Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)

LeanBench: comparing software stacks for batch and query processing of IoT data

João M. P. Moreira, Helena Galhardas, Miguel L. Pardal*

INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa, Portugal

Abstract

Companies produce data in large volumes and in multiple varieties. This trend is intensifying with the deployment of Internet of Things (IoT) devices. Companies need to process data more efficiently and at the edge of the network if they are to remain capable of making timely business decisions based on data. Apache *Hadoop* and *Hive* are two widely used data processing systems. However, they rely on complex software stacks that cannot run in a typical IoT gateway device, i.e., a computer with low hardware specifications. An approach to solve this problem is to replace the software with a leaner system with the same functionality. This approach is the value proposition of *Unicage*, a data processing system based on Unix shell scripting, that promises better performance by using the operating system directly for execution and inter-process communication.

In this paper, we benchmark data processing systems with workloads that compare Unicage with Hadoop and with Hive. The results show that the complexity of the software stack is indeed a significant bottleneck in data processing.
© 2016 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.

Keywords: Internet of Things, Data Processing, Benchmarking, Software Stacks, Performance Measurement

1. Introduction

Data production continues to increase with the progression of technology. The Internet of Things (IoT)[1] will further increase the volume of data produced, as data will be gathered automatically by sensors, and transmitted machine-to-machine without human intervention.

Apache *Hadoop* and Apache *Hive* are two widely used systems for data processing. However, they have complex software stacks, with the Java Virtual Machine and several libraries on top of it. For example, the dependency tree of the "Hadoop MapReduce Core" source code artifact in version 3.1.0 shows more than 100 Java libraries being required, just for the core functionality. As a consequence, it is difficult to run these systems directly on smaller devices with low system specifications, like typical IoT gateways. One possible solution to this problem is to use

* Corresponding author. Tel.: +351.213100300; fax: +351.213145843.
E-mail address: miguel.pardal@tecnico.ulisboa.pt

1877-0509 © 2016 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.

alternative tools that provide the same functionality with a leaner software stack. One specific example of such a tool is the *Unicage* system[2] that is made of individual, single-purpose, shell commands. These commands are written in the C language, and are combined in shell script. The corresponding programs run on Unix/Linux as processes and use pipes to exchange data.

In this paper, we assess the performance of Unicage when compared with the Hadoop and Hive data processing systems. We called this effort *LeanBench* as we analyze the performance of a *lean* software stack with existing *bench*marks (presented in Section 3).

2. Background

This Section presents Hadoop, Hive, and Unicage.

Apache *Hadoop* is an open-source framework for the MapReduce programming model, designed for processing large datasets across clusters of computers. MapReduce[3] is a programming model designed to process large datasets. Programs written using MapReduce are automatically parallelized as `map` and `reduce` tasks can be assigned to multiple machines.

Apache *Hive*[4] is a data warehousing system built on top of Hadoop. Hive enables users to query large datasets that reside in a distributed storage form, such as the HDFS system[5], without relying on defining complex low level `map` and `reduce` functions. Hive provides *HiveQL*, a query language similar to *SQL*. Internally, the queries written in HiveQL are compiled and translated to MapReduce jobs, which are executed by Hadoop. Hive manages and organizes the stored data in: *tables*, similar to the tables in relational database systems that support filter, projection, join and union operations; *partitions* that specify the location of the data in the file system; and *buckets* stored as files in the partition.

Unicage is a set of shell commands which enable users to construct data processing systems[6]. The Unicage development approach consists on writing a script using a Unix/Linux shell script. The program uses common commands, such as `find` and `sort`, together with commands specific to the Unicage tool set. These commands include string formatting functions, system functions for file input/output, common operations used in relational queries (such as joins, selections), mathematical and statistical functions. Each individual command is designed to provide a single function. Each command is written directly in the C programming language in order to take advantage of low-level input/output and memory manipulation techniques[7].

3. Related Work

There are multiple benchmarking libraries and tools to evaluate the performance of data processing systems. We chose Big Data tools to be unconstrained by data sizes in our experiments.

The Apache Hadoop distribution includes tools designed to test and benchmark various aspects of the framework, such as the processing component (MapReduce) and the storage component (HDFS). These tools can be used for evaluating the execution of tasks in Apache Hive as well, since Hive mainly relies on the Hadoop MapReduce and HDFS systems for the processing and storage capabilities respectively.

BigBench[8] is a benchmark proposal for MapReduce systems. For evaluating the *volume* of data, BigBench proposes a data generator that can be configured accordingly to a scaling factor in order to produce various quantities of data; for *variety* of data, it is able to produce structured and unstructured data; for *velocity*, it is able to provide continuous input of data. The continuous input of data varies in volume, depending on the queries being performed and also on the defined scale factor.

BigDataBench[9] is an implementation of BigBench proposal, including unstructured, semi-structured and structured datasets. The different datasets are representative of real world applications, such as off-line analytics, real-time analytics and on-line services. BigDataBench uses real world datasets instead of randomly generated data. However, to allow scalability of dataset sizes, BigDataBench introduces the *Big Data Generator Suite* (BDGS) that is capable of producing very large sets of synthetic data based on real datasets.

The *AMP Big Data Benchmark*[10] is a standalone tool for evaluating the performance of data warehousing systems based on MapReduce systems like Apache Hive. The main metric used is the query execution time. The datasets are

both structured and unstructured. The AMP Benchmark workloads are based on a previous proposal [11], and include simple scans, aggregations, joins and other more complex user-defined functions.

Finally, *HiBench* [12] [13] is a benchmarking tool for MapReduce systems. HiBench includes many workload categories, composed of a wide set of Hadoop MapReduce jobs for many small tasks, as well as larger tasks based on real world applications. To evaluate job execution times, throughput and usage of system resources, HiBench mainly uses smaller benchmarking tasks, such as sorting and word counting. For larger workloads, it uses tasks that are normally used in the real world, such as PageRank [14].

The existing benchmarking tools and libraries presented here do not provide support for Unicage, and therefore could not be used directly. In the next Section we detail our proposal, *LeanBench*, and how it reuses existing tools.

4. LeanBench

The *LeanBench* benchmark is split into two different modules: *batch processing operations* such as Sort, Grep and Word count that enable the comparison between Unicage and Hadoop; *querying operations* such as Selection, Aggregation and Join that enable the comparison between Unicage and Hive. Table 1 summarizes the benchmark operations selected for each system.

Table 1 Benchmark operations for each system.

	Unicage	Apache Hadoop	Apache Hive
Sort	✓	✓	
Grep	✓	✓	
Word count	✓	✓	
Select Query	✓		✓
Aggregation Query	✓		✓
Join Query	✓		✓

4.1. Datasets

All of the datasets are provided by the *Big Data Generator Suite* (BDGS), from BigDataBench presented in Section 3. BDGS enables the generation of large amounts of data based on real datasets. For generating unstructured data for batch processing operations, the tool uses: *English Wikipedia* entries, and *Amazon movie review* entries. For generating structured data for querying operations, the tool uses an additional real dataset, which is based on anonymous *e-commerce transaction data*.

The unstructured dataset sizes are measured in Megabytes (MB) or Gigabytes (GB). The structured dataset sizes are measured as the total *number of rows* where each row takes up 85 Bytes in size.

4.2. Operation Implementation

To support the operations of LeanBench on the various systems, we have to implement several versions of the same operation in different programming languages for each system: *Shell Script* for Unicage, *Java* for Hadoop, and *HiveQL* for Hive. The batch processing operations were chosen from BigDataBench and HiBench, and include batch sorting and word counting. The database querying operations were chosen from the AMP Big Data Benchmark, BigDataBench and HiBench, and include Selection, Aggregation, and Join queries.

4.3. Metrics

LeanBench considers three main metrics: the first metric is the *Execution Time* of an operation. This metric is used to measure the performance of each operation in the different systems. The second metric is the *Data Processed per Second* (DPS), obtained by dividing the input dataset size by the total processing time, in seconds. The third metric is the *Usage of System Resources* that allows the comparison of how each system uses the available resources, such as *CPU* and *Memory* usage for each operation, and also allows identifying performance bottlenecks.

5. Evaluation

For the experiments, the versions of the data processing systems that we used were the following: Unicage `uspTukubai`, Hadoop `2.7.3` and Hive `2.1.1`. Unicage, Hadoop and Hive were deployed and configured in a single machine to allow the identification of CPU bound, memory bound and I/O bound bottlenecks. Two sets of experiments were performed. Each data point presents the average value of 10 samples, with the exception of the datasets larger than 2GB where only 2 or less samples were collected due to the large execution times.

The first set of experiments, served the purpose of identifying the processing limitations of a machine with low hardware specifications, designated as *IoT gateway device*. These experiments intend to represent tasks carried out on devices with limited capabilities that are scattered across many physical locations, near the edge of the network, where physical world sensors are located. They are positioned to filter data in a decentralized way [15].

A second set of experiments was performed on a different machine with higher hardware specifications, similar to a *commodity server*, allowing it to process datasets of larger volume. The commodity server experiments intend to study the performance of a single node to be used in a cluster setup that could be deployed in a cloud data center [16].

The results of the benchmark present the relative performance between two systems, calculated according to the formula presented in Equation 1. Variables s and f represent the slowest and fastest systems respectively, p represents the performance in percentage. For example, a value of $p = 10$ represents a 10% performance increase.

$$p = \left(\frac{\text{Avg. Operation Execution Time}(s)}{\text{Avg. Operation Execution Time}(f)} \times 100 \right) - 100 \tag{1}$$

5.1. Experiments with IoT gateway device

The first experiment set has been performed on a machine with low system specifications, described in Table 2.

Table 2 IoT gateway hardware used in the experiments.

Machine Description
OS: Linux 4.4.0-66-generic # 87-Ubuntu SMP x86_64 GNU/Linux
CPU: Intel Core i3-2350M @ 2.30GHz
RAM: 2048MB DDR3-1333

The initial tests consisted on performing batch and querying operations on smaller datasets. Figure 1 describes the

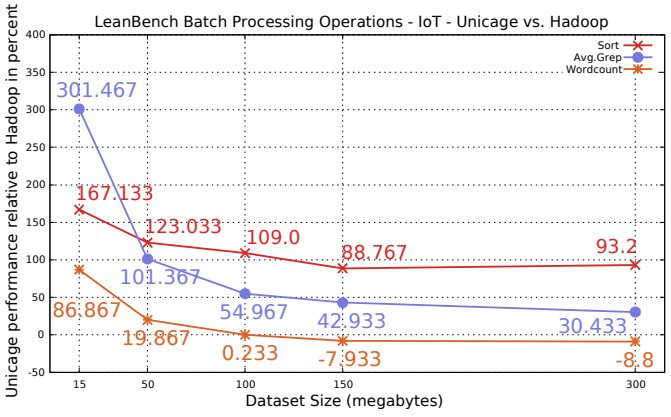

Fig. 1: Unicage vs. Hadoop in an IoT gateway device.

performance of Unicage relative to Hadoop in percentage for each batch processing operation. A value of 100% means that Unicage was twice as fast when compared to Hadoop. In the plots, higher values represent higher performance. For example, for the 50MB Dataset, the Average Grep operation performance in Unicage was 101.367% better in

relation to Hadoop. The tests performed show that Unicage performs better than Hadoop in all operations except Word count for datasets above 100MB, as shown in Figure 1. In addition, it is observed in all operations performed that the performance advantage of Unicage decreases as the volume of data to process increases.

Figure 2 shows the results for the querying tasks. Unicage performed better than Hive in all operations. Similarly to batch processing operations, a decrease in performance is observed as the volume of data to process is increased, however, it is more significant in comparison to batch processing drop.

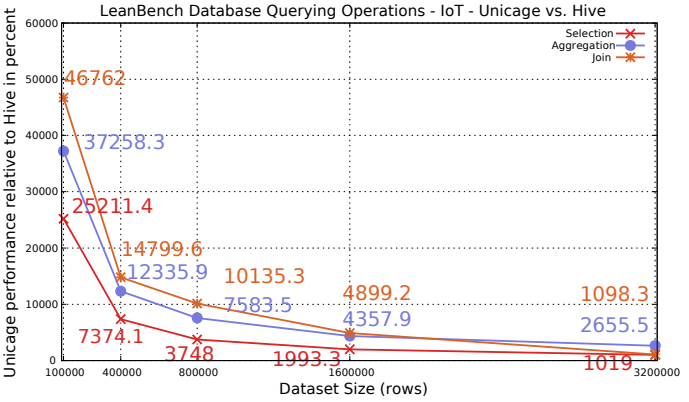

Fig. 2: Unicage vs. Hive in an IoT gateway device.

Figure 3 presents the percentual performance of Unicage relative to Hadoop for batch processing operations of LeanBench for larger datasets. The results confirm that Unicage is faster than Hadoop for the Sort and Grep operations but there is a decrease in performance as the volume of data to process increases. For the Word count operation, the Unicage performance is lower and also continues to decline when compared to Hadoop.

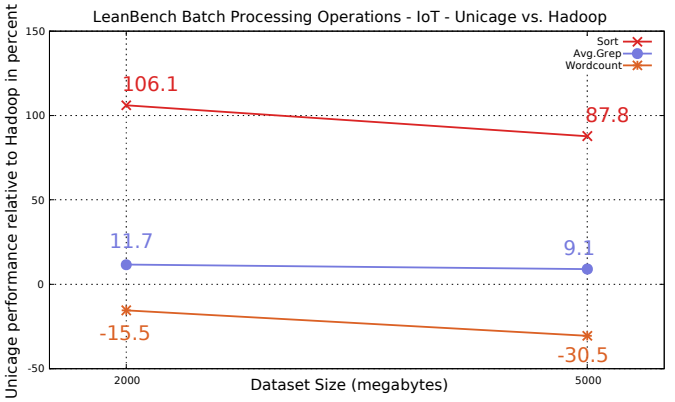

Fig. 3: Unicage vs. Hadoop in an IoT gateway device with larger datasets.

Figure 4 shows the percentual performance of Unicage relative to Hive for querying operations. Unicage is faster than Hive in all of the benchmark queries, confirming the initial results obtained previously with smaller datasets.

5.2. Experiments with Commodity Server

After reaching the limitations of the machine used in the first experiment set, we used a second machine, with higher hardware specifications, summarized in Table 3. Figure 5 describes the percentual performance of Unicage relative to Hadoop. Unicage remains faster than Hadoop for the Sort and Grep operations, using the 5, 10 and 20 GB datasets, but there is a significant drop in performance as the dataset sizes increase, similar to previous results.

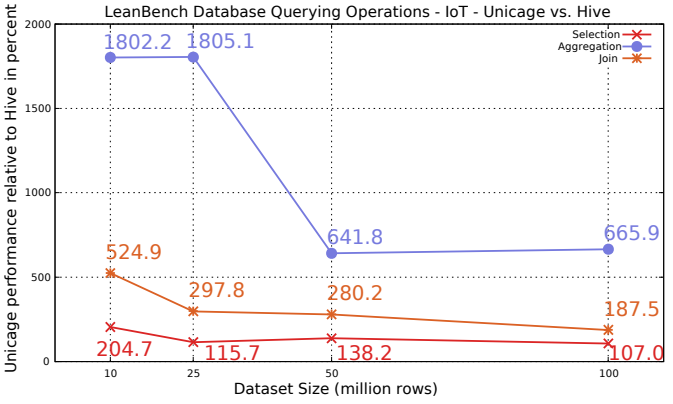

Fig. 4: Unicage vs. Hive in an IoT gateway device with larger datasets.

Table 3 Commodity server hardware used in the experiments.

Machine Description
OS: Linux 3.13.0-117-generic #164-Ubuntu SMP x86_64 GNU/Linux
CPU: Intel Xeon E5506 @ 2.13GHz (Using two cores)
RAM: 10GB DDR3-1066

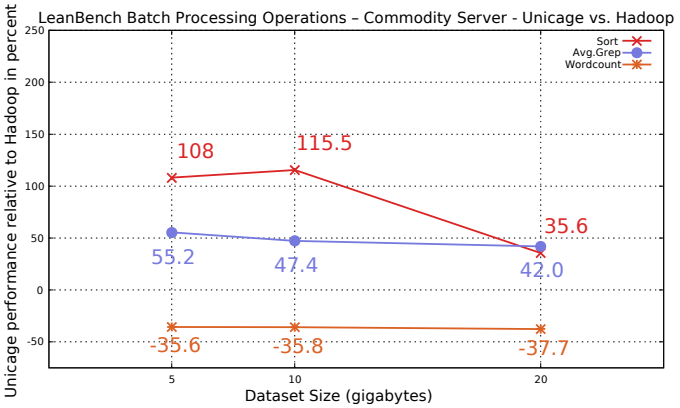

Fig. 5: Unicage vs. Hadoop in a Commodity Server.

Unicage remains slower than Hadoop for the Word count. Tests have been performed using the 40 and 60 GB datasets for Grep and Word count operations. Table 4 presents the average operation execution times obtained in these tests.

Table 4 Average execution times for Grep and Word count.

System	40 GB Dataset	60 GB Dataset
Unicage Avg. Grep	4458.262 seconds	*(failed execution)*
Hadoop Avg. Grep	6359.157 seconds	10669.153 seconds
Unicage Word count	23284.28 seconds	*(failed execution)*
Hadoop Word count	15840 seconds	25108.92 seconds

Unicage terminated execution in the middle of processing the 60GB dataset, failing to produce a valid output. While some of the Grep operations performed are not the most complex to match, the expressions match a large

portion of the entries in the Dataset, resulting in a large volume of entries matched to count, which may explain why Unicage failed to process the data, similar to the Word count operation, which counts the entries of the entire dataset.

Figure 6 shows the variation of the percentual performance of Unicage relative to Hive. When it comes to the querying operations of LeanBench, Unicage produced faster results than Hive when processing datasets of 200 and 400 million rows in all the benchmark queries.

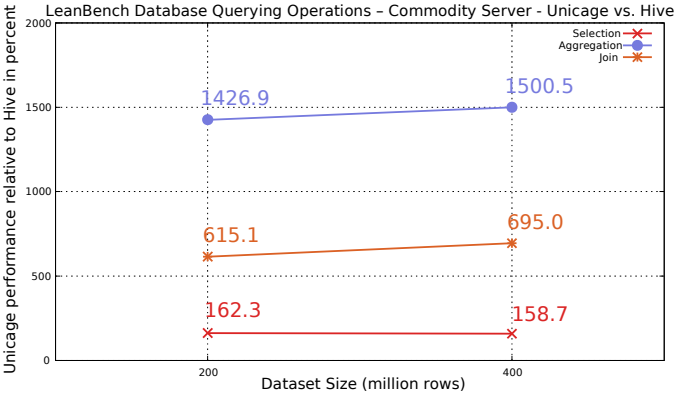

Fig. 6: Unicage vs. Hive in a Commodity Server.

5.3. Discussion

The results obtained when executing the *batch processing* operations show that Unicage performed better than Hadoop in all operations for datasets sized below 100MB. However, Unicage performs worse than Hadoop when processing the Word count operation with datasets larger than 100MB. While Unicage remains faster than Hadoop for the remaining batch processing operations, the performance declines as the dataset size increases, indicating that there is a point where Unicage starts performing worse than Hadoop. In particular, this fact was observed in the second set of experiments performed, using a commodity server, where Unicage failed to produce a valid output when performing Grep and Word count operations on a 60GB dataset.

For *data querying* benchmark operations, Unicage performed better than Hive. The performance of Unicage, when compared to Hive, also declined as the dataset sizes increased, but there are some exceptions. In the second set of experiments, a performance increase was observed in Unicage when performing the Aggregation and Joining queries for the 400 million row dataset in comparison to the 200 million row dataset.

6. Conclusion

In this paper we benchmarked the Unicage, Hadoop, and Hive data processing systems with diverse operations. The experiments allowed a comparison between Unicage and Hadoop for batch processing tasks, and between Unicage and Hive for database querying tasks.

For *batch processing*, Hadoop is the best choice if the processing task requires sorting volumes of data that are larger than 100MB when using an IoT gateway device, or 400MB when using a commodity server with larger amounts of system memory. These are small thresholds but if the processing task does not involve sorting large volumes of data then Unicage is a faster alternative to Hadoop.

When it comes to *database querying* tasks, Unicage has an advantage over Hive, which can be explained by its reduced software stack. While Hive relies on the entire Hadoop stack for performing the processing, Unicage exclusively performs the individual processing commands specified in the querying script of the task. However, the querying language of Hive does present an advantage to Unicage as users can directly write queries in *HiveQL*, a process very similar to writing *SQL* queries. In Unicage, it is not possible to write queries directly in a declarative language, and queries must instead be written using individual low-level processing commands.

The best processing system to use depends largely on the data processing task at hand. Figure 7 presents a flowchart with the best system choice criteria for some common processing tasks and requirements.

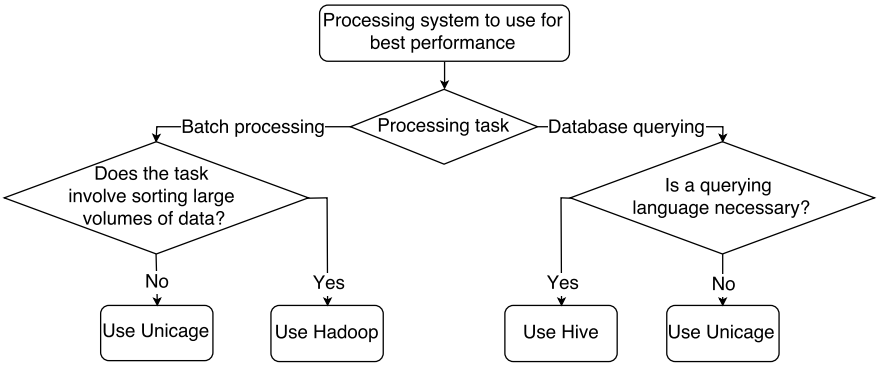

Fig. 7: Best processing system according to different processing tasks and specific requirements.

Unicage is especially suited to low complexity tasks, like data filtering. Fortunately, this type of task is very common in IoT applications [1]. The additional software layers of Hadoop and Hive provide ease of use for developers by supporting storage and computation distribution that make them suitable for more complex processing tasks. These tools are also prepared for cluster configurations, that will allow the use of much larger datasets, if required.

For future work, the results could be further improved by using datasets originated from actual IoT deployments. This will also allow a more detailed analysis to the data processing architecture and node placement.

Acknowledgements This work was supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID)

The authors also thank USP Lab – Japan and Europe – for providing Unicage licenses and technical support.

References

1. Uckelmann, D., Harrison, M., Michahelles, F.. *Architecting the Internet of Things*. Springer Publishing Company, Incorporated; 1st ed.; 2011. ISBN 3642191568, 9783642191565.
2. USP Labs, . Universal shell programming. http://en.usp-lab.com/; 2016.
3. Dean, J., Ghemawat, S.. Mapreduce: simplified data processing on large clusters. In: *OSDI 04: proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation*. USENIX Association; 2004, .
4. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., et al. Hive: A warehousing solution over a map-reduce framework. *Proc VLDB Endow* 2009;**2**(2):1626–1629. doi:10.14778/1687553.1687609. URL `http://dx.doi.org/10.14778/1687553.1687609`.
5. Borthakur, D.. Hdfs architecture guide. Tech. Rep.; 2008.
6. USP Labs, . How to Analyze 50 Billion Records in Less than a Second without Hadoop or Big Iron. Tech. Rep.; 2013. MIT-CRIBB, Universal Shell Programming Laboratory.
7. USP Labs, . Unicage FAQ. 2016. Universal Shell Programming Laboratory.
8. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., et al. Bigbench: towards an industry standard benchmark for big data analytics. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. ACM; 2013, p. 1197–1208.
9. Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., et al. Bigdatabench: A big data benchmark suite from internet services. In: *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE; 2014, p. 488–499.
10. University of California at Berkeley, . Big data benchmark. https://amplab.cs.berkeley.edu/benchmark/; 2017.
11. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., et al. A comparison of approaches to large-scale data analysis. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM; 2009, p. 165–178.
12. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: *New Frontiers in Information and Software as Services*. Springer; 2011, p. 209–228.
13. Huang, S., Huang, J., Liu, Y., Yi, L., Dai, J.. Hibench: A representative and comprehensive hadoop benchmark suite. In: *Proc. IEEE International Conference on Data Engineering (ICDE)*. 2012, .
14. Brin, S., Page, L.. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks* 2012;**56**(18):3825–3833.
15. Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., et al. Edge-centric computing: Vision and challenges. *SIGCOMM Comput Commun Rev* 2015;**45**(5):37–42.
16. Buyya, R., Vecchiola, C., Selvi, S.T.. *Mastering cloud computing: foundations and applications programming*. Newnes; 2013.