander Download wiktionary");
    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();
        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);
}
```

Reducer Code

```java
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);
```
### Inner Join Code

```java
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);
            }
        }
    }
} ...
```

### Left outer Join Code

```java
... 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

```java
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 (many) small data set(s) that can be performed on the map-side

• 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 running*

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
- **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

- **Join (Comment, user, on userID)**

```java
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());
        // 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
- Join (Comment, user, on userID)

```java
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

<table>
<thead>
<tr>
<th>Hash Key</th>
<th>Dataset A</th>
<th>Dataset B</th>
</tr>
</thead>
<tbody>
<tr>
<td>%5 = 0</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>%5 = 1</td>
<td>James</td>
<td>James</td>
</tr>
<tr>
<td>%5 = 2</td>
<td>William</td>
<td>Donald</td>
</tr>
<tr>
<td>%5 = 3</td>
<td>Chris</td>
<td>Chris</td>
</tr>
<tr>
<td>%5 = 4</td>
<td>Frank</td>
<td>Frank</td>
</tr>
</tbody>
</table>
Questions?