The Distribution Problem Of Unstructured Data When Solving Data Mining Tasks On Computer Clusters

Zein Ali Najyevich[1] and Borisova Svetlana Vyacheslavovna[1]

[1] Moscow Power Engineering Institute (National Research University), Krasnokazarmennaya 14, Moscow, Russian Federation
http:\\www.mpei.ru

Abstract. In the modern world, the amount of information is growing exponentially, with every minute new data appears for further storage, processing and analysis.Big data sources generate huge amounts of data and most of them are presented in a format that does not correlate well with traditional structured database formats. The three main “commandments” of big data (3V) are Volume (large amount of information), Variety (heterogeneity and lack of structure) and Velocity (data processing speed) [1].
Imagine that there is a large amount of published information that needs to be sorted into sections, which will greatly speed up and simplify its reading and processing [2]. By machine learning can be attributed various methods of cluster analysis, with which you can automate the classification by finding patterns or patterns that describe this data. Using the method of cluster analysis of "K-means", it is possible to achieve the distribution of the K-number of clusters at the greatest possible distance from each other, thereby allowing us to assess how much they differ. Ideally, the differences will be quite large, which will show the quality of execution, thereby significantly accelerating the process of analysis and information content.

Keywords: Big data, Hadoop, clustering, k-means, Map Reduce, data mining.

1 Introduction

Big data is a combination of technologies that are designed to perform three operations. First of all, it handles large amounts of data compared to “standard” scenarios. Secondly, to be able to work with quickly loaded data in very large volumes. Thirdly, they should be able to work with structured and poorly structured data in parallel threads and different aspects. Big data suggests that the input algorithms receive a stream of not always structured information and that more than one idea can be extracted from it.

Data mining — a complex of methods and decision algorithms such as clusters, decision trees, the associative method, genetic algorithms and others. The main goal of the data mining task is to identify the most suitable method, the results of which allow you to find the best prediction or classification result in a minimum amount of

ICID-2019 Conference

time. To achieve this goal, it is necessary to conduct a comparative analysis of various ways of machine learning.

MapReduce is a programming model suitable for processing of big data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel by definition, and they are very useful for performing large-scale data analysis using multiple machines in the cluster.

MapReduce is a parallel computing model used to process very large data.

2 MapReduce stages

Processing and analysis of large amounts of data occurs in three stages.

1. Stage Map. The work of this stage is to filter data.

The Map operation converts the input key / value pair into a set of intermediate pairs. The MapReduce library groups all intermediate values associated with the same intermediate Key # 1 and passes them to the Reduce function.

2. Stage Shuffle. The output of the Map function is sorted by “baskets”, and each “basket” corresponds to one key obtained in the previous step.

3. Stage Reduce. Each “basket” with values gets to the input of the Reduce function, where the user sets the function and calculates the final result for each of the baskets. All values obtained at this stage are the final result of the MapReduce model. [3]

The Reduce function combines key and a set of values for this key that presents the smallest possible set of values. Usually, each call to Reduce produces an output value of 0 or 1. Intermediate values are passed to the user's Reduce function. This allows us to process lists of values that are too large to fit in memory.

Exactly the same steps are performed on ETL systems when extracting, converting, and loading data. A MapReduce nask produces some useful information from the raw data that other storage systems consume. In a sense, any MapReduce implementation can be considered a parallel infrastructure for executing ETL procedures.

There have been attempts to implement ETL procedures inside the database server using SQL. But historically, industrial-grade ETL systems have been separate from DBMSs. Typically, DBMSs do not attempt to perform ETLs, and ETL systems do not support DBMS functions.

The advantage of MapReduce model is that these stages can be performed in parallel on multiple nodes, which makes this system highly scalable and reliable.

The main disadvantage of this model can be that the intermediate results of Map are saved to disk, which leads to an increase in operating time.

3 MapReduce problem

Programs using MapReduce will not always work fast. The main advantage of this programming model is the optimized distribution of data between nodes and the small amount of code that a programmer needs to implement in . However, in practice, the user of the program must consider the data distribution stage, in particular, the data

separation function and the amount of data at the output of the Map function, that can greatly affect performance. Additional modules, such as the Combiner feature, can help reduce the amount of data written to disk and transmitted over the network.

When writing a program, the user must find and choose a good compromise between computational and communication complexity. Communication complexity is superior to computational complexity, and many MapReduce implementations have been developed to record all communications in distributed storage for recovery.

For tasks that are solved quickly on unallocated systems, where the input data is placed in the RAM of one computer or a small cluster, the MapReduce framework will be inefficient. Since these frameworks are designed to be able to recover entire cluster nodes during computations, they write intermediate results of work to the distributed storage. In MapReduce tasks there is protection against crashes, but it is a very expensive procedure and pays off only when many computers are involved in the calculations, and when one of them fails, it is easiest to restart the task assigned to it on another node.

In some MapReduce tasks, you can see the following situation: one or several Reduce tasks run much longer than others, as a result, they prevent the next stage in data processing from starting. We can assume that all this is due to the fact that the volumes of processed data on different reducers are very different, due to the unsuccessful distribution of intermediate keys.

4 Word count task

A classic and at the same time simple example of a MapReduce program is "Word count", the task is about to count the number of occurrences of a word in a given text. We can briefly describe the work of this task at the Map stage: a string of text goes into the input, the map function breaks the string into separated words and counts them, then we get an array [word, 1] that presents the fact that the "word" was located in the text. At the reduce stage, intermediate pairs are grouped by word and all units are summed, and as a result a pair is formed (word, number of its occurrences in the text). We will write this program, and then execute it first in SQL, and then in java, then lunched on the Hadoop platform.

For the experiment we are using a computing cluster of 10 nodes with 35 cores; The amount of RAM: 48 GB. The SQL script is presented in Table 1.

Table 1. SQL script

```
drop table if exists [dbo].[text]
create table [dbo].[text] (line varchar(max))
load data inpath '' overwrite into [dbo].[text]

drop table if exists [dbo].[word_counts]
create table [dbo].[word_counts](word varchar(256), cnt
int, PRIMARY KEY CLUSTERED(word));
```

```
with words AS
(
    select string_split(line, ' ') as word
    from [dbo].[text];
),
with words_counts AS
(
    select word, COUNT(*) as cnt
    from words
    group by word;
)

INSERT INTO [dbo].[word_counts](word, cnt)
Select word, cnt
from word_counts
```

The JAVA script is presented in Table 2.

Table 2. JAVA script

```
public class WordCountJob extends Configured implements
Tool
{
  static public class WordCountMapper extends Mapper <
LongWritable, Text, Text, IntWritable >
{
  private final static IntWritable one = new IntWrita-
ble(1);
  private final Text word = new Text();
  @Override 7

protected void map(LongWritable key, Text value, Con-
text context) throws IOException, InterruptedException
{
  // breaking the string into words and 1: word, 1
      StringTokenizertokenizer = new StringTokeniz-
er(value.toString());
  while (tokenizer.hasMoreTokens())
  {
    text.set(tokenizer.nextToken());
    context.write(text, one);
  }
} //map
} //WordCountMapper
static public class WordCountReducer extends Reducer <
Text, IntWritable, Text, IntWritable >
{
```

```
  @Override
  protected void reduce(Text key, Iterable < IntWrita-
ble > values, Context context) throws IOException, In-
terruptedException
  { // all units from one word come to the reducer
    intsum = 0;
    for (IntWritable value: values)
    { sum += value.get(); }
    context.write(key, new IntWritable(sum));
  } //reduce
} //WordCountReducer
@Override
public int run(String[] args) throws Exception
{
  Job job = Job.getInstance(getConf(), "WordCount");
  job.setJarByClass(getClass()); //путь до файла
  TextInputFormat.addInputPath(job, new Path(args[0]));
// input string
  job.setInputFormatClass(TextInputFormat.class);
//MapReduce classes
       job.setMapperClass(WordCountMapper.class);
  job.setReducerClass(WordCountReducer.class);
  job.setCombinerClass(WordCountReducer.class);
//result file  TextOutputFormat.setOutputPath(job, new
Path(args[1]));
  job.setOutputFormatClass(TextOutputFormat.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  return job.waitForCompletion(true) ? 0 : 1;
} //run
public static void main(String[] args) throws Exception
{
  int exitCode = ToolRunner.run( new WordCountJob(),
args);
  System.exit(exitCode);
} //main
} //WordCountJob
```

The experiment was carried out on log-files (Hadoop + Yarn), consisting of 5,000,000 lines, occupying about 457 megabytes, received from all events in the program, which builds "data marts" in the Hadoop environment. This log is well suited to our experiment, since this file contains both unique words and many replicated ones. All words are separated by spaces.

After lunching the program, we can construct two histograms: the first histogram with the number of intermediate values displays the "load distribution of the Reduce task" on each of the ten nodes of the cluster, where reducers are displayed on the X

axis, and the number of intermediate ones is displayed on the Y axis values (millions) (see Fig. 1).

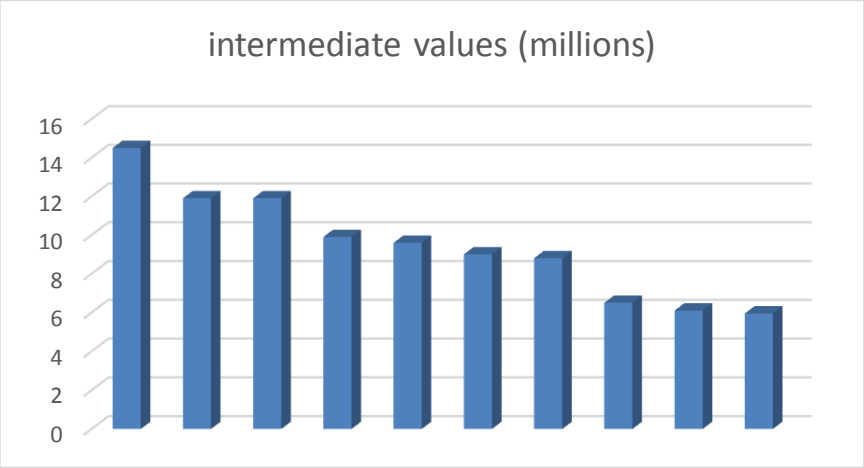

Fig. 1. A histogram of the load distribution of Reduce tasks.

This histogram maps each Reduce machine to the number of intermediate values that it received.

The second histogram (Fig.2) represents the time spent on the Reduce task, where on the X axis are Reducers and on the Y axis are seconds.

As well we can present the duration of Reduce task performed on each of the cluster nodes (see Fig. **1**2).

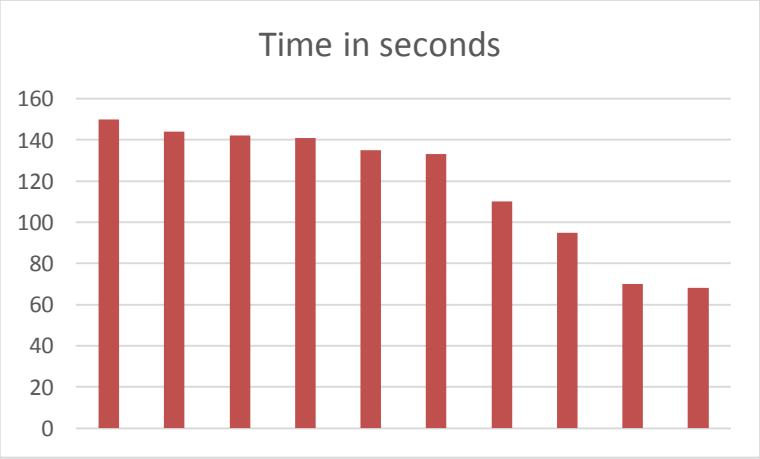

Fig. 2 Time spent on performing the Reduce tasks.

This histogram shows how many seconds the Reduce task performed on each of the cluster nodes. All Reducers in the histograms (Fig. 1 and Fig. 2) are sorted in descending order of the execution time of the Reduce task.

5 Conclusion

After analyzing the histogram from Fig. 1, we can pay attention to the fact that the computational load on the nodes is not uniform. As a result, the time required to complete the operation by the first machine differs from the tenth by more than two times. The full cycle of the Reduce task will depend on the slowest node. Therefore, there are prerequisites for a more uniform distribution of data across all nodes, since this will allow more efficient execution of such programs and their execution time can be significantly reduced. One of the solutions to this problem can be a statistical method for distributing intermediate keys and values that were obtained in previous runs to optimize the load on Reducers. After solving the problem of optimizing the load on Reducers, you can continue to research Data Mining and implement (investigate) various methods of cluster analysis, including the k-means method and its modifications.

6 Additional features

Just a couple of years ago, database management systems would simply be superseded by MapReduce technologies. This caused a natural discontent among the database community, whose reputable representatives tried to prove that trying to replace a DBMS with some MapReduce implementation was immoral, and inefficient.

However, it soon became clear that MapReduce technology could be useful for the parallel DBMS themselves. In many respects, the formation and implementation of this idea was promoted by start-up companies, introducing new massively parallel analytical DBMSs to the market and seeking competitive advantages. University research teams in close cooperation with these start-up companies have made and continue to make their contribution.

Today it is already clear that MapReduce technology can be effectively used inside a parallel analytical DBMS, serve as an infrastructure of fault-tolerant parallel DBMS, and also maintain its autonomy in a symbiotic union with a parallel DBMS. All this not only hinders the development of parallel DBMS technology, but, on the contrary, contributes to its improvement and dissemination.

However, a big data programmer should take care of data distribution when using a computer cluster, otherwise a small data analysis task may overload the system.

References

1. Author, F.: Article title. Journal 2(5), 99–110 (2016).
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016).
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999).
4. Author, F.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010).
5. LNCS Homepage, http://www.springer.com/lncs, last accessed 2016/11/21.