The Journal of Systems and Software 125 (2017) 133–151

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark

Ilias Mavridis*, Helen Karatza

Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 22 April 2016
Revised 19 October 2016
Accepted 23 November 2016
Available online 25 November 2016

Keywords:
Log Analysis
Cloud
Apache Hadoop
Apache Spark
Performance Evaluation

ABSTRACT

Log files are generated in many different formats by a plethora of devices and software. The proper analysis of these files can lead to useful information about various aspects of each system. Cloud computing appears to be suitable for this type of analysis, as it is capable to manage the high production rate, the large size and the diversity of log files. In this paper we investigated log file analysis with the cloud computational frameworks Apache™Hadoop® and Apache Spark™. We developed realistic log file analysis applications in both frameworks and we performed SQL-type queries in real Apache Web Server log files. Various experiments were performed with different parameters in order to study and compare the performance of the two frameworks.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Log files are a very important source of information, useful in many cases. However as the scale and complexity of the systems are being increased, the analysis of log files is becoming more and more demanding. The effort of collecting, storing and indexing a large number of logs is aggravated even more when the logs are heterogeneous.

The log production rate can reach to several TeraBytes (TB) or PetaBytes (PB) per day. For example Facebook dealt with 130 TB of logs every day in 2010 (https://www.facebook.com/notes/facebook-engineering/scaling-facebook-to-500-million-users-and-beyond/409881258919) and in 2014 they have stored 300 PB of logs (https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb). Due to the size of these datasets conventional database solutions cannot be used for the analysis, instead virtual databases combined with distributed and parallel processing systems are considered more appropriate (Kumar et al., 2015).

Although there are still open challenges, cloud or even interconnected cloud systems (Moschakis and Karatza, 2015) meet the needs of log analysis. Cloud is a field proven solution that is used for years by many big companies like Facebook, Amazon, eBay, etc. to analyze logs. Also in academia there are many studies that investigated cloud computing (mainly Hadoop) to analyze logs (Kotiyal et al., 2013; Wang et al., 2014; Narkhede and Baraskar, 2013; Yu and Wang, 2012; Wei et al., 2011; Yang et al., 2013; LIN et al., 2013; Liu et al., 2010; Vernekar and Buchade, 2013; Kumar and Hanumanthappa, 2013; Kathleen and Abdelmounaam, 2013). By the aforementioned, we can assume that log analysis is a big data use case and as a result many of the important challenges of cloud-based log file analysis are actually big data challenges. These challenges like data variety, data storage, data processing and resource management are studied in many works (Assunoa et al., 2015).

The data variety research issues are related to the handling of the increasing volume of data, to the extraction of meaningful content and to the aggregation and correlation of streaming data from multiple sources. Data storage related issues are about efficient methods to recognize and store important information, techniques to store large volumes of information in a way it can be easily retrieved and migrated between data centers. The data processing and resource management are about programming models optimized for streaming and multidimensional data, engines able to combine multiple programming models on a single solution and resource usage and energy consumption optimization techniques.

Also the analysis of log files comes with some extra challenges. Since many systems are distributed and heterogeneous, logs from a number of components must be correlated first (Oliner et al., 2012). Moreover, maybe there are missing, duplicate or misleading data that make log analysis more complex or even impossible. Furthermore many analytical and statistical modeling techniques do not always provide actionable insights (Oliner et al., 2012). For example, statistical techniques could reveal an anomaly

* Corresponding author. fax: +30-2310-997974.
E-mail addresses: imavridis@csd.auth.gr (I. Mavridis), karatza@csd.auth.gr (H. Karatza).

http://dx.doi.org/10.1016/j.jss.2016.11.037
0164-1212/© 2016 Elsevier Inc. All rights reserved.

in network utilization, but they do not explain how to address this problem.

Hadoop is the framework that has mainly been used for log analysis in cloud by many big companies and in academia too (Kotiyal et al., 2013; Wang et al., 2014; Narkhede and Baraskar, 2013; Yu and Wang, 2012; Wei et al., 2011; Yang et al., 2013; LIN et al., 2013; Liu et al., 2010; Vernekar and Buchade, 2013; Kumar and Hanumanthappa, 2013; Kathleen and Abdelmounaam, 2013). Hadoop was originally designed for batch processing providing scalability and fault tolerance but not fast performance (http://wiki.apache.org/hadoop/). It enables applications to run even in thousands of nodes with Petabytes of data. Hadoop manages the large amount of log data by breaking up the files into blocks and it distributes them to the nodes of the Hadoop cluster. It follows a similar strategy for computing by breaking jobs into a number of smaller tasks. Hadoop supports quick retrieval and searching of log data, scales well, supports faster data insertion rates than traditional database systems and is fault tolerant (Anton et al., 2013).

Hadoop follows a processing model that frequently writes and reads data from the disk; this affects its performance and makes it unsuitable for real-time applications (Stavrinides and Karatza, 2015). On the other hand, Spark uses more effectively the main memory and minimizes these data transfers from and to disk, by performing in-memory computations. Also Spark provides a new set of high-level tools for SQL queries, stream processing, machine learning and graph processing (https://databricks.com/spark).

The main contribution of this work is the performance evaluation of log file analysis with Hadoop and Spark. Although log analysis in cloud has been studied in many works like (Kotiyal et al., 2013; Wang et al., 2014; Narkhede and Baraskar, 2013; Yu and Wang, 2012; Wei et al., 2011; Yang et al., 2013; LIN et al., 2013; Liu et al., 2010; Vernekar and Buchade, 2013; Kumar and Hanumanthappa, 2013; Kathleen and Abdelmounaam, 2013), most of them are about Hadoop based tools and they ignore the powerful Spark. Our work complements the existing studies by investigating and comparing the performance of realistic log analysis applications in both Hadoop and Spark. To the best of our knowledge this is the first work that investigates and compares the performance of real log analysis applications in Hadoop and Spark with real world log data. We conducted several experiments with different parameters in order to study and compare the performance of the two frameworks. This paper extends our previous work (Mavridis and Karatza, 2015) by introducing two additional performance indicators, scalability and resource utilization. Also we proposed a power consumption model and we make an utilization-based cost estimation. In order to further investigate the performance of log analysis we developed additional applications and we conducted a new series of experiments in both frameworks.

The rest of the paper is organized as follows. Section 2 provides an overview of related research. Section 3 describes briefly what is a log file and log file analysis in cloud. Section 4 outlines the two open source computing frameworks Hadoop and Spark. Section 5 describes the setup of our experiments. Section 6 presents the experimental results and the evaluation of the frameworks. Finally Section 7 concludes this paper.

2. Related work

Cloud computing for log file analysis is presented in several studies. In Kotiyal et al. (2013), the authors highlighted the differences between traditional relational database and big data. They claimed that log files are produced in higher rate than traditional systems can serve and they proposed and presented log file analysis with Hadoop cluster.

A weblog analysis system based on Hadoop HDFS, Hadoop MapReduce and Pig Latin Language was implemented by Wang et al. (2014). The proposed system aims to overcome the bottleneck of massive data processing of traditional relational databases. The system was designed to assist administrators to quickly analyze log data and take business decisions. It provides an administrator monitoring system, problem identification and system's future trend prediction.

Narkhede et al. presented a Hadoop based system with Pig for web log applications (Narkhede and Baraskar, 2013). A web application was created to store log files on Hadoop cluster, run MapReduce jobs and display results in graphical formats. The authors concluded that Hadoop can successfully and efficiently process large datasets by moving computation to the data rather than moving data to computation.

In agreement with Wang et al. (2014) and Narkhede and Baraskar (2013), Yu and Wang (2012) proposed mass log data processing and data mining methods based on Hadoop to achieve scalability and high performance. To achieve scalability and reliability, log data are stored in HDFS and are processed with Hadoop MapReduce. In this case, the experimental results showed that the Hadoop based processing and mining system improved the performance of statistics query.

Wei et al. (2011) presented a scalable platform named Analysis Farm, for network log analysis with fast aggregation and agile query. To achieve storage scale-out, computation scale-out and agile query, Open Stack was used for resource provisioning and MongoDB for log storage and analysis. In experiments with Analysis Farm prototype with ten MongoDB servers, the system managed to aggregate about three million log records in a ten minute interval time and effectively queried more than four hundred million records per day.

Yang et al. (2013) proposed a Hadoop based flow logs analyzing system. This system uses a new script language called Log-QL. Log-QL is a SQL-like language that was translated and submitted to the MapReduce framework. After experiments the authors concluded that their distributed system is faster and can handle much bigger datasets compared to centralized systems.

A cloud platform for log data analysis that combines Hadoop and Spark is presented by LIN et al. (2013). The proposed platform has batch processing and in-memory computing capabilities by using Hadoop, Spark and Hive/Shark at the same time. The authors claimed that their platform managed to analyze logs with higher stability, availability and efficiency than standalone Hadoop-based log analysis tools.

Liu et al. implemented a MapReduce-based framework to analyze log files for anomaly detection (Liu et al., 2010). The system monitors the distributed cluster status and detects its anomalies by following a specific methodology. First, system logs were collected from each node of the monitored cluster to the analysis cluster. Then, the K-means clustering algorithm was applied to integrate the collected logs. After that, a MapReduce-Based algorithm was executed to parse these clustered log files.

Vernekar and Buchade (2013) used log file analysis for system threats and problem identification. The proposed system used a MapReduce algorithm to identify the security threats and problems. With this technique the proposed system achieved a significant improvement in response time of large log files analysis that leads to a faster reaction by system's administrator.

Log file analysis can also be used for intrusion detection. Kumar and Hanumanthappa (2013) described the architecture and implemented an intrusion detection system based on log analysis. They described an approach that uses Hadoop MapReduce in order to enhance throughput and scalability. Various experiments showed that the system fulfills the initial expectations for scalability, fault tolerant and reliability.

Finally Kathleen and Abdelmounaam (2013) presented SAFAL, a Spatio-temporal Analyzer of FTP Access Logs that uses Hadoop

```
1 157.55.33.xx0 - - [16/Jan/2013:17:57:48 -0800] "GET /webboard/ HTTP/1.0" 503 441
2 27.130.254.xx9 - - [16/Jan/2013:17:57:48 -0800] "GET / HTTP/1.1" 200 6210
3 66.249.75.xx3 - - [16/Jan/2013:17:57:51 -0800] "GET /webboard/index.php?area=statistics;u=24404 HTTP/1.1" 503 441
4 157.56.93.xx3 - - [16/Jan/2013:17:58:45 -0800] "GET /webboard/index.php?topic=2635.0 HTTP/1.0" 503 441
5 61.19.227.xx0 - - [16/Jan/2013:17:58:53 -0800] "GET /uploads/menu_1_324.gif HTTP/1.1" 200 6018
6 223.205.40.xx9 - - [16/Jan/2013:17:58:54 -0800] "GET /index1.php?id=13 HTTP/1.1" 200 15095
7 69.171.228.xx7 - - [16/Jan/2013:17:58:54 -0800] "GET /index1.php?id=13 HTTP/1.1" 200 15095
8 223.205.40.xx9 - - [16/Jan/2013:17:58:55 -0800] "GET /poll/chartpoll2.php HTTP/1.1" 200 1220
9 223.205.40.xx9 - - [16/Jan/2013:17:58:55 -0800] "GET /poll/chartpoll.php HTTP/1.1" 200 1209
10 223.205.40.xx9 - - [16/Jan/2013:17:58:55 -0800] "GET /FCKeditor/UserFiles/Image/dr_sda.jpg HTTP/1.1" 200 17158
```

Fig. 1. Apache web access log.

MapReduce and aims to identify trends in GPS data usage. The specific log files contain massive amounts of data like meteorological and digital imagery data. After several experiments the authors found that SAFAL was able to analyze very efficiently millions of lines of FTP Access Logs and that it could be possible to create near real time maps.

The majority of the above studies are about Hadoop and they do not take into consideration the powerful Spark to analyze logs. Our work differentiates from the existing ones by investigating and comparing the performance of realistic log analysis applications in both Hadoop and Spark. To the best of our knowledge this is the first work that studies the performance of real log analysis applications in Hadoop and Spark with real world log data.

3. Log file analysis

Log file analysis is the analysis of log data in order to extract some useful information (https://databricks.com/spark). A proper analysis requires a good knowledge of the device or software that produces the log data. It must be clear how the system that produces the logs works and what is good, suspicious or bad for it. Worth noting that a same value in two different systems maybe is bad for one but completely normal for the other.

3.1. Log files

Each working computer system collects information about various operations. This information is stored in specific files which called log files (Leite, 2011). Log files are consisting of log messages or simply log(s). A log message is what a computer system, software, etc. generates in response to some sort of stimulation (Anton et al., 2013). The information that pulled out of a log message and declares the reason that the log message was generated is called log data (Anton et al., 2013).

A common log message contains the timestamp, the source, and the data. The timestamp indicates the time at which the log message was created. The source is the system that created the log message and the data is the core of the log message. Unfortunately this format is not a standard; a log message can be significantly different from system to system.

One of the most common used types of log files in the web is the log file that is generated by the Apache HTTP Server (http://httpd.apache.org/docs/1.3/logs.html). Fig. 1 shows the first ten lines of a real Apache web access log file.

As shown in Fig. 1, the first element of each row is the ip address of the client (e.g., 157.55.33.xx0) or may be the name of the node that made the request to the server. Then there are two dashes (- -) that means that there is no value for this two fields (http://httpd.apache.org/docs/1.3/logs.html). The first dash stands for the identity of the client specified in RFC 1413 (http://www.rfc-base.org/rfc-1413.html), and the second dash represents the user id of the person requesting the server. The fourth element in each of these log messages indicates the date and time that the clients request had been served by the server and in the same brackets there is the servers time zone (e.g., -0800). Next in double quotes is the request line from the client. The request line first contains the method that has been used by the client (e.g., GET), then is the requested source (e.g., /poll/chartpoll2.php) and finally the used protocol (e.g., HTTP/1.1). At the end of each row there are two numbers that follow the request line. The first number is the status code that the server returns to the client (e.g., 200) and the last number indicates the size of the object returned to the client and is usually expressed in bytes (e.g., 6210).

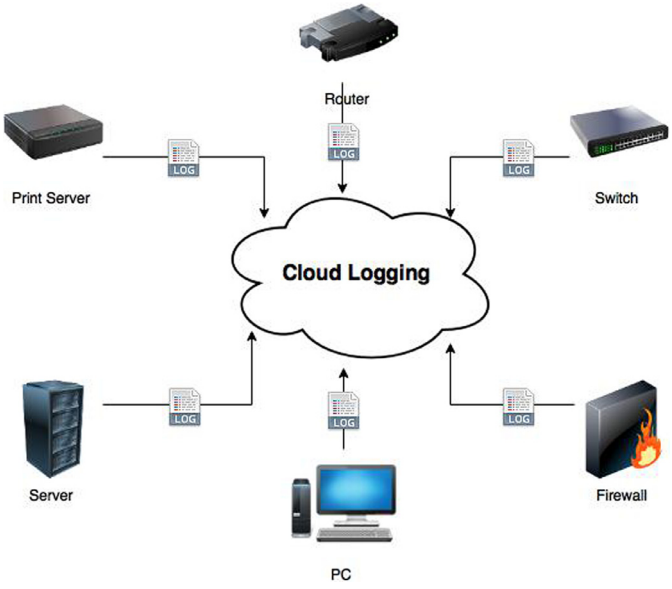

Fig. 2. Logging as a Service (LaaS).

3.2. Log analysis in the Cloud

The growing need for processing and storing resources for log analysis can be fulfilled by the virtually unlimited resources of cloud. By the combination of cloud and log analysis, a new term emerged the Logging as a Service (LaaS). With LaaS anybody can collect, store, index and analyze log data from different systems, without the need of purchasing, installing and maintaining his own hardware and software solutions.

There are some cloud service providers that maintain globally distributed data centers who undertake to analyze log files for one of their clients (Jayathilake, 2012). Usually these providers offer real-time, birds-eye view of the logs, a customized dashboard with easy to read line, bar, or pie charts. Users of such services can collect logs from various devices and software and submit them to the cloud using pre-built connectors for different languages and environments, as shown in Fig. 2.

To uploading logs to the cloud, most LaaS providers support Syslog (the most widespread logging standard), TCP/UDP, SSL/TLS and a proprietary API typically RESTful over HTTP/HTTPS

(Anton et al., 2013). Also many providers have custom APIs in order to support not Syslog logs and a customizable alerting system.

The LaaS is a quite new cloud service but there are already providers like (http://www.loggly.com) and (http://www.splunkstorm.com/) which offer different service products. There are some features and capabilities common to all and some others that may vary from provider to provider. Every LaaS must support file uploading from the user to the cloud, indexing of data (for fast search, etc.), long-term storage and a user interface for searching and reviewing the data (Anton et al., 2013). Most providers also support various types of log formats and have their own API (http://www.loggly.com, http://www.splunkstorm.com/). Moreover, the majority of providers charge their services with the model pay as you go, where the user is charged depending on the use of services he has made.

On the other hand, not every LaaS provider is the same. For example there are differences in the way that each provider has developed its system (Anton et al., 2013), some providers have built their services to anothers cloud infrastructure, while others in their own cloud infrastructure. Also, although all LaaS providers offer the possibility of long-term storage of data, the charges are not the same and the highest possible storage time differs also. Furthermore as it was expected the charges vary from provider to provider.

4. Cloud computational frameworks

We studied two of the most commonly used cloud computational frameworks, Hadoop and Spark. Hadoop is a well-established framework that is used for storing, managing and processing large data volumes for many years by web behemoths like Facebook, Yahoo, Adobe, Twitter, eBay, IBM, Linkedin and Spotify (http://wiki.apache.org/hadoop/PoweredBy). Hadoop is based on MapReduce programming model for batch processing. However the need for faster real-time data analysis led to a new general engine for large-scale data processing, Spark. Spark was developed by AMPLab (http://wiki.apache.org/hadoop/PoweredBy) of UC Berkeley and it can achieve up to one hundred times higher performance compared to Hadoop MapReduce (Xin et al., 2013) by using more effectively the main memory.

4.1. Apache Hadoop

Hadoop origins from Apache Nutch (http://nutch.apache.org/), which is an open source search engine. Basic elements to the creation of Hadoop were two Google studies, the first one was published in 2003 and describes the Google Distributed Filesystem (GFS) (Sanjay et al., 2003) and the second one was published in 2004, and describes MapReduce (Dean and Ghemawat, 2004). In February 2006 a part of Nutch became independent and created Hadoop. In 2010 a team from Yahoo began to design the next generation of Hadoop, the Hadoop YARN (Yet Another Resource Negotiator) or MapReduce2 (http://hortonworks.com/hadoop/YARN/).

YARN changed the resource management of Hadoop and supported a wide range of new applications with new features. YARN is more general than MapReduce (Fig. 3), in fact MapReduce is a YARN application. There are other YARN applications like Spark (http://wiki.apache.org/hadoop/PoweredByYARN), which can run parallel to the MapReduce, under the same resource manager.

4.1.1. Hadoop Distributed File System

Hadoop Distributed File System (HDFS) stores files into blocks of a fixed size in different nodes of Hadoop cluster (http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html) (Fig. 4). By dividing files into smaller blocks, HDFS can store much bigger files than the disk capacity of each node. The stored files follow the write-once, read-many approach and cannot be modified.

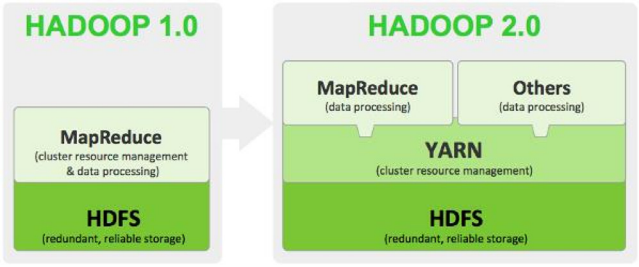

Fig. 3. Hadoop 1.0 to Hadoop 2.0 http://wiki.apache.org/hadoop/PoweredByYARN.

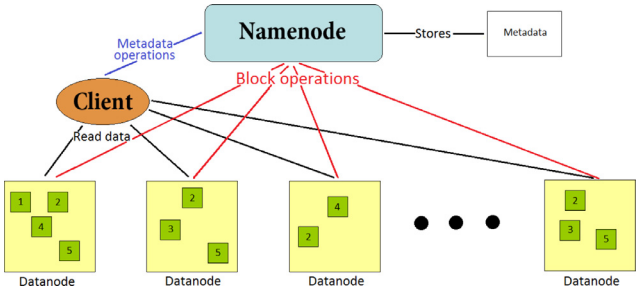

Fig. 4. HDFS architecture.

The HDFS follows the master/slave model. The NameNode is the master that manages the file system namespace and determines the access of the clients to the files. The slaves are called DataNodes and are responsible for storing the data and executing NameNodes instructions. For fault-tolerance, HDFS replicates each block of a DataNode to other additional nodes (http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).

4.1.2. MapReduce

MapReduce was presented by Google's paper (Dean and Ghemawat, 2004) and is a batch-based, distributed computing model. A MapReduce program consists of the Map Phase and the Reduce Phase (Dean and Ghemawat, 2004). Initially the data are processed by the map function and produce an intermediate result in the form of (Key, Value). After that follows the reduce function. The reduce function performs a summary operation that processes the intermediate results and generates the final result.

In the original version of the Hadoop MapReduce there are two types of nodes, the JobTracker (master) and TaskTrackers (slaves). In each MapReduce cluster there is a JobTracker that is responsible for resource management, job scheduling and monitoring (Wang et al., 2013). The TaskTrackers run processes that were assigned to them by the JobTracker.

With Hadoop YARN, there is no longer a single JobTracker that does all the resource management, instead the ResourceManager and the NodeManager manage the applications. The ResourceManager is allocating resources to the different applications of the cluster. The ApplicationMaster negotiates resources from the ResourceManager and works with the NodeManager(s) to execute and monitor the component tasks (http://hortonworks.com/hadoop/YARN/). The new Hadoop YARN is more scalable and generic than the previous version and it supports applications that do not follow the MapReduce model.

4.2. Apache Spark

Spark was developed in 2009 by AMPLab of UC Berkeley and became an open source project in 2010 (https://databricks.com/spark). In 2013, the program was donated to the Apache software

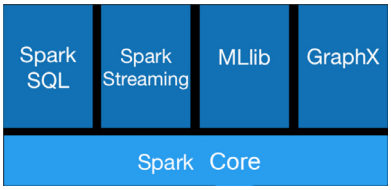

Fig. 5. Spark ecosystem.

foundation (https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50) and in November 2014, the engineering team at Databricks set a new record in large-scale sorting by using Spark (http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html).

Spark extends the popular MapReduce model and supports the combination of a wider range of data processing techniques, such as SQL-type queries and data flow processing. For ease of use, Spark has Python, Java, Scala and SQL APIs, and many embedded libraries.

One of the main features of Spark is the exploitation of main memory (https://spark.apache.org/). It may accelerate an application to one hundred times using memory and ten times using only the disc compared to Hadoop MapReduce (https://databricks.com/spark).

4.2.1. Spark ecosystem

Spark is a general purpose engine that supports higher-level items specialized to a particular kind of processing (https://databricks.com/spark). These components (Fig. 5) are designed to operate close to the core, and can be used as libraries for application development.

The components of the Spark ecosystem are (https://databricks.com/spark):

- Spark Core: Is the general execution engine for the Spark platform and every other functionality is built on top of it.
- Spark SQL: Is a Spark module for structured data processing. It can be considered as a distributed SQL query engine.
- Spark Streaming: Enables the real time processing of streaming and historical data.
- MLlib: Is a scalable machine learning library.
- GraphX:Is a graph computation engine that enables the manipulation and parallel processing of graphs.

4.2.2. Resilient Distributed Dataset

Resilient Distributed Dataset (RDD) is Spark's parallel and fault-tolerant data structure (Zaharia et al., 2012). Spark automatically distributes the RDD data into the cluster and performs parallel operations on them. RDDs can contain any object or class of Python, Java or Scala. One of the most important capabilities of Spark is the persisting or caching a dataset in main memory (https://databricks.com/spark). The cached RDDs are fault-tolerant and are processed much faster by the nodes of the cluster.

RDD supports two types of operations. Transformations which generate a new dataset from an existing one, and actions which return a value from a dataset (http://spark.apache.org/docs/latest/programming-guide.html). For example, map is a transformation that passes each element of a RDD to a function and results to a new RDD with the computed values. On the contrary, reduce is an action that passes each element of a RDD to a function and returns a single value as a result.

4.3. Key differences between Hadoop and Spark

The big difference between Hadoop and Spark is that Hadoop processes data on disk while Spark processes data in-memory and tries to minimize the disk usage. Spark avoids using the disk by keeping data in memory between Map and Reduce phases. By this way it achieves a performance boost especially for applications that do many iterative computations on the same data.

However, Spark needs a lot of memory to caching the data. If Spark runs on YARN with other resource-demanding services, or the data are more than the available memory, then it is possible for Hadoop to reach Spark's performance. Also the use of main memory affects the cost of the infrastructure. Memory resources are more expensive than disk and as a result Spark needs a more expensive infrastructure to run properly compared to Hadoop. However Spark is more cost-effective due to its better performance.

Spark has also build-in modules for stream processing, machine learning and graph processing, without the need of any additional platforms like Hadoop does. Hadoop MapReduce is suitable for batch processing but it needs another platform like Storm for stream processing, which requires additional maintenance. Also because Spark uses micro batches is more stable for stream processing than Hadoop. Hadoop is mainly for static data operations and for applications that do not need direct results. On the other hand for applications like analytics on streaming data, or applications with multiple operations Spark is the best choice.

Regarding security, the web UI of Spark can be secured via javax servlet filters. Spark can run on YARN and use Kerberos authentication, HDFS file permissions and encryption between nodes. Hadoop is more secured and has many security projects like Knox Gateway and Sentry.

4.4. SQL-type queries in Hadoop and Spark

SQL is a special purpose programming language that is used for many years to manage data in relational databases. Although SQL is not suitable for every data problem and maybe it can't be used for complicated analysis, it is being used by many enterprise developers and business analysts because it is easy to use. For these reasons we investigated the distributed SQL-type querying with Apache Hive and Spark SQL. We studied in which way and in how much time the semi-structured text log data that are stored in HDFS can be represented in tables of a virtual relational database and how Spark can access and process Hive data. Since we got the data into tables we performed several experiments to find out how much additional time a SQL query needs to run compared to a similar Hadoop or Spark application. After these experiments we concluded if the additional time of developing a Hadoop or Spark java application is justified by the execution time improvement.

Hive is developed by Facebook to run queries on large volume of data (http://hortonworks.com/big-data-insights/how-facebook-uses-hadoop-and-hive/). Hive uses a SQL type language the HiveQL and transforms SQL queries into MapReduce jobs. It does not create MapReduce applications, but Mapper and Reducer modules that were regulated by an XML file (https://cwiki.apache.org/confluence/display/Hive/Home). There are several interfaces that can be used to access Hive (https://cwiki.apache.org/confluence/display/Hive/Home). There is a command line interface (CLI) and various graphical user interfaces, commercial or open source such as Hue (http://www.cloudera.com/content/www/en-us/documentation/archive/cdh/4-x/4-2-0/Hue-2-User-Guide/hue2.html) of Cloudera. In this work we used the CLI interface.

For SQL-type querying with Spark we used a module that it is embedded in Spark framework, the Spark SQL (Armbrust et al., 2015). Spark SQL can be used in a Spark application as a library and also with external tools connected with JDBC/ODBC (https://databricks.com/blog/2014/03/26/spark-sql-manipulating-structured-data-using-spark-2.html). The JDBC server operates as an autonomous Spark driver and can be used by many users. Every user can store tables in memory and execute queries. Also Spark

Table 1
Cluster nodes.

Name	Virtual Cores	Memory	Disk	Network
Master	8	8 GB	40 GB	Local (192.168.0.1) Public (83.212.xx.xx)
Slave1	4	6 GB	40 GB	Local (192.168.0.2)
Slave2	4	6 GB	40 GB	Local (192.168.0.3)
Slave3	4	6 GB	40 GB	Local (192.168.0.4)
Slave4	4	6 GB	40 GB	Local (192.168.0.5)
Slave5	4	6 GB	40 GB	Local (192.168.0.6)

SQL can exist with Apache Hive and offers additional features such as access to Hive boards, UDFs (user-defined functions), SerDes (serialization and deserialization formats) and execution of SQL-type queries with HiveQL.

5. Experimental setup

We have conducted a series of experiments in order to evaluate the performance of Hadoop and Spark. We used an IaaS (Infrastructure as a Service) to create a private cloud infrastructure and we developed and ran realistic log analysis applications with real log files.

5.1. Cluster Architecture

For the experiments we used the virtualized computing resources of Okeanos (Koukis et al., 2013). Okeanos is an open source IaaS, developed by the Greek Research and Technology Network (Koukis & Louridas, 2012). It is offered to the Greek Research and Academic community and provides access to virtual machines, virtual ethernets, virtual disks, and virtual firewalls, over a web-based UI. Okeanos used Google Ganeti for the backend and Python/Django API for the frontend. The hypervisor is KVM and the frequency of CPUs is 2100 MHz.

With the available computing resources of Okeanos we created six virtual machines that are connected to a virtual private local network. In this cluster there is a master node and five slave nodes. As shown in Table 1 the master node is equipped with 8 virtual cores, 8 GB of memory and 40 GB of disk space. The five slave nodes are less powerful and each one of them has 4 virtual cores, 6 GB of memory and 40 GB of disk space.

The operating system deployed across all nodes is Debian Base 8.2. Also (as Hadoop requires) in every node has been installed the java version 1.7.0 and there is passwordless ssh between the nodes in order to be possible for Hadoop to start and stop various daemons and execute some other utilities (https://wiki.apache.org/hadoop/FAQ#Does_Hadoop_require_SSH.3F). Finally the master node except from the private local ip has also a public ip for administrative and monitor purposes.

5.2. The Dataset

For the experiments was used a real world Apache HTTP Server log file from a company's website. Part of the log file that is used in our experiments in shown in Fig. 1. This file was found after a relevant search on the internet and it consists of semi-structured text data. Its size is 1 GB and contains more than nine million lines of log messages. For the experiments that required bigger input file, the content of the original file was copied and merged to one new file. We did not apply any preprocessing to the file before the actual experiments.

To run the experiments we saved the data in HDFS, in blocks of 128 MB. Because the number of active nodes will be different during the experiments every data block was copied to all Datanodes. By this way even if there is only one active Datanode still it will have all the necessary blocks to recreate the original file.

Also in order to execute SQL queries the data must be structured in tables. As the logs are already in HDFS there is no need to re-upload the data. Hive can easily use this data, define the relationships and create a virtual relational database. Using Hive (https://hive.apache.org/javadocs/r0.10.0/api/org/apache/hadoop/hive/serde2/SerDe.html) and the appropriate regular expressions we created the database tables in a few seconds.

5.3. Metrics and monitoring tools

In our experiments we took measurements related to the application execution time and resource utilization. These metrics help us to understand the performance impact of different parameters such as input dataset, number of active nodes and type of application. We recorded utilization information about CPU, main memory, disk and network for every node for every moment that the experiments run.

Sysstat (http://sebastien.godard.pagesperso-orange.fr/) and Ganglia (Massiea et al., 2004) are the main tools that were used to analyze the resource utilization of the cluster nodes. Sysstat is a monitoring package for Linux systems with low overhead that can be easily used to gather system performance and usage activity information. There are Cloud specific monitoring tools such as Amazon CloudWatch, AzureWatch, Nimsoft and CloudKick (Fatemaa et al., 2014) but the main disadvantage is that many of them are commercial and provider-dependent. For instance, Amazon Cloud Watch monitors Amazon Web Services(AWS) such as Amazon EC2, Amazon RDS DB instances, and applications running on AWS.

On the other hand there are General purpose monitoring tools that were developed before the advent of Cloud Computing for monitoring of IT infrastructure resources. Many of these tools are evolved and adopted in Cloud. Such tools are Ganglia, Nagios, Collectd, Cacti and Tivoli. General purpose infrastructure monitoring tools usually follow a clientserver model by installing an agent in every system to be monitored.

We selected the open source Ganglia (Fig. 6) rather than other powerful tool like JCatascopia (Trihinas et al., 2014) because Ganglia is lightweight and in many cases adds the lowest overhead at the monitoring hosts compared to other tools. It has features such as scalability, portability and extensibility (Fatemaa et al., 2014) and it's well documented. Also Ganglia fulfills the monitoring requirements of our experiments as it supports Hadoop and Spark with more than one hundred special built-in metrics like HDFS and YARN metrics.

5.4. Experimental scenarios

In order to investigate the log file analysis with Hadoop and Spark we developed realistic applications in both frameworks. All applications are developed in java and are developed with the Eclipse IDE and Apache Maven. The log files that we used for our experiments are stored in HDFS and are accessible by both Hadoop and Spark applications. All applications read the input data from HDFS and produce output results in simple format that can be represented easily in charts and graphs. To parse the log data we used regular expressions.

As we are dealing with http log files, we considered that it would be quite possible for an administrator of a webpage to ask for information about traffic distribution, detection of possible threats and error identification.

Concerning the traffic distribution we developed two applications. The first application measures the requests to the web server and sorts them by day. By this way we can find out how the overall traffic is distributed into the seven days of the week and with

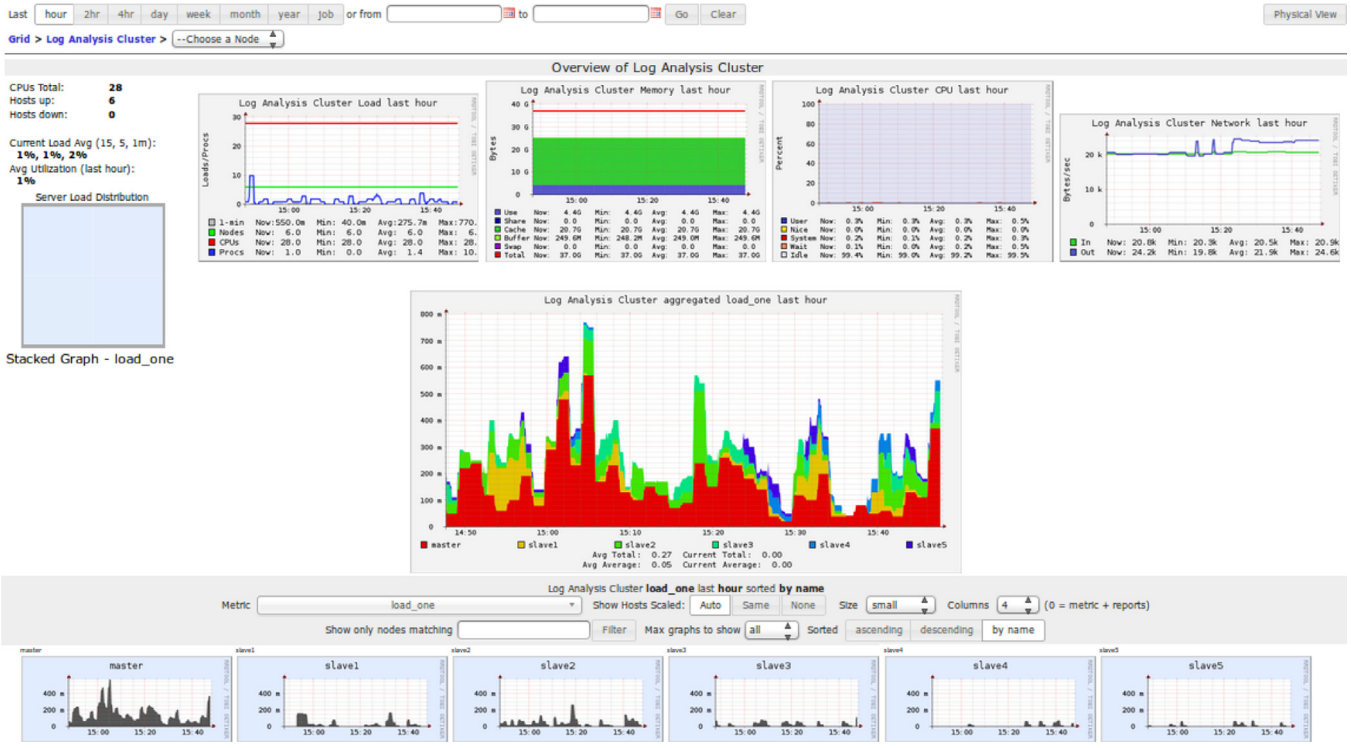

Fig. 6. Ganglia monitoring tool.

minor changes in the source code it could be possible to find the traffic per seconds till years. The second application finds the ten most popular requests and it displays them in descending order.

Taking into consideration that one of the most common attacks is denial of service attack (DoS attacks) (Lau et al., 2000), we developed two applications for detecting and help preventing this kind of malicious attack. So the third application that we developed for both frameworks searches for moments with abnormal big number of identical requests to the server and displays the time and the number of requests. The fourth application examines the suspicious for DoS attack moments, counts and displays the requests that every client made. By this way it is possible to detect the id of suspicious clients and take prevention measures like updating the system's firewall.

Finally we developed another two applications for error detection and correction. In the http log file when a response code is bigger than 300 that means that there is an error (http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html). So the fifth application identifies the type of errors and counts them. The final application takes the type of an error and returns the specific requests that create the majority of the errors. With these two applications a webpage administrator can identify, count and then correct the errors.

6. Performance analysis and evaluation

There are many works that investigated the performance of cloud computing and studied how various factors affect it (Conejero et al., 2014; Velkoski et al., 2013; Gu and Li, 2013; Vasconcelos and de Araujo Freitas, 2014; Feller et al., 2015; Batista et al., 2014; Zant and Gagnaire, 2015; Gao et al., 2011; Jiang et al., 2014; Chieu et al., 2011). In our study we focus on three performance indicators, execution time, resource utilization and scalability. We also estimated and compared the cost and power consumption of Hadoop and Spark based on resource utilization and execution time measurements.

6.1. Execution time

We conducted several experiments to examine how the total execution time of each application is affected by the number of active slave nodes, the size of the input file and the type of application.

6.1.1. Hadoop execution time

In Fig. 7 we see the execution times of six Hadoop applications that were executed with the same input file with different number of slave nodes. In Fig. 8 and Fig. 9 we see the results of the same experiments with different input file sizes.

In Figs. 7, 8 and 9 we observe an execution time reduction when the number of active nodes is increased and an execution time increment when the input file size gets bigger. These observations are completely reasonable because by these ways the processing volume for each node has been increased and as a result the processing time has been increased too.

We also see that in all cases the first three applications take considerably more time to run than the others. This happens because the Reduce Phase of the first three applications requires much more processing work than the Reduce Phase of the others. For example Fig. 10 shows the map and reduce execution times for two applications that presents big difference in overall execution time. As we can see for the first application the Map and Reduce execution times is almost the same whereas in the second application the Reduce execution time is significantly less.

In Fig. 11 we see the average execution times of the six Hadoop applications. We observe that there is a slight difference in execution times between two to five nodes for the smaller file. This makes sense because the file is relatively small and two nodes have enough computing power to execute the required processes. On the other hand for the same reason we see a bigger difference in execution times for the larger files where each additional node affects more the total execution time.

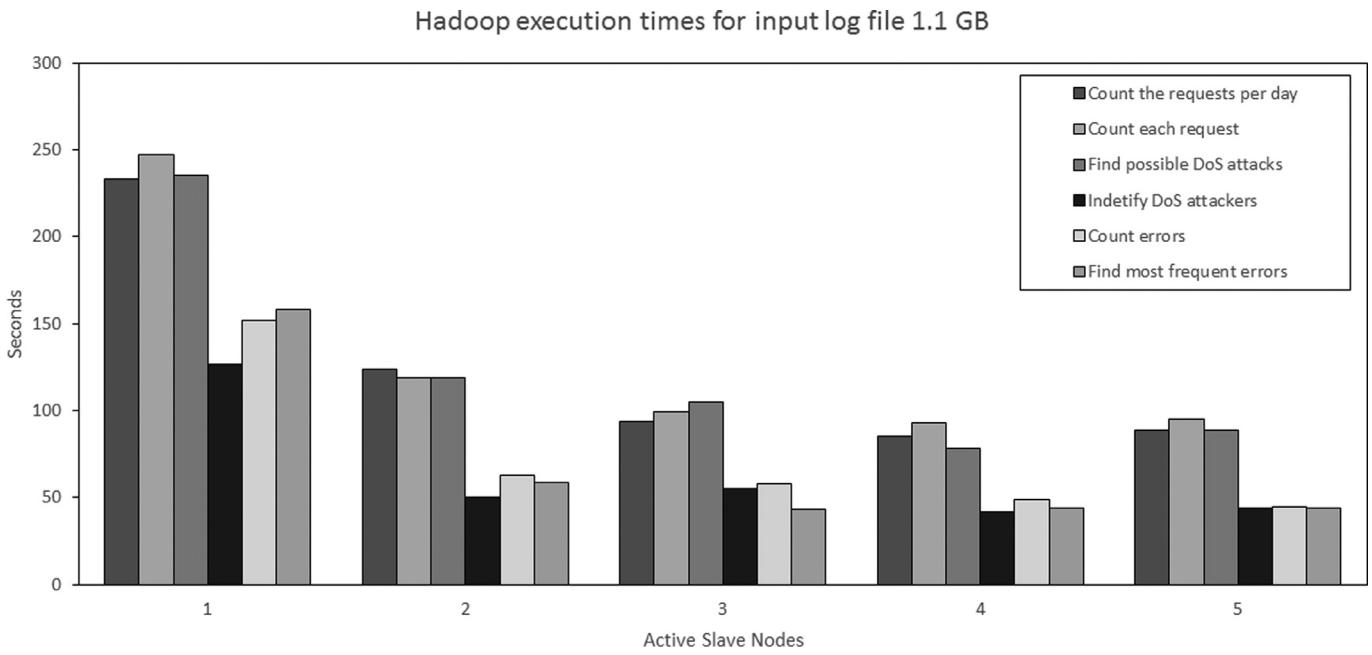

Fig. 7. Execution time of Hadoop applications with 1.1 GB input file.

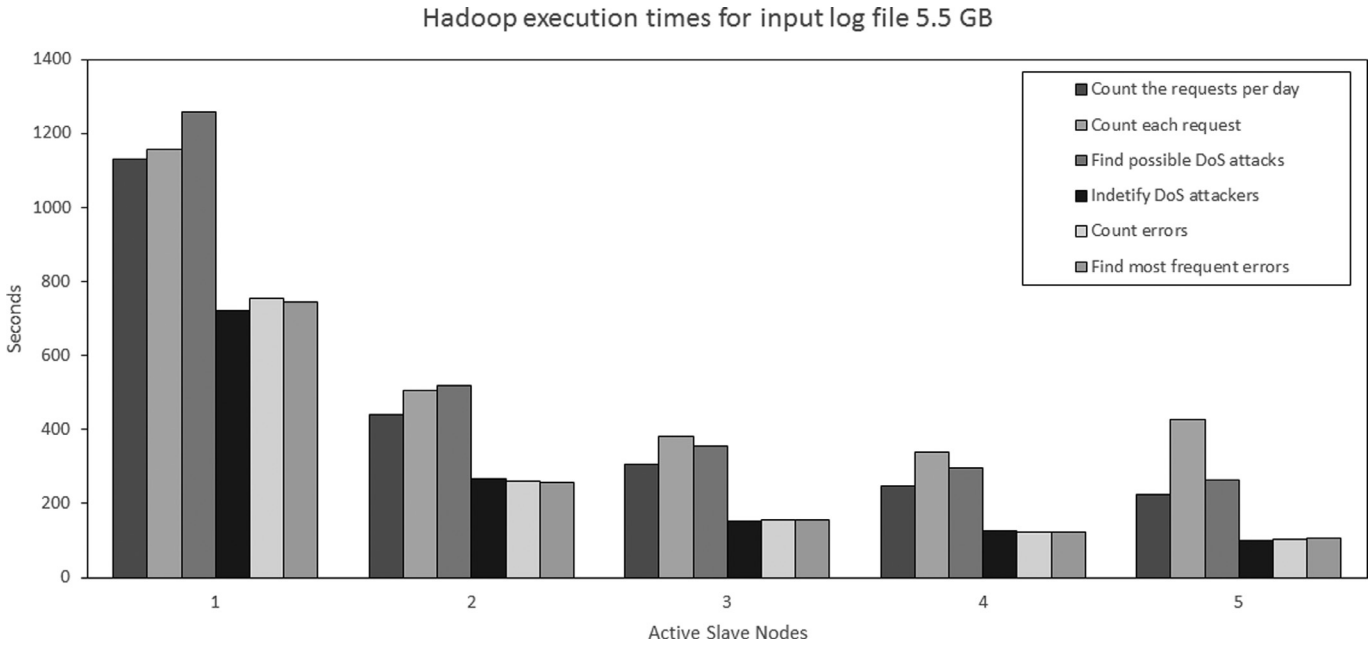

Fig. 8. Execution time of Hadoop applications with 5.5 GB input file.

6.1.2. Spark execution time

In correspondence to Hadoop experiments, relevant experiments carried out with Spark and the results are presented in Figs. 12, 13, 14. Spark applications can run as a standalone Spark applications or executed on YARN. For the following experiments the applications run as standalone Spark applications (both are supported from the developed system).

In these experiments we generally observed that Spark is significantly faster than Hadoop. As shown in Figs. 12, 13 and 14 the increment of the size of the input file or the reduction of active slaves increase the applications execution time especially when only one node remains active. Also the execution time differs from application to application due to different computing needs.

In addition we executed the find topten requests application with the same input file in the same cluster but in a different way in order to highlight the differences in execution time. Fig. 15 shows how the execution time is affected depending on whether the application runs on YARN or standalone. The execution of Spark applications on YARN offers additional features such as monitoring, dynamic resource management of the cluster, security through Kerberos protocol, possibility of parallel execution of various tools (e.g. MapReduce, Hive) and other features that are not supported by the standalone mode. However, we see that in Fig. 15 the execution of the applications on YARN is quite slower than standalone; this happens because YARN has a quite complex resource management and scheduling. Also as shown in Fig. 15, there are two types of YARN

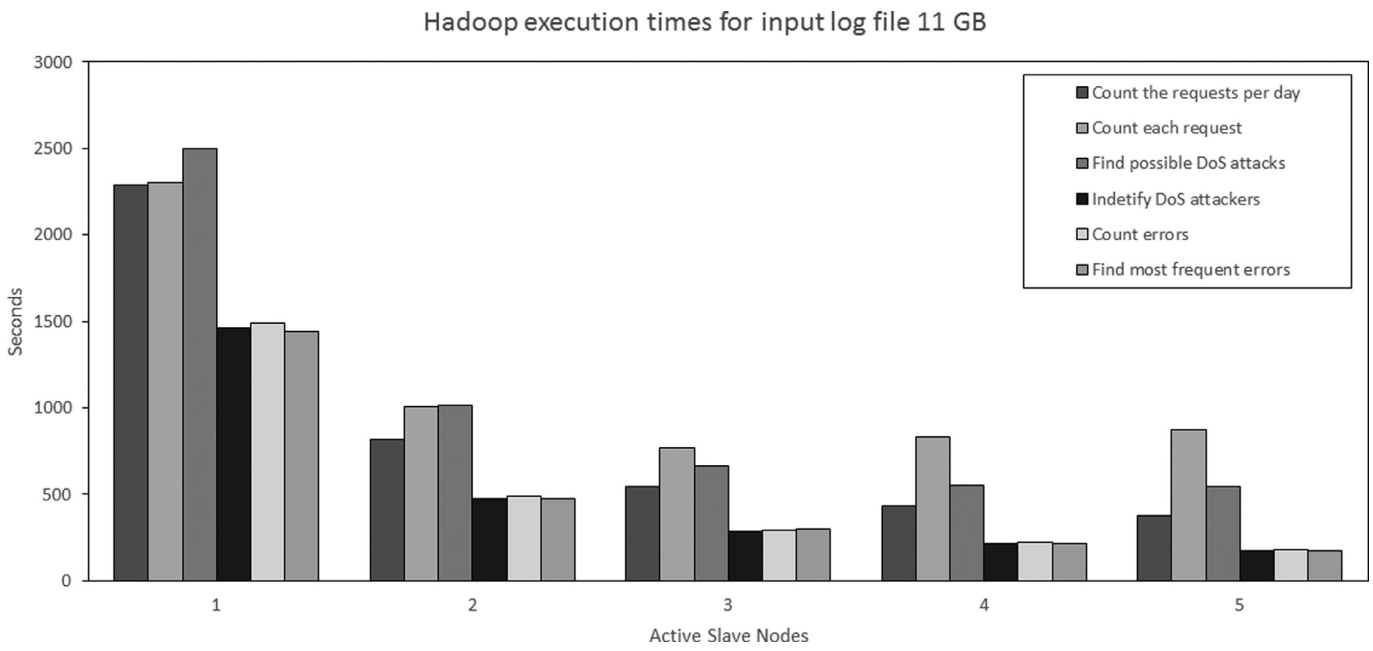

Fig. 9. Execution time of Hadoop applications with 11 GB input file.

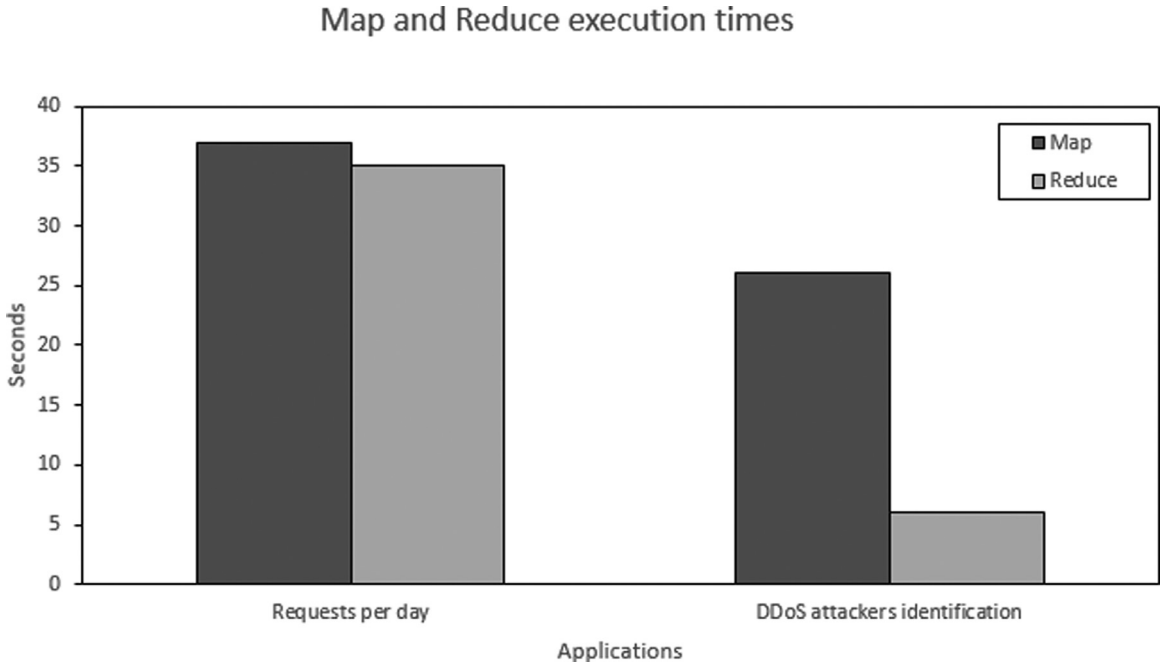

Fig. 10. Execution time of Map and Reduce phases for two different applications.

modes. In cluster-mode the driver runs in a process of the master who manages YARN. On the contrary in client-mode the driver runs on clients process (https://databricks.com/blog/2014/03/26/spark-sql-manipulating-structured-data-using-spark-2.html).

In Fig. 16 we see a direct comparison of mean execution time of Hadoop and Spark similar applications, for different data inputs, with different number of slave nodes. Is clear that Spark applications with the same input and number of slave nodes are much faster than similar Hadoop applications with the same options.

From Fig. 16 is clear that Spark is faster in every case. However is interesting to study how Spark will perform with much less available memory resources. We configured our cloud with less memory (1 GB per node) and we ran again the count error application with input file of 1.1 GB and 11 GB. We took the measurements that appeared in Table 2. And in this case Spark is faster than Hadoop. Spark achieved almost the same execution time as with the larger memory which was quite unexpected. This may happened because Spark uses techniques like lazy evaluation nad it claims that it can be up to ten times faster only by using the disk.

6.1.3. SQL-type execution times

Fig. 17 presents the execution times of Hadoop Hive and Spark SQL that are used for SQL-type querying. For these experiments Spark SQL ran in standalone mode and the execution time of Spark SQL queries improved significantly when the table was saved in main memory. Fig. 17 shows the execution time of the same SQL code that was executed in both Hive and Spark SQL and counts the

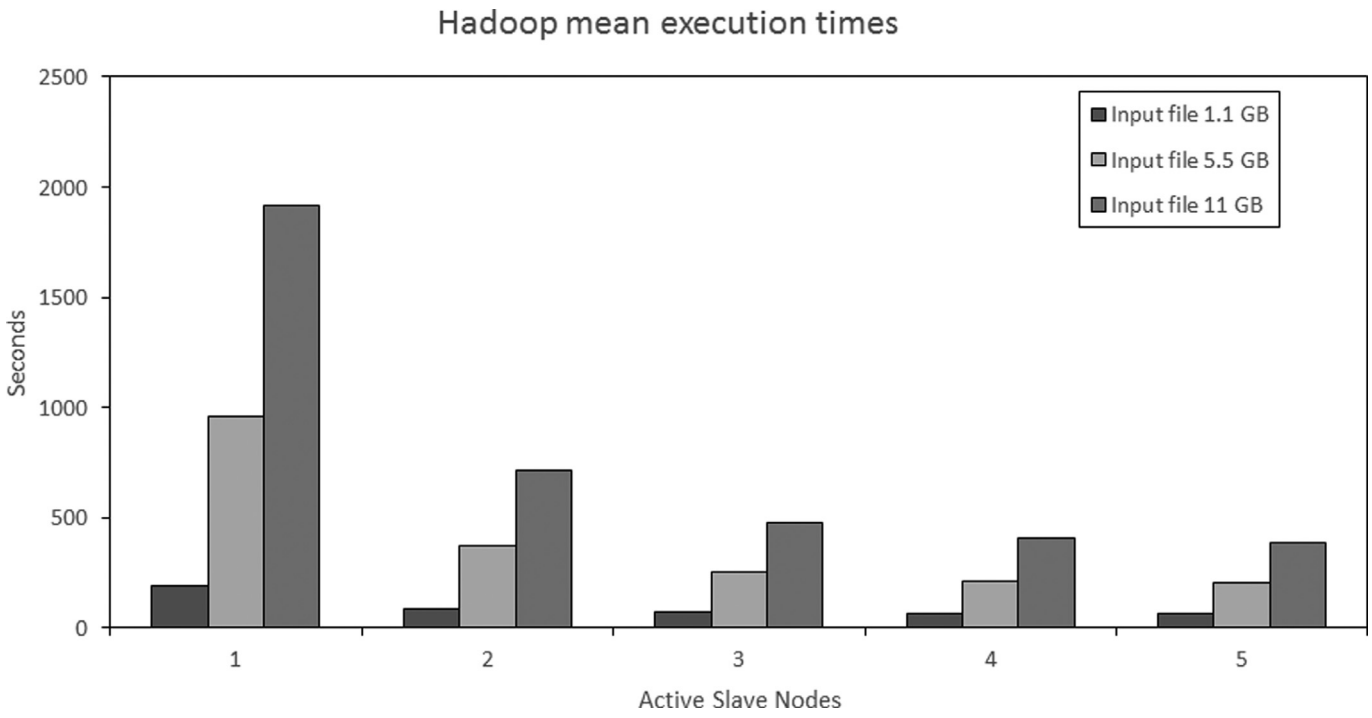

Fig. 11. Mean execution time of all Hadoop applications.

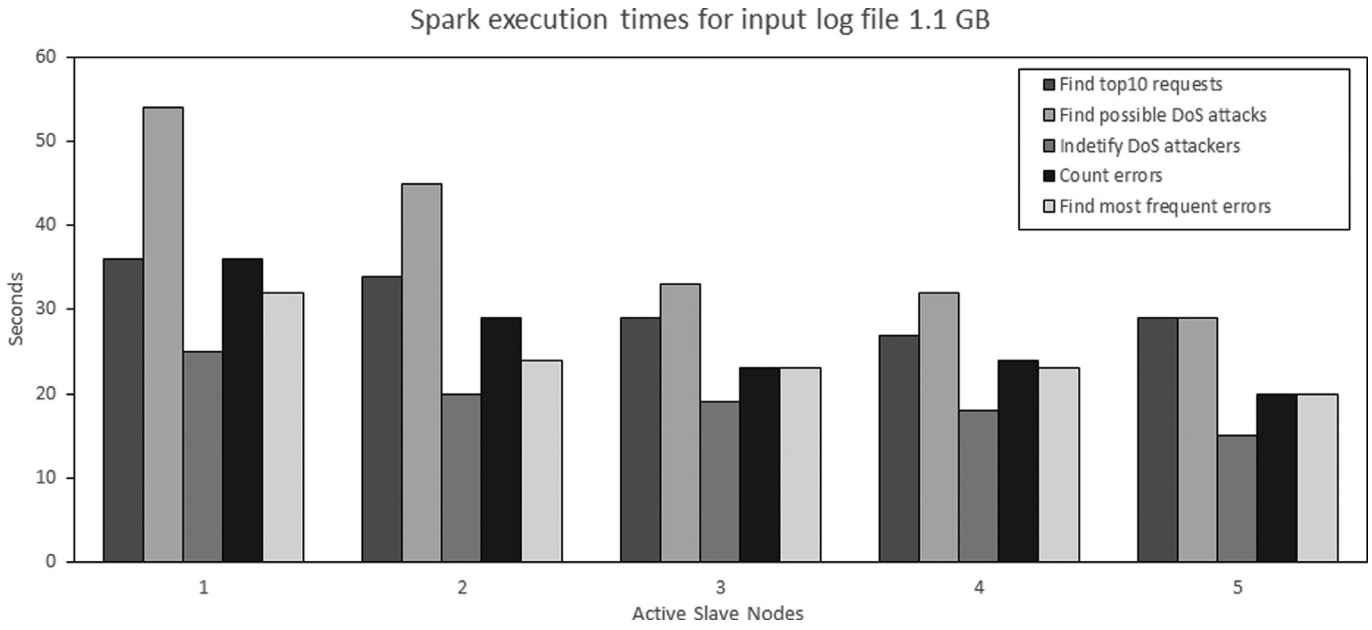

Fig. 12. Execution time of Spark applications with 1.1 GB input file.

Table 2
Execution times of count error application for different memory configuration.

Input file	Spark cluster with 1 GB memory per node	Hadoop cluster with 1 GB memory per node	Spark cluster with 6 GB memory per node	Hadoop cluster with 6 GB memory per node
1.1 GB	24 seconds	60 seconds	20 seconds	45 seconds
11 GB	57 seconds	221 seconds	58 seconds	176 seconds

Table 3
Execution times of Hadoop and Spark applications running alone and at the same time.

Input file	Spark alone	Hadoop alone	Spark execution times with Hadoop	Hadoop execution times with Spark
1.1 GB	20 seconds	45 seconds	240 seconds	180 seconds
11 GB	58 seconds	176 seconds	570 seconds	540 seconds

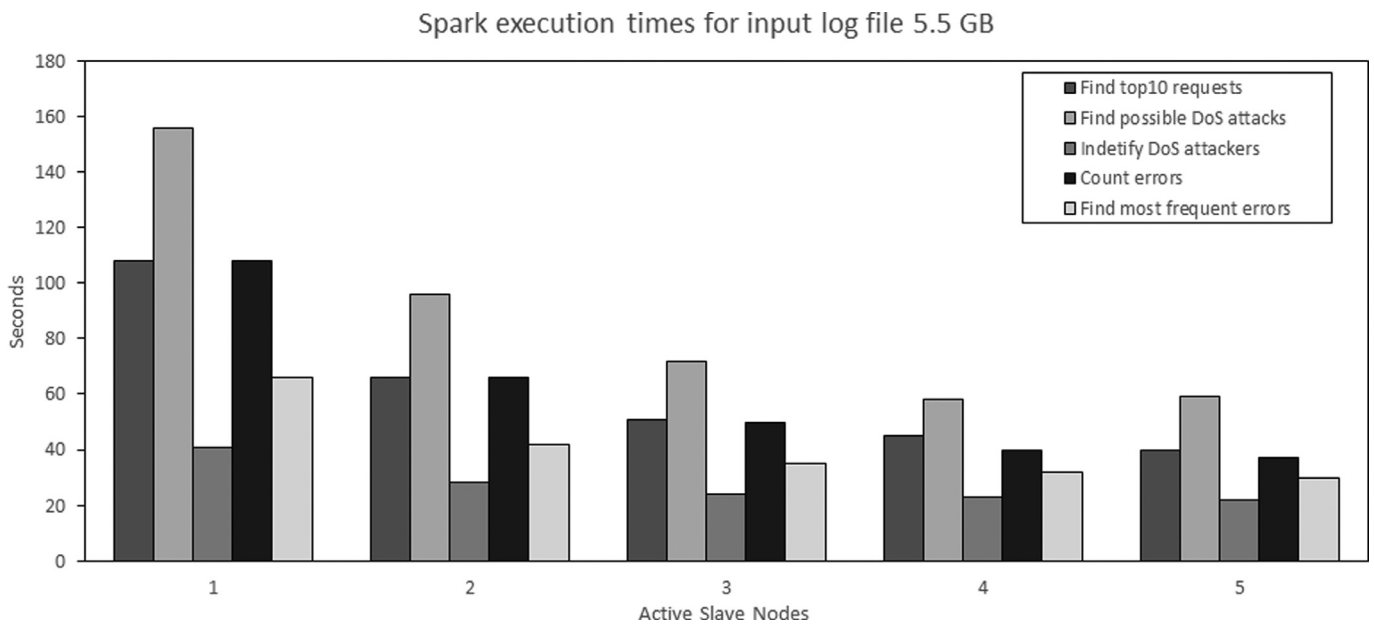

Fig. 13. Execution time of Spark applications with 5.5 GB input file.

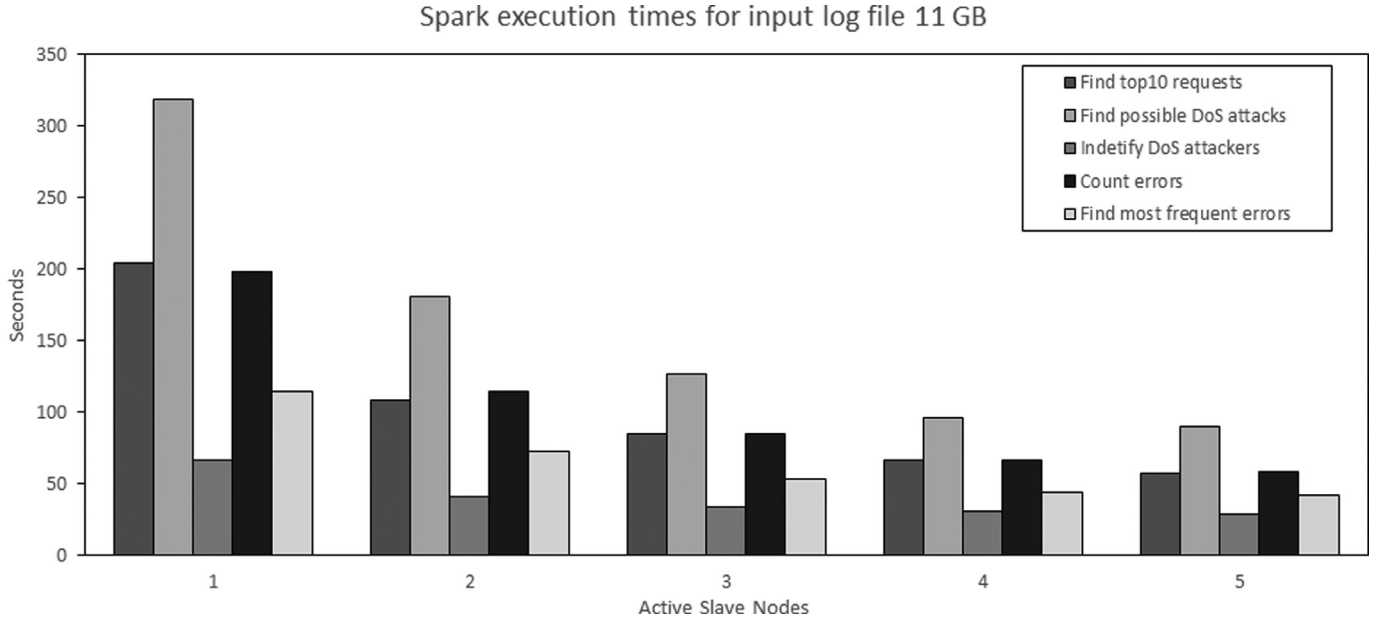

Fig. 14. Execution time of Spark applications with 11 GB input file.

errors in a log file. After experiments we concluded that Spark SQL is much faster than Hive. This happens because Spark SQL has a set of techniques to prevent reads and writes to disk storage, caching of tables in memory and optimizing efficiency.

6.1.4. Overall execution time results

Various measurements have shown how the size of the input file, the type of running application and the number of available nodes in the cluster affect the total execution time. To directly compare the two frameworks we developed two same applications and SQL queries that executed on both frameworks. The first application is about error counting and the second is about error finding. Fig. 18 shows the execution times for the first application, for both frameworks, for input files of 1.1 GB, 5.5 GB and of 11 GB. Spark is the fastest, follows the Spark SQL, and then the Hadoop MapReduce and Hive with big difference in execution times. These results are in complete agreement with what has been previously described and confirm that Spark in most cases is quite faster than Hadoop. The same conclusion is drawn from Fig. 19 that shows the execution times of the second application for both frameworks.

Except of the execution time, we also studied the internals of each query in order to identify some interesting characteristics of these applications. The applications that we developed are mainly I/O intensive and the number of bytes that been read from the HDFS are a little more than the actual log file and the written bytes are in general a few hundred bytes. Also as it was expected the number of maps is bigger than the reducers because is driven by the number of HDFS blocks in the input files. Furthermore we confirmed that in all cases the tasks are data-local task, that means that the block is physically on the same node like the computation. This happens because in the initial cluster setup we changed the

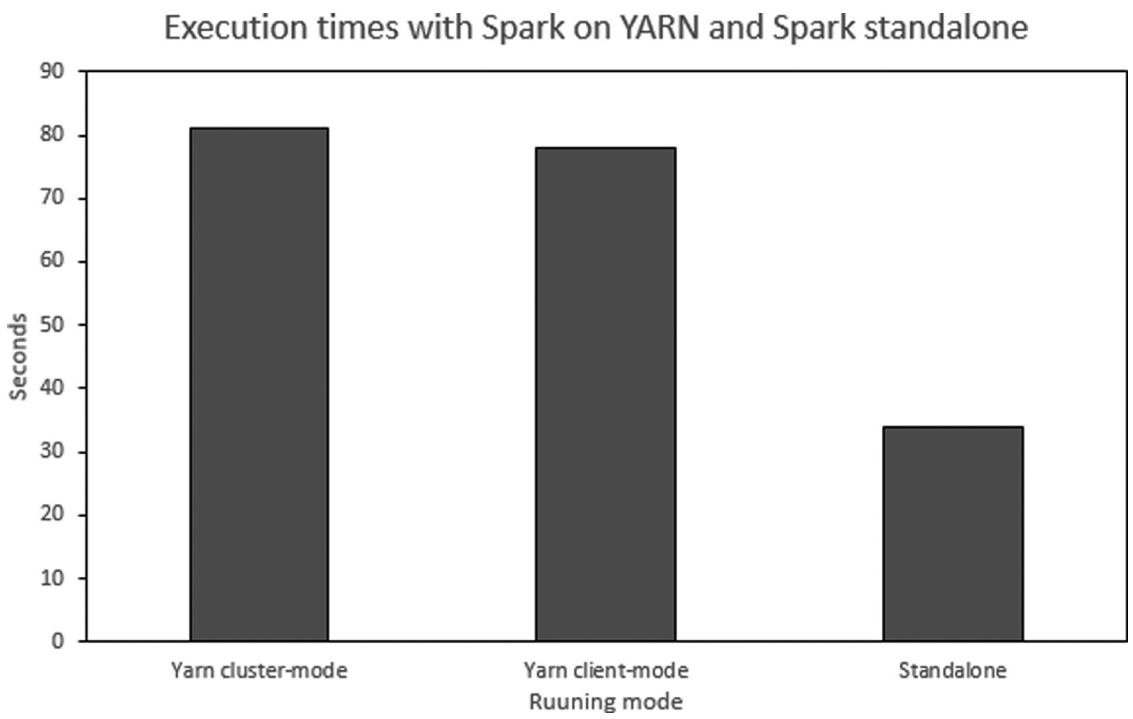

Fig. 15. Spark on YARN and Spark standalone.

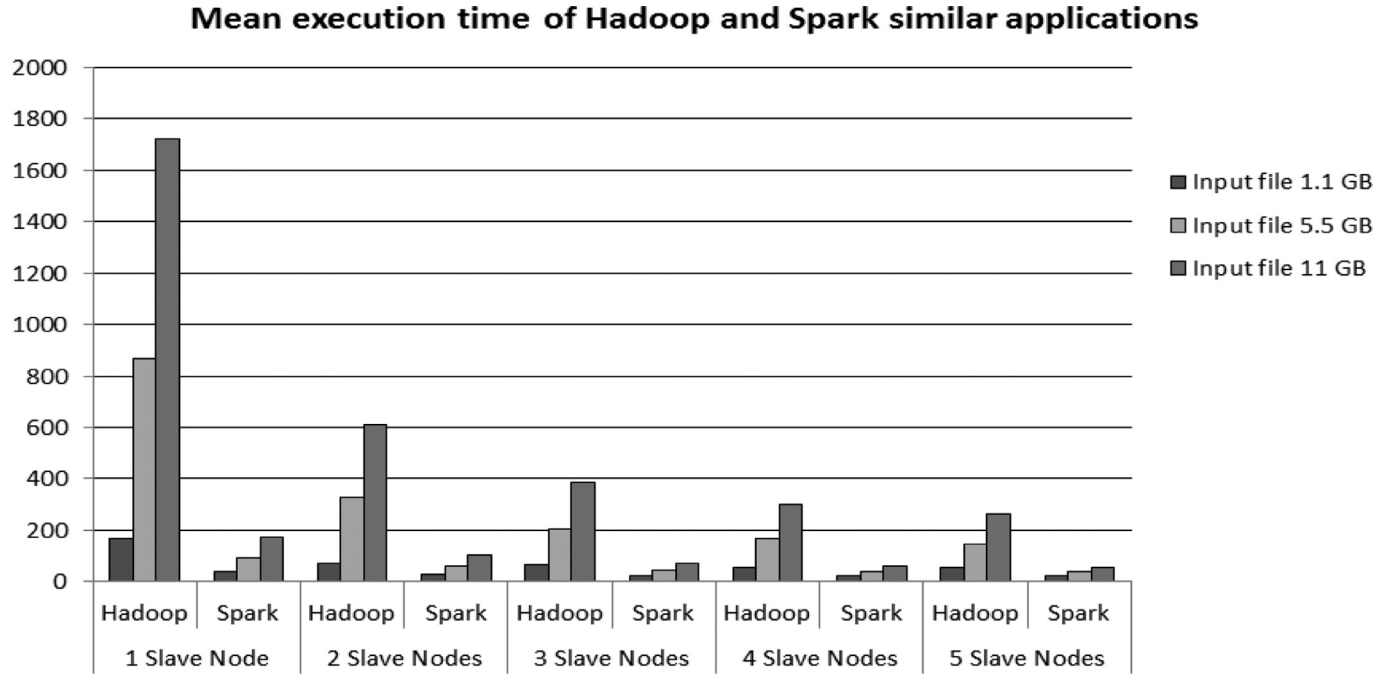

Fig. 16. Mean execution times of Hadoop and Spark similar applications.

replication factor to the number of the total nodes and as a result every node stores all the blocks of every HDFS file.

6.2. Scalability and cost evaluation

Scalability is the ability of an application to be scaled up to meet demand through replication and distribution of requests across a pool of resources (Chieu et al., 2011). This implies that scalable applications only require more computing resources to meet the additional load rather than other major changes.

Scalability is a critical factor to the success of many organizations because the resources requirements can vary significantly from time to time. Without scalable applications and infrastructure an organization has to pay a very high cost to always maintain the resources to meet the top requirements. On the other hand, if it reduces the cost to minimal resources then it will probably emerge performance and operational problems at peak hours.

Cloud Computing is the ideal infrastructure to develop and run scalable applications. By providing on-demand resources like processors, memory, storage and networking, a user can add or remove any time any kind of resources to satisfy the current needs.

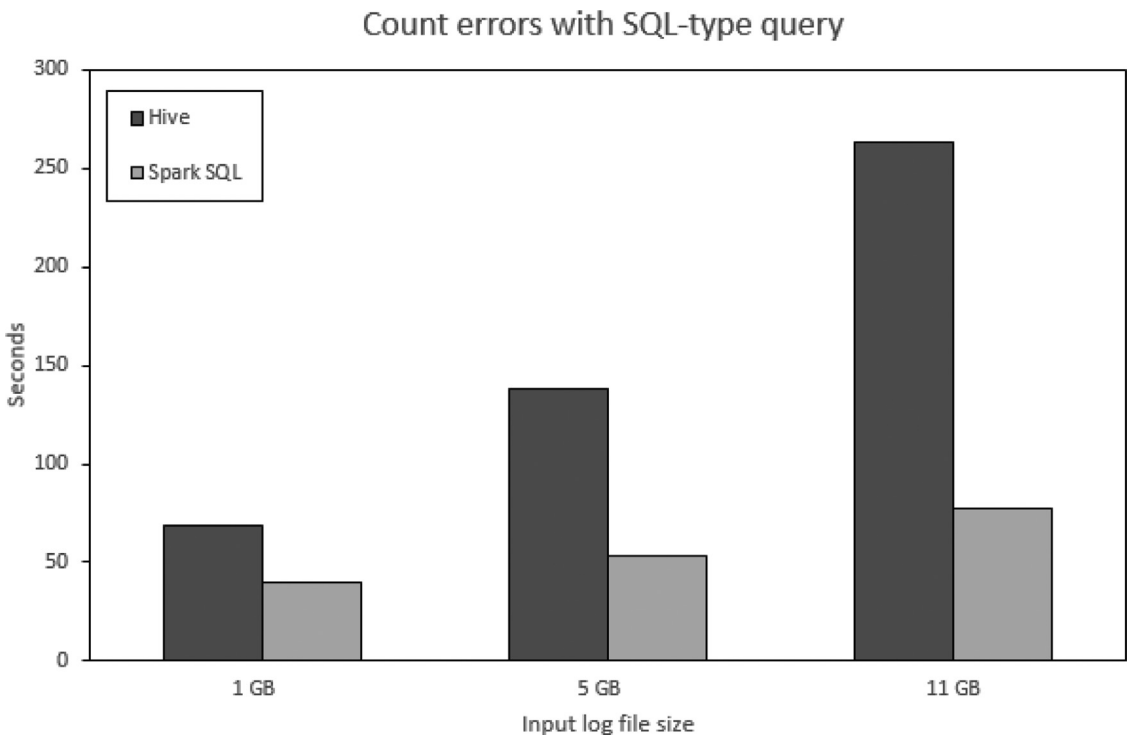

Fig. 17. Execution time of SQL-type queries.

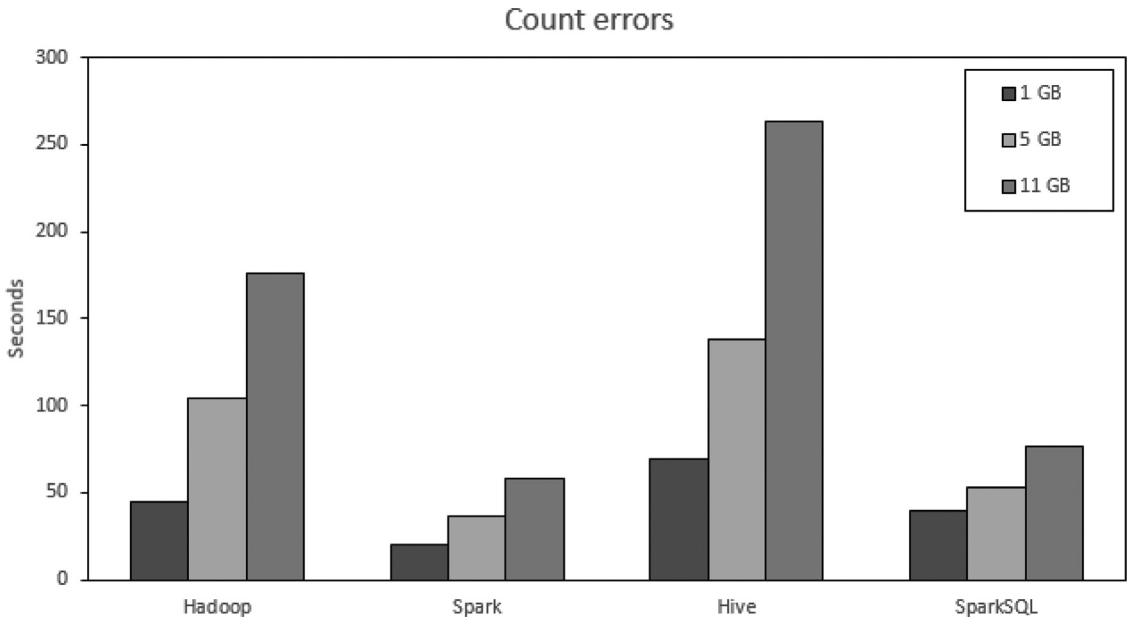

Fig. 18. Execution time of error counting application and queries in all frameworks.

Also because cloud providers follow the pay-as-you-go billing model with per second, minute or hour charges, cloud is very cost effective too.

However, not every application or computing framework is scalable optimized and maybe there are some factors or a point that the additional resources do not lead to equivalent performance improvement. As a result the evaluation of scalability of a cloud services is very important and it was studied as a quality attribute in many different ways and in various aspects (Lee & Kim, 2010). We evaluated Hadoops and Spark's scalability by investigating how different factors, such as number of cluster nodes and dataset size affect it.

We performed experiments by executing more than ten different realistic log file analysis applications in Hadoop and Spark with real log data and we took measurements for every case. In these experiments we used three datasets of 1.1, 5.5 and 11 GB each, with cluster size variation from one to five slave nodes.

Fig. 20 presents the mean execution time of all applications in Hadoop and how the size of the input file and the number of active nodes in the cluster affect it. In Fig. 21 we see how the same changes in the size of the input file and the number of nodes affect Spark applications execution time. At first glance we see in both Fig. 20 and Fig. 21 that Spark and Hadoop follow similar pattern although Spark is faster.

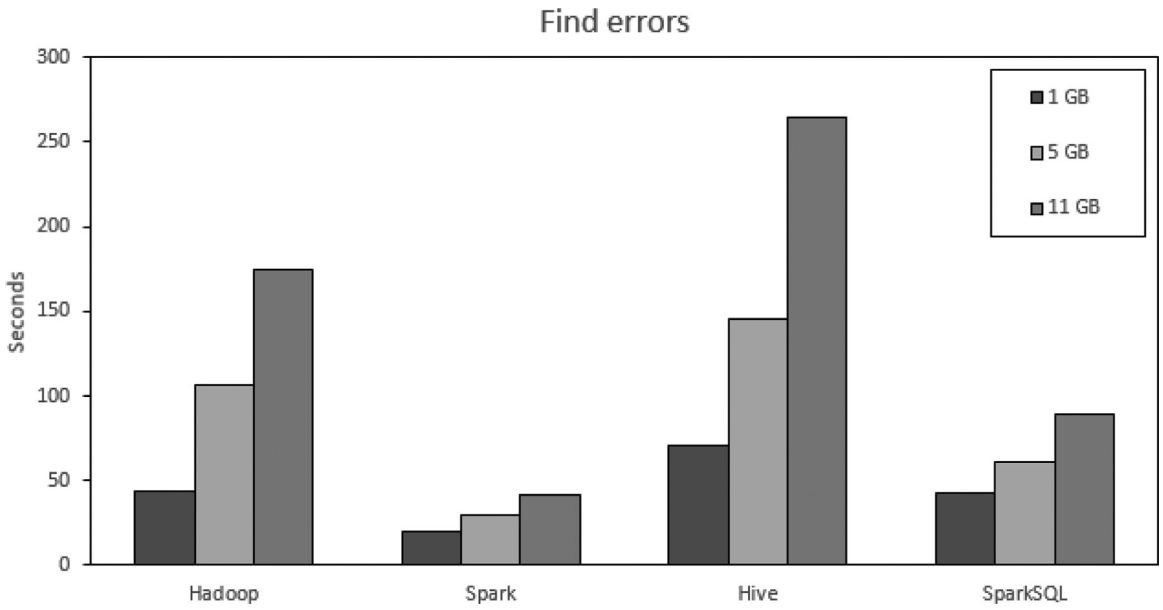

Fig. 19. Execution time of error finding application and queries in all frameworks.

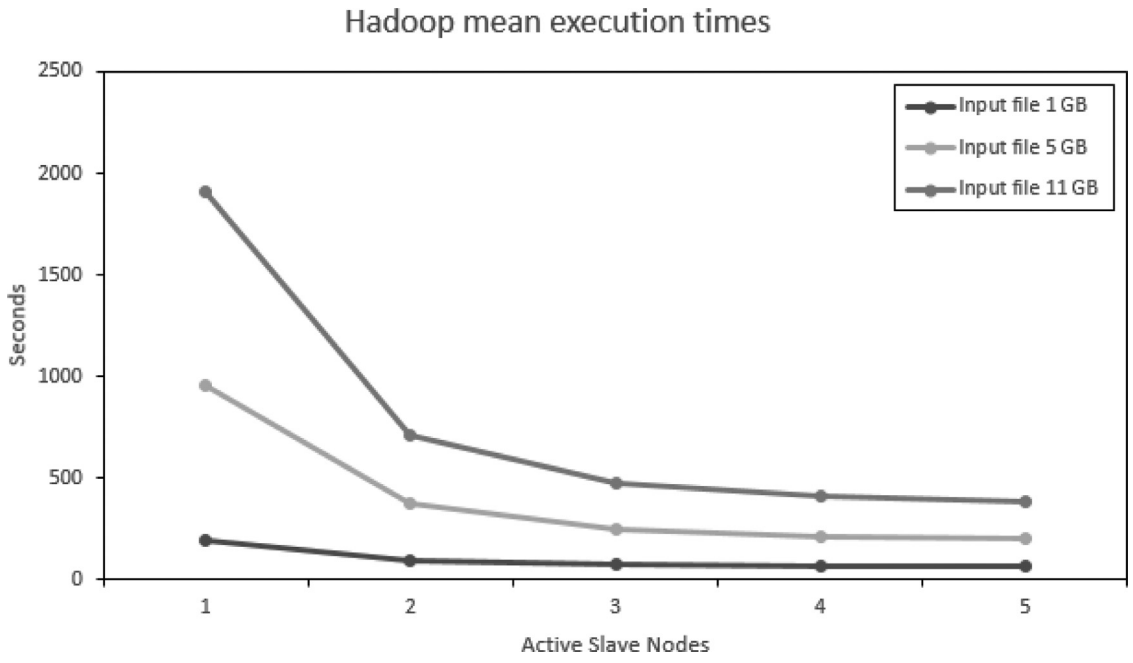

Fig. 20. Mean execution times of Hadoop applications.

In Figs. 20 and 21 we observe that for the same number of nodes when the input file from 5.5 GB increased to 11 GB the execution time almost was doubled. Also we see that for the two biggest files when the number of nodes dropped from four to two, the execution time almost was doubled too. On the other hand we see a different behavior for the small file of 1.1 GB. This happens because the file is relatively small and the extra nodes cannot improve further the execution time.

In Figs. 20 and 21 we observe also that especially for the smaller file and for three to five active slave nodes the total execution time is almost the same. Although the difference in execution time from three to five nodes is very small it comes with a big cost. In our study we used a free IaaS (Okeanos) and we used nodes with same computing and storing capabilities. If we assume that the renting cost for every node is the same, it seems reasonable to make a small sacrifice in total execution time in order to succeed a big cost improvement.

For the specific log analysis applications we study how we could take advantage of the scalability observations and achieve a significant cost reduction by accepting a burden in execution time. For the specific infrastructure and case studies we can achieve a mean 40% cost reduction by accepting a 11-19% burden in execution time, or 20% cost reduction by accepting a 1-6% burden in execution time for Hadoop applications. Also for Spark applications we can achieve a mean 40% cost reduction by accepting a 11-28% burden in execution time, or 20% cost reduction by accepting a 5-9% burden in execution time for the different input files.

In Fig. 20 we observe also a superlinear speedup from one to two slave nodes for the 11GB input file. Gunther et al. studied in depth this phenomenon and they characterized it as quite com-

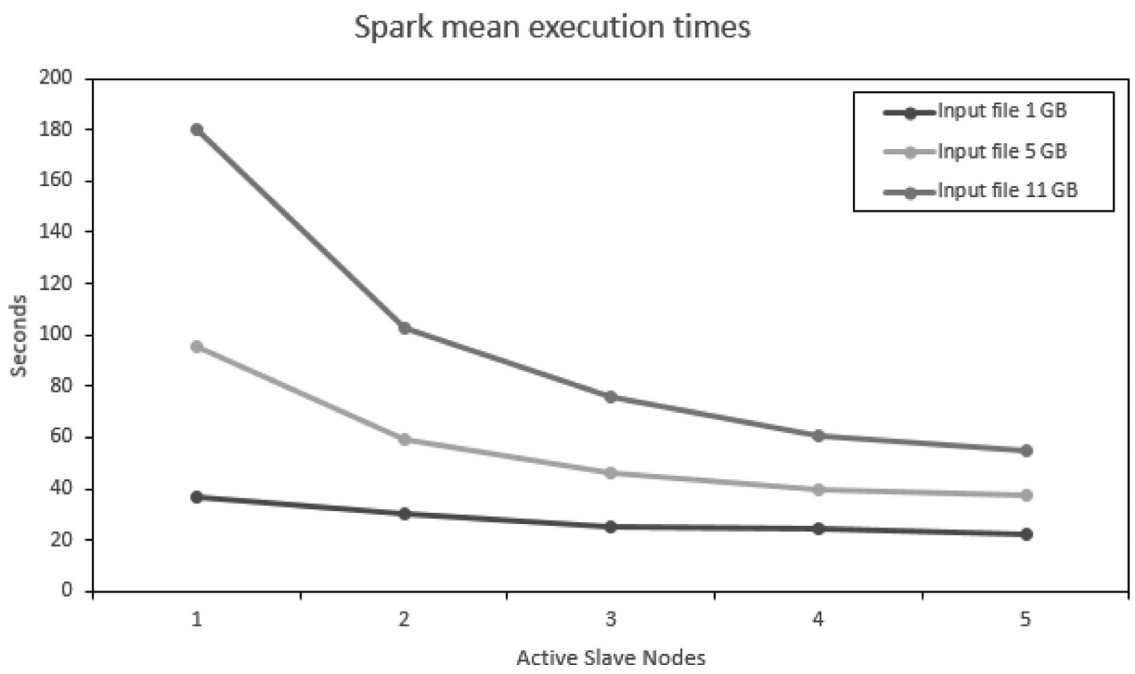

Fig. 21. Mean execution times of Spark applications.

mon performance illusion (Gunther et al., 2015). After several experiments with Terasort benchmark and Hadoop, they identified an I/O bottleneck that causes the Hadoop framework to restart the Reduce task file-write, which stretches the measured runtimes and leads to superlinear speedup measurements.

From these experiments we can safely conclude that both frameworks had great scalability and can handle the increasing computing demands for log file analysis under the condition that the cost of required additional nodes is affordable. Also we can take into consideration that we can achieve a significant cost reduction by taking advantage of the utilization measurements.

6.2.1. Resource utilization

We compared the resource utilization of the two frameworks by presenting the experimental utilization results for the same application in both frameworks. We ran the count error application with input log file of 5.5 GB and we recorded CPU, main memory, disk and network utilization information per second from all the nodes of the cluster. We executed the application for different cluster configuration with the same input file and we took the measurements that appear in Figs. 22–25.

In Fig. 22 we see the mean percentage utilization of the processors of the cluster for different number of active nodes. We took measurements for the percentage of CPU utilization that occurred while executing at the user level (%usr), percentage of CPU utilization that occurred while executing at the system level (%sys), percentage of time that the CPUs were idle during which the system had an outstanding disk I/O request (%iowait), percentage of time spent in involuntary wait by the virtual CPUs while the hypervisor was servicing another virtual processor (%steal), percentage of time spent by the CPUs to service software interrupts (%soft) and percentage of time that the CPUs were idle and the system did not have an outstanding disk I/O request (%idle). In this figure we observe that the two frameworks have almost the same behavior and the CPU utilization gets higher when remains only one active slave node.

In Fig. 23 we see the KiloBytes of memory needed for current workload in relation to the total amount of memory. By this way we can clearly see that Spark does better memory usage than Hadoop does. As we mentioned before Spark uses more effectively the main memory and achieves better performance.

In Fig. 24 we see the mean number of KiloBytes received (rxkB/s) and transmitted (txkB/s) per second. We observe that Spark uses on average more network resources than Hadoop does. This is reasonable because Spark processes the same amount of data and makes almost the same network transfers as Hadoop but in significantly less time.

Finally in Fig. 25 we see the mean number of total reads (rtps) and writes (wtps) requests per second that were issued to the disk. In first look we see that on average Spark uses disk more than Hadoop does, which is quite strange. This may happens because in our experiments Spark processes the same amount of data and makes the same disk accesses as Hadoop but in significantly less time. Also we observe that the majority of requests in both frameworks are read requests; this may vary from application to application and happens to the specific occasion because we measure the mean disk usage of a word count type application.

Also from these experiments we see that Hadoop is much more resource saving, which means that in a cloud environment it could allow other co-executing applications to run without interference. So we ran at the same time a Hadoop and Spark application at the same cluster to study how the performance of the one is affected by the other. We executed the count error application with input files of 1.1 GB and 11 GB with five active slave nodes, the results are in Table 3. We observed that when we ran together the two applications, Hadoop's execution time is better than Spark's execution time. This happens because Spark is more resource consuming and when executed with other application does not have enough resources to perform optimally.

6.2.2. Power Consumption

Ideally for accurate monitoring of power consumption every node of the cluster should be measured by physical watt-meters. Nevertheless due to the size of cloud computing infrastructures, the virtualization techniques and the special features of cloud computing frameworks, monitoring of power consumption is not a trivial task.

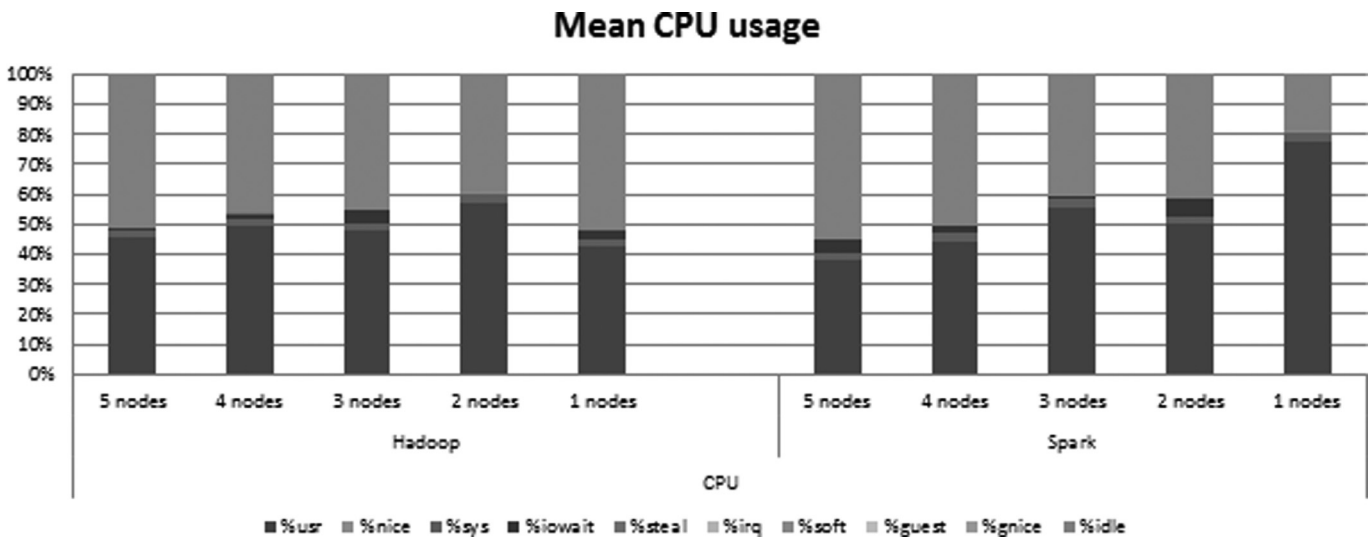

Fig. 22. Processor utilization.

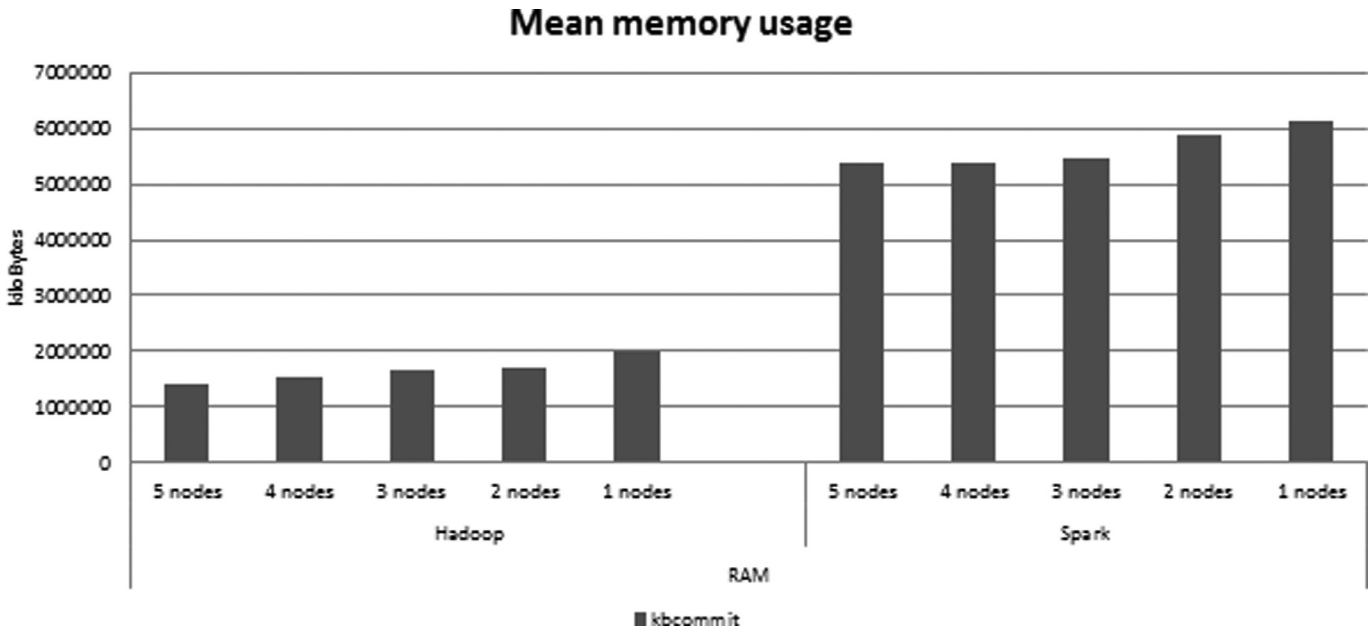

Fig. 23. Memory utilization.

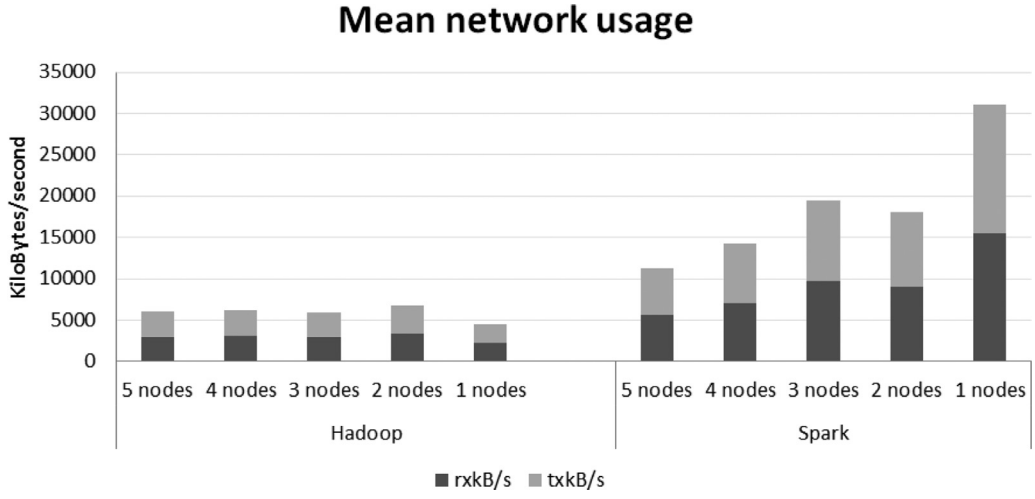

Fig. 24. Network utilization.

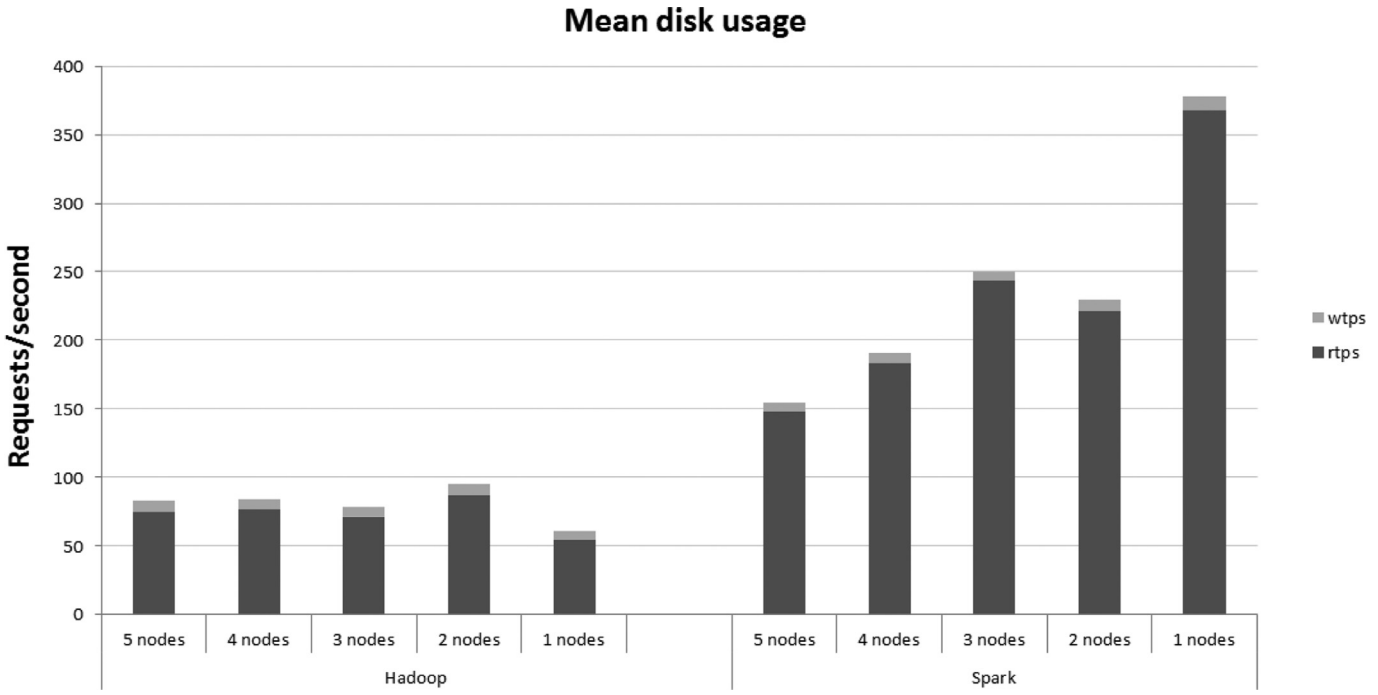

Fig. 25. Disk utilization.

To evaluate and compare the power consumption of Hadoop and Spark we requested from our IaaS provider (Okeanos) to physically monitoring our computing resources power consumption but as we are told this was infeasible. As we could not get any power consumption data to make a power consumption estimation, we used a power consumption model to predict it. Because all worker nodes in our cluster are homogeneous we made the assumption that same tasks in different nodes consume the same power and that the number of VMs is the same on every physical machine. For example we assume that reading and writing to memory or disk consume the same energy in every node.

In our case every node has the same components but this is not the case in many cloud infrastructures, a study from Google (Reiss et al., 2012) claims that there are more than ten generations of physical machines with different specifications in its production cluster. Also the number of VMs on a physical machine can differ also, with much higher energy consumption when the number of VMs configured on a physical machine increased (Ghosh et al., 2011).

There are a lot of studies that investigated how components like the CPU, memory, network interfaces and disk affect power consumption (Carpen-Amarie et al., 2013; Fan et al., 2007; Orgerie et al., 2010b; Yue and Zhu, 2007; Jimeno et al., 2008; Allalouf and Arbitman, 2009; Hylick and Sohan, 2008). The CPU is considered the most power demanding component. There are detailed analytical power models like Joseph (2001) which are real-time and highly accurate, but are not so general and portable because they are based on microarchitectural knowledge of a particular processor. Also these models rely on the assumption that CPU is the main power consumer, which is invalid for workloads that involve a large amount of I/O activity, as in the case of MapReduce (Bohra and Chaudhary, 2010).

Many studies are based on the correlation between power consumption and resource utilization. Like Lee and Zomaya (2010) that in their model assume that the relation between CPU utilization and energy consumption is linear. Chen et al. (2011b) propose a linear power model too. Their model take into consideration the power consumption from individual components to a single work node. Joulemeter (Kansal et al., 2011) monitors the resource usage of VMs and then converts it to energy consumed based on the power model of each individual hardware component.

Bohra and Chaudhary (2010) also proposed a power consumption model that is based on the correlation between the power consumption and components utilization. The authors proposed a four-dimensional linear weighted power model (Eq. 1) for the total power consumption.

$$P_{\text{total}} = c_0 + c_1 P_{\text{CPU}} + c_2 P_{\text{cache}} + c_3 P_{\text{DRAM}} + c_4 P_{\text{disk}} \quad (1)$$

In Eq. 1 the P_{CPU}, P_{cache}, P_{DRAM} and P_{disk} are specific performance parameters for CPU, cache, DRAM and disk, and c_0, c_1, c_2, c_3, c_4 are weights.

Chen et al. (2011a) modified the proposed model of Bohra et al. to the following model:

$$P_{\text{total}} = P_{\text{idle}} + c_1 P_{\text{CPU}} + c_2 P_{\text{HDD}} \quad (2)$$

We also further modify the Chen et al. model by adding the contribution of network hardware and the model transformed to Eq. 3.

$$P_{\text{total}} = P_{\text{idle}} + c_1 P_{\text{CPU}} + c_2 P_{\text{RAM}} + c_3 P_{\text{HDD}} + c_4 P_{\text{Network}} \quad (3)$$

To compare Spark and Hadoop-MapReduce power consumption we applied our proposed model based on the CPU, memory, disk and network utilizations. We assumed that the hardware weights are the same for Hadoop and Spark applications. Based on the utilization measurements for the count errors application with input file of 5.5 GB and five slave nodes cluster, we found the relationships of power consumption of Hadoop and Spark for the specific configuration (Eq. 4).

$$P_S = P_{\text{CPUHadoop}} + 1.9 P_{\text{RAMHadoop}} + 3.8 P_{\text{HDDHadoop}} + 1.9 P_{\text{NetworkHadoop}} \quad (4)$$

Where P_S is the Spark power consumption, $P_{\text{CPUHadoop}}$ is the cpu power consumption of Hadoop, $P_{\text{RAMHadoop}}$ is the memory power consumption of Hadoop, $P_{\text{HDDHadoop}}$ is the disk power consumption of Hadoop, and $P_{\text{NetworkHadoop}}$ is the network power consumption of Hadoop.

Subsequent studies like Orgerie et al. (2010a) proved that there is no linear relationship between CPU utilization and power consumption, as considered in several works (Steinder et al., 2008; Chen et al., 2008). They conducted experiments and they found that utilization has an impact on power consumption but the impact is not linear in all cases. The relation can be linear if the type of workload is the same and the external environment is the same also.

Also study (Orgerie et al., 2010a) showed that the energy used with the different ethernet link's data rates are not proportional to utilization. The authors even indicated that for some network devices, moving data consumes less energy than doing nothing. So, their findings are opposite to other studies like Lee and Zomaya (2010) which stated that switch power consumption depends on the number of sent bits.

The modeling of the energy consumption of cloud infrastructure is a complex task depending on various contexts (e.g. rack position). It is a hot topic with several studies and many of them are in conflict. As future work we could further improve our proposed model and validate it by using physical watt-meter measurements.

6.3. Discussion

We experimentally compared Hadoop and Spark and we evaluated the performance of them by investigating the execution time, scalability, resource utilization, cost and power consumption. From these experiments we concluded that Spark outmatch Hadoop in almost all cases. However Hadoop-MapReduce was built for batch processing and it is more suitable and cost-effective for truly Big Data processing.

From our application developing experience with Hadoop and Spark, we found that is more easy to develop Spark applications than Hadoop MapReduce applications. Spark has also build-in modules for stream processing, machine learning and graph processing that can been combined in the same application. For these reasons Spark can be used for more complicated and powerful applications.

Also with our experiments we demonstrated that Hadoop and Spark can coexist in the same cloud infrastructure and access and process the same HDFS data without any problem. According to the needs, the size of the data and available resources, a user can select Spark or Hadoop to process the HDFS data or even he can choose to run both of them at the same time.

7. Conclusions

In this work we studied the analysis of log files with the two most widespread frameworks in cloud computing, the well-established Hadoop and the rising Spark. In order to evaluate the performance of the frameworks we developed, executed and evaluated the performance of realistic log analysis applications in both of them.

Hadoop is one of the first frameworks for cloud computing, is widely known and is used for years by many big companies. Over the years Hadoop evolved and improved in order to meet the new era needs. These new needs led also to the creation of Spark. Spark is faster and more flexible than Hadoop. Spark achieved to dramatically reduce the execution time by saving the intermediate results in memory instead of the disk. Also, Spark except of MapReduce functions, supports a wide range of new capabilities that can be combined to generate new hitherto been impossible applications.

We conducted several experiments with different number of slave nodes, size of input file and type of application in order to confirm Sparks best performance. Initially we studied the total execution time of Hadoop and Spark log analysis applications and we analyzed the factors that affect it. We found that Spark due to the effective exploitation of main memory and the use of efficiency optimizing techniques was faster than Hadoop in every case.

Also as both frameworks were originally designed for scalability we conducted experiments in order to test the scalability of the two frameworks. In the experimental results we observed that the doubling of the active slave nodes leads almost to the halving of total execution time and the doubling of the input file leads almost to the doubling of total execution time for both frameworks. With these observations we made some cost estimations for the two frameworks and we investigated how we can achieve a significant cost reduction by accepting a burden in execution time.

Finally we took measurements related to the mean resource utilization of the cluster. From the various experiments we concluded that Spark had a higher mean utilization of the available resources compared to Hadoop. As it was expected Spark had higher memory utilization than Hadoop because saves intermediate results to memory. Also we found that Spark had a higher mean disk and network utilization which was quite unexpected. These unexpected experimental results can be explained by the fact that Spark processes the same amount of data but in significantly less time than Hadoop does and this leads to larger mean values of utilization related parameters. We also proposed a power consumption model and by using the utilization measurements we compared the power consumption of the two frameworks under specific experimental configuration.

The overall experiments showed Spark's best performance. However, the applications were implemented in such a way to make possible the comparison between the two frameworks. As future work could be implemented applications that make full use of Spark capabilities in order to evaluate the performance of the framework for more complex log analysis applications.

Acknwoledgements

The authors would like to thank Okeanos the GRNETs cloud service for the valuable resources.

References

Allalouf, M., Arbitman, Y., et al., 2009. Storage modeling for power estimation. In: Israeli Experimental Systems Conference (SYSTOR).

Anton, D., Chuvakin, A., Schmidt, K.J., 2013. Christopher Phillips, Partricia Moulder, Logging and Log Management: the authoritative guide to understanding the concepts surrounding logging and log management. Elsevier Inc. Waltham.

Assunoa, M., Calheirosb, R., Bianchic, S., Nettoc, M., Buyya, R., 2015. Big data computing and clouds: Trends and future directions. Journal of Parallel and Distributed Computing Volumes 79-80 3–15. Special Issue on Scalable Systems for Big Data Management and Analytics

Armbrust, M., Xin, .R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., Zaharia, M., 2015. Spark SQL: Relational data processing in spark. SIGMOD '15: ACM SIGMOD International Conference on Management of Data, Pages: 1383–1394, Melbourne, Australia.

Batista, B.G., Estrella, J.C., Santana, M.J., Santana, R.H.C., Reiff-Marganiec, S., 2014. Performance evaluation in a cloud with the provisioning of different resources configurations. 2014 IEEE World Congress on Services (SERVICES), Pages 309-316, Anchorage, Alaska.

Bohra, A., Chaudhary, V., 2010. VMeter: Power modelling for virtualized clouds. International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE,vol., no., pp. 1–8 19–23.

Carpen-Amarie, A., Orgerie, A., Morin, C., 2013. Experimental study on the energy consumption in iaas cloud environments. 6th International Conference on Utility and Cloud Computing (UCC), 2013 IEEE/ACM 9–12. Dresden, Germany

Chen, G., He, W., Lie, J., Nath, S., Rigas, L., Xiao, L., Zhao, F., 2008. Energy-aware server provisioning and load dispatching for connection- intensive internet services. In: NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. USENIX Association, pp. 337–350. Berkeley, CA, USA

Chen, Q., Grosso, P., van der Veldt, K., de Laat, C., Hofman, R., Bal, H., 2011a. Profiling energy consumption of VMs for green cloud computing, autonomic and secure computing (DASC). 2011 IEEE Ninth International Conference on Dependable 12–14. Sydney, Australia

Chen, Q., Grosso, P., Veldt, K.V.D., Laat, C.D., Hofman, R., Bal, H., 2011b. Profiling energy consumption of VMs for green cloud computing. In: 9th IEEE International Conference on Dependable Autonomic and Secure Computing (DASC 2011), pp. 768–775.

Chieu, T.C., Mohindra, A., Karve, A.A., 2011. Scalability and performance of web applications in a compute cloud. 2011 IEEE 8th International Conference on e-Business Engineering (ICEBE), Pages 317-323, Beijing, China 19–21.

Conejero, J., Caminero, B., Carron, C., 2014. Analysing hadoop performance in a multi-user iaas cloud. High Performance Computing and Simulation (HPCS), Bologna, Italy, Pages 399–406 21–25.

Dean, J., Ghemawat, S., 2004. Mapreduce: Simplified data processing on large clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA.

Fan, X., Weber, W., Barroso, L., 2007. Power provisioning for a warehouse-sized computer. In: Proc. of Int. Symposium on Computer architecture (ISCA), pp. 13–23.

Fatemaa, K., Emeakaroha, V.C., Healya, P.D., Morrisona, J.P., Lynnb, T., 2014. A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. Journal of Parallel and Distributed Computing 74 (10), 2918–2933.

Feller, E., Ramakrishnan, L., Morin, C., 2015. Performance and energy efficiency of big data applications in cloud environments: A hadoop case study. Journal of Parallel and Distributed Computing Special Issue on Scalable Systems for Big Data Management and Analytics 79-80, 80–89.

Gao, J., Pattabhiraman, P., Xiaoying, B., Tsai, W.T., 2011. Saas performance and scalability evaluation in clouds. 2011 IEEE 6th International Symposium on Service Oriented System Engineering (SOSE), Pages 61-71, Irvine, USA 12–14.

Ghosh, R., Naiky, V.K., Trivedi, K.S., 2011. Power-performance trade-offs in iaas cloud: A scalable analytic approach. 41st IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011) 152–157.

Gu, L., Li, H., 2013. Memory or time: Performance evaluation for iterative operation on hadoop and spark. 2013 IEEE 10th International Conference on High Performance Computing and Communications and 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC EUC),Zhangjiajie, China, Pages 721–727 13–15.

Gunther, N., Puglia, P., Tomasette, K., 2015. Hadoop superlinear scalability the perpetual motion of parallel performance. Communications of the ACM 13 (5), 46–55.

Hylick, A., Sohan, R., et al., 2008. An analysis of hard drive energy consumption. In: Proc. of the Int. Sym. on Modeling, Analysis and Simulation of Comp. and Telecomm. Systems (MASCOTS), pp. 1–10.

Jayathilake, D., 2012. Towards structured log analysis. 9th International Joint Conference on Computer Science and Software Engineering (JCSSE 2012), Bangkok, Thailand, 259–264.

Jiang, T., Zhang, Q., Hou, R., Chai, L., Mckee, S.A., Jia, Z., Sun, N., 2014. Understanding the behavior of in-memory computing workloads. 2014 IEEE International Symposium on Workload Characterization (IISWC), Pages 22-30, Raleigh, USA 26–28.

Jimeno, M., Christensen, K., Nordman, B., 2008. A network connection proxy to enable hosts to sleep and save energy. In: Proc. of Int. Performance, Computing and Comm. Conf. (IPCCC), pp. 101–110.

Joseph, R., 2001. Margaret martonosi, run-time power estimation in high performance microprocessors. In: ISLPED '01 Proceedings of the 2001 international symposium on Low power electronics and design, pp. 135–140. New York, NY, USA

Kansal, A., Zhao, F., Kothari, N., Bhattacharya, A.A., 2011. Virtual machine power metering and provisioning. In: 1st ACM Symposium on Cloud Computing (SoCC 2010), pp. 39–50.

Kathleen, H., Abdelmounaam, R., 2013. SAFAL: A mapreduce spatio-temporal analyzer for UNAVCO FTP logs. IEEE 16th International Conference on Computational Science and Engineering (CSE), Sydney, Australia,, Pages: 1083–1090 3–5.

Kotiyal, B., Kumar, A., Pant, B., Goudar, R., 2013. 'big data: Mining of log file through hadoop. IEEE International Conference on Human Computer Interactions (ICHCI'13), Chennai, India 1–7.

Koukis, E., Louridas, P., 2012. okeanos iaas. EGI Community Forum 2012 / EMI Second Technical Conference, Munich, Germany.

Koukis, V., Venetsanopoulos, C., Koziris, N., 2013. okeanos: Building a cloud, cluster by cluster. IEEE Internet Computing 17 (3), 67–71.

Kumar, M., Hanumanthappa, M., 2013. Scalable intrusion detection systems log analysis using cloud computing infrastructure. 2013 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Tamilnadu, India, Pages: 1–4. 26–8

Kumar, P., Bhardwaj, A., Doegar, A., 2015. Big data analysis techniques and challenges in cloud computing environment. International Journal of Advance Foundation And Research In Science & Engineering (IJAFRSE) Volume 1, Special Issue, ICCICT.

Lau, F., Rubin, S.H., Smith, M.H., Trajkovic, L., 2000. Distributed denial of service attacks. IEEE International Conference on Systems, Man, and Cybernetics, Nashville, TN, USA, 2275–2280.

Lee, J.Y., Kim, S.D., 2010. Software approaches to assuring high scalability in cloud computing. 2010 IEEE 7th International Conference on e-Business Engineering (ICEBE), Pages 300-306, Shanghai, China 10–12.

Lee, Y.C., Zomaya, A.Y., 2010. Energy efficient utilization of resources in cloud computing systems. Journal of Supercomputing Online First 1–13.

Leite, J.P., 2011. Analysis of log files as a security aid. 6th Iberian Conference on Information Systems and Technologies (CISTI), Lousada, Portugal,, Pages: 1–6 15–18.

LIN, X., WANG, P., WU, B., 2013. Log analysis in cloud computing environment with hadoop and spark. 5th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT 2013), Guilin, China, 273–276 17–19.

Liu, Y., Pan, W., Cao, N., 2010. G.i qiao, system anomaly detection in distributed systems through mapreduce-based log analysis. 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), Chengdu, China, Pages: V6–410-V6–413 20–22.

Massiea, M., Chunb, B., Cullera, D., 2004. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30 (7), 817–840.

Mavridis, I., Karatza, H., 2015. Log file analysis in cloud with apache hadoop and apache spark. 2nd International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015), Krakow, Poland,, Pages: 51–62 10–11.

Moschakis, I.A., Karatza, H.D., 2015. A meta-heuristic optimization approach to the scheduling of bag-of-tasks applications on heterogeneous clouds with multi--level arrivals and critical jobs. Simulation Modelling Practice and Theory, Elsevier 57, 1–25.

Narkhede, S., Baraskar, T., 2013. HMR log analyzer: Analyze web application logs over hadoop mapreduce. International Journal of UbiComp (IJU) 4 (3), 41–51.

Oliner, A., Ganapathi, A., Xu, W., 2012. Advances and challenges in log analysis. Communications of the ACM, ACM New York, NY, USA 55 (2), 55–61.

Orgerie, A., Lefevre, L., Gelas, J., 2010a. Demystifying energy consumption in grids and clouds. In: Green Computing Conference, 2010 International, pp. 15–18.

Orgerie, A.-C., Lefevre, L., Gelas, J.-P., 2010b. Demystifying energy consumption in grids and clouds. In: Proc. of Int. Green Computing Conf., pp. 335–342.

Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A., 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. Third ACM Symposium on Cloud Computing, New York, NY, USA.

Sanjay, G., Howard, G., Shun-Tak, L., 2003. The google file system. 19th ACM Symposium on Operating Systems Principles, Lake George, NY.

Stavrinides, G.L., Karatza, H.D., 2015. A cost-effective and qos-aware approach to scheduling real-time workflow applications in paas and saas clouds. In: Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud'15), Rome, Italy.

Steinder, M., Whalley, I., Hanson, J.E., Kephart, J.o., 2008. Coordinated management of power usage and runtime performance. In: IEEE Network Operations and Management Symposium (NOMS), pp. 387–394.

Trihinas, D., Pallis, G., Dikaiakos, M., 2014. JCatascopia: Monitoring elastically adaptive applications in the cloud, cloud and grid computing (CCGrid). 2014 14th IEEE/ACM International Symposium on Cluster, Chicago, Illinois, USA 26–29.

Vasconcelos, P.R.M., de Araujo Freitas, G.A., 2014. Performance analysis of hadoop mapreduce on an opennebula cloud with KVM and openVZ virtualizations. 2014 9th International Conference for Internet Technology and Secured Transactions (ICITST), Pages 471-476, London 8–10.

Velkoski, G., Simjanoska, M., Ristov, S., Gusev, M., 2013. CPU utilization in a multitenant cloud. IEEE EUROCON 2013,Zagreb, Croatia, Pages 242–249 1–4.

Vernekar, S., Buchade, A., 2013. Mapreduce based log file analysis for system threats and problem identification. Advance Computing Conference (IACC), 2013 IEEE 3rd International. Patiala, India, Pages: 831–835 22–23.

Wang, C., Tsai, C., Fan, C., Yuan, S., 2014. A hadoop based weblog analysis system. 7th International Conference on Ubi-Media Computing and Workshops (U-MEDIA 2014), Ulaanbaatar, Mongolia,, Pages: 72–77. 12–4

Wang, L., Tao, J., Ranjan, R., Marten, H., Streit, A., Chen, J., Chen, D., 2013. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. Future Generation Computer Systems 29 (3), 739–750.

Wei, J., Zhao, Y., Jiang, K., Xie, R., Jin, Y., 2011. Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis. International Conference on Cloud and Service Computing (CSC), Hong Kong, China,,Pages: 354–359 12–14.

Xin, R., Rosen, J., Zaharia, M., Franklin, M.J., Shenker, S., Stoica, I., 2013. Shark: SQL and rich analytics at scale. SIGMOD 2013, Pages 13-24, New York, USA 22–27.

Yang, J., Zhang, Y., Zhang, S., He, D., 2013. Mass flow logs analysis system based on hadoop. 5th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), Guilin, China,, Pages: 115–118 17–19.

Yu, H., Wang, D., 2012. Mass log data processing and mining based on hadoop and cloud computing. 7th International Conference on Computer Science and Education (ICCSE 2012), Pages: 197, Melbourne, Australia 14–17.

Yue, J., Zhu, Y., et al., 2007. Evaluating memory energy efficiency in parallel i/o workloads. In: IEEE Cluster, pp. 21–30.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mc-Cauley, M., Franklin, M.J., Shenker, S., Stoica, I., 2012. Resilient dis tributed datasets: A fault-tolerant abstraction for in-memory cluster computing. 9th USENIX conference on Networked Systems Design and Implementation, CA, USA,. 2–2

Zant, B.E., Gagnaire, M., 2015. Performance evaluation of cloud service providers. 2015 International Conference on Information and Communication Technology Research (ICTRC2015),Paris, France, Pages 302–305 17–19.