Report of MapReduce Facility

Project 3 of 15-640

Kailiang Chen(kailianc) & Yang Pan(yangpan)

## **Development environment**

This project is developed with Eclipse IDE for Java Developers, Luna Release (4.4.0), JDK 8u20. If you want to write a new example, implementing interface MRBase, you should work in the same environment. Details will be elaborated in the following tutorial.

# I/O Library

We simplify the Hadoop MapReduce library and provide two classes Pair and PairContainer for Application Programmer.

Pair is a class which has a String type key and an ArrayList<String> type value. It has following APIs

1. Pair(String line) : constructor which will create a Pair if a line of String is input (the defined delimiter is colon )
2. Pair(String key, value) : constructor which will create a Pair if a String type key and a String type value is input
3. Pair(String key, Iterator<String> val) : constructor which will create a Pair if a String type key and a String iterator is input
4. String toString() : a Pair can convert to a String output, with key : value1,value2,…,valueN
5. String getFirst()/setFirst() : getter/setter for String type key
6. Iterator<String>getSecond/setSecond(): getter/setter for Iterator<String> type value
7. int compareTo (): Pair instances will need to be sorted by key in PairContainer so it has to implement Comparable

PairContainer is a class which maintains a series of Pair instances. It has a ArrayList<Pair> structure to put/get/sort if needed. It has the following APIs:

1. PairContainer(Iterator<Pair> itor) : constructor which will create a PairContainer if an iterator to a Pair data structure is input.
2. void emit(Pair pair) : put a Pair instance into management.
3. void emit(String key, String val) : construct and put a Pair instance into management if a String type key and String type value is input
4. void mergeSameKey() : sort and merge the values if the key is the same
5. Iterator<Pair> getInitialIterator() : get the iterator to the list of values
6. String toString() : a PairContainer can convert to a String, with each Pair per line
7. void restoreFromString() : create a PairContainer from a String input, a Pair per line, a key with a list of value per Pair

# Two Examples

We have provided two examples.

One example is the classic WordCounter. In the map phase every word is emitted with count “1”. In the reduce phase, the count of the same word is accumulated.

public class WordCounter implements MRBase {

@Override

public void reduce(String key, Iterator<String> values,

PairContainer output) {

Integer sum = 0;

while (values.hasNext()) {

sum += Integer.parseInt(values.next());

}

output.emit(key, sum.toString());

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public void map(String key, String value,

PairContainer output) {

Scanner scan = new Scanner(value);

scan.useDelimiter("\n");

while(scan.hasNext()) {

output.emit(scan.next(), "1");

}

}

}

The other example is a new Twitter information extraction. During the map phase, the userId and photoNum posted by this userId are extracted from the Twitter record line by line. During the reduce phase, the photoNum of the same userId is accumulated. The final result is two columns – left is userId, right is photoNum.

public class Twitter implements MRBase {

@Override

public void map(String key, String value, PairContainer output) {

Scanner scan = new Scanner(value);

scan.useDelimiter("\n");

String line = null;

while(scan.hasNext()) {

line = scan.next();

if (line != null && line.isEmpty()) {

continue;

}

JSONObject json = null;

try {

json = new JSONObject(line);

String userID = extractUserID(json);

JSONArray medias = extractMedias(json);

int numOfPhoto = 0;

if (medias != null) {

for (int i = 0; i < medias.length(); i++) {

JSONObject media = medias.getJSONObject(i);

if (media != null) {

String type = media.getString("type");

if (type != null && type.equals("photo")) {

numOfPhoto++;

} }

}

}

output.emit(userID, String.valueOf(numOfPhoto));

} catch (JSONException e) {

e.printStackTrace();

}

}

}

@Override

public void reduce(String key, Iterator<String> values,

PairContainer output) {

Integer sum = 0;

while (values.hasNext()) {

sum += Integer.parseInt(values.next());

}

output.emit(key, sum.toString());

}

public JSONArray extractMedias(JSONObject json) {

JSONArray medias = null;

try {

medias = json.getJSONObject("entities").getJSONArray("media");

} catch (JSONException e) {

return null;

}

return medias;

}

public String extractUserID(JSONObject json) throws JSONException {

String id = json.getJSONObject("user").getString("id\_str");

return id;

}

public String extractTweetID(JSONObject json) throws JSONException {

String id = json.getString("id\_str");

return id;

}

}

# How to run example

1. Obtain the source code of our facility, and copy the “Example/WordCounter” or “Example/Twitter” to the Master root directory
2. Start the facility, details can refer to “System\_Administrator\_Manual”
3. Start the example using “New XXX”, details can refer to “System\_Administrator\_Manual”

# Writing customized Mapper/Reducer

By implementing the MRBase class, you can customize your own map/reduce functions. Here, we haven’t divided the interface into Mapper/Reducer for simplicity.

The interface is as follows :

1. void map(String key, String value, PairContainer output)
2. void reduce(String key, Iterator<String> values, PairContainer output)

According to the paper “MapReduce: Simplified Data Processing on Large Clusters”,

in map function, the key is the file name, the value is the file’s contents, the output is a PairContainer; in reduce function, the key is a word, the values is a list of counts, the output is a PairContainer. Both in map and reduce, PairContainer’s emit() function is used to output the result.