Using Hadoop MapReduce in a Multicluster Environment

I. Tomašić, A. Rashkovska and M. Depolli
Jožef Stefan Institute/Department of Communication Systems, Ljubljana, Slovenia
ivan.tomasic@ijs.si, aleksandra.rashkovska@ijs.si, matjaz.depolli@ijs.si

Abstract - Hadoop MapReduce has become one of the most popular tools for data processing. Hadoop is normally installed on a cluster of computers. When the cluster becomes undersized, it can be scaled by adding new computers and storage devices, but it can also be extended by real or virtual resources from another computer cluster. We present a utilization of the MapReduce paradigm on a Hadoop installation extended across two clusters connected over the Internet. We measured execution times of Map and Reduce tasks in the multicluster environment, and compared them to the corresponding times obtained while only computers from a single cluster are used. The results show that there might be a decrease in MapReduce performance depending on: the concrete data analyses application, the ratio of the number of local and remote computers, and connection bandwidth to remote computers. Additionally, the investigation suggests an upgrade to the Apache Hadoop MapReduce, making it more adjusted to the multicluster environment.

I. Introduction

Apache Hadoop [1] is a highly popular set of open source modules for distributed computing. The key Hadoop components are the Hadoop Distributed File System (HDFS) and the MapReduce - a data-processing component. Since 2004, when it was introduced [2], the MapReduce paradigm has become one of the most popular tools for batch-processing and generating large datasets, mostly because it allows users to build complex distributed programs using a simple model.

Hadoop has proved its ability to store and analyze huge datasets often referred to as the BigData [3]. It has been used by Yahoo and Facebook. If a Hadoop cluster becomes undersized, a commonly used approach is to scale the cluster by adding new computers and storage devices. Other possibility is to resort for resources on another computer cluster. The additional resources can be virtual or real computers.

In our previous work [4], we presented the steps and configurations needed for extending a HDFS installation with computers from another cluster accessed over the Internet. The presented benchmarks results, for one additional computer, showed a drop in read and write operations by approximately one order of magnitude. In the present study, we investigate the applicability and efficiency of the Hadoop MapReduce on the same hardware and network topology (see [4] for details). For the present study, the Cloudera Apache Hadoop distribution CDH 4.1.0 has been used [5].

The research was funded in part by the European Union, European Social Fund, Operational Programme for Human Resources, Development for the Period 2007-2013.

For evaluating the MapReduce performance in a multicluster environment, we have measured Map and Reduce tasks execution times, and compared them to the corresponding times obtained on a single cluster. The multicluster environment is achieved by connecting an additional server using a 100Mb/s link (note that the network bandwidth inside the cluster is 1 Gb/s). We developed MapReduce jobs for analyzing tabular data coming from heat transfer simulation in a biomedical application, in particular, cooling of a human knee after surgery [6]. Similar approaches are applicable also in other scientific areas related to multi-parametric simulations [7], environmental data analysis [8], high energy physics [9], etc.

A. *MapReduce Paradigm*

The MapReduce user specifies two functions called Map and Reduce, which operate on data arranged in key/value pairs. The Map takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce, which aggregates or merges together these values to form a new, possibly smaller, set of values. The Map and Reduce of the Hadoop MapReduce implementation are executed in separate Java Virtual Machines (JVMs) [10]. Their executions are referred to as Map and Reduce tasks. A set of tasks executed for one application are referred to as a MapReduce job.

B. *Apache Hadoop MapReduce Implementation*

At the beginning of a MapReduce job, the Hadoop divides the input data into fixed-size pieces called splits, and creates a separate Map task for each split. Each Map task runs the Map function for each record in the split. The default split size is the same as the default size of an HDFS block, which is 64 MB. In addition to splitting, the Hadoop also performs data locality optimization by running the Map task on the node where the input data resides in the HDFS.

The Map tasks write their outputs to their local disks, and partition their outputs, creating one partition for each Reduce task. Each partition may contain various keys and associated values. Each Map task offers all the records for a given key only to a single Reduce task. This is achieved by packing all the records for a specific key in a single

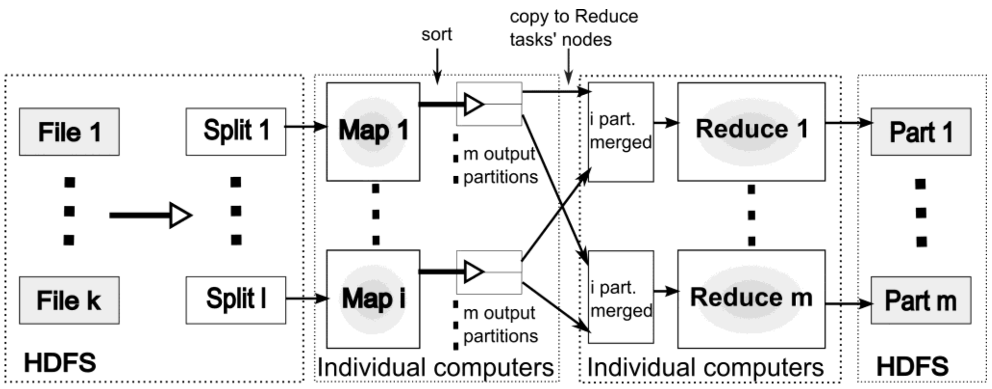


Figure 1. Schematic representation of MapReduce data flow (Combiner functions are not presented).

partition. The records are distributed to Reduce tasks and merged in a way that all records sharing the same key are processed by the same Reduce task. Fig. 1 shows schematically the MapReduce data flow.

In the Apache Hadoop, all Reduce tasks wait until all Map tasks complete or fail [11]. Reduce tasks, unlike Map tasks, cannot convey on data locality because the input to a single Reduce task is generally formed from outputs of multiple Map tasks. Each Map task output is first sorted and then transferred across the network to the node where its corresponding Reduce task is running. On this node, the sorted map outputs are merged before being passed to the Reduce task. The data flow between Map and Reduce tasks is known as “shuffle”, whereas the merging is logged as a part of a Reduce task. The number of Reduce tasks is specified independently for a given job. Each Reduce task outputs a single file, which is usually stored in the HDFS.

Hadoop allows a user to specify an additional function, called Combiner, which receives all the data emitted by the Map tasks on the same node as the input, executes on that node, and forms an output that becomes an input to a Reduce function. The Combiner function is used to achieve data reduction that consequently minimizes data transfer over the network and reduces the impact of the limited communication bandwidth on the performances of a MapReduce job.

II. METHODS

A. System Arhitecture

The utilized system hardware and networking architecture has been described in our previous study [4], where we used the original Apache Hadoop distribution [1]. For the current study, we have used the Cloudera distribution [5] that was installed on a cluster of six computing nodes. The nodes are connected with Gigabit Ethernet, whereas the additional external node is connected through a 100Mb/s channel.

One of the nodes is designated as the namenode, while others are the datanodes. The namenode also hosts the jobtracker. All machines in the cluster run an instance of a datanode and a tasktracker. For a description of the HDFS and MapReduce nodes refer to [11], [12]. After joining the additional node, the obtained Hadop multicluster has been rebalanced by executing Cloudera’s HDFS balancer [13], to allow the data blocks to be equally distributed to the additional node.

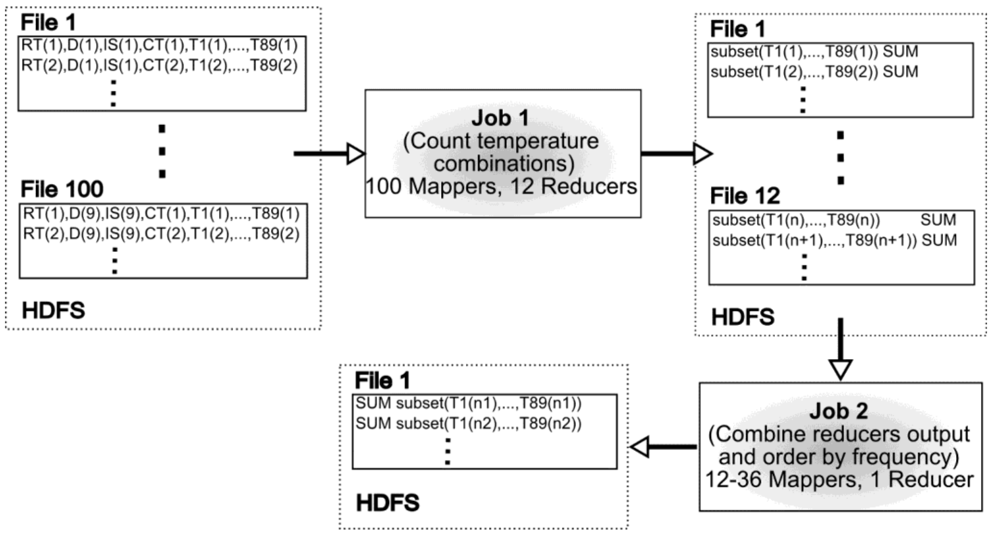
B. Test Dataset

A computer simulation of two hours cooling of a human knee after surgery was performed for 10 different knee sizes, 10 different initial temperature states before cooling, and 10 different temperatures of the cooling pad. This resulted in 1000 simulation cases. The results of those simulation cases were gathered in 100 files. Each file contained 71970 rows or approximately 44 MB of data. Each data row was composed of the following parameters, i.e. columns: RT, D, IS, CT, T1, T2, … , T85, where are: RT - relative time in a simulation case, D - knee size, IS –initial state, CT – cooling temperature, T1-T85 – inner and outer knee temperatures, i.e., temperatures at a particular location in the knee center, 8 locations on the knee skin and 8 respective locations under the cooling pad, all taken in the current and in previous time steps. In order to assess the periodicities in the knee simulation results, we assigned the MapReduce to count the occurrences of the same value arrays for a subset of knee temperatures T; more precisely, to count the occurrences of identical rows after having projected only columns of T that are of interest. We will refer to the number of occurrences of identical rows as temperatures’ frequencies.

We defined and examined 8 cases with different sets of T. The cases are given in Table I.

TABLE I. LIST OF TEST CASES

Case	Parameters
1	T1
2	T1-T5
3	T1,T6,T11,T16,T21
4	T1-T21
5	T1,T6,T11,T16,T21,T46,T51,T56,T61
6	T1-T21,T46-T61
7	T1,T6,T11,T16,T21,T26,T31,T36,T41,T46,T51, T56,T61,T66,T71,T76,T81
8	T1-T85



File 1
RT(1),D(1),IS(1),CT(1),T1(1),...,T89(1)
RT(2),D(1),IS(1),CT(2),T1(2),...,T89(2)
File 100
RT(1),D(9),IS(9),CT(1),T1(1),...,T89(1)
RT(2),D(9),IS(9),CT(2),T1(2),...,T89(2)
HDFS
Job 1
(Count temperature combinations)
100 Mappers, 12 Reducers
File 1
subset(T1(1),...,T89(1)) SUM
subset(T1(2),...,T89(2)) SUM
File 12
subset(T1(n),...,T89(n)) SUM
subset(T1(n+1),...,T89(n+1)) SUM
HDFS
Job 2
(Combine reducers output and order by frequency)
12-36 Mappers, 1 Reducer
File 1
SUM subset(T1(n1),...,T89(n1))
SUM subset(T1(n2),...,T89(n2))
HDFS

Figure 2. MapReduce jobs pipeline.

C. MapReduce jobs

The MapReduce jobs pipeline for solving our test cases is illustrated in Fig. 2. Job 1 counts the occurrences of the same value arrays for a subset of temperatures, whereas Job2 combines outputs from Job 1 and orders the combined rows by temperature frequency.

The sizes of the input files are smaller than the HDFS block size (in our case: 64 MB). Hence, the number of input Map tasks in Job 1 is equal to the number of input files [14], i.e., each input file is processed by a different Map task. The number of Reduce tasks is set to 12 for Job 1, hence the output of Job 1 now consists of 12 files. Each file contains a unique combination of temperatures and the number of their occurrences. Job 2 combines Reduce tasks' outputs from Job 1 into a single file (in Job 2, the number of Reduce tasks is set to 1). It also sorts the input columns in the output file by temperatures' frequencies. The number of Map tasks in Job 2 depends on the test case (Table I) and varies between 12 for Case 1 and 36 for Case 8. Job 1 has the same Combiner and Reduce functions, whereas the Combiner function for the Job 2 is not specified, since there are no multiple rows for the same key in the Map tasks output data. The block diagram of the distribution of files and the implementation of both Jobs is given in Fig. 3.

In the Map function of Job 1, from each input row, only the relevant columns (see Table I) are extracted. For example, in test case 2, only the columns belonging to T1-T5 will be extracted in the SearchString variable. Reduce functions sum, i.e., count the number of occurrences of each combination of temperatures (the key) and outputs it as the new value for the current key. Because all the values for one key are processed by a single Reduce task, it is evident that the output of Job 1 consists of unique combinations of temperatures and the number of their occurrences.

In Job 2, the Map function inverts its key/value pairs, making temperature occurrences the keys, and emits them to the Reduce function that outputs the received pairs. The sorting by the occurrence is done by the framework as explained in Section I.B.

Note that default MapReduce configuration parameters have been used. For the specification please refer to the Cloudera documentation [5].

```
//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 I) 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());
}
```

Figure 3. Java code segments of Map and Reduce tasks for Job 1 and Job 2.

TABLE II. MAPREDUCE TASKS EXECUTION TIMES DIFFERENCE BETWEEN MULTICLUSTER AND SINGLE CLUSTER CASE.

	Job 1								Job 2							
Case:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Total time spent by all maps (s)	96	77	53	43	25	69	71	52	1	0	-4	-1	1	11	6	38
Total time spent by all reduces (s)	5	-12	56	97	131	308	29	2082	0	1	5	83	15	2	0	284
Map tasks avg. time (s)	1	1	0	1	0	1	0	0	0	0	-1	0	0	1	1	1
Worse performing map task (s)	1	1	1	-1	0	-1	3	0	0	0	0	0	0	0	0	4
Shuffle avg. time (s)	0	-1	0	1	1	2	3	14	0	0	0	14	0	13	0	12
Worse performing Shuffle (s)	1	-2	0	0	0	4	6	50	0	0	0	14	0	13	0	12
Reduce tasks avg. time (s)	0	0	4	7	10	22	0	160	0	0	5	69	15	-11	-1	272
Worse performing reduce task (s)	0	0	5	38	14	34	-1	255	0	0	5	69	15	-11	-1	272
CPU time spent (s)	-2	-8	-2	-2	0	-8	-6	4	0	1	-3	4	4	-2	1	16
Total duration (s)	**0**	**-1**	**8**	**42**	**16**	**37**	**6**	**264**	**0**	**0**	**2**	**83**	**14**	**7**	**0**	**284**
No. Map tasks on remote node	**4**	**8**	**5**	**3**	**2**	**3**	**9**	**6**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**2**
No. Reduce tasks on remote node	**2**	**2**	**2**	**2**	**2**	**2**	**0**	**2**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**

The last two rows show the number of Map/Reduce tasks executed on the remote node. A negative value indicates that a particular time value vas smaller for the multicluster.

III. RESULTS

Table II shows MapReduce tasks' execution times for Job 1 and Job 2, for each test case. The presented time values are the differences between the execution times for multicluster and single cluster runs. The table also shows the total CPU time differences and total jobs' durations differences. The last two rows, relevant only for the multicluster run, indicate the number of Map and Reduce tasks executed on the remote computer.

Fig. 4 shows total durations of the MapReduce jobs in the single cluster and in the multicluster run, for each test case.

IV. DISCUSSION

It is evident from Fig. 4 that the introduction of a remote node may introduce a significant slowdown in MapReduce performance. The slowdown depends on two parameters. One is the amount of data coming out from Map tasks, whereas the other is the number of tasks executed on a remote node (the last two rows in Table II). The amount of data coming from Map tasks increases with the test case, as the number of columns in the data increases (see Table I). The jobs' execution times are similar until Case 4, regardless whether there are tasks' executions on the remote node or not. The data amount significantly increases in Case 4, which evidently influences the execution times of the Shuffle and Reduce tasks (Table II). This result was expected, because in this case the remote node was allocated for Map and Reduce

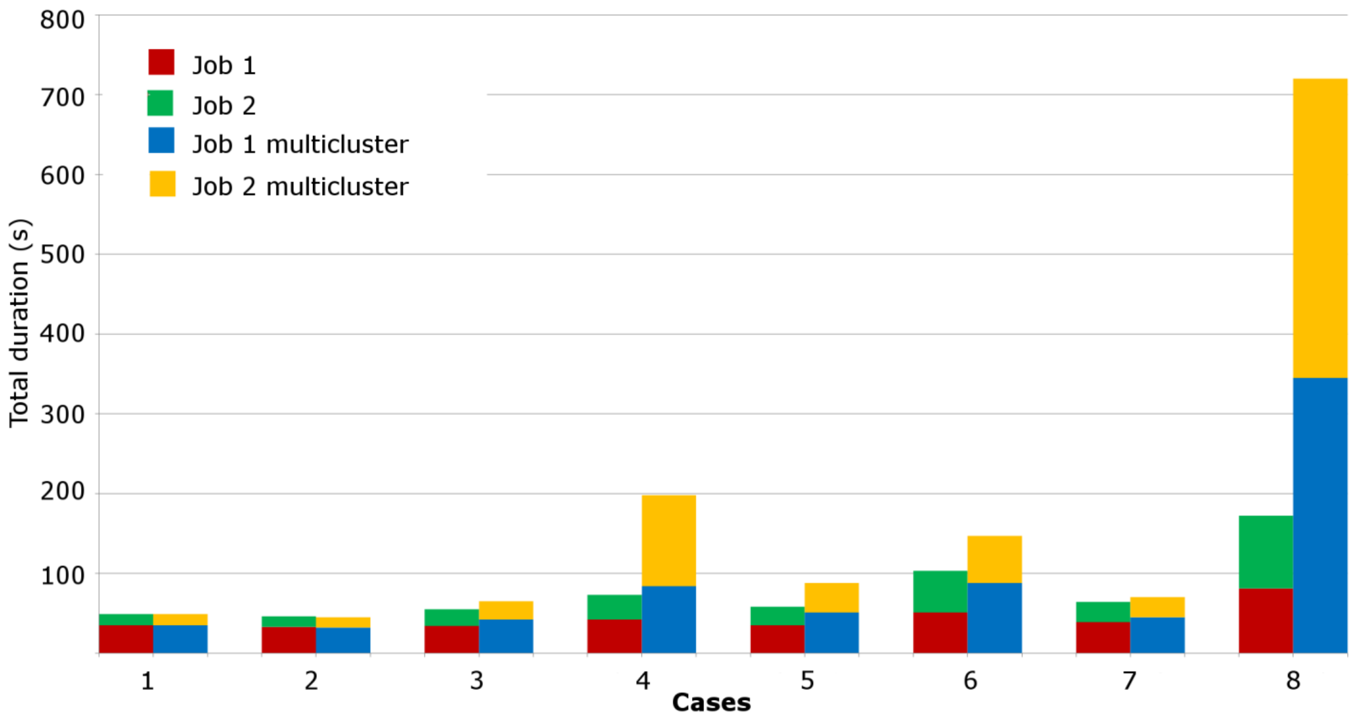


Figure 4. Jobs' total execution times for the single cluster (left columns) and for the multicluster (right columns)

tasks, which imposed a heavy data communication between local and remote nodes (Fig. 1) with significantly slower communication than in the local cluster.

The same pattern can be observed for all the succeeding cases. Particularly interesting is Case 7 with a low increase in execution time for Job 1, and no increase in execution time for Job 2. Table II reveals that there were no Reduce tasks for Job 1 on the remote node, which restricted the necessary communication with the remote node only in one direction – from the remote node to the local nodes. That obviously decreased the performance but not as significantly as if the remote node was also used for Reduce tasks. For Job 2, all processing was local, so there was no decrease in performance.

For Case 8, the amount of the data is the highest and consequently is the slowdown. For Job 2, there were no Reduces on the remote node, but the worst Reduce task's execution time is relatively very long compared to the increase in the Shuffle tasks' execution times. We suspect that it is probably because of possible instabilities in the Reduce tasks' merging procedures, influenced by the introduction of the remote node.

A comparison of the MapReduce performance in our multicluster, to the performances previously investigated for the HDFS [4], reveals that the MapReduce performance is not as decreased as the HDFS performance. Even in the Case 8, which is the worst case, it is not decreased by a level of magnitude as the HDFS performance approximately is. This result was to be expected, because Hadoop does its best to run MapReduce jobs locally on the computers where the data resides. Furthermore, the communication to and from the distant node is consisted of the data coming out of Map tasks that can be, depending of a job at hand, significantly smaller than the source data. Additionally, the amount of data sent to Reduce tasks may be decreased by using Combiner functions.

The addition of the remote node in the presented use case means increase of the cluster storage resources for 20%, since the initial number of computes is five. The small number of the computers used, means that the chances for the remote computer to be allocated for a Map or Reduce task are higher than in the case when there are more computers in the cluster. Hence, the MapReduce performances should be higher as the ratio of the number of local and remote computers increases.

V. CONCLUSION

Adding remote nodes to a Hadoop cluster is a viable option for increasing storage resources. The MapReduce performances may or may not decrease, which depends on the concrete data analyses application, the ratio of the number of local and remote computers, and on connection bandwidth to remote computer or computers.

We have observed that the Reduce tasks execution times may increase even without obvious justifications in communication latencies. We suspect this result is due to possible instabilities in the merging processes on Reduce nodes. This issue needs to be further explored.

The Apache Hadoop MapReduce implementation may be upgraded for a multicluster environment with a decision algorithm that would prefer local computers to the remote. The algorithm should take into account the amount of data expected to come from Map or Combiner tasks into, which depends on the concrete application, and consider the connection bandwidth to the remote nodes.

ACKNOWLEDGMENT

The authors are grateful to Turboinštitut d.d. and CHS d.o.o., both from Ljubljana, Slovenia, for granting a full access to their storage and computational resources.

REFERENCES

[1] "Welcome to Apache™ Hadoop®!," Oct., 2012; http://hadoop.apache.org/.

[2] J. Dean, and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in OSDI'04, 2004.

[3] B. Franks, "What is big data and why does it matter?," Taming the big data tidal wave: finding opportunities in huge data streams with advanced analytics, pp. 3-29, Hoboken, New Jersey: John Wiley & Sons, Inc., 2010.

[4] I. Tomasic, J. Ugovsek, A. Rashkovska, and R. Trobec, "Multicluster Hadoop Distributed File System," in MIPRO, 2012 Proceedings of the 35th International Convention, 2012, pp. 314-318.

[5] I. Cloudera. "CDH Proven, enterprise-ready Hadoop distribution – 100% open source," Oct, 2012; http://www.cloudera.com/hadoop/.

[6] R. Trobec, M. Šterk, S. Almawed, and V. M., "Computer simulation of topical knee cooling," Comput. biol. med, vol. 38, pp. 1076-1083, 2008.

[7] G. Kosec, Šarler, Božidar, "Solution of a low Prandtl number natural convection benchmark by a local meshless method.," International journal of numerical methods for heat & fluid flow, vol. 23, pp. 22, 2013.

[8] U. Stepišnik, and G. Kosec, "Modelling of slope processes on karst," Modeliranje pobočnih procesov na krasu, vol. 40, no. 2, pp. 267-273, 2011.

[9] L. Wang et al., "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," Future Generation Computer Systems, vol. 29, no. 3, pp. 739-750, 2013.

[10] T. White, "How MapReduce Works," Hadoop: The Definitive Guide, pp. 167-188, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2010.

[11] T. White, "MapReduce," Hadoop: The Definitive Guide, pp. 15-40, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2010.

[12] T. White, "The Hadoop Distributed Filesystem," Hadoop: The Definitive Guide, pp. 41-73, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2010.

[13] I. Cloudera, "Cloudera Manager Free Edition User Guide," Dec, 2012.

[14] T. White, "MapReduce Types and Formats," Hadoop: The Definitive Guide, pp. 189-224, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2010.