Databases and Information Systems X
A. Lupeikiene et al. (Eds.)
© 2019 The authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/978-1-61499-941-6-195

Moving Hadoop to the Cloud for Big Data Analytics

Irina ASTROVA[a, 1], Arne KOSCHEL[b], Felix HEINE[b] and Ahto KALJA[a]
[a] *Department of Software Science, School of IT, Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia,*
[b] *Faculty IV, Department of Computer Science, Hannover University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany*

Abstract. Hadoop is a Java-based open source programming framework, which supports the processing and storage of large volumes of data sets in a distributed computing environment. On the other hand, an overwhelming majority of organizations are moving their big data processing and storing to the cloud to take advantage of cost reduction – the cloud eliminates the need for investing heavily in infrastructures, which may or may not be used by organizations. This paper shows how organizations can alleviate some of the obstacles faced when trying to make Hadoop run in the cloud.

Keywords. Big data analytics, cloud computing, education, Hadoop, OpenStack, MapReduce, Pregel, word counting, shortest path, PageRank

1. Introduction

Big data are usually big in two different senses. They are big in the quantity and variety of data that are available to be stored and processed. They are also big in the scale of analysis (or analytics) that can be applied to those data. Both kinds of "big" depend on the existence of supportive infrastructure. Such an infrastructure is increasingly being provided by the cloud like OpenStack or Amazon EC2. On the other hand, Apache Hadoop [1] has become the de-facto standard for processing and storing large data sets. Many organizations have come to rely on Hadoop for dealing with the ever-increasing quantities and varieties of their data. As a consequence, over the past few years, there has been an increase in organizations that are moving Hadoop to the cloud.

"Moving Hadoop to the cloud" means running Hadoop clusters on storage and compute resources offered by cloud provider [5]. It is hard to overstate how many advantages come with moving Hadoop to the cloud. The most important is scalability, meaning that the underlying infrastructure can be expanded or contracted according to the actual demand on resources. This paper presents a scalable big data infrastructure, one that gets automatically adjusted if more computing power or storage capacity is needed.

[1] Corresponding author, Irina Astrova, Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia, E-mail: irina@cs.ioc.ee, ahto@cs.ioc.ee.

2. Motivation

Not only is cloud computing used in the industry, but also at educational institutions, where it is quite common that hardware resources to be used by teachers and students (either individually or in groups) are very limited, with most of the resources only being available in the form of shared, pre-installed environments often used by multiple users at the same time. While this essentially reduces costs for the hardware, it also requires that very restrictive permissions should be granted on the user accounts to prevent multiple users from interfering with each other. In such situations, cloud computing can provide educational institutions with flexibility for the allocation of hardware resources, which can be easily repurposed amongst multiple users.

Scenarios where educational institutions take advantage of cloud computing in their programs of study are becoming ever more diverse, ranging from general platforms like Google G Suite for Education to specialized platforms like eduDScloud. eduDScloud (**edu**cational **D**ata analytics and **S**ervice-oriented architecture **cloud**) [14] has been developed at the University of Applied Sciences and Arts Hannover, Germany. It is used for teaching service-oriented and microservices architectures, data mining and big data analytics. It provides a private-cloud-based infrastructure to which virtual labs can be easily deployed at a fingertip. Virtual labs are composed of virtual machines that are operated in sandboxed environments of dynamically assignable and expandable resources. eduDScloud helps students to gain practical experience, e.g., in high-performance big data frameworks like Hadoop and Spark. At the same time, it helps teachers to create exercises for students with a minimum of administrative work because virtual labs can be easily instantiated from pre-built templates. As a data store, eduDSCloud uses HDFS so that each of the users are given their own distributed file system, where they have full rights to insert, update and delete data without interfering with other users.

While eduDScloud is already capable of scaling out across multiple machines, its current implementation limits it to private cloud providers. This can become a serious hindrance when, e.g., due to busy schedules, a high number of lab instances have to run in parallel, potentially exhausting all the available resources. To solve these challenges, upcoming projects will migrate eduDScloud to a hybrid cloud, thereby allowing it to scale out using resources of a public cloud, while maintaining the privacy protection for sensitive data in a private cloud. Part of such migration is the evaluation of open-source virtualization platforms like OpenStack, as it would enable seamless integration with various public cloud providers like Amazon. Furthermore, switching to one of those cloud-based solutions would allow eduDScloud to run solely on open source software.

3. Challenges

The cloud computing technology is based on the concept of virtualization. However, the virtualization of a Hadoop cluster or more specifically HDFS (Hadoop Distributed File System) cluster is a challenging task. Unlike most common server applications, Hadoop has some special requirements for a cloud architecture. In particular, Hadoop requires for topology information about the utilized infrastructure. Hadoop then uses this information to manage replications in HDFS. If the infrastructure consists of more than one cluster, Hadoop ensures that at least one replication is stored in a different hardware cluster than the other replications to allow for data access even when the whole cluster is unavailable. Moreover, Hadoop tries to perform computing tasks near the required data to avoid network transfers, which are often slower than local data access.

A cloud architecture abstracts the physical hardware to hide the infrastructure details from the hosted instances. Furthermore, shared storage pools are often used to store instances instead of having a dedicated storage in every computing node. Shared storage pools and missing topology information of the Hadoop instances might lead to multiple HDFS replications onto the same physical storage pool. Also Hadoop's paradigm to avoid network traffic by allocating computing tasks near the data storage would be broken, since shared storage pools are often connected via a network. As a consequence, the performance of cluster would probably be massively decreased due to unnecessary replications and increased network traffic.

4. Related Work

Cloud providers have started to offer prepackaged services that use Hadoop under the hood, but do most of the cluster management work themselves. The users simply point the services to data and provide the services with jobs to run, and the services handle the rest, delivering results back to the users. The users still pay for the resources used, as well as the use of the services, but save on all of the management work [5].

Examples of prepackaged services include:

- **Elastic MapReduce:** This is Amazon Web Services' solution for managing Hadoop clusters through an API. Data are usually stored in Amazon S3 or Amazon DynamoDB. The normal mode of operation for Elastic MapReduce is to define the parameters for a Hadoop cluster like its size, location, Hadoop version and variety of services, point to where data should be read from and written to, and define steps to run jobs. Elastic MapReduce launches a Hadoop cluster, performs the steps to generate the output data and then tears the cluster down. However, the users can leave the cluster running for further use and even resize it for greater capacity.
- **Google Cloud Dataproc:** It is similar to Elastic MapReduce, but runs within Google Cloud Platform. Data are usually stored in Google Cloud Storage.
- **HDInsight:** This is Microsoft Azure's solution, which is built on top of Hortonworks Data Platform. HDInsight works with Azure Blob Storage and Azure Data Lake Store for reading and writing data used in jobs. Ambari is included as well for cluster management through its API.

Despite their advantages like ready availability and ease of use, prepackaged services only work on the cloud providers offering them. Organizations can be worried about being "locked in" to a single cloud provider, unable to take advantage of competition between the providers. Moreover, it may not be possible to satisfy data security or tracking requirements with the services due to a lack of direct control over the resources [5].

In addition to prepackaged services, cloud providers offer Hadoop-as-a-Service (HaaS) for big data analytics [6]. However, HaaS offerings share the same disadvantages as prepackaged services in terms of moving further away from the open source world and jeopardizing interoperability. Moreover, since unlike to prepackaged services they are not explicitly based on Hadoop, there is a separate learning curve for them, and the effort could be wasted if they are ever discarded in favor of an application that works on Hadoop or on a different cloud provider [5]. In contrast, this paper presents open source software that is able to support multiple cloud providers.

Finally, a new category of cloud-based solutions called Big-Data-as-a-Service (BDaaS) [15] has emerged. Most of these offerings are being delivered in a public cloud, but there are also offerings available for deployment to a private cloud. However, it is not possible to have exacting, precise control over their cloud architectures.

5. Big Data Infrastructure

Figure 1 gives an overview of the architecture of our big data infrastructure that is based on both Hadoop and the cloud. This architecture is multilayered – different layers allocate the different responsibilities of the infrastructure. An upper layer uses a lower layer as a service.

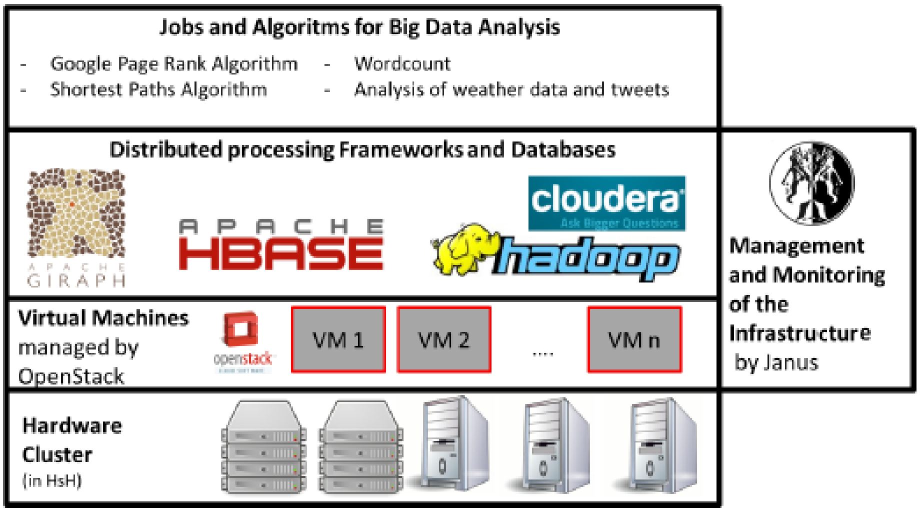
Jobs and Algoritms for Big Data Analysis
- Google Page Rank Algorithm
- Shortest Paths Algorithm
- Wordcount
- Analysis of weather data and tweets
Distributed processing Frameworks and Databases
APACHE GIRAPH
APACHE HBASE
cloudera
hadoop
Virtual Machines managed by OpenStack
openstack
VM 1
VM 2
....
VM n
Management and Monitoring of the Infrastructure by Janus
Hardware Cluster (in HsH)

Figure 1. Architecture of big data infrastructure [7].

The architecture comprises the following layers:

- **Hardware (or physical) clusters:** This is the bottom layer. It is composed of all physical machines that are wired together either by the Internet or by a direct network connection. A physical cluster is abstracted by means of virtualization.
- **Virtual machines:** On top of the hardware clusters, we installed virtual machines to form a hardware abstraction layer. A virtual machine acts like a physical computer except that software running on the virtual machine is separated from the underlying hardware resources. This layer is managed by OpenStack, which eases the management of virtual machines by a standardized API.

- **Distributed processing frameworks and databases:** This layer is composed of Hadoop, HDFS, HBase, Giraph and Cloudera. Cloudera is a distribution, which delivers many Apache products, including Hadoop and HBase.
- **Jobs and algorithms for big data analytics:** This is the application layer, which is responsible for big data analytics. Here we implemented algorithms like word counting, shortest path and PageRank (see Section 7). All the algorithms were implemented in a parallel manner using the MapReduce model of Hadoop [2] or the Pregel model of Giraph [3].
- **Management and monitoring of the infrastructure:** This layer is responsible for managing both the physical and virtual clusters. Here we implemented Janus, our own written software, which provides an API to automate the launching, management and resizing of Hadoop clusters. Figure 2 shows an overview of architecture of Janus that can be viewed as a link between the OpenStack's cloud manager and the Cloudera Manager. Janus connects both sides by utilizing classes, which implement two abstract interfaces: `iCloud` and `Hadoop-Mgr`. This is done to ensure that the core logic of Janus will not be changed even if another cloud provider like Amazon is added. On the cloud management side, these are classes of OpenStack or Amazon EC2, which implement the `iCloud` interface. On the Hadoop cluster management side, these could be the Hortonworks Ambari Manager Wrapper or the Cloudera Manager Wrapper classes, which implement the `Hadoop-Mgr` interface.

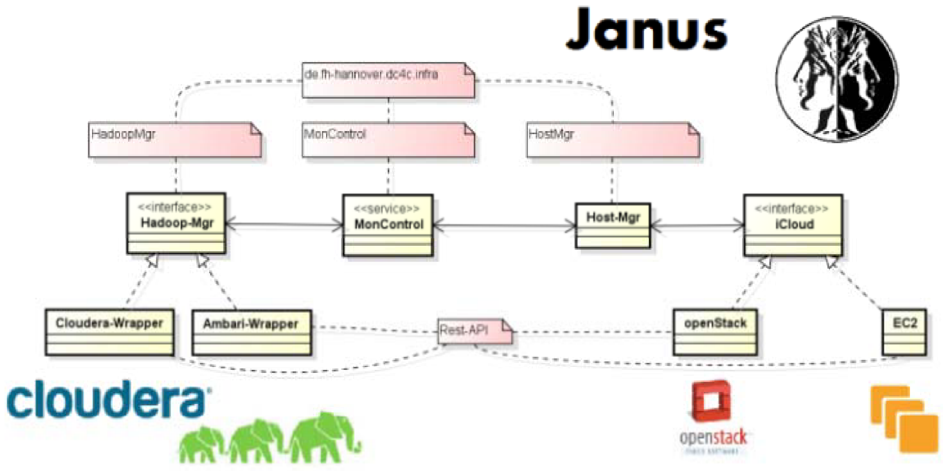

Figure 2. Architecture of Janus [7].

6. Application Scenarios

We identified two major application scenarios for our big data infrastructure. One was about the storage capacity offered to the users. If the storage capacity becomes scarce, the infrastructure will automatically increase the size of the HDFS. If the physical limits of the hardware get reached, the infrastructure will automatically contact another cloud provider. This could be a private cloud provider like another university or partner organization or a public cloud provider like Amazon. The users get the possibility to define prioritization of external clouds to minimize the expenses, which arise when commercial public clouds are used. Whenever a Hadoop cluster needs to be extended, Janus searches for a new suitable host system in all managed OpenStack clouds. Thereby currently used clouds are preferred, so that a Hadoop cluster will only be extended into a new cloud as a last resort. Within each managed cloud, Janus searches for hosts, which are currently not used by the Hadoop cluster that should be expanded. If more than one host system could run a new instance, the host with the lowest count of running instances is selected. In either case, such expansion should be considered as a temporal and rapid solution to prevent data loss, which would occur if the cloud could not store any further data. In the long term, it would be necessary to buy additional hardware to offer more storage capacity within the cloud to release expensive public cloud instances.

It could be beneficial to co-locate the allocations of a job on the same rack (affinity constraints) to reduce network costs, but spread the allocations across multiple machines (anti-affinity constraints) to minimize resource interference. If multiple Hadoop instances are placed on the same machine, replicated data will become unavailable if that machine fails.

Janus takes care of the anti-affinity of Hadoop instances. One virtual machine host should never run more than one instance from the same Hadoop cluster. This would endanger the redundancy of HDFS and might decrease the general performance with unnecessary replications. Instances forming a Hadoop cluster are anti-affine to all other instances of the same cluster but not anti-affine to instances of other Hadoop clusters. This means that one host system could generally run more than one Hadoop instance, but they cannot be from the same Hadoop cluster.

We had two separate OpenStack installations, each having one OpenStack master and several OpenStack computing nodes. The first cloud consisted of five servers with a quad-core processor and 16 GB RAM. The second cloud consisted of three servers with a quad-core processor and 128 GB RAM. All the servers were utilizing a local RAID-0 array as their data storage to ensure highest storage performance. Redundancy was achieved by the internal replication mechanisms of HDFS. To simulate the scaling mechanism into a public cloud, we configured Janus to treat the second cloud as the public cloud. Janus broke with the anti-affinity of instances in a Hadoop cluster in the simulated public cloud and launched new instances wherever hardware resources were available.

Another application scenario concerned the computation power of the cloud. As more and more different users would use the cloud for their big data analytics, a single job could get really slow if the virtual computing nodes reach their limits. The solution for this scenario is to start further virtual computing nodes in the cloud to take over additional analysis jobs. If the physical limits of the hardware also get reached, additional computation power will be obtained from an external cloud. An important point, which has to be taken into consideration when expanding into a public cloud, is the storage location of sensitive data. The users may want not to offer those data to a public cloud provider just because the infrastructure is running out of storage. In this case, the users are given the possibility to mark their data as sensitive so that the infrastructure can avoid the exposure of those data. To implement such a scenario, the cloud solution has to move other non-sensitive data to the public cloud to free storage for the sensitive data.

Based on the application scenarios, we created two rules to react to storage capacity or computing power bottlenecks. Both rules are checked in a cyclic interval. If a rule gets violated, the defined action will be started and no further checks of any other rule are done until the violation is fixed. One rule is `HdfsCapacityRule`. This rule is used to monitor the free disk space in HDFS; it creates a new Hadoop node if a given threshold is violated for a given timeframe. Another rule is `MapRedSlotsRule`. It is triggered when a given percentage of the available MapReduce slots have already been in use. When this rule is violated for a given timeframe, a new Hadoop node is created too.

7. Algorithms

To take advantage of the scalability of our big data infrastructure, we implemented algorithms for the big data analytics. Examples of these algorithms included:

- Word counting;
- Shortest path;
- PageRank.

All the algorithms needed the data procurement of Wikipedia, which consisted in the extraction of a Wikipedia dump. This dump was a huge .XML file downloaded from Wikipedia. It contained all Wikipedia articles in MediaWiki syntax. Additionally, for the shortest path and PageRank algorithms, a web graph of Wikipedia articles was extracted. However, MediaWiki syntax has a lot of keywords and meta-characters, which make the process of the web graph extraction difficult. Moreover, it was necessary to filter out all invalid hyperlinks. These links are pages without any content. In Wikipedia, invalid hyperlinks are highlighted in red when a page is viewed in a browser.

7.1. Word Counting

We employed the word counting algorithm to find out how often a particular word was present in Wikipedia articles. Figure 3 gives an overview of the steps involved in the algorithm.

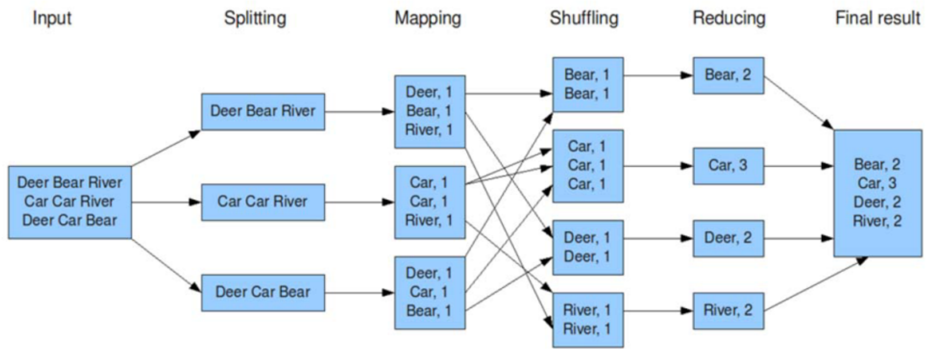

Figure 3. Word counting [12].

The word counting algorithm was implemented using MapReduce. In the Map phase, key-value pairs were formed, where a key was the word and a value was always 1.

```
Mapper{
      map(Text text){
               for each(String w in text){
                        output.collect(w, 1);
               }
      }
}
```

In the Map phase, punctuation characters like dots, commas, quotation marks and brackets were also filtered out, by replacing them with whitespaces or removing them. The following regular expressions show patterns for finding the punctuation characters.

```
1 replace((?<=\W)[\.\-_:,;\(\)"'~\?\[\]\!], " ")
2 replace([\.:,;\-_\(\)"'~\?\[\]\!](?=\W|\z), " ")
3 replace((?<=\\w)[\.:,;\(\)"'~\?\[\]\!](?=\w), "")
```

The first regular expression finds the punctuation characters at the beginning of a word, the second one at the end of a word or sentence, and the third one within a word. In some cases, it is difficult to decide if a punctuation character needs to be replaced by a whitespace or removed. For example, in the word `(Fach)Hochschule`, when the first bracket is removed, the second one should be removed as well. If both brackets are left, the words `Fachhochschule` and `(Fach)Hochschule` cannot be grouped together.

In the Shuffle phase, the values of the corresponding keys were aggregated. The result is a key (the word) with a list of values (ones). For example: <Bear: 1, 1>, <Car: 1, 1, 1>, etc.

In the Reduce phase, the values in each list were summed up to find out the word counts. For example: <Bear: 2>, <Car: 3>, etc.

```
Reducer{
      reduce(Word key, List values){
               int sum=0;
               for each(int v in values){
                        sum=sum+v;
               }
               output.collect(key, sum);
      }
}
```

7.2. Shortest Path

The aim of a Wikipedia game called five steps to Jesus [11] is to find the shortest path to a Wikipedia article on Jesus. The basic idea behind this game can be described as follows:

- Choose a random Wikipedia article;
- Choose a random hyperlink in the article;

- Repeat the previous step until arriving at the article on Jesus.

Figure 4 illustrates how the game works.

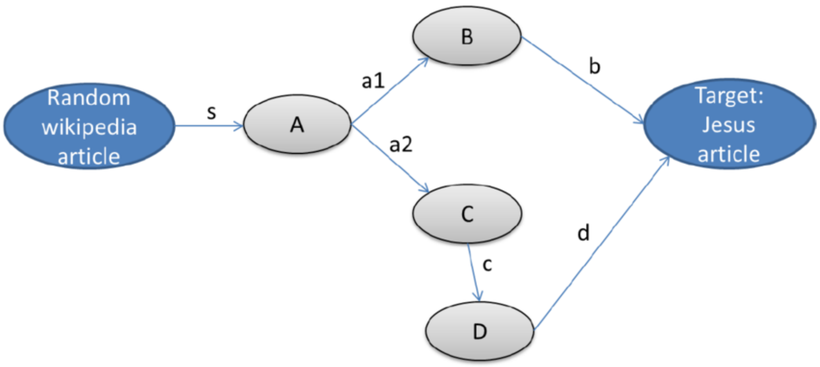

Figure 4. Web graph for five steps to Jesus.

The web graph in the figure can be written in the following format, where every vertex lists the target vertices in the braces (one line per vertex).

```
B{Jesus}
C{D}
D{Jesus}
```

As can be seen from the figure, there are two paths (written as a list of edges) to get from a randomly chosen Wikipedia article to the target one about Jesus:

```
1 {s, a1, b}
2 {s, a2, c, d}
```

Since the first path has a length of three whereas the second path has a length of four, it is the shortest path. To find the shortest path, a pre-processing step was necessary to invert all edges of the web graph, as shown in Figure 5. This step was implemented using MapReduce.

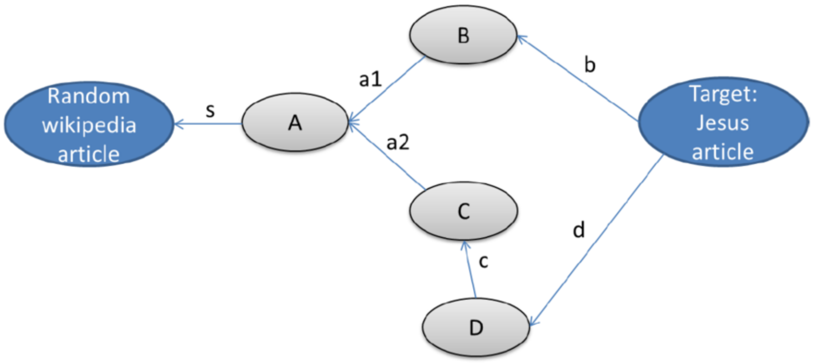

Figure 5. Inverted web graph for five steps to Jesus.

In the Map phase, all `target_vertex_ID` were mapped to the new `source_vertex_ID` and vice versa so that in the Reduce phase, all `target_vertex_ID` could be collected for the new `source_vertex_ID`.

```
Mapper{
    map(source_vertex_ID, List(target_vertex_ID), output ){
        for each target_vertex_ID {
            var kvl = new KeyValueList();
            kvl.setKey(target_vertex_ID);
            kvl.setValue(vertex_ID);
            output.collect(kvl);
        }
    }
}
```

```
Reducer {
    reduce(Key target_vertex_ID, List(source_vertex_ID, output){
        output.collect(target_vertex_ID, List(source_vertex_ID);
    }
}
```

After the vertices of the web graph had been inverted, the shortest path algorithm was used. This algorithm was implemented using Pregel.

```
Shortest Path Algorithm in Pregel
Input: list of vertices wired by edges to web graph
Output: list of vertices with minimum value for “hyperlink count”
repeat super-steps iterations until every vertex v is halted; halted vertices are waked up if they have received new messages {
    for each vertex as v which is not halted{
        if(getSuperstep() == 0){
            v.setValue(infinity);
        }
        var double minDist;
        if(v.getID().startsWith(“Jesus”)) {
            minDist = 0;
        } else {
            minDist = infinity;
        }
        for each v.getMessages() as m {
            minDist = min(minDist, m.getValue());
        }
        if(minDist <v.getValue()){
            v.setValue(minDist);
            v.sendMessageToAllEdgesWithValue(minDist+1);
        }
        v.voteToHalt();
    }
}
```

The shortest path algorithm iterates over the whole web graph until all the vertices are halted. At first, the values of all the vertices are initiated to infinity, meaning that there is no path to a Wikipedia article on Jesus (lines 3 to 5). Next it is checked if the current vertex is an article on Jesus (line 7). Since Wikipedia has many articles on Jesus (e.g., "Jesus Christ", "Jesus of Nazareth", etc.), all articles, starting with the word Jesus, are specified as the target articles. Vertices representing the target articles are initiated by zero. Neighbored vertices are connected by edges to the current vertex and exchange information on the shortest path by messages. In lines 12 and 13, it is found which of the neighbored vertices has the shortest path to an article on Jesus. If such a path is found, it will be saved as a new value for the current vertex and sent to the neighbored vertices (line 16). At the end of the iteration, all the vertices get halted. When the vertices receive new messages, they are waked up and get active again. After a few iterations the shortest path will be found. According to the empirical data, the smallest number of hyperlinks that lead to an article on Jesus does not exceed five [11].

7.3. PageRank

We used Google PageRank algorithm to find out the ranks of Wikipedia articles. The algorithm works by counting the number and quality of hyperlinks to a page to determine the PageRank's value of that page. This value is in the range of 0 and 1. It can be viewed as a rough estimate of how popular the page is. The underlying assumption is that more popular pages are likely to receive more hyperlinks from other popular pages. The algorithm was implemented using both MapReduce and Pregel.

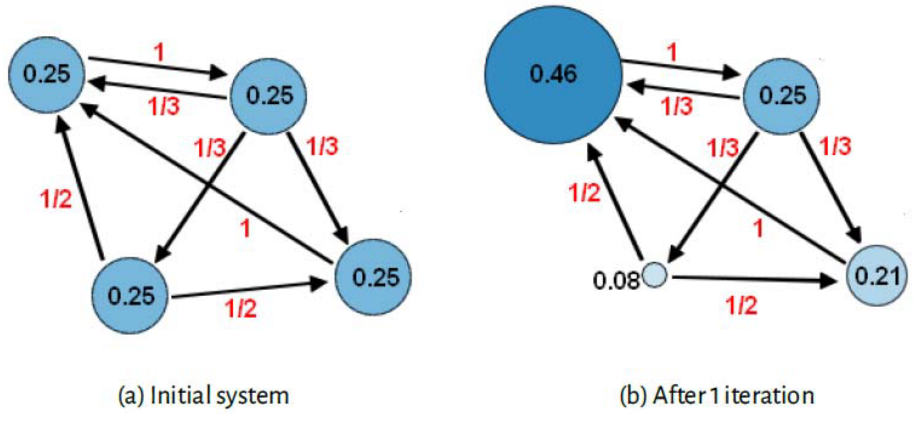

Figure 6. PageRank algorithm applied to web graph.

In the Map phase, there are two different types of output. The first output is the PageRank's value mapped to the target of the linked pages so that in the Reduce phase every page will get the PageRank of the pages linked to it (line 4). The second output is the same key-value pair to forward the information on the web graph structure to the Reduce phase (line 6). The web graph structure is represented as follows:

```
key: page
value: pair of {pagerank, list of linked_pages}
```

In the Reduce phase, all the PageRank's values of pages, which have a link to the current page, are summed up in line 7 to calculate the total PageRank in line 12. The second output of the Map phase is fetched in line 9 to reconstruct the origin vertex record in line 13. The PageRank algorithm is iterated until it converges.

```
Google PageRank in MapReduce
Input: a list of vertices wired by edges to graph
Output: a list of vertices with the value "PageRank"
1 Mapper {
2      map(page, page_value, PageList(linked_page), output) {
3             for each linked_page {
4                    output.writeKeyValue(linked_page,
new PageValue(page_value / sizeof(PageList(linked_page))));
5      }
6      output.writeKeyValue(page, PageList(linked_page));
7 }


1 Reducer {
2      reduce(page, List(value), output) {
3             var PageList linked_pages = null;
4             var double sum;
5             for each value {
6                    if(value instance of PageValue);
7                           sum += value.getPageValue();
8                    } else {
9                           linked_pages = value.getPageList();
10                   }
11            }
12            var double pagerank = (1-damping) / numberOfSites +
damping * sum;
13            output.writeKeyValue(page,                         new
PageValueAndLinkedPages(sum, linked_pages));
14     }
15 }
```

Pregel adapts MapReduce to perform graph processing on large data sets. Therefore, Pregel's implementation of the PageRank algorithm is more efficient than MapReduce's implementation. The algorithm starts with the first superstep zero and repeats until all vertices are halted, which will happen when the value of MAX_SUPERSTEP is achieved in line 11. In the first superstep zero, lines 3 to 9 will be skipped so that vertices send to all other vertices their initial PageRank's values divided by the number of edges. The initial PageRank's value is one divided by the total number of pages (see Figure 6).

In the second superstep one and higher, the new PageRank based on the received PageRank's values will be calculated in lines 4 to 9. In line 12, the new PageRank is sent to the linked pages so that these can re-calculate their PageRank's values and so on. This recursive iteration stops when the condition in line 11 is met. The value of MAX_SUPERSTEPS is set by the user before the calculation has been started. The more superstep iterations, the more accurate PageRank's values.

```
Google PageRank in Pregel
Input: a list of vertices wired by edges to graph
Output: a list of vertices with the value "Page Rank"
1 repeat supersteps until all vertices are halted {
2     for each vertex as v, which are not halted, compute{
3             if(getSuperstep() >= 1){
4                     var double sum = 0;
5                     var List(message) = v.getMessages();
6                     for each message of List(message){
7                             sum+= message.getValue();
8                     }
9                     v.setValue((1-damping) / getTotalVertics() +
damping * sum);
10            }
11            if( getSuperstep() < MAX_SUPERSTEPS ) {
12                    sendMessageToAllEdges(     v.getValue()      /
v.getNumberOfEdges() );
13            } else {
14                    v.votetoHalt();
15            }
16    }
17 }
```

Pregel is a computational model for large-scale graph processing, whereas MapReduce is not optimized for such computation. Comparing the two implementations of the PageRank algorithm, the Pregel implementation is a more intuitive than the MapReduce implementation. More specifically, in lines 5 to 9 the PageRank is computed based on the received massages to send the new value of the PageRank to the linked pages in line 12. By contrast, the MapReduce implementation is not intuitive and split up in two phases. In the Map phase, a vertex (Wikipedia article) propagates its PageRank to the linked pages by mapping the PageRank's value to a key "linked page". This technique is also known as inverted index: changing a key to a value and vice versa. It is used to simulate sending messages to the linked vertices.

In the Reduce phase, the Reducer gets a list of PageRank's value for each page. Based on these values, the PageRank of the page is re-computed. Since the PageRank algorithm is recursive, there is the need of chaining those iterative steps. The Pregel computational model is a native model for recursive algorithms with a termination support. Every superstep is an iteration, which is synchronized by the framework. The information synchronization between the vertices is done by sending messages over the network. The recursive termination mechanism is the major advantage of Pregel over MapReduce. If a certain condition occurs, a large part of the graph can be excluded from the computation by calling the `voteToHalt` method. (The shortest path algorithm also used this mechanism. If a vertex does not receive any more messages, it will stop computing the shortest path for that vertex.)

The input of the MapReduce implementation is the vertex data set, viz., a vertex with its value and the linked pages. This data set has to be carried from the Mapper in line 6 to the Reducer. The Reducer has to write the whole data set structure down, so that the next chaining MapReduce iteration gets the whole data set of the vertex with the value and edges. By contrast, in Pregel each processor is a vertex with the knowledge of edges, which communicates with other vertices via messages. Giraph has a paradigm "Think like a vertex that can send messages to any other vertex in the graph" using the bulk synchronous parallel (BSP) model. At first, processors compute independently and in parallel until a process reaches a barrier point. In the meanwhile, the processors communicate asynchronously by message exchange over the network and waits until all other processes have finished their computation. The barrier point is used for synchronizing the processors. This procedure is called superstep and is repeated several times until the PageRank values are stabilized.

8. Conclusion

Hadoop was designed and built around the assumption that distributed parallel processing and the minimization of storage and network latencies are a key to maximizing data query performance over large data sets. This assumption places constraints on the cloud architecture and deployment for the underlying infrastructure. For example, Hadoop has mandated that the co-location of computation and storage resources is essential for high performance. These requirements could indicate that Hadoop is ill-suited for the cloud, where the physical distance between computation and storage resources is often large or unknown. However, in this paper, we have presented a big data infrastructure to run Hadoop clusters on the cloud.

The value proposition includes the cloud benefits of cost reduction, flexibility and scalability. However, these advantages are offset of the cloud challenges of performance and data security. The challenges of performance can be further broken into two factors: storage access performance and computation performance [15]. To evaluate storage access performance, we used our big data infrastructure for storing large data sets onto HDFS and HBase [8]. As for computation performance, we implemented the PageRank algorithm using MapReduce and Pregel, and ran the two implementations on large data sets [4].

Challenges in data security were also addressed. The levels of data security offered in public clouds vary from provider to provider, but in general they continue to improve [13]. However, some organizations have regulatory or privacy requirements that prevent them from moving their sensitive data to a public cloud. For those organizations, our big data infrastructure provides additional control on data security, while minimizing data movement.

Finally, our big data infrastructure was recognized as being worthy of application in the area of Estonian e-Government, which also needs to deal with big data analytics [9, 10].

9. Future Work

In the current version of Janus, rules monitor only a single property of Hadoop cluster and check if that property violates a certain threshold. These rules were primarily used to prove that a Hadoop cluster can be automatically adjusted when the CPU usage per node increases or the HDFS capacity gets too low. A future work could be to extend the existing rule engine to support rules, which monitor multiple properties. Another enhancement could be to monitor complex events occurred in a Hadoop cluster, e.g., when every Friday night a weekly computation is started and from week to week the performance gets lower. For this purpose, historical measurements have to be stored and evaluated.

Acknowledgement

Irina Astrova's and Ahto Kalja's work was supported by the Estonian Ministry of Education and Research institutional research grant IUT33-13.

References

[1] T. White, Hadoop: The Definitive Guide 3, O'Reilly Media, 2012.

[2] A. Shook, MapReduce Design Patterns, O'Reilly Media, 2013.

[3] G. Malewicz, A. Matthew, A. Bik, J. Dehnert, I. Horn, N. Leiser and G. Czajkowski, "Pregel: A system for large-scale graph processing", in Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, New York, USA, 2010.

[4] A. Koschel, F. Heine, I. Astrova, F. Korte, T. Rossow and S. Stipkovic, "Efficiency experiments on Hadoop and Giraph with PageRank", in Proceedings of 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Heraklion Crete, Greece, IEEE, pp. 328-331, 2016.

[5] B. Havanki, Moving Hadoop to the cloud: Harnessing cloud features and flexibility for Hadoop clusters, O'Reilly Media, 2017.

[6] I. Astrova, A. Koschel, M. H. Lennart and H. Nahle, "Offering Hadoop as a cloud service", in Proceedings of the 2016 SAI Computing Conference, London, UK, IEEE, pp. 589-595, 2016.

[7] I. Astrova, A. Koschel, F. Heine and A. Kalja, "Scalable Hadoop-based infrastructure for big data analytics", in Proceedings of the 13th International Baltic Conference on Databases and Information Systems, Trakai, Lithuania, Springer (Communications in Computer and Information Science; 838), pp. 233-242, 2018.

[8] A. Koschel, F. Heine and I. Astrova, "Harnessing cloud scalability to Hadoop clusters", in Proceedings of the 15th European, Mediterranean and Middle Eastern Conference on Information Systems, Limassol, Cyprus, to appear in IEEE, 2018.

[9] A. Kalja, A. Reitsakas and N. Saard, "e-Government in Estonia: best practices," in Technology Management: A Unifying Discipline for Melting the Boundaries, eds.: T. R. Anderson, T. U. Daim, D. F. Kocaoglu and N.J. Piscataway, IEEE, pp. 500–506, 2005.

[10] A. Kalja, T. Robal and U. Vallner, "New generations of Estonian e-Government components", in Proceedings of the 2015 PICMET, Portland, Oregon, USA, IEEE, pp. 625-631, 2015.

[11] A. James, "Fun with Wikipedia: Click to Jesus", 2009. http://boingboing.net/2009/12/28/fun-with-wikipedia-c.html

[12] X. Zhang, "A simple example to demonstrate how does the MapReduce work", 2014. http://xiaochongzhang.me/blog/?p=338

[13] I. Astrova, A. Koschel, M. L. Henke, "IaaS platforms: How secure are they?" In Proceedings of IEEE 30th International Conference on Advanced Information Networking and Applications Workshops, Crans-Montana, Switzerland, NJ: IEEE Computer Society, pp. 843-848, 2016.

[14] A. Koschel, F. Heine, I. Astrova, I. Astrov, "A private cloud for data mining education", In Proceedings of the 6th International Conference on Enterprise Systems, Limassol, Cyprus, IEEE, pp. 129−133, 2018.

[15] T. Phelan, J. Baxter, "Hadoop in the cloud: Making a case for Big-Data-as-a-Service", 2016. https://www.oreilly.com/ideas/hadoop-in-the-cloud

[16] C. François, "Lexical disambiguation", MSc Project 2014-2015 - Progress Report, 2015. https://studentnet.cs.manchester.ac.uk/resources/library/thesis_abstracts/ProjProgReptsMSc15/Francois Christophe-Progress-Report.pdf