Distributed Operating Systems- unit 5

(HADOOP- The Configuration API)

Dr.K.Geetha
Associate Professor of Computer Science
Periyar Arts College
Cuddalore

OVERVIEW

- The Configuration API
- Configuring the Development Environment
- Running Locally on Test Data
- Running on a Cluster
- The Map Reduce Web UI
- Using a Remote Debugger
- Tuning a Job
- Map Reduce Workflows

Reference:Tom White, “Hadoop: The Definitive Guide”, Published by O’Reilly Media, Third Edition, 2009

Configuration Class

An instance of the Configuration class (found in the org.apache.hadoop.conf package)represents a collection of configuration properties and their values.

Each property is named by a String,

The type of a value may be one of several types, including
Boolean
 Int
long
float
and other useful types such as String, Class, java.io.File, and collections of Strings.
Configurations read their properties from resources—XML files with a simple structure

A simple configuration file

```
<?xml version="1.0"?>

<configuration>

<property>

<name>color</name>

<value>yellow</value>

<description>Color</description>

</property>

<property>
```

A simple configuration file

```
name>size</name>

<value>10</value>

<description>Size</description>

</property>

<property>

<name>weight</name>

<value>heavy</value>

<final>true</final>
```

A simple configuration file

```
<description>Weight</description>

</property>

<property>

<name>size-weight</name>

<value>${size},${weight}</value>

<description>Size and weight</description>

</property>

</configuration>
```

A simple configuration file

this configuration file is in a file called configuration-1.xml, we can access its properties using a part of code like this:

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
assertThat(conf.get("color"), is("yellow"));
assertThat(conf.getInt("size", 0), is(10));
assertThat(conf.get("breadth", "wide"), is("wide"));
```

Combining resources

```
xml version="1.0"?>
<configuration>
<property>
<name>size</name>
<value>12</value>
</property>
<property>
<name>weight</name>
<value>light</value>
</property>
</configuration>
```

Combining Resources

Resources are added to a Configuration in order:

```
Configuration conf = new Configuration();

conf.addResource("configuration-1.xml");

conf.addResource("configuration-2.xml");
```

Mapper

Unit test for Max Temperature Mapper

```
import static org.mockito.Matchers.anyObject;

import static org.mockito.Mockito.*;

import java.io.IOException;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapred.OutputCollector;

import org.junit.*;

public class MaxTemperatureMapperTest {
```

Mapper

```
public void processesValidRecord() throws IOException {

MaxTemperatureMapper mapper = new MaxTemperatureMapper();

Text value = new Text("0043011990999991950051518004+68750+023550FM-
12+0382" +

// Year ^^^^

"99999V0203201N00261220001CN9999999N9-00111+99999999999");
// Temperature ^^^^^

OutputCollector<Text, IntWritable> output = mock(OutputCollector.class);

mapper.map(null, value, output, null);

verify(output).collect(new Text("1950"), new IntWritable(-11));

}

}
```

First version of a Mapper that passes MaxTemperatureMapperTest

```
public class MaxTemperatureMapper extends MapReduceBase
implements Mapper<LongWritable, Text, Text, IntWritable> {
public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
String line = value.toString();
String year = line.substring(15, 19);
int airTemperature = Integer.parseInt(line.substring(87, 92));
output.collect(new Text(year), new IntWritable(airTemperature));
}
}
```

Reducer

Reducer

The reducer has to find the maximum value for a given key. Here’s a simple test for
this feature:

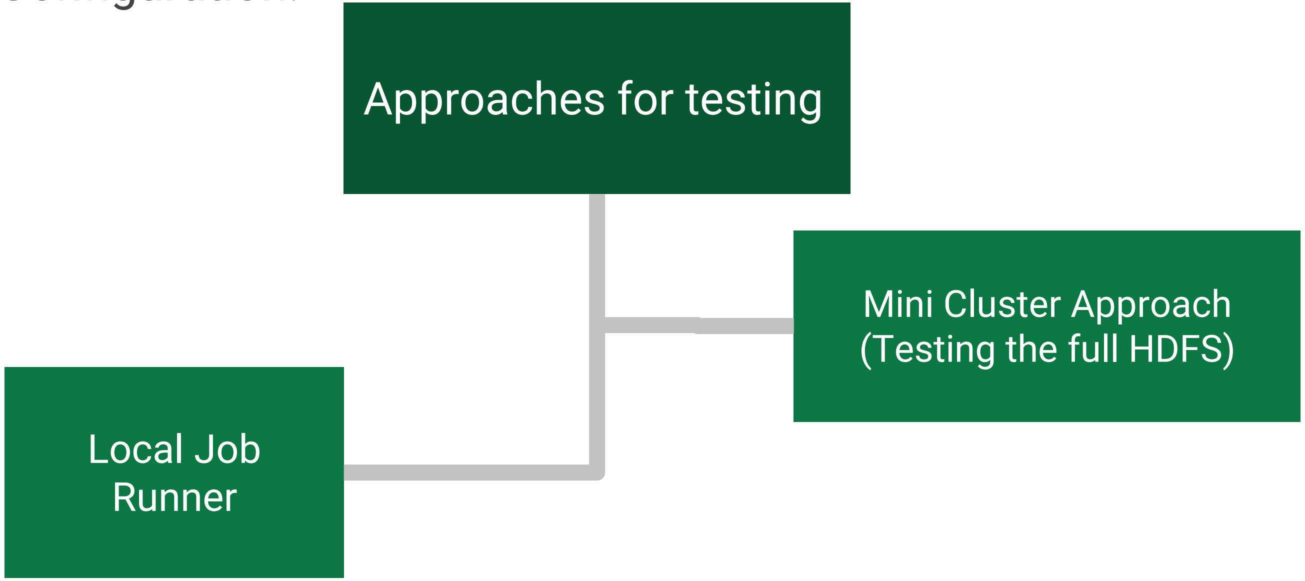
```
@Test
public void returnsMaximumIntegerInValues() throws IOException {
MaxTemperatureReducer reducer = new MaxTemperatureReducer();
Text key = new Text("1950");
Iterator<IntWritable> values = Arrays.asList(
new IntWritable(10), new IntWritable(5)).iterator();

OutputCollector<Text, IntWritable> output = mock(OutputCollector.class);

reducer.reduce(key, values, output, null);

verify(output).collect(key, new IntWritable(10));

}
```

Running Locally on Test Data

- The local job runner is only designed for simple testing of MapReduce programs,
- It differs from the full MapReduce implementation.
- The biggest difference is that it can’t run more than one reducer. (It can support the zero reducer case, too.)
- Most applications can work with one reducer, although on a cluster you would choose a larger number to take advantage of parallelism.
- the local runner will silently ignore the setting and use a single reducer.

Testing the Driver

Apart from the flexible configuration it is made more testable because it allows to inject an arbitrary Configuration.



The MapReduce Web User Interface

Hadoop comes with a web UI for viewing information about your jobs. It is useful for following a job’s progress while it is running, as well as finding job statistics and logs after the job has completed.

Job History
Job history refers to the events and configuration for a completed job.

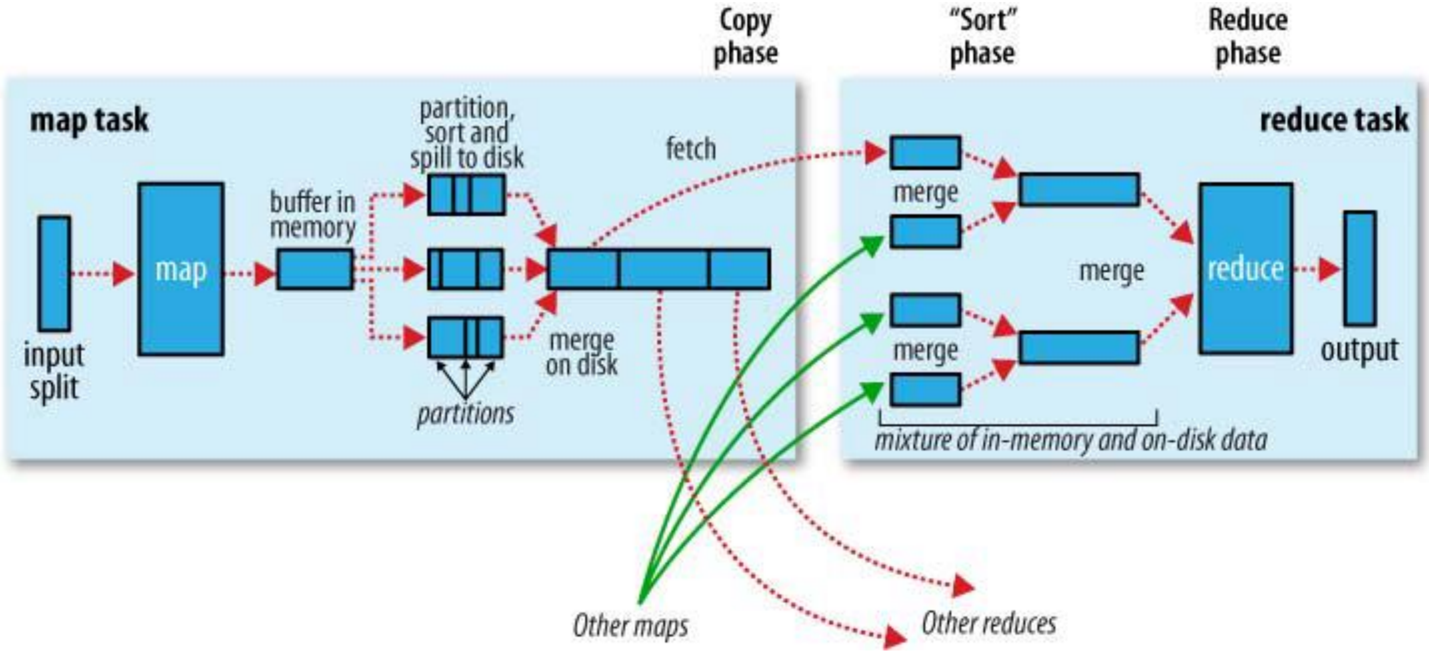
Tuning a Job-checklist

Area	Best practice
Number of mappers	How long are you mappers running for? If they are only running for a few seconds on average, then you should see if there's a way to have fewer mappers and make them all run longer, a minute or so, as a rule of thumb. The extent to which this is possible depends on the input format you are using.
Number of reducers	For maximum performance, the number of reducers should be slightly less than the number of reduce slots in the cluster. This allows the reducers to finish in one wave, and fully utilizes the cluster during the reduce phase.
Combiners	Can your job take advantage of a combiner to reduce the amount of data in passing through the shuffle?
Intermediate compression	Job execution time can almost always benefit from enabling map output compression.
Custom serialization	If you are using your own custom `Writable` objects, or custom comparators, then make sure you have implemented `RawComparator`.
Shuffle tweaks	The MapReduce shuffle exposes around a dozen tuning parameters for memory management, which may help you eke out the last bit of performance.

MapReduce Workflows

- Decomposing a Problem into MapReduce Jobs
- Running Dependent Jobs

Work flows


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition, sort and spill to disk
fetch
buffer in memory
input split
map
merge
merge
merge
reduce
output
merge on disk
partitions
mixture of in-memory and on-disk data
Other maps
Other reduces

Reference:Tom White, “Hadoop: The Definitive Guide”, Published by O’Reilly Media, Third Edition, 2009