Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 11 (2017) 695 – 702

Procedia MANUFACTURING

27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy

BigBench workload executed by using Apache Flink

Sonia Bergamaschi, Luca Gagliardelli, Giovanni Simonini* and Song Zhu

Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy

Abstract

Many of the challenges that have to be faced in Industry 4.0 involve the management and analysis of huge amount of data (e.g. sensor data management and machine-fault prediction in industrial manufacturing, web-logs analysis in e-commerce). To handle the so-called Big Data management and analysis, a plethora of frameworks has been proposed in the last decade. Many of them are focusing on the parallel processing paradigm, such as *MapReduce*, *Apache Hive*, *Apache Flink*. However, in this jungle of frameworks, the performance evaluation of these technologies is not a trivial task, and strictly depends on the application requirements. The scope of this paper is to compare two of the most employed and promising frameworks to manage big data: *Apache Flink* and *Apache Hive*, which are general purpose distributed platforms under the umbrella of the Apache Software Foundation. To evaluate these two frameworks we use the benchmark *BigBench,* developed for *Apache Hive*. We re-implemented the most significant queries of *Apache Hive BigBench* to make them work on *Apache Flink*, in order to be able to compare the results of the same queries executed on both frameworks. Our results show that *Apache Flink*, if it is configured well, is able to outperform *Apache Hive*.

Peer-review under responsibility of the scientific committee of the 27th International Conference on Flexible Automation and Intelligent Manufacturing

Keywords: Big Data Management;Apache Flink; Apache Hive; BigBench; Benchmarking

* Corresponding author.
E-mail address: giovanni.simonini@unimore.it

Peer-review under responsibility of the scientific committee of the 27th International Conference on Flexible Automation and Intelligent Manufacturing
doi:10.1016/j.promfg.2017.07.169

1. Introduction

In the last few years the volume of the data generated by the industries has grown really fast, and the trend is to grow more and more [21], this because Industry 4.0 uses different smart machines that produce a large amount of data. For example, a Consumer Packaged Goods company that produces a personal care product, generates 5,000 data samples every 33 milliseconds, that result in 4 trillion samples per year [22]. Analysing in real time these large amount of data (Big Data) and extracting valuable information is strategic for the Industry. Referring to the previous example, the company needs to analyse its data to check if everything works fine very quickly.

With the rising volume of the data, the projects related to the so-called Big Data have proliferated, both in the academy and in the industry [1, 13, 12, 14]. To manage and analyze this huge amount of data, distributed and parallel computing is the most promising approach. In particular, one of the newest technologies pioneers, Google, designed a new paradigm of parallel processing to manage its huge volume of data, *Google File System* and *Map Reduce*. This new paradigm simplifies the development of distributed applications on commodity hardware, and allows a high scalability of these applications. Indeed, after the publication of the *Google File System* [15], the *Map Reduce* paradigm is widespread. Existing frameworks inspired by this paradigm [16, 17, 18] have their own characteristics, for instance: API for programming languages, data mining and machine learning algorithms support. In this scenario, it is hard to evaluate which is suit for a specific application, business case or company.

The goal of this work is to evaluate the performance of *Apache Flink* under different aspects of Big Data analytics. *Flink* is an open source framework for distributed Big Data analytics like *Hadoop* and *Hive*. However, the *Flink* core is a distributed streaming data-flow engine written in *Java* and *Scala*. The aims of *Flink* project is to bridge the gap between *Map-Reduce-like* systems and *share-nothing parallel database management systems*. The advantage of such system is to provide both batch and streaming processing in one single framework. This makes it the ideal choice to handle both the traditional need of batch processing for business intelligence (e.g. for datawarehouse), and the demand of real-time processing for the Industry 4.0 (e.g. for managing sensor data in real-time). To evaluate *Flink* we choose to employ *BigBench*, a business-oriented benchmark, ideal for the assessing of big data framework in an industrial context. *BigBench* is an Open Source Big Data benchmark developed by Oracle, TeraData, Intel in collaboration with Middleware Systems Research Group, University of Toronto, Canada. Since BigBench is written for Apache Hive, we adapted the benchmark to work over the Flink engine, in order to compare the two platforms.

The rest of the paper is structured as follows. In Section 2, an overview and related works about *Flink* and *BigBench* is presented. Section 3 is dedicated to our development of *BigBench* workload by using the *Flink* framework. The results of evaluation and comparison are presented in Section 4. Finally, we draw our conclusion about results in Section 5.

2. Background and related work

Apache Flink. *Flink* (formerly Stratosphere) is a distributed massively parallel system for data analytics. Its aim is to relief the programmers from the burden of explicitly programming software that has to run on distributed architectures. It achieves this by providing a high-level set of APIs to program user-defined functions (*UDFs*) that are automatically translated in a parallel distributed computation and executed on the *Flink* framework. Hence, we can see *Flink* as a general purpose distributed platform that aims to offer a complete and extensible solution for a data processing engine layer.

Basically, *Flink* allows to process the user-defined functions (UDFs) code through the system stack illustrated in Figure 1a. The *Flink* core provides data distribution, communication and fault tolerance for distributed computations over data streams. On the top of it, there are several APIs used to create applications. The *Flink* has a master-slave architecture, composed of the *Job Manager* and one or more *Task Manager*, as illustrated by Figure 1b. The *Job Manager* is the master node, which coordinates all the computations in the *Flink System*, while the *Task Managers* are workers, that actually execute the distributed programs. This architecture is completely transparent to the programmers, which only have to know the exposed API to write programs. Defining its own serializers *Flink* is able to provide a cheap data representation and to perform, through low-level

optimized algorithms, transformations directly on raw data, without needing to deserialize objects from their binary representation. In terms of performance, that means that the ability to compute common operations (e.g., sorting and hashing) can be very efficient, even on complex datatypes, significantly reducing overheads.

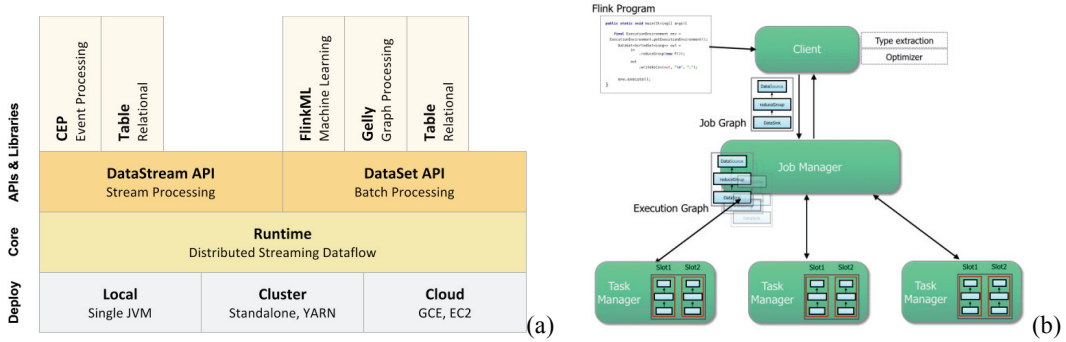


Fig. 1. (a) Flink component stack; (b) Flink execution lifecycle.

BigBench. The scope of this work is to evaluate *Flink* on as many as possible aspects of Big Data application, thus we chose the *BigBench* [19] as benchmark for this purpose. This benchmark has a data model that covers 3 categories of data type, structured, semi-structured and unstructured data; in addition, its workload is designed to cover various business cases and technical aspects.

Most of the fundamental aspects that characterize the *BigBench* data model, in particular as regards the structured relational tables, is adapted from the *TCP-DS* (*TPC Benchmark DS*: 'The' Benchmark Standard for decision support solutions including Big Data) [2]. In addition to structured relational data model from the *TCP-DS*, *BigBench* enriched the structured part with semi-structured and unstructured data. A simplified data model of *BigBench* is illustrated in Figure 2. The structured data depicts a product retailer; while semi-structured data are composed by clicks on retailer's web site; as well unstructured data are product reviews submitted by customers.

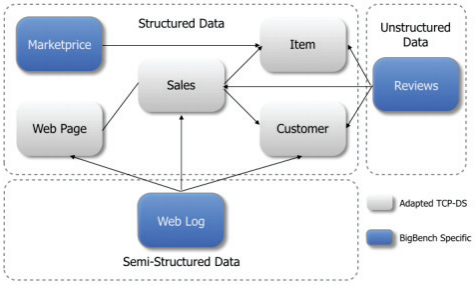


Fig. 2. Simplified BigBench data model

BigBench employs an extended version of *Parallel Data Generation Framework (PDGF)* [3] for the synthetic data generation. *PDGF* is a parallel data generator, which is able to generate a large volume of data for an arbitrary schema. While the standard *PDGF* can be employed to generate only structured data, *BigBench* developed an extended version of *PDGF* to generate semi-structured and unstructured data. *BigBench* can produce different volume of data by using scale factors.

In order to better highlight data volume changes according to tested scale factors, through which 1, 50, 100, 150 and 200 GB data have been respectively generated, in figure 4a shown size of table with scale factor 1 and the scaling method, and in the graphs of figure 4b report the size information about the main tables and their related data volume trends with different scale factors.

Query	Input Datatype	Processing Model	Query	Input Datatype	Processing Model
#1	Structured	Java MR	#16	Structured	Java MR (OpenNLP)
#2	Semi-Structured	Java MR	#17	Structured	HiveQL
#3	Semi-Structured	Python Streaming MR	#18	Unstructured	Java MR (OpenNLP)
#4	Semi-Structured	Python Streaming MR	#19	Structured	Java MR (OpenNLP)
#5	Semi-Structured	HiveQL	#20	Structured	Java MR (Mahout)
#6	Structured	HiveQL	#21	Structured	HiveQL
#7	Structured	HiveQL	#22	Structured	HiveQL
#8	Semi-Structured	HiveQL	#23	Structured	HiveQL
#9	Structured	HiveQL	#24	Structured	HiveQL
#10	Unstructured	Java MR (OpenNLP)	#25	Structured	Java MR (Mahout)
#11	Unstructured	HiveQL	#26	Structured	Java MR (Mahout)
#12	Semi-Structured	HiveQL	#27	Unstructured	Java MR (OpenNLP)
#13	Structured	HiveQL	#28	Unstructured	Java MR (Mahout)
#14	Structured	HiveQL	#29	Structured	Python Streaming MR
#15	Structured	Java MR (Mahout)	#30	Semi-Structured	Python Streaming MR

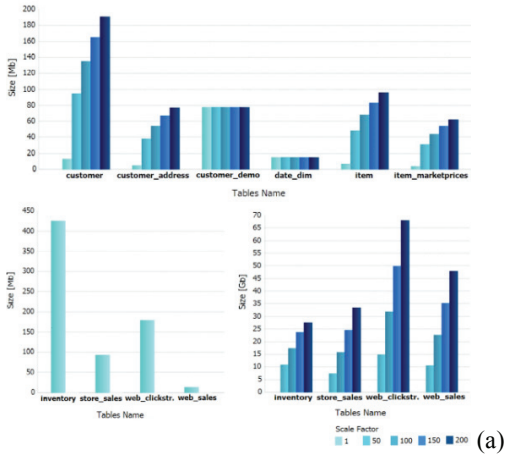
Fig. 3 BigBench queries

Workload. The main part of the *BigBench* workload is a set of queries to be executed against the data generated by using itself. These queries are defined in terms of 30 business questions. In Figure 3 is shown a table with the characteristics of the 30 queries. Ten of these are derived from *TPC-DS* workload against structured data. The remains 20 were designed based on business use case identified by the McKinsey report on Big Data use cases and opportunities [7]. Seven of these 20 queries run against the semi-structured data and 5 run over the unstructured portion of the data model.

From the queries implementation point of view:

- 14 are pure *HiveQL* queries;
- 4 are implemented by using *Python*;
- 2 are *Java-based MR jobs*;
- 5 exploit the *OpenNLP* libraries to implement sentiment analysis and named-entity recognition;
- 5 employ *Mahout* to perform machine-learning algorithms.

In the standard release of *BigBench* all queries use *Hive*. Our work consists of the development of queries with the *Flink* framework, and the comparison of the performance of two frameworks.



(a)

Table Name	# Rows SF 1	Bytes/Row	Scaling
date	109573	141	static
time	86400	75	static
ship_mode	20	60	static
household_demographics	7200	22	static
customer_demographics	1920800	40	static
customer	100000	138	square root
customer_address	50000	107	square root
store	12	261	square root
warehouse	5	107	logarithmic
promotion	300	132	logarithmic
web_page	60	134	logarithmic
item	18000	308	square root
item_marketprice	90000	43	square root
inventory	23490000	19	square root * logarithmic
store_sales	810000	143	linear
store_returns	40500	125	linear
web_sales	810000	207	linear
web_returns	40500	154	linear
web_clickstreams	6930000	27	linear
product_reviews	98100	670	linear

(b)

Fig. 4. (a) BigBench tables scaling (b) BigBench volume with different scale factors.

Related works. Evaluate Big Data frameworks is not an easy issue. There are emerging benchmarks to fullfill this goal. However, each benchmark has own characteristics and performs the evaluation on some specific aspects of Big

Data applications. An analysis on Big Data benchmarks is made by Geoffrey C. Fox et al [4], where a classification of Big Data benchmarks is presented.

Another work about *BigBench* similar to ours, but by using the *Spark engine*, was done by Todor Ivanov et al [5], where the authors implemented BigBench queries by using *Spark SQL* and compare the performance of *Hive over Hadoop engine* with the performance of workload implemented with *Spark SQL*. In this case, *Spark* outperforms *Hadoop* in many cases, hence *Spark* uses main memory abstraction while *Hadoop* uses Map Reduce on disk. The few cases where *Spark* performs poorly are due to the join operation, and they are resolved in the new Spark release. On other hand, Juwei Shi et al [6] have established that, in some cases Hadoop performs better than *Spark*. These works demonstrate that there are no absolute winners in the Big Data frameworks. Every framework can take advantage in a specific aspect of Big Data analytics. With this point of view, we approach the performance evaluation of *Flink*. There are few works on *Flink*, hence it is still a new technology with respect to *Spark*. A comparison between Flink and Spark is analyzed by Ovidiu-Cristian Marcu et al [7], where two frameworks are compared by using various algorithms such as: word Count; grep; Tera Sort; K-Means; Page Rank; Connected Components. Our approach is to use a standard benchmark, and implement the workload by using *Flink engine*.

3. Development

We have taken 13 of the 30 *BigBench* queries, translated them through the *Flink DataSet API* and then executed on the *Flink* engine. In particular, referring to figure 3 we have translated the queries number 1, 5, 6, 8, 11, 13, 15, 17, 20, 23, 24, 25, 29. We choose these queries because they cover a wide cross-functional spectrum: market basket analysis, machine learning algorithms and clustering processing.

Most of the queries, implemented through a MapReduce approach on unstructured text proposed within the Big Bench workload, involve *UDFs* used to extract sentiment from the plan text. The import of these built-in functions would not involve any further *engine-oriented* implementations. We report two significant conversion examples.

Note that, the complexity of our implementation of the *BigBench* queries remains the same of the original *Hive* implementations: the difference between the results depends on how the two frameworks manage the memory and the tasks. *Hive* is built on *Hadoop* that uses the classical *MapReduce* implementation [23], thus every transformation is performed in two phases: a *map* and a *reduce*. Between these two phases, the results are written to disk. *Flink* uses a different paradigm, that combines the *map* and *reduce* operations into a single *job* [17]. The intermediate results are kept in memory. As the memory access is much faster than the disk one, *Flink*'s implementation outperforms the *Hive* one.

Query 11: Pearson product-moment correlation coefficient. *BigBench* documentation reports that this query "For a given product, measure the correlation of sentiments, including the number of reviews and average review ratings, on product monthly revenues within a given time frame". In particular, given a fixed period of time the query computes the total number of reviews and the average review ratings for each sold item. Then, using the correlation function included in the *Hive Built-in Aggregate Functions* (*UDAF*), for each item computes the *Pearson* correlation [8] between the number of total reviews and the average rate, in order to discover any relationship between these variables. The *Flink Pearson* correlation coefficient *UDF* has been developed referencing to the implementation proposed in [9], the correlation formula is shown in (1), where $x = X - \overline{X}$, $y = Y - \overline{Y}$, X is the number of reviews per item and Y is the average ratings per item.

$$\rho = \frac{\sum x \cdot y}{\sqrt{\sum x^2 \cdot \sum y^2}} \quad (1)$$

The *Flink* implementation workflow is shown in Figure 5. One of the most challenging aspect of this implementation has been dealing with temporary aggregated values. Because *Flink* works use the *MapReduce* paradigm, so it is not possible to pass over data multiple times for further subsequent processing, since rows are continuously pushed to next operators, also on different machines, as soon as they have been processed. For this

reason, in order to perform a correct computation it has been necessary to join the rows with their respective mean vector and finally, after collecting the aggregated values required, extract the correlation coefficient through a *FlatMap* transformation on a temporary DataSet.

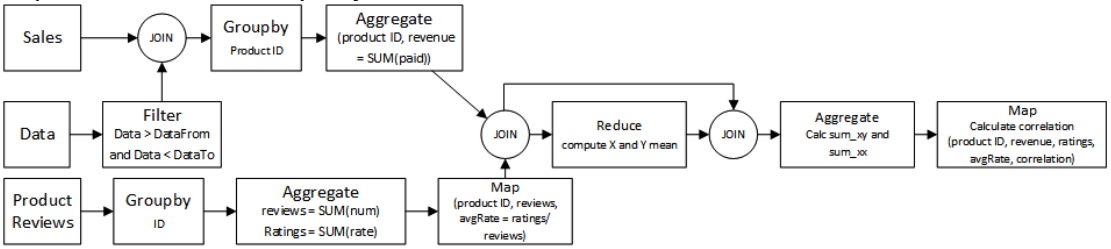


Fig. 5 Query 11, Flink workflow

Query 25: customers K-Means clustering. As reported in the *BigBench* documentation [19] this query performs a "Customer segmentation analysis: Customers are separated along the following key shopping dimensions: recency of last visit, frequency of visits and monetary amount. Use the store and online purchase data during a given year to compute. After model of separation is build, report for the analyzed customers to which "group" they were assigned.

The query generates a dataset for K-Means from two views that respectively report the customers and the purchases information. The feature vector for each customer includes:

- *Customer ID*;
- *Recency*: a boolean value that indicates if the customer has made purchases in the last 60 days;
- *Frequency*: total amount of purchases;
- *Total spend*: total amount spent.

To implement this query in *Flink* we used the Spark Machine Learning Library (*Spark MLlib*) [20], which provides a high-level access to a number of different machine learning algorithms, among which the K-Means. The main difference between *Hive* and *Flink* implementations is the source from which a given ML algorithm receives its input dataset. *Hive* indeed, saves the temporary result of the query execution directly in a resulting metastore on HDFS, instead *Flink* stores the temporary results in a CSV file.

Figure 6 shows the workflow of the *Flink* execution: first it joins the customer data with the purchases data, after it groups the data for the customer ID, then for each customer it calculates the feature vector using an UDF reduce function; the partial results are saved in an CSV file, finally the feature vectors are elaborated using MLlib's K-Means algorithm.

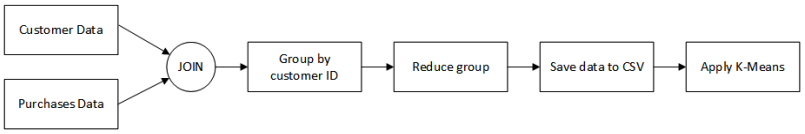


Fig. 6 Query 25, Flink workflow

The impact of the variation of the parameter k of the K-means algorithm is not a purpose of our work. Since our goals is to compare the performance of the two frameworks. We thus set the same parameter k value equal to 8 in both implementation.

4. Results analysis

This section aims to provide a running time analysis over the tested workload. The tests have been done on the IBM DIMA departmental cluster, in particular each machine of the cluster has 48 CPUs running at 3.7 GHz and 50 GB of RAM. The applications used in the tests are: *Apache Flink* 0.10.0, *Apache Hive* 1.2.1, *Apache Spark 1.6.0* with *spark-ml* library, Apache Hadoop 2.4.1.

It is important to remark that the results reflect the proposed implementation shown in the previous chapter, also the running time may be subject to changes from the *Flink* engine settings, in particular with regards to memory management parameters specification. For the queries showed in details in the previous chapter it is possible to see the performance to the growth of the Scale Factor in Figure 7.

More generally, Table 1 reports for each query that was implemented in *Flink* and the Scale Factor level, the time saved in percentage with respect to *Hive*.

As it is shown, our *Flink* query implementations outperforms the *Hive* one in each test, allowing the user to save a lot of time. These results can be explained observing the structure of the two systems, *Hive* is built on *Hadoop*, so for each operation of *MapReduce* it writes the result on a disk, and this is a slow operation. Instead, *Flink* works mainly in memory, and this thus, moreover it has advanced strategies[†] to optimize the parallel JOIN operations, which are the most used operations in the query processing task.

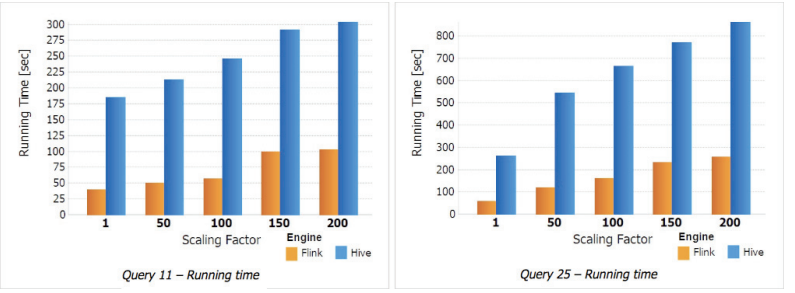


Fig. 7 Query 11 and 25 running time at the growt of the Scale FactorConclusion

Table 1. Time savings percentage *Flink* vs *Hive*.

Query	SF 1	SF 50	SF 100	SF 150	SF 200
1	88.3%	90.0%	88.8%	88.7%	88.0%
6	78.2%	91.1%	86.0%	85.2%	81.7%
11	78.4%	76.5%	76.8%	65.8%	66.1%
13	79.3%	86.9%	84.0%	78.4%	77.5%
15	87.6%	84.2%	77.6%	68.5%	69.4%
17	79.6%	90.9%	89.4%	90.1%	87.6%
20	83.6%	82.0%	81.1%	80.4%	76.8%
24	81.1%	80.2%	71.6%	73.2%	61.4%
25	77.5%	77.8%	75.7%	69.8%	70.0%
29	78.4%	89.8%	89.4%	87.3%	83.6%
Total	80.8%	86.1%	83.7%	81.4%	78.3%

5. Conclusions and Future works

In this work we provided an overview of the *Apache Flink* platform, its structure and its characteristics as well as a description of the *BigBench* benchmark, a benchmark developed for *Apache Hive*. We used *BigBench* to evaluate the performance of *Apache Flink* and to compare it to *Apache Hive*. In particular, we used the most relevant 13 of a total of 30 queries of the *BigBench* benchmark, able to cover a wide use cases spectrum. Finally, we have shown the performance of *Apache Flink* on the translated queries and how it outperforms *Apache Hive* as a significantly shorter execution time is used on these queries: *Flink* can achieve a time save in percentage around 80%.

[†] https://flink.apache.org/news/2015/03/13/peeking-into-Apache-Flinks-Engine-Room.html

In future, we are planning to translate all the *BigBench* queries into *Flink* to complete this work, and extend it for different kinds of workload and application [10, 11], such as the predictive analysis of sensor data for Industry 4.0. Moreover, an interesting work that is possible to do is to try to translate the *BigBench* queries in *Apache Spark* using *Spark SQL* with the *DataFrames*, which provides a native support to query structured data.

6. Acknowledgements

Part of this work is derived from the Master Thesis of Guido Mazza that was developed under the supervision of the Dr. Tilmann Rabl at the Technische Universität, the full work can be found on the MoreThesis website.

References

[1] S. Bergamaschi, E. Carlini, M. Ceci, B. Furletti, F. Giannotti, D. Malerba, M. Mezzanzanica, A. Monreale, G. Pasi, D. Pedreschi, R. Perego, S. Ruggieri, Engineering volume 2, pages 163-170, 2016

[2] Tpc-ds homepage. Available: http://www.tpc.org/tpcds

[3] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch. A Data, Generator for Cloud-Scale Benchmarking. In TPCTC, pages 41–56, 2010.

[4] Fox, GC Jha, S Qiu, J Ekanazake, S and Luckow, Andre, Towards a comprehensive set of big data benchmarks, Big Data and High Performance Computing, volume 26, 2015.

[5] Ivanov, Todor and Beer, Max-Georg, Evaluating Hive and Spark SQL with BigBench, arXiv preprint arXiv:1512.08417, 2015

[6] Shi, Juwei and Qiu, Yunjie and Minhas, Umar Farooq and Jiao, Limei and Wang, Chen and Reinwald, Berthold and Ozcan, Fatma, Clash of the titans: MapReduce vs. Spark for large scale data analytics, Proceedings of the VLDB Endowment, volume 8, pages 2110-2121, 2015

[7] Marcu, Ovidiu-Cristian and Costan, Alexandru and Antoniu, Gabriel and Perez, Maria S, Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks, in Cluster 2016-The IEEE 2016 International Conference on Cluster Computing, 2016

[8] D. M. Lane. Values of the pearson correlation. [Online]. Available: http://onlinestatbook.com/2/describing bivariate data/pearson.html

[9] Computing pearson's r. [Online]. Available: http://onlinestatbook.com/2/describing bivariate data/calculation.html

[10] G. Simonini, S. Bergamaschi, H. V. Jagadish. BLAST: a loosely schema-aware meta-blocking approach for entity resolu- tion - PVLDB 9.12 (2016): 1173-1184

[11] S. Bergamaschi, D. Ferrari, F. Guerra, G. Simonini, Y. Velegrakis - Providing Insight into Data Source Topics - Journal on Data Semantics (2016): 1-18

[12] G. Simonini, Z. Song. Big data exploration with faceted browsing - IEEE HPCS 2015

[13] G. Simonini, F. Guerra. Using big data to support automatic Word Sense Disambiguation - IEEE HPCS 2014.

[14] F. Guerra, G. Simonini , M. Vincini . Supporting Image Search with Tag Clouds: a Preliminary Approach - Advances in Multimedia, 2015.

[15] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." In ACM SIGOPS operating systems review, vol. 37, no. 5, pp. 29-43. ACM, 2003.

[16] Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. "Hive: a warehousing solution over a map-reduce framework." Proceedings of the VLDB Endowment 2, no. 2 (2009): 1626-1629.

[17] Carbone, Paris, Stephan Ewen, Seif Haridi, Asterios Katsifodimos, Volker Markl, and Kostas Tzoumas. "Apache flink: Stream and batch processing in a single engine." Data Engineering (2015): 28.

[18] Zaharia, Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pp. 2-2. USENIX Association, 2012.

[19] Ghazal, Ahmad, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. "BigBench: towards an industry standard benchmark for big data analytics." In Proceedings of the 2013 ACM SIGMOD international conference on Management of data, pp. 1197-1208. ACM, 2013.

[20] Meng, Xiangrui, et al. "Mllib: Machine learning in apache spark." Journal of Machine Learning Research 17.34 (2016): 1-7.

[21] Yin, Shen, and Okyay Kaynak. "Big data for modern industry: challenges and trends [point of view]." Proceedings of the IEEE 103.2 (2015): 143-146.

[22] General Electric Intelligent Platforms,"The rise of industrial big data", 2012, White Paper.

[23] Taylor, Ronald C. "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics." BMC bioinformatics 11.12 (2010): S1.