CLASS Conference 2012

Cloud Assisted Services

BigData and MapReduce with Hadoop

Ivan Tomašić

M. Sc.

Jožef Stefan Institute, Ljubljana, Slovenia

Outline

- Introduction
- MapReduce paradigm
- MapReduce application example
- Conclusion

Introduction

- What kind of data is BigData and where do we find it?
- How do we store BigData?
- How do we process and analyze BigData?
- MapReduce - mechanism for parallel processing of big data

MapReduce paradigm

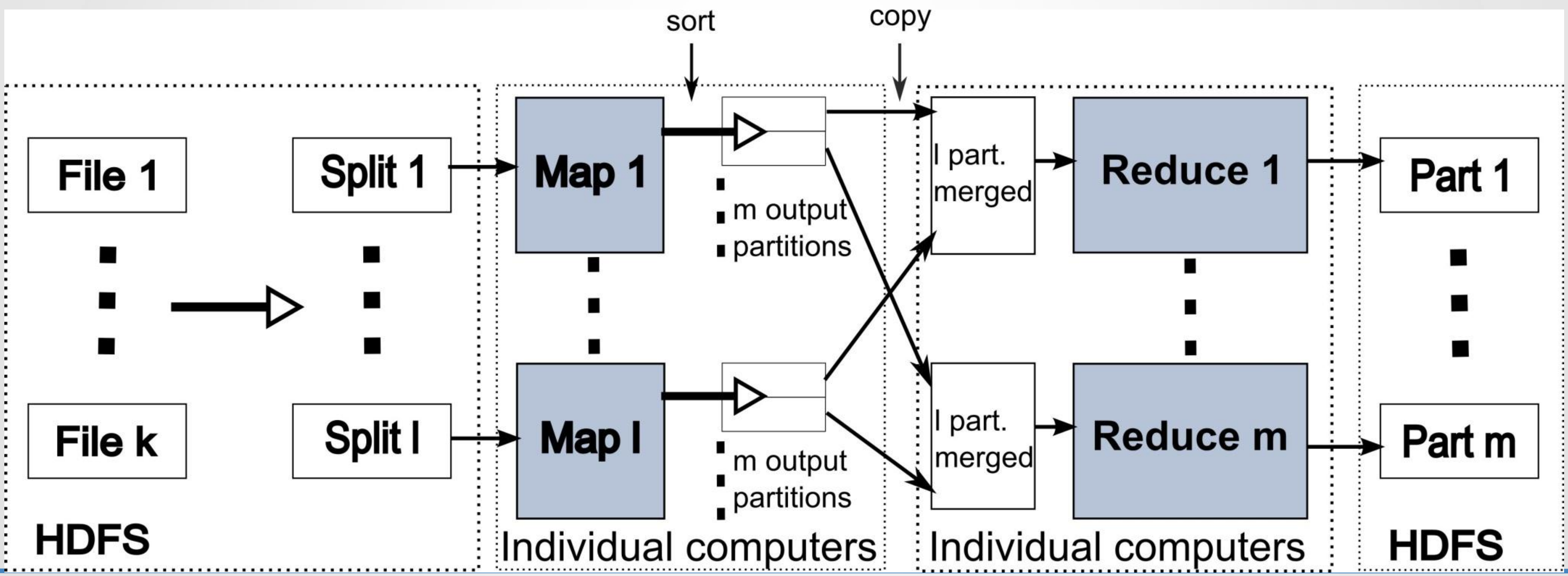
- MapReduce is a programming model and an associated implementation for processing and generating large data sets.
- MapReduce user specifies two functions:
  - *Map*
  - *Reduce*
- We have used MapReduce implemented in Apache Hadoop and distributed in Cloudera Hadoop distribution

Apache Hadoop MapReduce implementation

- Apache Hadoop is an open-source software framework aimed for developing data-intensive distributed applications that can run on large clusters of commodity hardware.
- The Hadoop project is comprised of four modules:
  - Hadoop Common – support for other modules,
  - Hadoop Distributed File System (HDFS),
  - Hadoop YARN - a framework for job scheduling and cluster resource management
  - Hadoop MapReduce

Hadoop MapReduce data flow

- Hadoop divides the input data to be processed into fixed-size pieces called splits
- Each *Map* task runs the *Map* function for each record in the split
- *Map* tasks partition their outputs, each creating one partition for each *Reduce* task.
- Each *Map* task's output is firstly sorted and then transferred across the network to the node where its corresponding *Reduce* task is running. The sorted map outputs are merged, before being passed to the *Reduce* task.

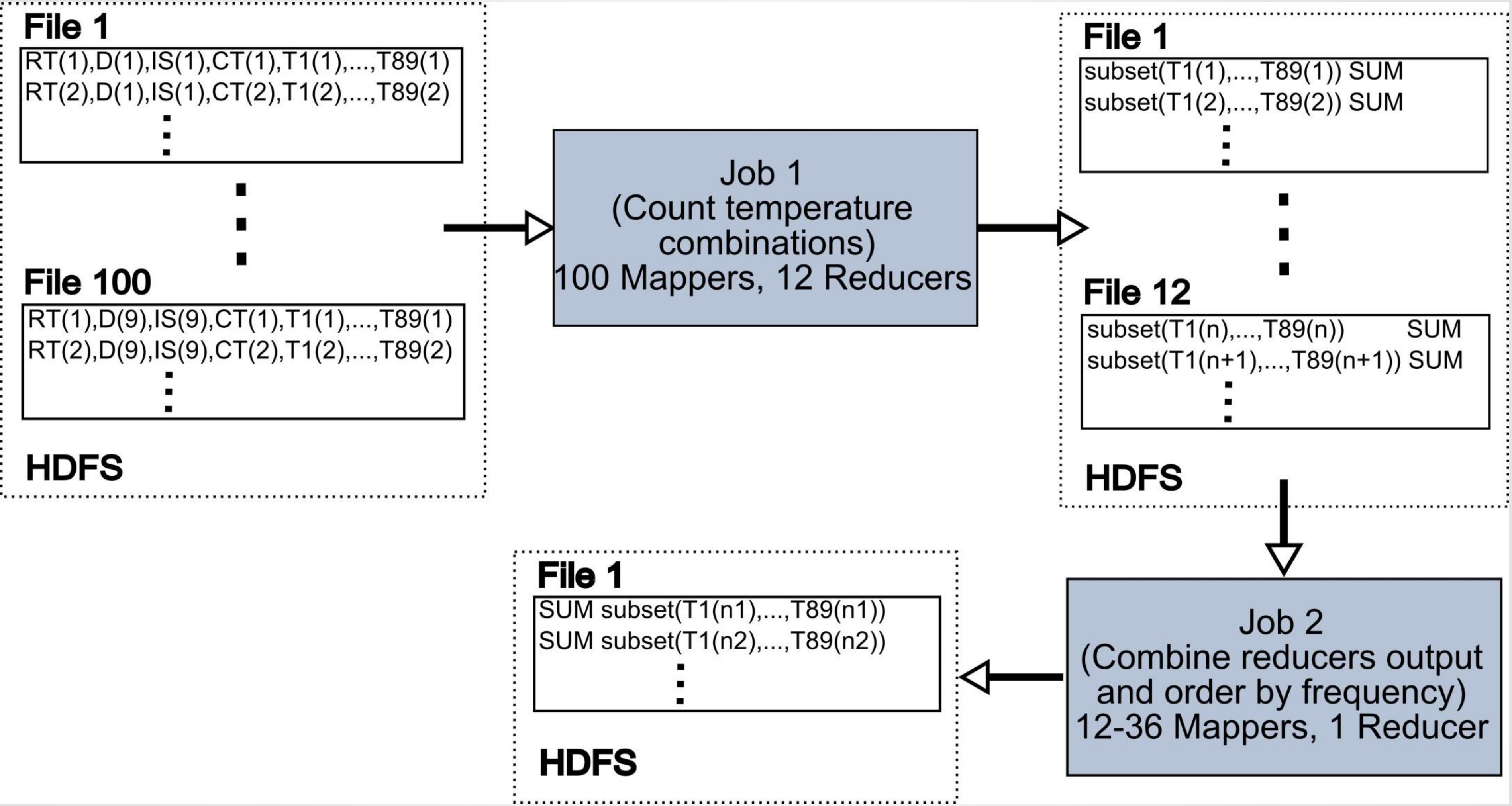


Analyzing computer simulation data with MapReduce

- Input data:
  - Computer simulation of two hours cooling of a human knee after surgery
  - Each data row is composed of following comma separated parameters: RT, D, IS, CT, T1, T2, … , T85
- The results are in 100 files each approximately 44 MB.
- Task: find the number of occurrences for 8 sets of T parameters with the same values – 8 cases.

The MapReduce jobs pipeline

(for solving our test cases)



MapReduce jobs
(implementation details)

- Job 1:
  - Map - Extarction of relevat columns
  - Reduce – counts the number of occurances for each combination of temp.

- Job 2:
  - Map - inverts its key/value pairs
  - Reducer – outputs received key/value pairs
  - Framework does the sorting

```
//Job 1
public void map(LongWritable key,Text value,OutputCollector<Text,IntWritable> output,
Reporter reporter) throws IOException{
        String line = value.toString();
        String[] lineElements  = line.split(",");
        String SearchString = null
        //depending on a case (Table 1) concatenate different lineElements in
        //SearchString
        ...
        word.set(SearchString);
        output.collect(word, new IntWritable(1));
}

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException{
        int sum = 0;
        while (values.hasNext()){
        sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));

}
```

```
//Job 2

public void map(LongWritable key, Text value, OutputCollector<IntWritable,Text> output,
Reporter reporter) throws IOException{

        String line = value.toString();
        //\t is the default delimiter used by a reducer
        String[] lineElements  = line.split("\t");
        output.collect(new IntWritable(Integer.parseInt(lineElements[1])),
                new Text(lineElements[0]));
}
public void reduce(IntWritable key, Iterator<Text> values, OutputCollector<IntWritable,
Text> output, Reporter reporter) throws IOException{
        //there is only one value
        output.collect(key, values.next());
}
```

Results

- The ten highest numbers of temperature occurrences, for each test case:

Case 8	Case 7	Case 6	Case 5	Case 4	Case 3	Case 2	Case 1
294	298	294	298	387	391	8933	11159
224	228	224	228	319	323	8860	11097
211	215	217	227	294	298	8778	10945
181	199	216	221	267	271	8351	10924
168	194	211	220	232	264	7807	10729
165	185	187	215	231	256	7695	10720
161	185	181	215	224	253	7626	10706
159	183	175	199	224	248	7551	10645
158	172	168	199	216	247	7504	10602
154	172	165	195	216	245	7456	10591

Results

(execution times)

- The total time spent for Maps and Reduces in Job 1 and Job 2 for all test cases and on all executing nodes is:
  ts = 9903 + 1264 + 941 + 139 = =12247
- total duration of the complete MapReduce analysis is:
  tm = 377 + 248 = 625 s
- The ratio ts/tm is 19.6.

Job1									
Case:	1	2	3	4	5	6	7	8	Total
Total time spent by all maps in (s)	1,122	1,080	1,119	1,187	1,121	1,287	1,162	1,826	9,903
Total time spent by all reduces (s)	100	80	91	148	108	207	118	413	1,264
Map tasks avg. time (s)	11	10	11	11	11	12	11	18	
The last Map task finished at (s)*	33	31	32	35	33	35	32	54	
Shuffle avg. time (s)	5	3	3	7	5	7	5	14	
The last Shuffle task finished at (s)*	36	33	33	36	35	39	35	57	
Reduce tasks avg. time (s)	2	2	3	5	3	9	4	20	
The last Reduce task finished at(s)*	39	36	37	42	39	49	39	79	
CPU time spent (s)	588	618	667	790	686	933	719	1,494	6,494
Total duration (s)	40	37	38	43	49	51	40	79	377

Job2									
Case:	1	2	3	4	5	6	7	8	Total
Total time spent by all maps in (s)	32	31	51	78	59	184	64	443	941
Total time spent by all reduces (s)	4	4	10	16	12	31	12	50	139
Map tasks avg. time (s)	2	2	3	6	4	7	5	12	
The last Map task finished at (s)*	7	7	12	14	15	13	12	20	
Shuffle avg. time (s)	1	1	6	5	6	7	4	8	
The last Shuffle task finished at (s)*	8	10	15	16	19	22	13	30	
Reduce tasks avg. time (s)	1	1	3	10	5	23	8	40	
The last Reduce task finished at(s)*	10	12	19	26	24	45	21	71	
CPU time spent (s)	7	9	55	95	70	185	73	330	823
Total duration (s)	13	14	22	28	26	48	24	73	248

Conclusion

- We successfully implemented the analysis of a large amount of simulation results with two MapReduce jobs.
- The pipelining between jobs can be further refined if a need occurs.
- The can be applied in a similar way on data sets coming from computer simulations of hydro turbines which is performed at Turboinštitut Ljubljana (TI), Slovenia.
- Further tests on data from TI

Thank You

Ivan Tomašić
Jožef Stefan Institute, Ljubljana, Slovenia

ivan.tomasic@ijs.si