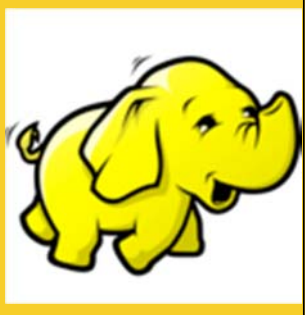
Map Reduce Features

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/

© 2012 coreservlets.com and Dima May



Agenda

- **Counters**
- **Speculative Execution**
- **Distributed Cache**

Counters

- **Instrument Job's metrics**
  - Gather statistics
  - Quality control – confirm what was expected
  - Diagnostics
- **Framework provides a set of built-in metrics**
  - For example bytes processed for input and output
- **User can create new counters**
  - Number of records consumed
  - Number of errors or warnings
- **Counters are divided into groups**
- **Tracks total, mapper and reducer counts**

Built-in Counters

- **Maintains and sums up counts**
- **Several groups for built-in counters**
  - Job Counters – documents number of map and reduce tasks launched, number of failed tasks
  - File System Counters – number of bytes read and written
  - Map-Reduce Framework – mapper, reducer, combiner input and output records counts, time and memory statistics

Job Counters

- **Web UI exposes counters for each Job**

User Defined Counters

- **You can create new counters**
- **Increment counters in Reducer and/or Mapper classes**
  - Framework accurately sums up counts between various stages and produces totals
    - Accounts for failed tasks

Implement User-Defined Counters

1. **Retrieve Counter from Context object**
   - Framework injects Context object into map and reduce methods
2. **Increment Counter's value**
   - Can increment by 1 or more

1: Retrieve Counter from Context

- **Utilize Context object**
  - void map(Key key, Value value, Context context)
  - void reduce(Key key, Iterable<Value> values, Context context)
- **Although map's and reduce's Context type is not the same they both extend from**

  `org.apache.hadoop.mapreduce.TaskAttemptContext`

- **TaskAttemptContext provides two ways to retrieve counters**
  - public Counter getCounter(String groupName, String counterName);
  - public Counter getCounter(Enum<?> counterName);
    - Figures out group name from fully qualified classname of enum - enum.getDeclaringClass().getName()

2: Increment Counter's Value

- **Increment or even set the value**
  - void setValue(long value);
  - void increment(long incr);

StartsWithCountMapper with Counters

- **Recall the StartsWithCountJob**
- **Update Mapper to document counts for**
  - Total tokens processed
  - Number of tokens that start with uppercase
  - Number of tokens that start with lowercase
- **First create an enum to reference these counters:**

```
public enum Tokens {
      Total, FirstCharUpper, FirstCharLower
}
```


StartsWithCountMapper_UserCounters.java

```
@Override
protected void map(LongWritable key, Text value, Context context)
              throws IOException, InterruptedException {

  StringTokenizer tokenizer = new StringTokenizer(value.toString());
  while (tokenizer.hasMoreTokens()) {
       String token = tokenizer.nextToken();
       reusableText.set(token.substring(0, 1));
       context.write(reusableText, countOne);


       context.getCounter(Tokens.Total).increment(1);
       char firstChar = token.charAt(0);

       if ( Character.isUpperCase(firstChar)){
              context.getCounter(Tokens.FirstCharUpper).increment(1);
       } else {
              context.getCounter(Tokens.FirstCharLower).increment(1);
       }
  }
}
```

Keep count of total tokens processed

Stats on tokens that start with upper case vs. lowercase

Run StartsWithCountMapper_UserCounters

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar
mr.wordcount.StartsWithCountJob_UserCounters
/training/data/hamlet.txt /training/playArea/wordCount/
...
...
...
        Map output records=34189
        Map output bytes=205134
        Combine input records=34189
        Combine output records=69
        Reduce input records=69
        Reduce output records=69
  mr.wordcount.StartsWithCountMapper_UserCounters$Tokens
        FirstCharLower=26080
        FirstCharUpper=8109
        Total=34189
  File Input Format Counters
        Bytes Read=211294
  File Output Format Counters
        Bytes Written=385
```

Job configures new mapper with counts

Group was derived from the class name

Total # of tokens should match Map output records

Customize Counter's Names

- **Can customize counter and group names when using enums**
  1. Create a properties file <classname>.properties defining counter name properties
     - Inner classes are substituted by underscore
     - For example: org.com.MyMapper$CustomEnum would be MyMapper_CustomEnum.properties
  2. Place properties file in the same package as the class that defines Enum

1: Create Properties File

- **In our case the enum was defined in**
  - mr.wordcount.StartsWithCountMapper_UserCounters$Tokens
- **Therefore the file is to be named**
  - StartsWithCountMapper_UserCounters_Tokens.properties
- **Define Group and Counter names:**

```
CounterGroupName = Token Processed
Total.name=Total Tokens Processed
FirstCharUpper.name=Tokens start with Uppercase
FirstCharLower.name=Tokens start with Lowercase
```


Test Counter Re-Naming

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar
mr.wordcount.StartsWithCountJob_UserCounters
/training/data/hamlet.txt /training/playArea/wordCount/
...
...
...
        Map output records=34189
        Map output bytes=205134
        Combine input records=34189
        Combine output records=69
        Reduce input records=69
        Reduce output records=69
    Token Processed
        Tokens start with Lowercase=26080
        Tokens start with Uppercase=8109
        Total Tokens Processed =34189
    File Input Format Counters
        Bytes Read=211294
    File Output Format Counters
        Bytes Written=385
```

New Names are mapped by the framework

Retrieving Counters

1. **Web-UI**

2. **Command line**
   - $ mapred job -status <job_id>
3. **Java API**
   - Further analyze counts
   - Store in a database

Java API to Retrieve Counters

- **Print all the counters after the job is done**
  - Snippet from StartsWithCountJob_PrintCounters.java

```
int resultCode = job.waitForCompletion(true) ? 0 : 1;

System.out.println("Job is complete! Printing Counters:");
Counters counters = job.getCounters();

for ( String groupName : counters.getGroupNames()){
  CounterGroup group = counters.getGroup(groupName);
  System.out.println(group.getDisplayName());

  for (Counter counter : group.getUnderlyingGroup()){
       System.out.println("  " + counter.getDisplayName() +
                          "=" + counter.getValue());
  }
}
```

Speculative Execution

- **Job is decomposed into small tasks**
- **Job is as fast as the slowest task**
- **Given 100s or even 1000s of tasks**
  - Few tasks may run very slowly (hardware issues, other activity on that machine, configuration, etc...)
- **MapReduce framework strives to resolve slow running tasks by spawning the same task on a different machine**
  - Doesn't start speculative tasks immediately

Speculative Execution

- **Will spawn a speculative task when**
  - All the tasks have been started
  - Task has been running for an extended period of time
    - over a minute
  - Did not make significant progress as compared to the rest of the running tasks
- **After task's completion duplicates are killed**
- **Just an optimization**

Speculative Execution

- **Can be turned off by setting these properties to false**
  - mapred.map.tasks.speculative.execution
    - Turn on/off speculative execution for map phase
  - mapred.reduce.tasks.speculative.execution
    - Turn on/off speculative execution for reduce phase
- **When should I disable Speculative Execution?**
  - Task is outputting directly to a shared resource; then starting a duplicate task may cause unexpected results
  - Minimize cluster and bandwidth usage; duplicate tasks use up resources

Distributed Cache

- **A mechanism to distribute files**
- **Make them available to MapReduce task code**
- **yarn command provides several options to add distributed files**
- **Can also use Java API directly**
- **Supports**
  - Simple text files
  - Jars
  - Archives: zip, tar, tgz/tar.gz

Distributed Cache via $ yarn Command

- **Update StartWithCount job to exclude specified start letters**
  1. Load a file which contains start letters to exclude onto distributed cache
     - utilize -files parameter with the yarn command
  2. Update Map code to utilize the exclude file

1: Load Exclude File Onto DistributedCache

```
$ cat $PLAY_AREA/data/startWithExcludeFile.txt
b
c
d
e
f
G
```

Exclude tokens that start with these letters

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar \
  mr.wordcount.StartsWithCountJob_DistCache \
  -files $PLAY_AREA/data/startWithExcludeFile.txt \
  /training/data/hamlet.txt \
  /training/playArea/wordCount/
```

Using -files option yarn command will place the file onto DistributedCache.

2: Utilize Exclude File in the Map Code

```
public class StartsWithCountMapper_DistCache extends
       Mapper<LongWritable, Text, Text, IntWritable> {
  private Logger log =
       Logger.getLogger(StartsWithCountMapper_DistCache.class);
  private final static IntWritable countOne =
       new IntWritable(1);
  private final Text reusableText = new Text();

  public final static String EXCLUDE_FILE =
       "startWithExcludeFile.txt";
  private final Set<String> excludeSet = new HashSet<String>();
...
...
...
```

Will be able to directly reference the file without absolute path;
Constructs an exclude set

2: Utilize Exclude File in the Map Code

Framework takes care of all the magic; the file is now stored locally and can be referenced with just a name

```
...
...
@Override
protected void setup(Context context) throws IOException,
     InterruptedException {
   FileReader reader = new FileReader(new File(EXCLUDE_FILE));
   try {
     BufferedReader bufferedReader = new BufferedReader(reader);
     String line;
     while ((line = bufferedReader.readLine()) != null) {
       excludeSet.add(line);
       log.info("Ignoring words that start with ["+line+"]");
     }
   } finally {
     reader.close();
   }
}
...
...
```

2: Utilize Exclude File in the Map Code

```
...
...
  @Override
  protected void map(LongWritable key, Text value,
      Context context) throws IOException, InterruptedException {

    StringTokenizer tokenizer =
        new StringTokenizer(value.toString());

    while (tokenizer.hasMoreTokens()) {
      String firstLetter =
          tokenizer.nextToken().substring(0, 1);

      if (!excludeSet.contains(firstLetter)) {
        reusableText.set(firstLetter);
        context.write(reusableText, countOne);
      }
    }
  }
}
```


Output of Distributed Cache Job

```
$ hdfs dfs -cat /training/playArea/wordCount/part-r-00000
...
...
[       122
_       1
a       2370
h       1724
i       1468
j       65
k       222
...
...
```


Exclude letters are missing

Distributed Cache Inner-Workings

- **Accepts two types: files and archives**
  - Archives are unarchived on the local node
- **Items specified to the $yarn command via -files, -libjars and -archives are copied to HDFS**
- **Prior to task execution these files are copied locally from HDFS**
  - Files now reside on a local disk – local cache
- **Files provided to the -libjars are appended to task's CLASSPATH**
- **Locally cached files become qualified to be deleted after all tasks utilizing cache complete**

Distributed Cache Inner-Workings

- **Files in the local cache are deleted after a 10GB threshold is reached**
  - Allow space for new files
  - Configured via yarn.nodemanager.localizer.cache.target-size-mb property
- **Local cache is stored under**

`${yarn.nodemanager.local-dirs}/usercache/$user/filecache`

  - Task code is not aware of the location
  - Symbolic link is created for each file, that's why we were able to reference a file without the path
    - FileReader reader = new FileReader(new File(EXCLUDE_FILE))

Java API - DistributedCache

- **Typically don't need to use directly**
  - Delegate to $yarn command (with -files, -libjars or -archive options)
- **However when programmatic involvement is necessary use DistributedCache class**
  - File(s) to be cached must exist on HDFS
    - With $yarn command framework takes care of this step for you
  - In the Tool – Place data into cache via methods on Job
  - In the Task – Retrieve the file from the cache

1: File(s) to be Cached Must Exist on HDFS

- **Add file to HDFS, either via command line or via FileSystem Java API**
- **Major difference from using $yarn command where framework adds it to HDFS on your behalf**

```
$ hdfs dfs -put startWithExcludeFile.txt /training/data/
$ hdfs dfs -ls /training/data/
Found 3 items
  0 2011-12-24 11:21 /training/data/glob
 22 2011-12-20 22:12 /training/data/readMe.txt
 12 2012-06-17 16:08 /training/data/startWithExcludeFile.txt
```

2: In the Tool – Place Data Onto Cache via Methods on Job

StartsWithCountJob_DistCacheAPI.java

```
....
public int run(String[] args) throws Exception {
  Job job = Job.getInstance(getConf(), getClass().getSimpleName());
  job.setJarByClass(getClass());
...
...
...
  Path toCache = new Path("/training/data/startWithExcludeFile.txt");
  job.addCacheFile(toCache.toUri());
  job.createSymlink();


  return job.waitForCompletion(true) ? 0 : 1;
}
...
```

Add file to DistributedCache

Create symbolic links for all files in DistributedCache; without the links you would have to use fully qualified path, in this case "/training/data/startWithExcludeFile.txt"

3: In the Task – Retrieve the File From the Cache

- **Same as before:**

StartsWithCountMapper_DistCache.java

```
...
...
@Override
protected void setup(Context context) throws IOException,
    InterruptedException {
  FileReader reader = new FileReader(new File(EXCLUDE_FILE));
  try {
    BufferedReader bufferedReader = new BufferedReader(reader);
    String line;
    while ((line = bufferedReader.readLine()) != null) {
      excludeSet.add(line);
      log.info("Ignoring words that start with ["+line+"]");
    }
  } finally {
    reader.close();
  }
}
...
```

Wrap-Up

Summary

- **We learned about**
  - Counters
  - Speculative Execution
  - Distributed Cache

Questions?