**CS435 BIG DATA**

**PART 1. LARGE SCALE DATA ANALYSIS USING MAPREDUCE**

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

---

**Today’s topics**

- FAQs
- MapReduce design pattern
  - Filtering
- Review for midterm

---

**FAQs**

- Quizzes (1-5) will be emailed with answers

---

**Filtering Pattern 3. Top 10**

- Retrieves a relatively small number (top $K$) of records, according to a ranking scheme in your dataset, no matter how large the data

- Known uses
  - Outlier analysis
  - Selecting interesting data
  - Catchy dashboards

---

**The structure of Top 10 pattern**

![Diagram of Top 10 pattern]
Mapper

```java
public static class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
    private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Map<String, String> parsed = transformXmlToMap(value.toString());
        String userId = parsed.get("Id");
        String reputation = parsed.get("Reputation");
        // Add this record to our map with the reputation as the key
        repToRecordMap.put(Integer.parseInt(reputation), new Text(value));
        // If we have more than ten records, remove the one with the lowest rep
        // As this tree map is sorted in descending order, the user with
        // the lowest reputation is the last key.
        if (repToRecordMap.size() > 10) {
            repToRecordMap.remove(repToRecordMap.firstKey());
        }
    }
}
```

Reducer

```java
public static class TopTenReducer extends Reducer<NullWritable, Text, NullWritable, Text> {
    private TreeMap<Integer, Text> repToRecordMap = new TreeMap<Integer, Text>();

    public void reduce(NullWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        for (Text value : values) {
            Map<String, String> parsed = transformXmlToMap(value.toString());
            repToRecordMap.put(Integer.parseInt(parsed.get("Reputation")), new Text(value));
            // If we have more than ten records, remove the one with the lowest rep
            // As this tree map is sorted in descending order, the user with
            // the lowest reputation is the last key.
            if (repToRecordMap.size() > 10) {
                repToRecordMap.remove(repToRecordMap.firstKey());
            }
        }
        for (Text t : repToRecordMap.descendingMap().values()) {
            // Output our ten records to the file system with a null key
            context.write(NullWritable.get(), t);
        }
    }
}
```

Filtering Pattern 4. Distinct

- You have data that contains similar records and you want to find a unique set of values

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());
        String userId = parsed.get("UserId");
        outUserId.set(userId); // Set our output key to the user's id
        context.write(outUserId, NullWritable.get());
    }
}
```

MapReduce Design Patterns

Filtering Patterns

- Distinct

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());
        String userId = parsed.get("UserId");
        outUserId.set(userId); // Set our output key to the user's id
        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 previous MapReduce software using a Combiner?

This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012

Customizing input and output

- Do we always want to load or store data the way Hadoop MR does out of the box?
  - Injecting data from original source without storing data in HDFS
  - Feeding the MapReduce output to the next process

Patterns discussed in this section

1. Generating data
2. External source input
3. Partition pruning
Modify the way data is loaded on disk
- Approach 1: Configuring how contiguous chunks of input are generated from blocks in HDFS
  - InputFormat
- Approach 2: Configuring how records appear in the map phase
  - RecordReader

Modify the way data is stored on disk
- Approach 1: Configuring how contiguous chunks of output are generated from blocks in HDFS
  - OutputFormat
- Approach 2: Configuring how records are stored after the map phase
  - RecordWriter

Roles of InputFormat in Hadoop
1. Validate the input configuration for the job (i.e., checking that the data is there).
2. Split the input blocks and files into logical chunks of type InputSplit, each of which is assigned to a map task for processing.
3. Create the RecordReader implementation to be used to create key/value pairs from the raw InputSplit. These pairs are sent one by one to their mapper.

Methods of the InputFormat abstract
- getSplits()
  - retrieves the configured input using the JobContext object
  - returns a List of InputSplit objects
    - getLocations() of InputSplit returns the list of hostnames where the input split is located
    - This provides data to the system to determine the TaskTracker to process the map task
    - Good place to throw any necessary exceptions
      - Before the job is actually submitted to the JobTracker
- createRecordReader()
  - Called by framework and generates RecordReader

Accessing your input file in MapReduce
- STEP 1. Validates the input for the job by checking whether all of the input paths exist
- STEP 2. Splits each input file logically based on the total size of the file in bytes
  - Block size is the upper bound
    - E.g. 160MB in HDFS will generate three blocks
      - 2 x 64MB and 1x38MB
- STEP 3. Each map task will be assigned exactly one of these input splits
- STEP 4. RecordReader will generate key/value pairs for Mapper input

RecordReader (1/2)
- Generates key/value pairs
- Fixing boundaries
  - Input split boundary might not exactly match the record boundary
  - E.g. TextInputFormat reads text files using a LineRecordReader to create key/value pairs
    - Will the chunk of bytes for each input split be lined up with a new line character, to mark the line for the LineRecordReader?
      - Those bits that are stored on a different node are streamed from a DataNode hosting the block
        - Handled by the FBInputStream class
RecordReader (2/2)

- Reads Bytes from the input source
- Generates WritableComparable key and Writable value
- An object-oriented way to present information to a mapper
- Example
  - TextInputFormat grabs each line
  - "<?xml version="1.0"?>" and "<quiz>" will be injected to the different Mappers
  - Customized RecordReader can read lines after the input split boundary
  - Each RecordReader should starts at the beginning of an XML element

Methods of the RecordReader (abstract)

- initialize()
- getCurrentKey() and getCurrentValue()
- nextKeyValue()
- getProgress()
- close()

Schema on read

- InputSplit represents a byte-oriented view of the split
- RecordReader prepares data for a mapper
  - Only the RecordReader maintains the schema

OutputFormat

- Similar to an input format
- Tasks
  - Validate the output configuration for the job
  - Create the RecordWriter implementation that will write the output of the job
- FileOutputFormat
  - File based output
  - Most output from MapReduce job is written to HDFS
  - TextOutputFormat (extended FileOutputFormat)
    - Stores key/value pairs to HDFS at a configured output directory with a tab delimiter
    - Validates the output file directory

Storing data in an External DB

- MapReduce job is not restricted to storing data to HDFS
- MapReduce can do a parallel bulk write
- Your storage should be able to handle the large number of connections from the many tasks
- E.g. DBOutputFormat<K DBWritable, V>
  - Objects that read from/written to a database should implement DBWritable
  - If we have the following table in the database:

```sql
CREATE TABLE MyTable (
    counter INTEGER NOT NULL,
    timestamp BIGINT NOT NULL,
);
```

Writing your output to a DB (1/2)

```java
public class MyWritable implements Writable, DBWritable {
    // Some data
    private int counter;
    private long timestamp;

    // WritableWrite implementation
    public void write(DataOutput out) throws IOException {
        out.writeInt(counter);
        out.writeLong(timestamp);
    }

    // WritableReadFields implementation
    public void readFields(DataInput in) throws IOException {
        counter = in.readInt();
        timestamp = in.readLong();
    }
}
```
Writing your output to a DB (2/2)

```java
public void write(PreparedStatement statement) throws SQLException {
    statement.setInt(1, counter);
    statement.setLong(2, timestamp);
}

public void readFields(ResultSet resultSet) throws SQLException {
    counter = resultSet.getInt(1);
    timestamp = resultSet.getLong(2);
}
```

**PreparedStatement** is an object that represents a precompiled SQL statement.

```java
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES
    SET SALARY = ? WHERE ID = ?");
pstmt.setBigDecimal(1, 153833.00);
pstmt.setInt(2, 110592);
```

I/O Pattern 1: Generating Data

- Generates a lot of data from scratch
- This pattern does not load data
- Use cases:
  - Generating random data
  - Generating artificial data as part of a benchmark
  - TeraGen/TeraSort and DFSIO
- This pattern is map-only

**Identity Mapper**

- Implements Mapper<K, V, K, V>
- conf.setMapperClass(IdentityMapper.class);
- Identity Mapper takes input key/value pair and returns without any processing
- Other implementations of Mapper
  - InverseMapper, TokenCountMapper, ChainMapper, Etc.

**Identity Reducer**

- Implements Reducer<K, V, K, V>
- Performs no reduction, writing all input values directly to the output.
- What is the difference between Identity Reducer and 0 reducer?
  - Identity reducer still sort and shuffle output data from the mappers
  - No aggregation

I/O Pattern 1: Generating Data: Example

- Goal
  - Generates random StackOverflow data
  - Take a list of 1,000 words and make random blurbs
Driver code

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    int numMapTasks = Integer.parseInt(args[0]);
    int numRecordsPerTask = Integer.parseInt(args[1]);
    Path wordList = new Path(args[2]);
    Path outputDir = new Path(args[3]);
    Job job = new Job(conf, "RandomDataGenerationDriver");
    job.setJarByClass(RandomDataGenerationDriver.class);
    job.setNumReduceTasks(0);
    job.setInputFormatClass(RandomStackOverflowInputFormat.class);
    RandomStackOverflowInputFormat.setNumMapTasks(job, numMapTasks);
    RandomStackOverflowInputFormat.setNumRecordPerTask(job, numRecordsPerTask);
    RandomStackOverflowInputFormat.setRandomWordList(job, wordList);
    TextOutputFormat.setOutputPath(job, outputDir);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(NullWritable.class);
    System.exit(job.waitForCompletion(true) ? 0 : 2);
}
```

InputSplit Code

```java
public static class FakeInputSplit extends InputSplit implements Writable {
    public void readFields(DataInput arg0) throws IOException {
    }
    public void write(DataOutput arg0) throws IOException {
    }
    public long getLength() throws IOException, InterruptedException {
        return 0;
    }
    public String[] getLocations() throws IOException, InterruptedException {
        return new String[0];
    }
}
```

InputFormat code

```java
public static class RandomStackOverflowInputFormat extends InputFormat<Text, NullWritable> {
    public static final String NUM_MAP_TASKS = "random.generator.map.tasks";
    public static final String NUM_RECORDS_PER_TASK = "random.generator.num.records.per.map.task";
    public static final String RANDOM_WORD_LIST = "random.generator.random.word.file";
    public List<InputSplit> getSplits(JobContext job) throws IOException {
        // Get the number of map tasks configured for int
        numSplits = job.getConfiguration().getInt(NUM_MAP_TASKS, -1);
        // Create a number of input splits equivalent to the number of tasks
        ArrayList<InputSplit> splits = new ArrayList<InputSplit>();
        for (int i = 0; i < numSplits; ++i) {
            splits.add(new FakeInputSplit());
        }
        return splits;
    }
    public static void setNumMapTasks(Job job, int i) {
        job.getConfiguration().setInt(NUM_MAP_TASKS, i);
    }
    public static void setNumRecordPerTask(Job job, int i) {
        job.getConfiguration().setInt(NUM_RECORDS_PER_TASK, i);
    }
    public static void setRandomWordList(Job job, Path file) {
        DistributedCache.addCacheFile(file.toUri(), job.getConfiguration());
    }
}
```

I/O Pattern 2: External Source Output

- Writing MapReduce output to a nonnative location
- In a MapReduce approach, the data is written out in parallel
The Structure of the external source output pattern

External Source OutputFormat

- Input
- Split
- Mapper
- External Source
- OutputFormat

The OutputFormat verifies the output specification of the job configuration prior to job submission.

The RecordWriter writes all key/value pairs to the external source

Example

- Writing the results to a number of Redis instances
  - Redis is an open-source, in-memory, key-value store
  - Redis provides Jedis (Java client of Redis)
  - A Redis hash is a map between string fields and string values
    - Similar to a Java HashMap

OutputFormat Code

```java
public static class RedisHashOutputFormat extends OutputFormat<Text, Text> {

    public static final String REDIS_HOSTS_CONF = "mapred.redishashoutputformat.hosts";
    public static final String REDIS_HASH_KEY_CONF = "mapred.redishashoutputformat.hashKey";

    public static void setRedisHosts(Job job, String hosts) {
        job.getConfiguration().set(REDIS_HOSTS_CONF, hosts);
    }

    public static void setRedisHashKey(Job job, String hashKey) {
        job.getConfiguration().set(REDIS_HASH_KEY_CONF, hashKey);
    }

    public RecordWriter<Text, Text> getRecordWriter(TaskAttemptContext job) throws IOException, InterruptedException {
        return new RedisHashRecordWriter(job.getConfiguration().get(REDIS_HASH_KEY_CONF), job.getConfiguration().get(REDIS_HOSTS_CONF));
    }

    public void checkOutputSpecs(JobContext job) throws IOException {
        String hosts = job.getConfiguration().get(REDIS_HOSTS_CONF);
        if (hosts == null || hosts.isEmpty()) {
            throw new IOException(REDIS_HOSTS_CONF + " is not set in configuration.");
        }

        String hashKey = job.getConfiguration().get(REDIS_HASH_KEY_CONF);
        if (hashKey == null || hashKey.isEmpty()) {
            throw new IOException(REDIS_HASH_KEY_CONF + " is not set in configuration.");
        }
    }
}
```

RecordWriter Code

```java
public static class RedisHashRecordWriter extends RecordWriter<Text, Text> {

    private HashMap<Integer, Jedis> jedisMap = new HashMap<Integer, Jedis>();

    public RedisHashRecordWriter(String hashKey, String hosts) {
        this.hashKey = hashKey;
        // Create a connection to Redis for each host
        // Map an integer 0-( numRedisInstances - 1) to the instance
        int i = 0;
        for (String host : hosts.split(\"\", \")\") {
            Jedis jedis = new Jedis(host);
            jedis.connect();
            jedisMap.put(i, jedis);
            ++i;
        }
    }
```
continued

```java
public void write(Text key, Text value) throws IOException, InterruptedException {
    // Get the Jedis instance that this key/value pair will be written to Jedis
    Jedis j = jedisMap.get(Math.abs(key.hashCode()) % jedisMap.size());
    // Write the key/value pair
    j.hset(hashKey, key.toString(), value.toString());
    public void close(TaskAttemptContext context) throws IOException, InterruptedException {
        // For each jedis instance, disconnect it for (Jedis jedis : jedisMap.values()) {
        jedis.disconnect();
    }
}
```

`Mapper Code`

```java
public static class RedisOutputMapper 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());
        String userId = parsed.get("Id");
        String reputation = parsed.get("Reputation");
        // Set our output key and values
        outkey.set(userId);
        outvalue.set(reputation);
        context.write(outkey, outvalue);
    }
}
```

`Driver Code`

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Path inputPath = new Path(args[0]);
    String hosts = args[1];
    String hashName = args[2];
    Job job = new Job(conf, "Redis Output");
    job.setJarByClass(RedisOutputDriver.class);
    job.setMapperClass(RedisOutputMapper.class);
    job.setNumReduceTasks(0);
    job.setInputFormatClass(TextInputFormat.class);
    TextInputFormat.setInputPaths(job, inputPath);
    job.setOutputFormatClass(RedisHashOutputFormat.class);
    RedisHashOutputFormat.setRedisHosts(job, hosts);
    RedisHashOutputFormat.setRedisHashKey(job, hashName);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    int code = job.waitForCompletion(true) ? 0 : 2;
    System.exit(code);
}
```

`I/O Pattern 3: Partition Pruning`

- Configures the way the framework picks input splits and drops files from being loaded into MapReduce based on the name of the file
- Partitions data by a predetermined value
- Use cases
  - Organizing your data based on your analysis patterns
  - Change analytics? Or, change data input format?

`Writing InputSplit`

```java
public static class RedisLastAccessInputSplit extends InputSplit implements Writable {
    private String location = null;
    private List<String> hashKeys = new ArrayList<String>[
    public RedisLastAccessInputSplit() {
        // Default constructor for reflection
    }
    public RedisLastAccessInputSplit(String redisHost) {
        this.location = redisHost;
    }
    public void addHashKey(String key) {
        hashKeys.add(key);
    }
    public void removeHashKey(String key) {
        hashKeys.remove(key);
    }
    public List<String> getHashKeys() {
        return hashKeys;
    }
}
```
public void readFields(DataInput in) throws IOException {
    location = in.readUTF();
    int numKeys = in.readInt();
    hashKeys.clear();
    for (int i = 0; i < numKeys; ++i) {
        hashKeys.add(in.readUTF());
    }
}

public void write(DataOutput out) throws IOException {
    out.writeUTF(location);
    out.writeInt(hashKeys.size());
    for (String key : hashKeys) {
        out.writeUTF(key);
    }
}

public long getLength() throws IOException, InterruptedException {
    return 0;
}

public String[] getLocations() throws IOException, InterruptedException {
    return new String[] { location };
}