# Performance Evaluation of Big Data Analysis

Roberto R. Expósito, Jorge Veiga, and Juan Touriño
Computer Architecture Group, Universidade da Coruña, CITIC, A Coruña, Spain

## Synonyms

Big data performance characterization

## Definitions

Big Data analysis can be defined as the strategy of collecting, storing, processing, and analyzing very large and complex datasets to extract patterns and other useful information. Currently, there exist multiple Big Data systems specifically oriented to different fields such as machine learning, data mining, or graph processing.

At the core of any Big Data system, the data processing engine is usually in charge of ultimately processing the input dataset, and thus its performance is key for scalable Big Data analysis. To provide horizontal scalability, most data processing frameworks generally provide a distributed mode so that they can be deployed over the nodes of a cluster. As each cluster has a certain set of features (e.g., CPU microarchitecture, available memory, networking technology), the performance of these frameworks is affected not only by the software component but also by the available hardware resources.

Most performance evaluation studies are oriented to compare several distributed processing frameworks with different configuration parameters. To do so, a set of experiments is carried out, which involves generating the input datasets, processing them using representative workloads, and extracting the corresponding performance metrics. The use of benchmarking tools to perform all these tasks is widespread.

This chapter first presents an overview of the most popular distributed processing frameworks. Then, a summary of previous studies that have assessed the performance of these frameworks is provided together with the benchmarking tools that are most commonly used to perform this task.

## Overview

According to the literature, it has been established that distributed frameworks can be classified into three main groups according to the specific capabilities of their underlying data processing engine and targeted use cases: (1) batch-only, (2) stream-only, and (3) hybrid.

Batch-only frameworks enable to store and process static, very large datasets in a scalable and easy-to-program way. According to the MapReduce paradigm originally proposed by Google (Dean and Ghemawat 2008), input data are split and processed in chunks by using the two

© Springer Nature Switzerland AG 2022
S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_143-2

phases that name the model, which are derived from functional programming. Apache Hadoop (2008), as an open-source implementation of the MapReduce model, has long been one of the most popular frameworks for large-scale batch processing over the last decade. Basically, Hadoop consists of three components or layers: (1) the Hadoop MapReduce as data processing engine; (2) the Hadoop Distributed File System (HDFS) (Shvachko et al. 2010) as data storage layer; and (3) the Yet Another Resource Negotiator (YARN) (Vavilapalli et al. 2013) as resource management layer. Furthermore, the vast ecosystem around Hadoop has become the most commonly used platform to solve Big Data problems. This ecosystem consists of multiple open-source projects such as Apache Mahout for machine learning, Apache Giraph for graph processing, Apache Hive for SQL-like query processing on HDFS, or Apache HBase as a non-relational, columnar database implemented on top of HDFS, among many other projects.

It is also worth mentioning some modifications of Hadoop that adapt it to specific networking technologies such as RDMA-Hadoop (Wasi-Ur-Rahman et al. 2013) or that seek overall performance improvements like Flame-MR (Veiga et al. 2016b). On the one hand, RDMA-Hadoop adapts several Hadoop subsystems to take advantage of Remote Direct Memory Access (RDMA) networks in order to achieve better communication efficiency. On the other hand, Flame-MR presents a novel design that includes several optimizations such as an event-driven architecture, pipelined data movements, and efficient iterative support.

The second group of distributed processing frameworks, stream-only, was developed to allow building pipelines that process data which arrives continuously and from different sources, even in real time. In this scenario, batch processing is not applicable due to time constraints, the possibility of having an unending stream, and the lack of support for real-time processing. Popular stream processing projects are Apache Storm (Iqbal and Soomro 2015), Apache Heron (Kulkarni et al. 2015), Apache Samza (Noghabi et al. 2017), and the Kafka Streams library (Isah et al. 2019). All of them follow a different approach than the MapReduce model, generally creating a graph-based architecture using pipelines and direct acyclic graphs. Data management in stream frameworks is also different from the batch-only approach, which mainly uses HDFS. The streaming paradigm introduces the idea of data sources and sinks. A source is defined as the origin of the data into the streaming architecture, whereas the sink is the end where output data is persisted. A data ingestion layer at the front end is responsible for accepting streams of data into the system. This layer ensures scalable and fault-tolerant data distribution across the cluster from multiple input data streams, thus decoupling the data sources from other parts of the system. To play this role for data ingestion, message-oriented queuing systems such as Apache Kafka (Kreps et al. 2011) are usually deployed together with the stream processing framework.

Finally, hybrid frameworks try to offer a unified solution for data processing by covering both batch and streaming scenarios. These solutions inherit some of the functionalities offered by batch models like MapReduce, as well as the new features from streaming architectures. Currently, Apache Spark (Zaharia et al. 2016) and Apache Flink (Carbone et al. 2015) are the most popular open-source frameworks under this category. On the one hand, Spark provides a batch processing engine based on a novel data structure, Resilient Distributed Datasets (RDDs), which are in-memory data collections partitioned over the cluster nodes. As RDDs keep data in memory, Spark can avoid disk traffic and alleviate some of the issues that hinder Hadoop performance, especially for iterative workloads. Initially, Spark supported stream processing by providing the Streaming API, which was based on a micro-batch processing model: streams are split into finite chunks of data (i.e., batches) that are then processed in parallel in batch mode. More recently, Spark 2.2 introduced the Structured Streaming API to support real-time processing instead of using micro-batches (Armbrust et al. 2018). Structured Streaming is built on top of the Spark SQL engine so that it manages streaming data as of a relational table where data is continuously appended to it. Furthermore, Spark includes built-in libraries for machine learning (MLlib) and graph algorithms (GraphX). On the other hand, Flink specifically targets streaming scenarios. The basic building blocks of Flink programs are streams and transformations, defined as data sources and data operators, respectively. Unlike Spark Streaming, Flink provides a native stream engine that allows handling incoming data on an item-by-item basis as a true stream. Batch processing is also supported by simply considering batches to be data streams with finite boundaries. Similar to Spark, Flink also includes built-in libraries for machine learning (FlinkML) and graph algorithms (Gelly).

In terms of the physical architecture, most current frameworks can be deployed in a cluster of commodity machines, usually following a master/slave architecture. Regarding resource management, YARN and Mesos (Hindman et al. 2011) are widely supported as cluster managers, but many of the frameworks can also run in stand-alone mode or even provide support for container-based platforms such as Kubernetes.

As a summary, Table 1 shows the main characteristics of the most relevant frameworks discussed previously.

## Key Research Findings

Evaluating Big Data systems has been an active area of research over the last decade. Most studies have assessed and characterized the performance of distributed processing frameworks at multiple levels in an effort to provide useful guidelines for identifying the most appropriate one for a specific use case. This section provides a summary of the main research results.

### Data Processing Engine

The underlying data processing engine defines the type of data operators that can be performed over the input datasets or streams of data, and so its performance is a crucial factor that determines the overall throughput. As mentioned before, MapReduce has been one of the most popular batch engines so far, with Hadoop as its de facto standard implementation. Consequently, many early works have thoroughly assessed Hadoop performance using representative workloads (Fadika et al. 2011; Dede et al. 2014; Jakovits and Srirama 2014).

However, the underlying Hadoop architecture presents various limitations that hinder its overall performance, such as the writing of intermediate results to HDFS. Thus, it has been acknowledged that Hadoop cannot be the one-size-fits-all solution, which has led to the development of multiple alternatives that try to optimize its performance. So, later studies focus on in-memory data processing engines such as Spark due to their better flexibility and performance. For instance, Shi et al. (2015) compared Hadoop with Spark, showing that the latter can reduce the execution time up to 80% for batch iterative workloads. Further evaluations are conducted by Veiga et al. (2016a), comparing Hadoop with Spark and Flink for batch processing. Their results have shown that Spark and Flink can reduce Hadoop runtimes by 77% and 70% on average, respectively.

When comparing Spark with Flink, most works conclude that their performance is highly dependent on the workload executed. Spangenberg et al. (2015) have compared both frameworks using standard batch workloads (e.g., WordCount, KMeans, PageRank), showing that Flink outperforms Spark except in one of them. Another study by Bertoni et al. (2015) using three genomic applications has shown that Flink beats Spark in two of them. Some internal design characteristics of both frameworks are addressed by Marcu et al. (2016), identifying a set of configuration parameters that have a major influence on their execution time and scalability.

Other works have focused on the data processing capabilities for streaming scenarios. Some studies (Chintapalli et al. 2016) have shown that Spark Streaming outperforms Flink and Storm in terms of peak throughput at the cost of providing higher latencies. Samosir et al. (2016) have assessed Spark Streaming, Storm, and Samza using both quantitative and qualitative tests, showing that Samza is the worst framework under evaluation. The study conducted by Karimov et al.

**Performance Evaluation of Big Data Analysis, Table 1** Overview of distributed processing frameworks and libraries

| | Paradigm | Cluster managers | Data sources and sinks | Iterative support | Real time |
|---|---|---|---|---|---|
| Hadoop | Batch | YARN, Mesos | Distributed filesystems (e.g., HDFS), object storage (e.g., S3) | No | No |
| RDMA-Hadoop | Batch | YARN | Distributed filesystems (e.g., HDFS), parallel filesystems (e.g., Lustre), object storage (e.g., S3) | No | No |
| Flame-MR | Batch | YARN | Distributed filesystems (e.g., HDFS), object storage (e.g., S3) | Yes | No |
| Storm | Stream | YARN, Mesos, Docker, Kubernetes, stand-alone | Distributed filesystems (e.g., HDFS), databases (e.g., Cassandra), message systems (e.g., Kafka) | Yes | Yes |
| Heron | Stream | YARN, Mesos, Aurora, Kubernetes, stand-alone | Distributed filesystems (e.g., HDFS), databases (e.g., Cassandra), object storage (e.g., S3), message systems (e.g., Kafka) | Yes | Yes |
| Samza | Stream | YARN, stand-alone | Distributed filesystems (e.g., HDFS), message systems (e.g., Kafka) | Yes | Yes |
| Kafka Streams | Stream | YARN, stand-alone | Distributed filesystems (e.g., HDFS), databases (e.g., Cassandra), object storage (e.g., S3), message systems (e.g., Kafka) | Yes | Yes |
| Spark | Hybrid | YARN, Mesos, Kubernetes, stand-alone | Distributed filesystems (e.g., HDFS), databases (e.g., Cassandra), object storage (e.g., S3), message systems (e.g., Kafka) | Yes | Yes |
| Flink | Hybrid | YARN, Mesos, Docker, Kubernetes, stand-alone | Distributed filesystems (e.g., HDFS), databases (e.g., Cassandra), object storage (e.g., S3), message systems (e.g., Kafka) | Yes | Yes |

(2018) confirmed that the general performance for windowed operations of Spark Streaming and Flink is better than that of Storm. A more recent work (van Dongen and Van den Poel 2020) has assessed the performance of the newest Spark Structured Streaming API, comparing it with its predecessor (Spark Streaming), Flink and the Kafka Streams library. By running four workloads that cover different processing scenarios, the results have shown that when sub-second latency is critical, Flink would be the best choice. Otherwise, Structured Streaming offers a high-level, SQL-like API that provides high throughput with minimal tuning.

## File System

Most Big Data processing frameworks supports HDFS in order to distribute the storage of large datasets over the nodes of a cluster, collocating storage and compute services on the same nodes. However, HDFS is not widespread in High Performance Computing (HPC) clusters, which usually separate compute and storage services by relying on parallel file systems like Lustre, OrangeFS, or GPFS. Some works have evaluated the performance of both approaches, concluding that GPFS behaves better than HDFS at low concurrency scenarios, while HDFS is more suited to high concurrency one (Fadika et al. 2012). Some other works have demonstrated that parallel file systems can provide performance improvements over HDFS. For instance, Fadika et al. (2014) propose MARIANE, a custom MapReduce framework implemented on top of GPFS. Another approach presented by Xuan et al. (2017) relies on a two-level storage system that integrates OrangeFS with Tachyon, an in-memory file system, in order to obtain higher performance than just using HDFS.

### Disk Technology

Traditional magnetic disk drives are being progressively replaced by solid-state drives (SSDs), which obtain significantly better performance but at higher cost per byte. Using SSDs has been reported to significantly improve the performance of Hadoop by reducing disk bottlenecks when executing I/O-bound workloads (Hong et al. 2016; Bakratsas et al. 2018). Regarding Spark, its performance can be improved by 23% when using SSDs to store intermediate results compared to a memory-only approach as shown by Choi et al. (2015).

### Network Interconnects

The network plays a key role in the performance of those Big Data workloads where data shuffling is the most demanding phase. In streaming scenarios, the network is also of utmost importance to scale applications to a large number of nodes with a reasonable latency.

Most frameworks implement the network support on top of a socket-based interface (e.g., Hadoop, Spark), which can limit their potential performance especially on HPC systems where specialized networking technologies (e.g., InfiniBand, Intel Omni-Path) are widespread. Consequently, some early works (Islam et al. 2012; Wasi-Ur-Rahman et al. 2013) have been focused on adapting Hadoop components (e.g., HDFS) to take full advantage of such networks. As an example, a performance improvement of 32% has been achieved when running the TeraSort workload with Hadoop on an InfiniBand HPC cluster using eight nodes.

Later studies have also analyzed the impact of implementing this advanced networking support in other frameworks such as Spark (Lu et al. 2016b) or Heron (Kamburugamuve et al. 2017). For instance, a performance improvement up to 46% for batch workloads has been measured in the case of Spark.

### Memory Management

As Big Data applications process large amounts of data, managing the available memory resources efficiently is another important factor to take into account to achieve scalable performance. Most current Big Data frameworks are written in some object-oriented, managed language (e.g., Java, Scala) executed by the Java Virtual Machine (JVM). In this context, objects are automatically tracked by the JVM in order to release unused memory once they stop being referenced. This process, performed by the JVM garbage collector, can cause significant performance overheads when processing large datasets.

Modifying the JVM memory management to adapt it to the characteristics of Big Data systems can lead to significant performance improvements. For instance, Broom (Gog et al. 2015) has proposed a region-based algorithm to allocate data objects efficiently. Another example is Yak (Nguyen et al. 2016), which implements a hybrid approach that utilizes generation- and region-based algorithms for control and data objects, respectively. As these memory managers are implemented in Java, they can be used with any JVM-based framework.

Other solutions are specific to a certain framework. For example, Deca (Lu et al. 2016a) modifies the management of data containers in Spark to estimate their lifetime and allocating memory regions accordingly. This mechanism provides a maximum speedup of 41.6 in scenarios with heavy disk spilling.

### Manycore Accelerators

The great majority of Big Data frameworks rely on CPUs to perform the computations. Some other works propose the use of specialized manycore accelerators typically available in heterogeneous systems, like GPUs or FPGAs.

Due to the high degree of data parallelism provided by GPUs, they have become a suitable option to accelerate Big Data workloads. For instance, Mars (Fang et al. 2011) accelerates MapReduce workloads using CPU, GPU, or hybrid computations, providing a maximum speedup of 2.8. The exploitation of GPUs has also been employed to improve the performance of Spark (Rathore et al. 2018) and Flink (Chen et al. 2018). For instance, the performance of Flink when running KMeans is improved by 5× on a 10-node GPU cluster.

FPGAs are hardware devices that can be programmed to build custom accelerators. Neshatpour et al. (2015) have assessed the benefits of accelerating typical machine learning and data mining applications by offloading some of their kernels to FPGAs, obtaining a maximum speedup of 2.72. More recently, Hou et al. (2018) have proposed an FPGA-based Spark implementation that provides a modest speedup of 1.79 over the CPU-only counterpart.

### CPU Microarchitecture

The specific CPU microarchitecture of the cluster nodes has been the focus of some previous works. For instance, the performance and energy efficiency of "big" nodes based on high-performance server architectures (e.g., Intel Xeon x86-64) have been compared to "little" nodes based on low-powered counterparts (e.g., ARM, Intel Atom).

Loghin et al. (2015) have stated that big Xeon servers are more efficient for CPU-intensive jobs, while little ones based on ARM can perform better for I/O-intensive workloads. The study conducted by Kalyanasundaram and Simmhan (2017) has shown that Hadoop performance on an ARM64 server is comparable to an x86-64 Xeon counterpart for integer-based workloads and only lags behind for floating-point-intensive benchmarks like PageRank. However, the ARM64 server has a 3× smaller base power load than the x86-64 Xeon one. According to Malik et al. (2019), big Xeon servers are more efficient than Atom ones as the computational size of the problem increases while also providing a clear performance advantage for I/O-intensive Hadoop applications.

## Examples of Application

Most existing benchmarking tools provide multiple kinds of representative workloads to assess the performance of Big Data processing frameworks under different use cases. Early efforts have been originally designed for supporting batch-only workloads on Hadoop, while later ones have been mainly focused on the streaming scenario.

HiBench (Huang et al. 2010) is probably the first benchmark suite to evaluate and characterize the performance of Hadoop, being later extended to support in-memory frameworks such as Spark and Flink, even including a streaming component. Another long-lived, popular project is BigDataBench (Wang et al. 2014), which supports more than 30 workloads classified in 5 different application domains (e.g., social networks, e-commerce). Moreover, this suite focuses on improving the quality of the input data by generating them from real-world datasets. BigBench (Ghazal et al. 2013) proposes a standard end-to-end benchmark that covers a representative number of application profiles, with all major characteristics in the lifecycle of Big Data systems. There also exists some suites specifically oriented to evaluate the performance of in-memory processing frameworks. For instance, SparkBench (Li et al. 2017) features machine learning, graph processing, SQL queries, and streaming computations on Spark.

Regarding streaming scenarios, one of the most popular projects is the Linear Road Benchmark (LRB), originally proposed by Arasu et al. (2004), which is focused on application-level performance. LRB contains a toolkit comprising a data generator, a data sender, as well as a result validator. The Yahoo! Streaming Benchmark (YSB 2015) allows comparing multiple frameworks (e.g., Storm, Spark, Flink) by providing a streaming workload that simulates a real-world advertising analytics task. Another interesting project is StreamBench (Qian et al. 2016), which proposes a standard method to measure the throughput and latency of streaming engines with the use of a mediator between the data source and the data processing engine under evaluation.

Other tools not only provide a set of benchmarks but also ease the execution of the experiments. MRBS (Sangroya et al. 2012) is able to automatically set up a Hadoop cluster in a public cloud provider. Once the cluster is running, it injects the input dataset and executes the selected workloads, allowing to obtain multiple metrics related to execution time, throughput, and cost. BDEv (Veiga et al. 2018) is another example of this kind of tools but extending the support for other frameworks such as Spark and Flink. Once the user configures the experiments, BDEv deploys the frameworks over the cluster, generates the input datasets, and performs the selected workloads in a fully automatic manner. This tool also allows recording several metrics during the execution of the experiments, including execution time, resource utilization, energy efficiency, and hardware performance counters.

Finally, other tools focus on providing further insights into the performance of Big Data frameworks, even in real time. For instance, the BDWatchdog project (Enes et al. 2018) provides not only resource usage metrics in real time but also profiling information about the execution of the Java bytecode which can be extremely useful to detect performance bottlenecks.

## Future Directions for Research

As explained in previous sections, the overall performance of Big Data systems is affected by multiple factors, both in terms of hardware and software. So, the development of evaluation tools that can bring more meaningful insights to end users is key to get a more accurate view of the performance of Big Data systems.

Distributed processing frameworks are becoming increasingly complex to deploy, manage, and configure properly. Although there are some tools that ease their deployment and the execution of the workloads over a cluster, future ones must also cooperate with the frameworks in a more active way to help end users to make decisions. In addition to only obtaining raw performance metrics (e.g., runtime, latency, throughput), new tools should take advantage of such information to feed back the system and thus improve performance. Furthermore, as the configuration parameters of a framework can have a large influence on its performance in a certain scenario, automatic fine-tuning tools would be of great interest.

## Cross-References

- ▶ Analytics Benchmarks
- ▶ Big Data Benchmarking
- ▶ Energy Efficiency in Big Data Analysis
- ▶ Metrics for Big Data Benchmarks
- ▶ Stream Benchmarks
- ▶ Tools and Libraries for Big Data Analysis

## Recommended Reading

Apache Hadoop (2008) http://hadoop.apache.org. [Last visited: May 2020]

Arasu A et al. (2004) Linear road: a stream data management benchmark. In: 30th international conference on very large data bases (VLDB'04), Toronto, Canada, pp 480–491

Armbrust M et al. (2018) Structured Streaming: a declarative API for real-time applications in Apache Spark. In: ACM international conference on management of data (SIGMOD 2018), Houston, TX, USA, pp 601–613

Bakratsas M, Basaras P, Katsaros D, Tassiulas L (2018) Hadoop MapReduce performance on SSDs for analyzing social networks. Big Data Res 11:1–10

Bertoni M, Ceri S, Kaitoua A, Pinoli P (2015) Evaluating cloud frameworks on genomic applications. In: 2015 IEEE international conference on big data (IEEE BigData 2015), Santa Clara, CA, USA, pp 193–202

Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache Flink: stream and batch processing in a single engine. In: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 36(4)

Chen C, Li K, Ouyang A, Zeng Z, Li K (2018) GFlink: an in-memory computing architecture on heterogeneous CPU-GPU clusters for Big Data. IEEE Trans Parallel Distrib Syst 29(6):1275–1288

Chintapalli S, et al. (2016) Benchmarking streaming computation engines: Storm, Flink and Spark streaming. In: 30th IEEE international parallel and distributed processing symposium workshops, (IPDPSW'16), Chicago, IL, USA, pp 1789–1792

Choi IS, Yang W, Kee YS (2015) Early experience with optimizing I/O performance using high-performance SSDs for in-memory cluster computing. In: 2015 IEEE international conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, pp 1073–1083

Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Dede E, Fadika Z, Govindaraju M, Ramakrishnan L (2014) Benchmarking MapReduce implementations under different application scenarios. Fut Gener Comput Syst 36:389–399

Enes J, Expósito RR, Touriño J (2018) BDWatchdog: real-time monitoring and profiling of Big Data applications and frameworks. Fut Gener Comput Syst 87:420–437

Fadika Z, Dede E, Govindaraju M, Ramakrishnan L (2011) Benchmarking MapReduce implementations for application usage scenarios. In: 12th IEEE/ACM international conference on grid computing (GRID'11), Lyon, France, pp 90–97

Fadika Z, Govindaraju M, Canon R, Ramakrishnan L (2012) Evaluating Hadoop for data-intensive scientific operations. In: 5th IEEE international conference on cloud computing (CLOUD'12), Honolulu, HI, USA, pp 67–74

Fadika Z, Dede E, Govindaraju M, Ramakrishnan L (2014) MARIANE: using MapReduce in HPC environments. Fut Gener Comput Syst 36:379–388

Fang W, He B, Luo Q, Govindaraju NK (2011) Mars: accelerating MapReduce with graphics processors. IEEE Trans Parallel Distrib Syst 22(4):608–620

Ghazal A et al. (2013) BigBench: towards an industry standard benchmark for Big Data analytics. In: ACM international conference on management of data (SIGMOD'13), New York, NY, USA, pp 1197–1208

Gog I et al. (2015) Broom: sweeping out garbage collection from Big Data systems. In: 15th workshop on hot topics in operating systems (HotOS'15), Kartause Ittingen, Switzerland

Hindman B, et al. (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: 8th USENIX symposium on networked systems design and implementation (NSDI'11), Boston, MA, USA, pp 295–308

Hong J, Li L, Han C, Jin B, Yang Q, Yang Z (2016) Optimizing Hadoop framework for solid state drives. In: 2016 IEEE international congress on Big Data (BigData Congress 2016), San Francisco, CA, USA, pp 9–17

Hou J, et al. (2018) A case study of accelerating Apache Spark with FPGA. In: 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on Big Data science and engineering (TrustCom/BigDataSE 2018), New York, NY, USA, pp 855–860

Huang S, Huang J, Dai J, Xie T, Huang B (2010) The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 26th IEEE international conference on data engineering workshops (ICDEW'10), Long Beach, CA, USA, pp 41–51

Iqbal MH, Soomro TR (2015) Big Data analysis: Apache Storm perspective. Int J Comput Trends Technol 19(1):9–14

Isah H, Abughofa T, Mahfuz S, Ajerla D, Zulkernine F, Khan S (2019) A survey of distributed data stream processing frameworks. IEEE Access 7:154, 300–154, 316

Islam NS, et al. (2012) High performance RDMA-based design of HDFS over InfiniBand. In: International conference for high performance computing, networking, storage and analysis (SC'12), Salt Lake City, UT, USA, pp 35:1–35:12

Jakovits P, Srirama SN (2014) Evaluating MapReduce frameworks for iterative scientific computing applications. In: 2014 international conference on high performance computing & simulation (HPCS'14), Bologna, Italy, pp 226–233

Kalyanasundaram J, Simmhan Y (2017) ARM wrestling with Big Data: a study of commodity ARM64 server for Big Data workloads. In: 24th IEEE international conference on high performance computing (HiPC'17), Jaipur, India, pp 203–212

Kamburugamuve S, Ramasamy K, Swany M, Fox G (2017) Low latency stream processing: Apache Heron with Infiniband & Intel Omni-Path. In: 10th international conference on utility and cloud computing (UCC'17), Austin, TX, USA, pp 101–110

Karimov J, Rabl T, Katsifodimos A, Samarev R, Heiskanen H, Markl V (2018) Benchmarking distributed stream data processing systems. In: 34th IEEE international conference on data engineering (ICDE'18), Paris, France, pp 1507–1518

Kreps J, et al. (2011) Kafka: a distributed messaging system for log processing. In: 6th international workshop on networking meets databases (NetDB II), Athens, Greece, pp 1–7

Kulkarni S, et al. (2015) Twitter Heron: stream processing at scale. In: ACM international conference on management of data (SIGMOD'15), Melbourne, Australia, pp 239–250

Li M, Tan J, Wang Y, Zhang L, Salapura V (2017) SparkBench: a Spark benchmarking suite characterizing large-scale in-memory data analytics. Cluster Comput 20(3):2575–2589

Loghin D, Tudor BM, Zhang H, Ooi BC, Teo YM (2015) A performance study of Big Data on small nodes. Proc VLDB Endowment 8(7):762–773

Lu L, et al. (2016a) Lifetime-based memory management for distributed data processing systems. Proceedings of the VLDB Endowment 9(12):936–947

Lu X, Shankar D, Gugnani S, Panda DK (2016b) High-performance design of Apache Spark with RDMA and its benefits on various workloads. In: 2016 IEEE international conference on Big Data (IEEE BigData 2016), Washington, DC, USA, pp 253–262

Malik M, et al. (2019) Big vs little core for energy-efficient Hadoop computing. J Parallel Distrib Comput 129:110–124

Marcu OC, Costan A, Antoniu G, Pérez-Hernández MS (2016) Spark versus Flink: understanding performance in Big Data analytics frameworks. In: 2016 IEEE international conference on cluster computing (CLUSTER'16), Taipei, Taiwan, pp 433–442

Neshatpour K, Malik M, Ghodrat MA, Sasan A, Homayoun H (2015) Energy-efficient acceleration of Big Data analytics applications using FPGAs. In: 2015 IEEE international conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, pp 115–123

Nguyen K, et al. (2016) Yak: a high-performance Big-Data-friendly garbage collector. In: 12th USENIX symposium on operating systems design and implementation (OSDI'16), Savannah, GA, USA, pp 349–365

Noghabi SA, et al. (2017) Samza: stateful scalable stream processing at LinkedIn. Proc VLDB Endowment 10(12):1634–1645

Qian S, Wu G, Huang J, Das T (2016) Benchmarking modern distributed streaming platforms. In: 2016 IEEE international conference on industrial technology (ICIT 2016), Taipei, Taiwan, pp 592–598

Rathore MM, Son H, Ahmad A, Paul A, Jeon G (2018) Real-time Big Data stream processing using GPU with Spark over Hadoop ecosystem. Int J Parallel Program 46(3):630–646

Samosir J, Indrawan-Santiago M, Haghighi PD (2016) An evaluation of data stream processing systems for data driven applications. In: International conference on computational science (ICCS'16), San Diego, CA, USA, pp 439–449

Sangroya A, Serrano D, Bouchenak S (2012) MRBS: towards dependability benchmarking for Hadoop MapReduce. In: 18th international Euro-Par conference on parallel processing workshops (Euro-Par'12), Rhodes Island, Greece, pp 3–12

Shi J et al. (2015) Clash of the titans: MapReduce vs. Spark for large scale data analytics. Proc VLDB Endowment 8(13):2110–2121

Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: IEEE 26th symposium on mass storage systems and technologies (MSST'2010), Incline Village, NV, USA, pp 1–10

Spangenberg N, Roth M, Franczyk B (2015) Evaluating new approaches of Big Data analytics frameworks. In: 18th international conference on business information systems (BIS'15), Poznań, Poland, pp 28–37

van Dongen G, Van den Poel D (2020) Evaluation of stream processing frameworks. IEEE Trans Parallel Distrib Syst 31(8):1845–1858

Vavilapalli VK, et al. (2013) Apache Hadoop YARN: yet another resource negotiator. In: 4th annual symposium on cloud computing (SOCC'13), Santa Clara, CA, USA, pp 5:1–5:16

Veiga J, Expósito RR, Pardo XC, Taboada GL, Touriño J (2016a) Performance evaluation of Big Data frameworks for large-scale data analytics. In: 2016 IEEE international conference on Big Data (IEEE BigData 2016), Washington, DC, USA, pp 424–431

Veiga J, Expósito RR, Taboada GL, Touriño J (2016b) Flame-MR: an event-driven architecture for MapReduce applications. Fut Gener Comput Syst 65:46–56

Veiga J, Enes J, Expósito RR, Touriño J (2018) BDEv 3.0: energy efficiency and microarchitectural characterization of Big Data processing frameworks. Fut Gener Comput Syst 86:565–581

Wang L, et al. (2014) BigDataBench: a Big Data benchmark suite from Internet services. In: 20th IEEE international symposium on high-performance computer architecture (HPCA'14), Orlando, FL, USA, pp 488–499

Wasi-Ur-Rahman M, et al. (2013) High-performance RDMA-based design of Hadoop MapReduce over InfiniBand. In: 27th IEEE international parallel and distributed processing symposium workshops and PhD forum (IPDPSW'13), Boston, MA, USA, pp 1908–1917

Xuan P, Ligon WB, Srimani PK, Ge R, Luo F (2017) Accelerating Big Data analytics on HPC clusters using two-level storage. Parallel Comput 61:18–34

YSB (2015) https://github.com/yahoo/streaming-benchmarks, [Last visited: May 2020]

Zaharia M, et al. (2016) Apache Spark: a unified engine for Big Data processing. Commun ACM 59(11):56–65