HCMC University Of Technology
Faculty of Computer Science & Engineering

Course: Parallel Processing & Distributed Systems Hadoop & Mapreduce

Thin Nguyen, Tu Do

November 22, 2016

1 Distributed File System (DFS)

To exploit cluster computing, files must look and behave somewhat differently from the conventional file systems found on single computers. This new file system, often called a distributed file system or DFS (although this term has had other meanings in the past), is typically used as follows.
There are several distributed file systems of the type we have described that are used in practice. Among these:

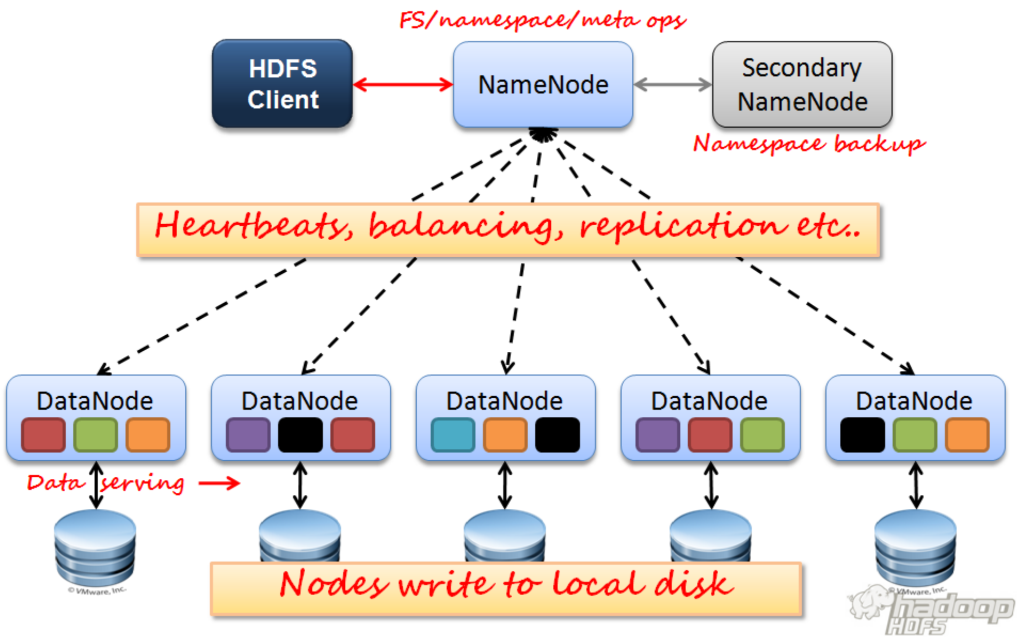
1. The Google File System (GFS), the original of the class.
2. Hadoop Distributed File System (HDFS), an open-source DFS used with Hadoop, an implementation of MapReduce (see Section 2.2) and distributed by the Apache Software Foundation.
3. CloudStore, an open-source DFS originally developed by Kosmix.

2 Hadoop Distributed File System (HDFS)

Key HDFS Features:

- **Scale-Out Architecture** - Add servers to increase capacity
- **High Availability** - Serve mission-critical workflows and applications
- **Fault Tolerance** - Automatically and seamlessly recover from failures
- **Flexible Access** Multiple and open frameworks for serialization and file system mounts
- **Load Balancing** - Place data intelligently for maximum efficiency and utilization
- **Tunable Replication** - Multiple copies of each file provide data protection and computational performance

- **Security** - POSIX-based file permissions for users and groups with optional LDAP integration



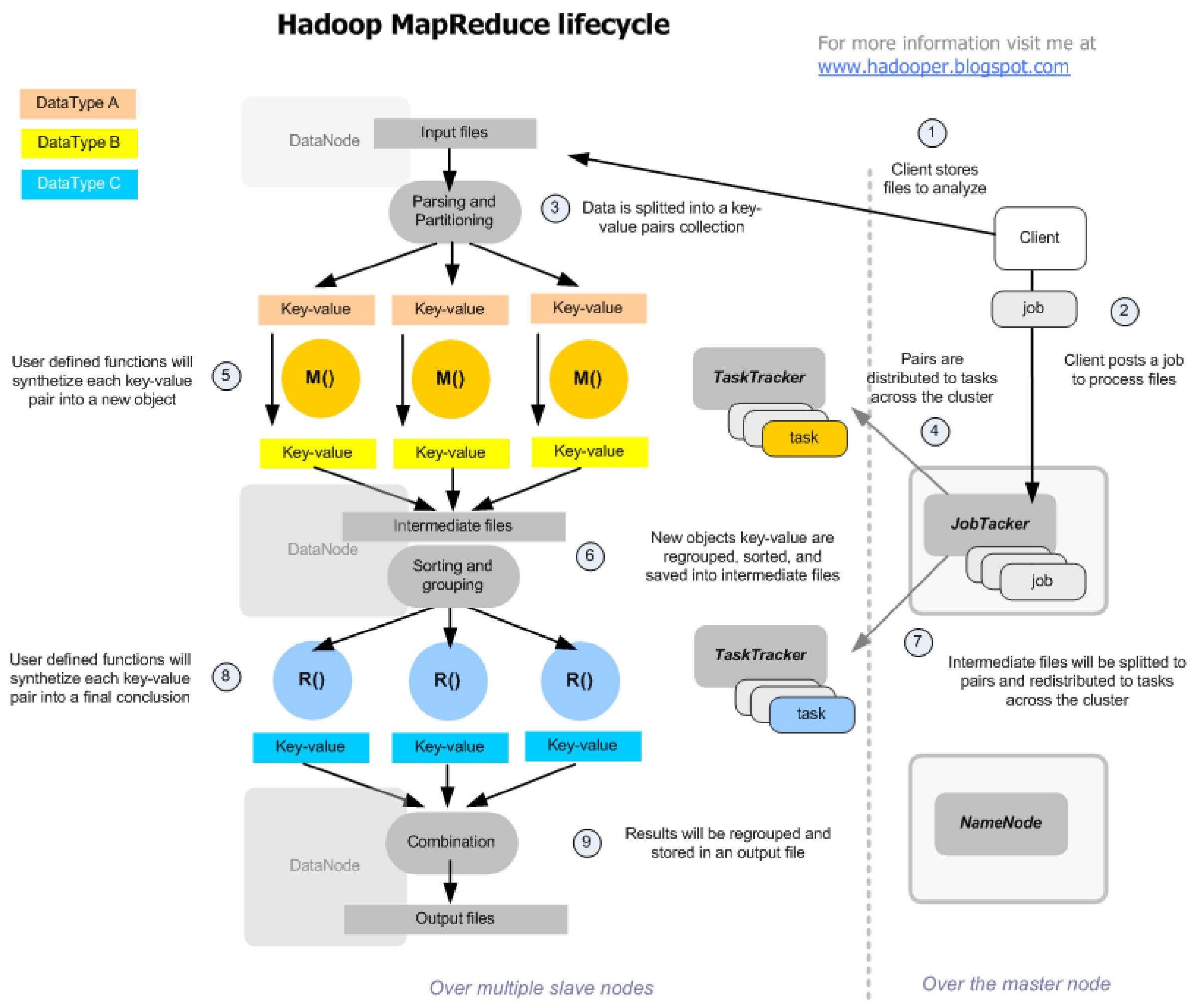
3 MapReduce

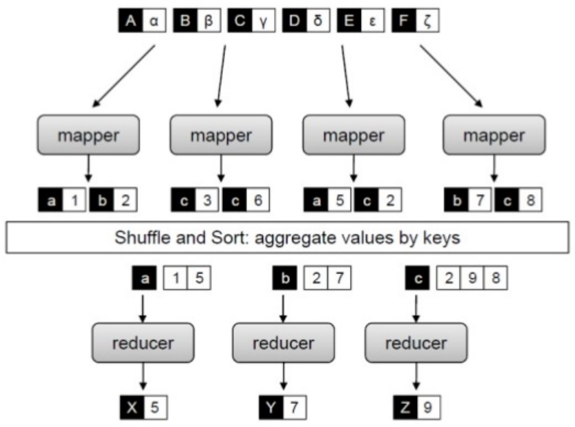
MapReduce is a style of computing that has been implemented in several systems, including Googles internal implementation (simply called MapReduce) and the popular open-source implementation Hadoop which can be obtained, along with the HDFS file system from the Apache Foundation. You can use an implementation of MapReduce to manage many large-scale computations in a way that is tolerant of hardware faults. All you need to write are two functions, called Map and Reduce, while the system manages the parallel execution, coordination of tasks that execute Map or Reduce, and also deals with the possibility that one of these tasks will fail to execute. In brief, a MapReduce computation executes as follows:

1. Some number of Map tasks each are given one or more chunks from a distributed file system. These Map tasks turn the chunk into a sequence of *key-value* pairs. The way key-value pairs are produced from the input data is determined by the code written by the user for the Map function.

2. The key-value pairs from each Map task are collected by a master controller and sorted by key. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task.

3. The Reduce tasks work on one key at a time, and combine all the values associated with that key in some way. The manner of combination of values is determined by the code written by the user for the Reduce function.


Hadoop MapReduce lifecycle
For more information visit me at
www.hadooper.blogspot.com
DataType A
DataType B
DataType C
DataNode
Input files
Parsing and Partitioning
3 Data is splitted into a key-value pairs collection
1 Client stores files to analyze
Client
job
2 Client posts a job to process files
Key-value
Key-value
Key-value
5 User defined functions will synthetize each key-value pair into a new object
M()
M()
M()
TaskTracker
task
Pairs are distributed to tasks across the cluster
4
JobTacker
job
Key-value
Key-value
Key-value
Intermediate files
DataNode
Sorting and grouping
6 New objects key-value are regrouped, sorted, and saved into intermediate files
7 Intermediate files will be splitted to pairs and redistributed to tasks across the cluster
8 User defined functions will synthetize each key-value pair into a final conclusion
R()
R()
R()
TaskTracker
task
Key-value
Key-value
Key-value
NameNode
DataNode
Combination
9 Results will be regrouped and stored in an output file
Output files
Over multiple slave nodes
Over the master node



3.1 Combiners

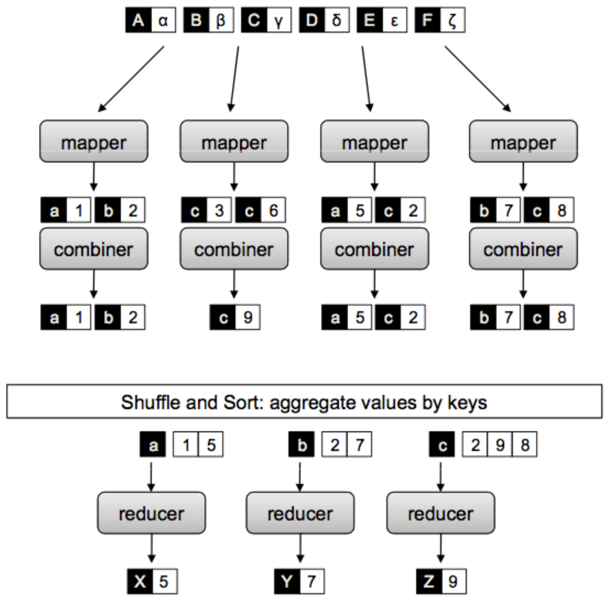
When the Reduce function is associative and commutative, we can push some of what the reducers do to the Map tasks.

- **Commutative** $a + b = b + a$

- **Associative** $a + (b + c) = (a + b) + c$

For example, in case of Max function and Mean function, they are both commutative. However, only Max function is associative while Mean function is not.

$$max(max(a, b), max(c, d, e)) = max(a, b, c, d, e)$$
$$mean(mean(a, b), mean(c, d, e))! = mean(a, b, c, d, e)$$



Advantage of combiners: Network transmission are minimized
Disadvantage of combiners:

- It does not guarantee the execution of a combiner can be executed 0, 1 or multiple times on the same input.
- Key-value pairs emitted from mapper are stored in local file system, and the execution of the combiner could cause expensive IO operations.

3.2 TopN Example

Find the top-n used words of text file.
Input Data: pg201.txt

3.2.1 TopN Mapper

```
public static class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private String tokens = "[_|$#<>\\^=\\[\\]\\*/\\\\,;,.\\-:()?!\"']";

    @Override
    public void map(Object key, Text value, Context context) throws IOException,
            InterruptedException {
        String cleanLine = value.toString().toLowerCase().replaceAll(tokens, " ");
        StringTokenizer itr = new StringTokenizer(cleanLine);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken().trim());
            context.write(word, one);
        }
    }
}
```

3.2.2 TopN Reducer

```
public static class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private Map<Text, IntWritable> countMap = new HashMap<>();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
        IOException, InterruptedException {

        // computes the number of occurrences of a single word
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }

        // puts the number of occurrences of this word into the map.
        // We need to create another Text object because the Text instance
        // we receive is the same for all the words
        countMap.put(new Text(key), new IntWritable(sum));
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        Map<Text, IntWritable> sortedMap = sortByValues(countMap);

        int counter = 0;
        for (Text key : sortedMap.keySet()) {
            if (counter++ == 20) {
                break;
            }
            context.write(key, sortedMap.get(key));
        }
    }
}
```

3.2.3 TopN Private Function sortByValues

```
private static <K extends Comparable, V extends Comparable> Map<K, V>
sortByValues(Map<K, V> map) {
    List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet());

    Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {
        @Override
```

```
            public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {
                return o2.getValue().compareTo(o1.getValue());
            }
        });

        //LinkedHashMap will keep the keys in the order they are inserted
        //which is currently sorted on natural ordering
        Map<K, V> sortedMap = new LinkedHashMap<K, V>();

        for (Map.Entry<K, V> entry : entries) {
            sortedMap.put(entry.getKey(), entry.getValue());
        }

        return sortedMap;
}
```

3.2.4 TopN Combiner

```
public static class TopNCombiner extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
    IOException, InterruptedException {

        // computes the number of occurrences of a single word
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

3.3 Exercises

3.3.1 Mean

Find the mean max temperature for every month
Input Data: milano_temps.csv
Temperature in Milan
(DDMMYYYY,MIN,MAX)
01012000,-4.0,5.0
02012000,-5.0,5.1
03012000,-5.0,7.7
04012000,-3.0,9.7
05012000,-0.8,9.0
...

Result should be
022012 7.230769230769231
022013 7.2
022010 7.851851851851852

```
022011 9.785714285714286
032013 10.741935483870968
032010 13.133333333333333
032012 18.548387096774192
032011 13.741935483870968
022003 9.278571428571428
...
```

3.3.2 Mean

Combine information from the users file with information from the posts file
Input Data - Users file: forum_users.tsv
“user_ptr_id” “reputation” “gold” “silver” “bronze”

```
“100006402” “18” “0” “0” “0”
“100022094” “6354” “4” “12” “50”
“100018705” “76” “0” “3” “4”
...
```

Input Data - Posts file: forum_nodes_no_lf.tsv
“id” “title” “tagnames” “author_id” “body” “node_type” “parent_id” “added_at” “score”

```
“5339” “Whether pdf of Unit and Homework is available?” “cs101 pdf” “100000458” “” “question” “
N” “2012-02-25 08:09:06.787181+00” “1”
...
```

Result should be
USER_ID REPUTATION SCORE

```
00081537 1019 3
100011949 12 1
100105405 36 1
...
```