CMPE 282 Cloud Services
MapReduce Design Patterns – Data Organization

Instructor: Kong Li

Content

- What and Why
- MapReduce refresher
- Summarization Patterns
- Filtering Patterns
- Data Organization Patterns
- Join Patterns

Data Organization Patterns

What: Reorganize, restructure

Why: I want to change the way my data is organized

- **Structured to hierarchical**
- **Partitioning**
- **Binning**
- Total order sorting
- Shuffling

Structured to Hierarchical 1/5

- Intent
  - Transform row-based data to a hierarchical format (JSON or XML)
- Motivation
  - Migrating data from an RDBMS to Hadoop
  - Reformatting data into a more conducive structure
- Applicability
  - You have data sources that are linked by FKs
  - Your data is structured and row-based

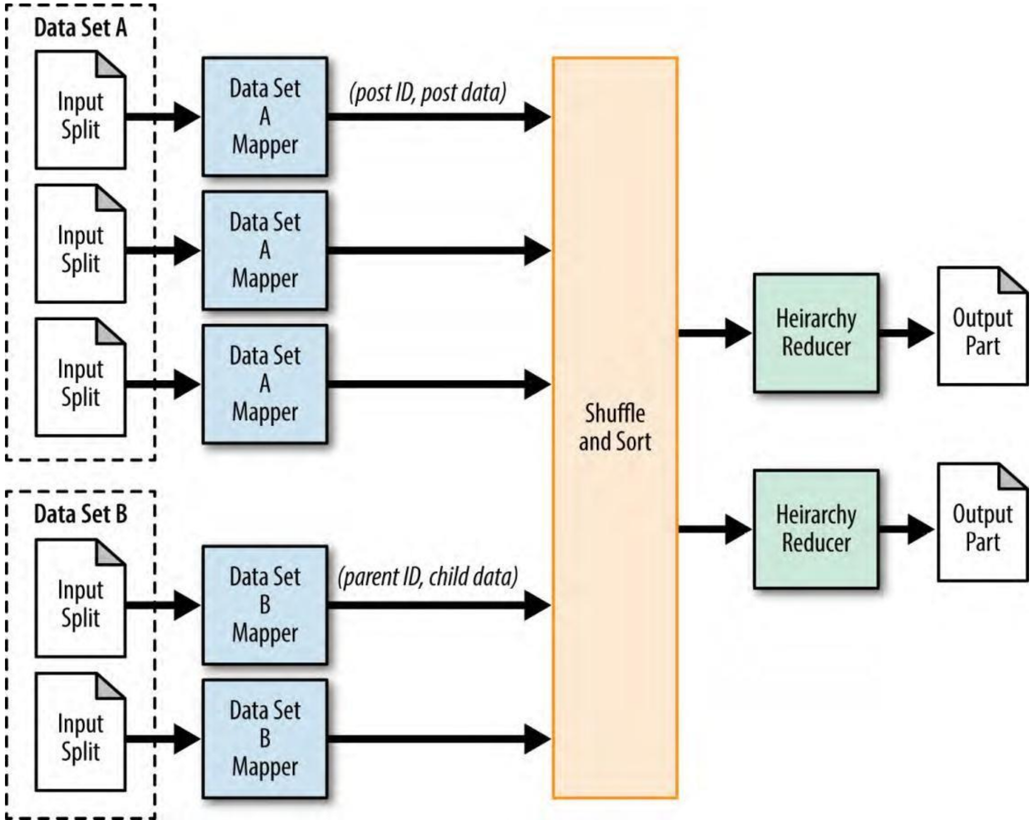
```
Posts
 Post
  Comment
  Comment
 Post
  Comment
  Comment
  Comment
```

Structured to Hierarchical 2/5

- Structure:
  - Mapper: load data and parse records into one cohesive format
  - Combiner: little use
  - Reducer: build the hierarchical data structure from the list of data items

Post

Comment



Posts
  Post (Id)
    Comment (PostId)
    Comment (PostId)
  Post (Id)
    Comment (PostId)
    Comment (PostId)
    Comment (PostId)

Structured to Hierarchical 3/5

- Known uses
  - Pre-joining data
  - Preparing data for HBase or MongoDB
- Performance analysis
  - How much data is being sent from mappers to reducers?
    - Almost all data is moved across network
  - The memory footprint of the obj that the reducer builds
    - For a post that has a million comments?

Structured to Hierarchical 4/5

- PostCommentBuildingDriver.java: Given a list of posts and comments, create corresponding XML hierarchy
- In-1: Posts.xml
- In-2: Comments.xml

```java
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);
}
```

```java
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);
    }
}

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);
    }
}
```

Structured to Hierarchical 5/5

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

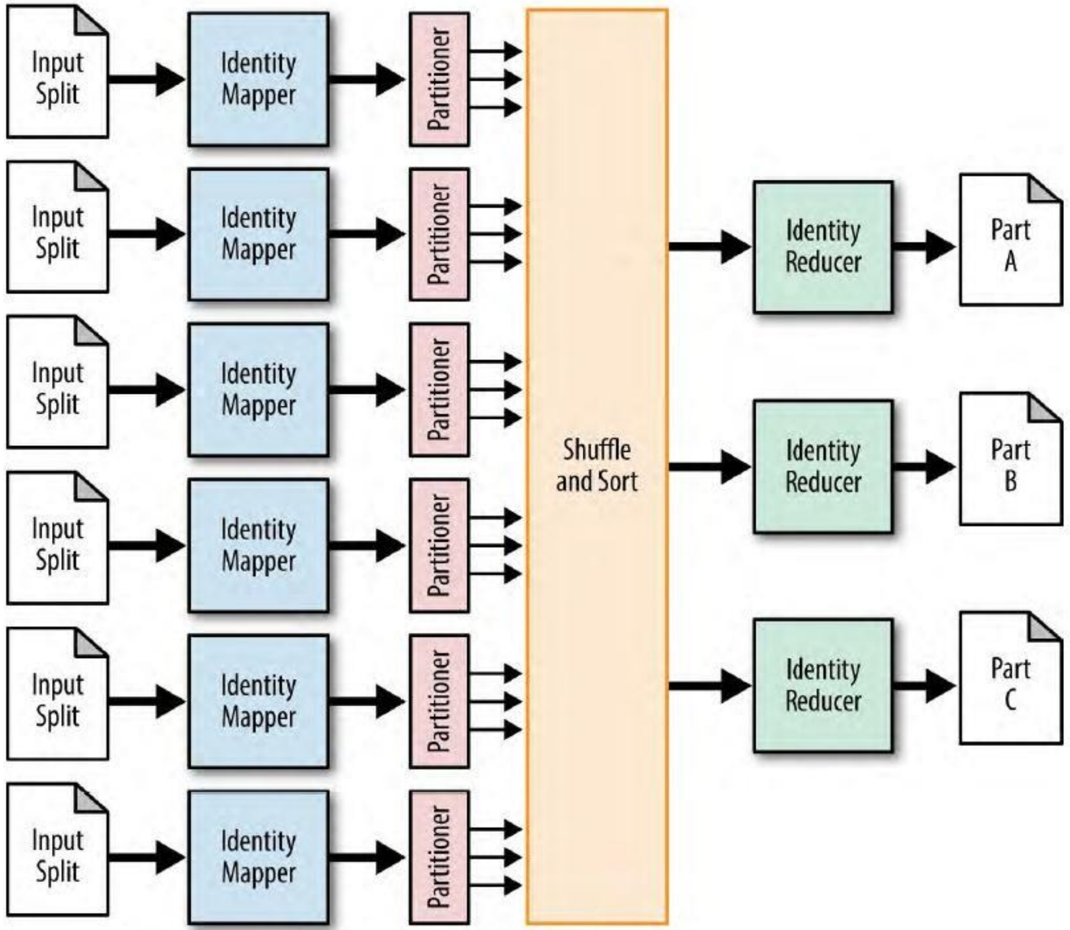
```java
private String nestElements(String post, List<String> comments) {
    // Create the new document to build the XML
    DocumentBuilder bldr = dbf.newDocumentBuilder();
    Document doc = bldr.newDocument();

    // Copy parent node to document
    Element postEl = getXmlElementFromString(post);
    Element toAddPostEl = doc.createElement("post");

    // Copy the attributes of the original post element to the new one
    copyAttributesToElement(postEl.getAttributes(), toAddPostEl);

    // For each comment, copy it to the "post" node
    for (String commentXml : comments) {
        Element commentEl = getXmlElementFromString(commentXml);
        Element toAddCommentEl = doc.createElement("comments");

        // Copy the attributes of the original comment element to
        // the new one
        copyAttributesToElement(commentEl.getAttributes(),
                toAddCommentEl);

        // Add the copied comment to the post element
        toAddPostEl.appendChild(toAddCommentEl);
    }

    // Add the post element to the document
    doc.appendChild(toAddPostEl);

    // Transform the document into a String of XML and return
    return transformDocumentToString(doc);
}

private Element getXmlElementFromString(String xml) {
    // Create a new document builder
    DocumentBuilder bldr = dbf.newDocumentBuilder();

    return bldr.parse(new InputSource(new StringReader(xml)))
            .getDocumentElement();
}
```

Partitioning 1/6

- Intent
  - Move records into categories; doesn’t care the order of records
  - Take similar records in a data set and partition them into distinct, smaller data sets
- Motivation
  - If you want to look at a particular set of data, the data items are normally spread out across the entire data set → requires an entire scan of all of the data
- Applicability
  - Knowing how many partitions ahead of time
    - Ex: by day of the week → 7 partitions

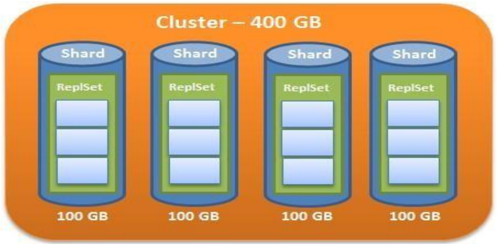
Partitioning 2/6

- Structure
  - exploits the fact that the partitioner partitions data
  - Custom partitioner: determine the partition for each record



Partitioning 3/6

- Known uses
  - Partition pruning by continuous value (e.g., date)
    - analyzes FROM and WHERE clauses in SQL statements to eliminate unneeded partitions when building the partition access list
  - Partition pruning by category
    - Country, phone area code, language
  - Sharding
    - Shard: a single server or replica set holding a part of sharded collection



- Performance analysis
  - The resulting partitions may not have similar # of records
    - Heavy workload for Reducers with high % of records
    - Workarounds:
      - Split very large partitions into several smaller partitions, even if just randomly
      - Assign multiple reducers to one partition and then randomly assign records into each to spread it out a bit better
  - Heavy traffic in shuffle-exchange network

Partitioning 4/6

- PartitionedUsers.java : Given a set of user info, partition the records based on the year of last access date, one partition per year (2008 - 2011)
- In: Users.xml
- Fix: extend to 2016

```
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);
    }
}
```

Partitioning 5/6

- Driver:

```
...
// 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);
...
```

- Reducer:

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

Partitioning 6/6

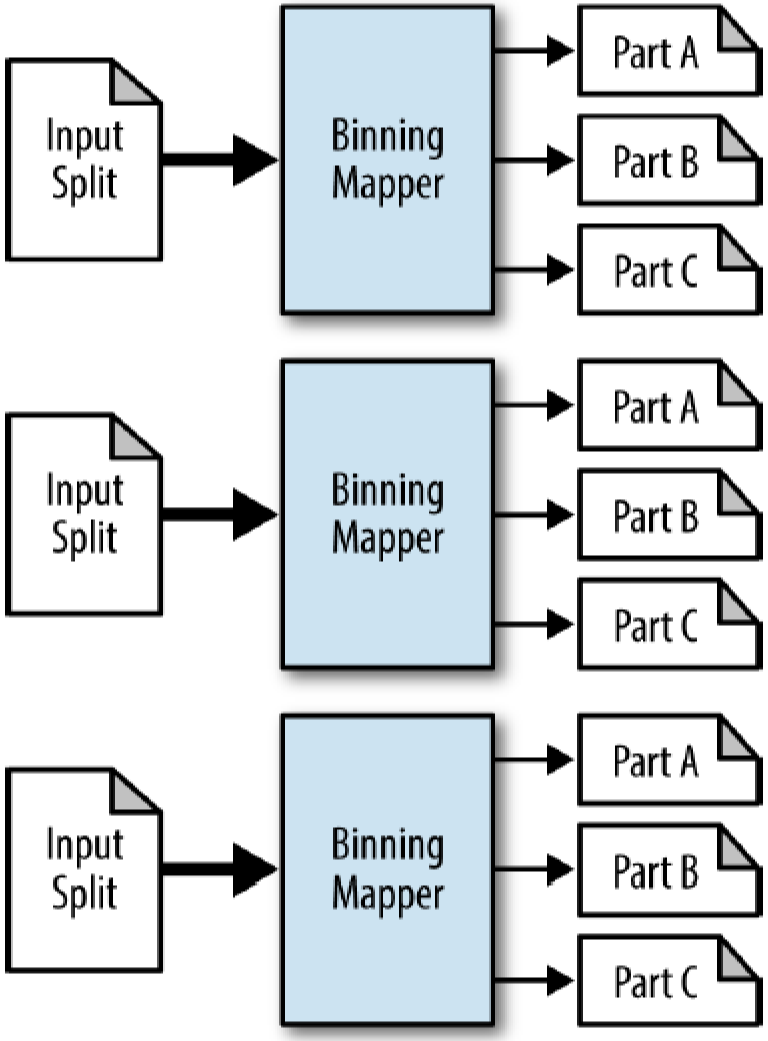
```java
public static class LastAccessDatePartitioner extends
        Partitioner<IntWritable, Text> implements Configurable {

    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);
    }
    public static void setMinLastAccessDate(Job job,
            int minLastAccessDateYear) {
        job.getConfiguration().setInt(MIN_LAST_ACCESS_DATE_YEAR,
                minLastAccessDateYear);
    }
}
```

2008 → 0
2009 → 1
2010 → 2
2011 → 3

Binning 1/4

- Intent
  - For each record in the data set, file each one into one or more categories
- Motivation
  - Binning: similar to partitioning and often can be used to solve the same problem
  - Binning splits data up in the map phase instead of in the partitioner
    - Possibility of handling dynamic # of categories
  - Each mapper has one file per possible output bin
    - 1000 Bins x 1000 Mappers = 1000,000 files

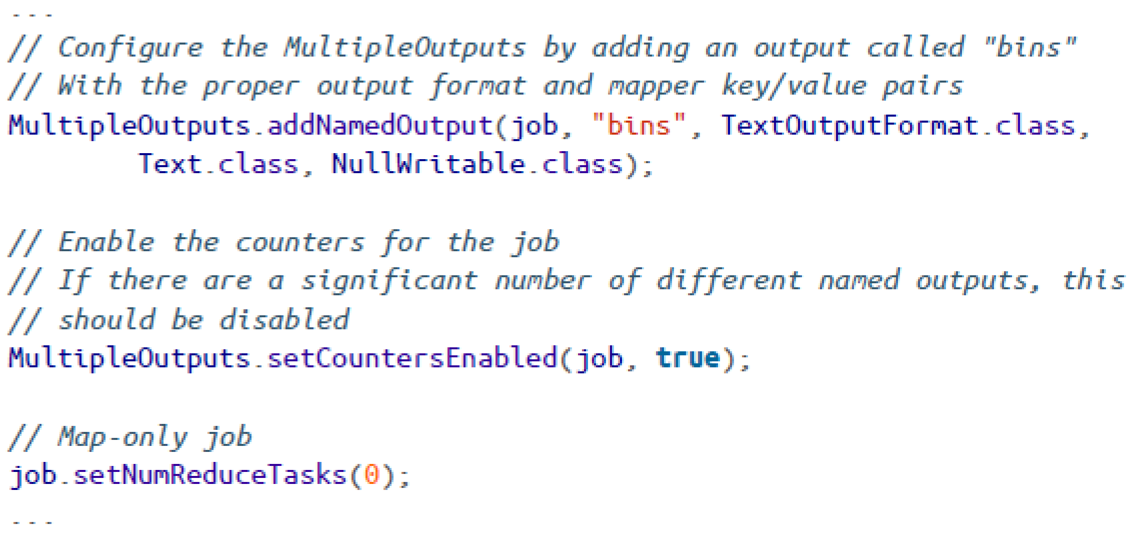
Binning 2/4



- Structure
  - Mapper: if the record meets the criteria, it is sent to that bin
  - No combiner, partitioner, or reducer
- Performance analysis
  - Each mapper outputs one small file per bin
  - map-only jobs scalability & performance
    - No sort, shuffle, or reduce to be performed
  - Most of the processing done on data that is local

Binning 3/4

- Binning.java: Given a set of StackOverflow posts, bin the posts into four bins based on the tags hadoop, pig, hive, and hbase. Also, create a separate bin for posts mentioning hadoop in the text or title
- In: Posts.xml



```
...
// Configure the MultipleOutputs by adding an output called "bins"
// With the proper output format and mapper key/value pairs
MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class,
        Text.class, NullWritable.class);

// Enable the counters for the job
// If there are a significant number of different named outputs, this
// should be disabled
MultipleOutputs.setCountersEnabled(job, true);

// Map-only job
job.setNumReduceTasks(0);
...
```

- MultipleOutputs
  - writing to additional outputs other than the job default output
  - Each additional output, or **named output**, may be configured with its own OutputFormat, with its own key class and with its own value class
  - void write(String namedOutput, K key, V value, String baseOutputPath)
    Write key and value to baseOutputPath using the namedOutput

Binning 4/4

```java
public static class BinningMapper extends
    Mapper<Object, Text, Text, NullWritable> {

    private MultipleOutputs<Text, NullWritable> mos = null;

    protected void setup(Context context) {
        // Create a new MultipleOutputs using the context object
        mos = new MultipleOutputs(context);
    }

    protected void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

        Map<String, String> parsed = MRDPUtils.transformXmlToMap(value
                .toString());

        String rawtags = parsed.get("Tags");

        // Tags are delimited by ><. i.e. <tag1><tag2><tag3>
        String[] tagTokens = StringEscapeUtils.unescapeHtml(rawtags).split(
                "><");

        // For each tag
        for (String tag : tagTokens) {
            // Remove any > or < from the token
            String groomed = tag.replaceAll(">|<", "").toLowerCase();

            // If this tag is one of the following, write to the named bin
            if (groomed.equalsIgnoreCase("hadoop")) {
                mos.write("bins", value, NullWritable.get(), "hadoop-tag");
            }
            if (groomed.equalsIgnoreCase("pig")) {
                mos.write("bins", value, NullWritable.get(), "pig-tag");
            }
            if (groomed.equalsIgnoreCase("hive")) {
                mos.write("bins", value, NullWritable.get(), "hive-tag");
            }
            if (groomed.equalsIgnoreCase("hbase")) {
                mos.write("bins", value, NullWritable.get(), "hbase-tag");
            }
        }

        // Get the body of the post
        String post = parsed.get("Body");

        // If the post contains the word "hadoop", write it to its own bin
        if (post.toLowerCase().contains("hadoop")) {
            mos.write("bins", value, NullWritable.get(), "hadoop-post");
        }
    }

    protected void cleanup(Context context) throws IOException,
            InterruptedException {
        // Close multiple outputs!
        mos.close();
    }
```

References

- Donald Miner and Adam Shook, *MapReduce Design Patterns*.
  - http://oreil.ly/mapreduce-design-patterns
  - https://github.com/adamjshook/mapreducepatterns