

# BDEv 3.0: energy efficiency and microarchitectural characterization of Big Data processing frameworks

Jorge Veiga\*, Jonatan Enes, Roberto R. Expósito, Juan Touriño

*Computer Architecture Group, Universidade da Coruña, Campus de A Coruña, 15071 A Coruña, Spain*

---

## Abstract

As the size of Big Data workloads keeps increasing, the evaluation of distributed frameworks becomes a crucial task in order to identify potential performance bottlenecks that may delay the processing of large datasets. While most of the existing works generally focus only on execution time and resource utilization, analyzing other important metrics is key to fully understanding the behavior of these frameworks. For example, microarchitecture-level events can bring meaningful insights to characterize the interaction between frameworks and hardware. Moreover, energy consumption is also gaining increasing attention as systems scale to thousands of cores. This work discusses the current state of the art in evaluating distributed processing frameworks, while extending our Big Data Evaluator tool (BDEv) to extract energy efficiency and microarchitecture-level metrics from the execution of representative Big Data workloads. An experimental evaluation using BDEv demonstrates its usefulness to bring meaningful information from popular frameworks such as Hadoop, Spark and Flink.

*Keywords:* Big Data processing, performance evaluation, energy efficiency, microarchitectural characterization

---

## 1. Introduction

Over the last several years, the datasets managed by current Big Data systems have been experiencing a steady increase in size, demanding higher data processing capabilities from distributed frameworks like Apache Hadoop [1]. As a consequence, analyzing the performance and scalability of these processing frameworks has gained huge attention. Furthermore, the appearance of emergent in-memory alternatives (e.g. Apache Spark [2]) has improved the performance of the algorithms and widened the range of programming models and data operators that Big Data applications can use to process their input datasets in a more efficient and flexible way. In recent years, there has been a spur of research activity in assessing the benefits of adopting these emergent frameworks, while also considering the possible drawbacks that they might present.

The evaluation of Big Data frameworks is often done only in terms of performance and scalability, taking into account the execution time of the workloads as a main metric. Despite being significant, the sole use of these metrics

---

\*Corresponding author: Tel: +34 881 011 212, Fax: +34 981 167 160

Email addresses: [jorge.veiga@udc.es](mailto:jorge.veiga@udc.es) (Jorge Veiga), [jonatan.enes@udc.es](mailto:jonatan.enes@udc.es) (Jonatan Enes), [rreye@udc.es](mailto:rreye@udc.es) (Roberto R. Expósito), [juan@udc.es](mailto:juan@udc.es) (Juan Touriño)

can be oversimplifying as it obviates considerable information about the underlying potential causes of performance overheads that may appear in the experiments. Hence, the use of more insightful metrics is crucial for extracting meaningful conclusions when comparing different Big Data frameworks and/or algorithms.

Resource utilization is a useful system metric generally used to identify performance bottlenecks in terms of CPU, memory, disk and network. This metric enables the characterization of the applications to determine the suitability of the frameworks for each type of workload. Moreover, modern processors provide interfaces to access the values of their internal hardware counters through special registers. These counters can be used to monitor some CPU events such as cache accesses, executed instructions or branches taken. Thus, hardware counters allow conducting a more fine-grained analysis of the executions, leading to a better understanding of the behavior of Big Data frameworks in a specific CPU microarchitecture.

Data centers are currently experiencing a dramatic increase both in their computational capability and size, which in turn significantly increases their energy consumption. In fact, the power consumed by data centers is expected to be 8% of the worldwide electricity supply by 2020 [3]. Moreover, energy consumption forms the major fraction of the operational cost for public cloud providers, which are the preferred platforms to run Big Data applications because of their cost efficiency, elasticity, flexibility and scalability [4]. Due to these reasons, the evaluation and optimization of the energy efficiency of Big Data processing frameworks has become a great research challenge [5]. Although there are several works regarding this topic, they are usually restricted to a specific system and they do not provide any automatic method or tool for performing the experiments in other systems and platforms.

Considering the state of the art, there is a need for an evaluation tool that enables the extraction of several types of metrics from the execution of data processing frameworks. This tool must focus on portability, being able to carry out the experiments on a great variety of systems. However, some of the characteristics of the underlying system (e.g. processor model, CPU microarchitecture) may affect the type and number of metrics that are available, and so this tool must also adapt to this. Taking these aspects into consideration, this work aims to tackle the evaluation of Big Data processing frameworks under several perspectives and provide an automatic tool to perform this task in a more user-friendly way. The main contributions of this paper are:

- A comprehensive state-of-the-art survey of how Big Data processing frameworks are evaluated and the benchmark suites and tools that are most commonly used to perform this task, classifying the works depending on the metrics and frameworks that they take into account.
- The proposal of two extensions developed for our Big Data Evaluator (BDEv) tool to provide energy efficiency information and hardware counter monitoring, turning BDEv into a holistic evaluation tool that can characterize performance, resource utilization, energy efficiency and microarchitectural-level behavior in a fully automatic way.
- An evaluation of Hadoop, Spark and Flink in terms of the different metrics supported by BDEv, showing the

potential of our tool to provide meaningful insights when characterizing Big Data processing frameworks. As BDEv has been designed focusing on portability, the experiments are easy to conduct on different systems.

The rest of the paper is organized as follows: Section 2 presents an overview of the state of the art regarding the evaluation of data processing frameworks, including a classification of methods and tools. Section 3 describes the overall design and main targeted use cases of BDEv, along with the proposed extensions to monitor energy consumption and processor events. Section 4 describes the experimental testbed where Hadoop, Spark and Flink have been evaluated using BDEv, and Section 5 analyzes the results obtained. Finally, Section 6 extracts the main conclusions of the paper and proposes future work.

## **2. State of the art in evaluating data processing frameworks**

This section aims to provide an overview of Big Data processing systems and how they are currently being evaluated. Once the most popular distributed processing frameworks (Section 2.1) and benchmark suites (Section 2.2) are introduced, several previous works that analyze the performance characteristics of Big Data workloads are presented (Section 2.3), including interesting metrics such as their energy efficiency (Section 2.4) and microarchitectural performance (Section 2.5).

### *2.1. Big Data processing frameworks*

Hadoop [1], along with its implementation of the MapReduce model [6], has long been one of the most popular frameworks for large-scale batch processing. Nowadays, recent requirements from the users have made the development of new paradigms, technologies and tools necessary. A clear example of this issue is the appearance of use cases that require iterative and/or stream processing, implying the use of more advanced frameworks. This is mandatory in order to build pipelines that handle and process data which arrives in a real-time manner from different sources, which cannot be done with traditional batch processing frameworks such as Hadoop. In general, current Big Data processing systems can be classified in three groups: 1) batch-only, 2) stream-only and 3) hybrid, according to their underlying data processing engine, framework topology and targeted use case.

Batch-only frameworks were the first to appear in order to handle big datasets in a scalable and easy-to-program way. According to the MapReduce paradigm as originally conceived by Google [6], input data is split and processed in chunks following no particular order and generally with no time constraints. This model computes the output data by using two phases: Map and Reduce. The Map phase extracts the relevant attributes for the computation and the Reduce phase operates them to get the final result. Currently, the most popular batch processing framework is Hadoop MapReduce, together with the Hadoop Distributed File System (HDFS) [7] to manage distributed data storage and Yet Another Resource Negotiator (YARN) [8] for resource management. In addition to running MapReduce jobs, the vast Hadoop ecosystem has become the most commonly used platform to solve Big Data problems, including multiple

open-source projects such as the machine learning library Mahout [9], the graph-processing engine Giraph [10], the HDFS-based database HBase [11] and many more.

It is worth mentioning several modifications of Hadoop that adapt it to specific interconnection networks such as RDMA-Hadoop [12], or that seek overall performance improvements like Flame-MR [13]. RDMA-Hadoop adapts several Hadoop subsystems (e.g. HDFS) to use Remote Direct Memory Access (RDMA) networks like InfiniBand, in order to achieve better communication efficiency (e.g. HDFS replication, MapReduce shuffle). In the case of Flame-MR, it presents a novel design that includes several optimizations such as an event-driven architecture and pipelined data movements, improving the overall performance of the MapReduce model in an automatic and transparent way (i.e. it is Hadoop-compatible).

The second group of Big Data frameworks, stream-only, were developed when the need to process large-sized data streams arose. This is a scenario where batch processing is not applicable due to time constraints, the possibility of having an unending stream and the lack of real-time support. Examples of stream processing frameworks are Storm [14], Heron [15] and Samza [16]. These frameworks follow a different approach than MapReduce, creating a graph-based architecture using pipelines and Direct Acyclic Graphs (DAGs). Data management in streaming frameworks is also different from the batch-only approach, which mainly uses HDFS. The streaming paradigm introduces the idea of sources and sinks. A source is defined as the origin of the data into the streaming architecture, while the sink is the end where output data is persisted. Although HDFS can still be used, higher-level storage solutions are preferred. Examples of such solutions are queue systems like Kafka [17] or databases like Cassandra [18], which adapt better to the flowing nature of data streams. Moreover, in order to deploy a stream processing architecture, another component is needed to properly manage the data throughout the dataflow. To play such a role, message-oriented middleware is required, such as the aforementioned Kafka, RabbitMQ [19] or ZeroMQ [20]. Finally, streaming frameworks may also use other resource management tools apart from YARN, especially in cases where the deployed streaming pipelines and architectures need to be running continuously (i.e. if any part of the framework fails, it has to be relaunched in an automatic way). Examples of such tools are Mesos [21] and Aurora [22], used by Heron, while Samza relies solely on YARN. Regarding Storm, it can be integrated with YARN, Mesos and Docker [23], or run stand-alone.

Finally, hybrid solutions such as Spark [2], Flink [24] or Apex [25] try to offer a unified solution for data processing by covering both the batch and stream processing scenarios. These solutions inherit the functionalities offered by batch processing models like MapReduce, as well as the new features of streaming frameworks. To provide a more efficient data processing but remaining usable for stream processing, these solutions follow the DAG design philosophy, while also implementing new architectures with further optimizations. Spark provides a batch processing engine based on a novel data structure, Resilient Distributed Datasets (RDDs) [26], which are in-memory data collections partitioned over the nodes of a cluster. As RDDs keep data in memory, Spark can avoid disk traffic and alleviate some of the issues that hinder the performance of Hadoop, especially for iterative workloads. Spark also provides stream processing capabilities through Spark Streaming, which implements a micro-batch processing model by buffering the stream in

Table 1: Comparison of batch, stream and hybrid frameworks

	Paradigm	Resource management	Data management	Real-time	Use case
<b>Hadoop</b>	batch-only	YARN	distrib. filesystems (e.g. HDFS), object storage (e.g. S3)	no	batch processing of non time-sensitive workloads
<b>RDMA-Hadoop</b>	batch-only	YARN	distrib. filesystems (e.g. HDFS), parallel filesystems (e.g. Lustre), object storage (e.g. S3)	no	native support for RDMA networks
<b>Flume-MR</b>	batch-only	YARN	distrib. filesystems (e.g. HDFS), object storage (e.g. S3)	no	transparent optimization for MapReduce workloads
<b>Storm</b>	stream-only	YARN, stand-alone	databases (e.g. Cassandra), queue systems (e.g. Kafka)	yes	low-latency and real-time processing pipelines
<b>Heron</b>	stream-only	YARN, Mesos, Docker, stand-alone	databases (e.g. Cassandra), queue systems (e.g. Kafka)	yes	improvements over Storm
<b>Samza</b>	stream-only	YARN	queue systems (e.g. Kafka)	yes	deployment of large pipelines and dataflows with emphasis on reliability and statefulness
<b>Spark</b>	hybrid	YARN, Mesos, stand-alone	distrib. filesystems (e.g. HDFS), databases (e.g. Cassandra), object storage (e.g. S3)	near real-time	batch and micro-batch processing with streaming support
<b>Flink</b>	hybrid	YARN, Mesos, stand-alone	distrib. filesystems (e.g. HDFS), databases (e.g. Cassandra), queue systems (e.g. Kafka)	yes	stream processing with support for traditional batch workloads
<b>Apex</b>	hybrid	YARN	distrib. filesystems (e.g. HDFS), databases (e.g. Cassandra), object storage (e.g. S3), queue systems (e.g. Kafka)	yes	unified stream and batch processing

sub-second increments which are sent as small fixed datasets for batch processing. Moreover, Spark includes other built-in libraries like MLlib [27] for machine learning and GraphX [28] for graph algorithms. Regarding Flink, it relies on the custom-developed Stratosphere platform [29] to specifically target stream processing. Flink defines streams and transformations as the data sources and operations, respectively. Unlike Spark, Flink provides a stream engine that allows us to handle incoming data on an item-by-item basis as a true stream. It also supports batch processing by simply considering batches to be data streams with finite boundaries. Like Spark, Flink also includes built-in libraries that support machine learning algorithms (FlinkML [30]) and graph processing (Gelly [31]). Finally, Apex has been recently released as a new proposal that aims to offer a mature platform that unifies batch and stream workloads. It provides developers with several libraries and tools in order to lower the barrier to entry and support a broad spectrum of data sources and sinks.

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

## 2.2. Benchmarking tools

This section offers an overview of existing benchmarking tools for evaluating data processing frameworks. The most long-lived projects were originally designed for analyzing the performance of batch-only workloads on Hadoop. That is the case of HiBench [32], a popular benchmark suite that supports 19 workloads in its current version (6.0),

including micro-benchmarks, machine learning algorithms, SQL queries, web search engines, graph benchmarks and streaming workloads. Apart from Hadoop, it also supports hybrid and stream-only frameworks like Spark, Flink and Storm and message queuing systems like Kafka. However, not all the workloads are available for all the frameworks. HiBench generates the input datasets needed for the workloads and reports the execution time, throughput and system resource utilization as main metrics. Another well-known project, BigDataBench [33], improves the quality of the input data by providing means to generate them from 15 real-world datasets. Furthermore, it supports 37 workloads classified in 5 different application domains: search engine, social networks, e-commerce, multimedia analytics and bioinformatics. From version 2.0 on, BigDataBench also includes DCBench [34] and CloudRank-D [35], other benchmark suites which were previously independent. MRBS [36] is a suite oriented to multi-criteria analysis as it takes into account different metrics like latency, throughput and cost. MRBS includes 32 MapReduce workloads from 5 application domains: recommendation systems, business intelligence, bioinformatics, text processing and data mining. Moreover, MRBS can automatically set up the Hadoop cluster on which the benchmark will run using a public cloud provider configured by the user. Once the cluster is running, MRBS injects the dataset and runs the workload, releasing the resources when the experiment concludes. Apart from evaluating the execution time of the workloads, users can also assess the multi-tenancy of a Hadoop cluster by using GridMix [37]. This benchmark launches several synthetic jobs which emulate different users and queues, being able to evaluate Hadoop features like the distributed cache load, data compression/decompression and jobs with high memory requirements or resource utilization (e.g. CPU, disk).

Benchmarking tools also exist which enable us to evaluate other Big Data systems built on top of Hadoop. That is the case of PigMix [38], which evaluates Pig, a high-level language for expressing data analytics workloads on top of Hadoop. Furthermore, some Big Data benchmarks focus on evaluating the adaptability of Hadoop to traditional database use cases. One example is MRBench [39], which implements 22 relational SQL queries (e.g. select, join). The authors of MRBench describe how these queries can be translated into MapReduce jobs, and the issues that may arise. BigBench [40] proposes a standard benchmark for Big Data that covers a representative number of application profiles. It includes a data model that represents the typical characteristics of Big Data systems (i.e. variety, velocity and volume), and a synthetic data generator that adopts some of its parts from traditional database benchmarks to support structured, semi-structured and unstructured data.

In the last years, new benchmark suites specifically oriented to in-memory processing frameworks have appeared, like SparkBench [41]. It includes 10 workloads with typical usage patterns of Spark: machine learning, graph processing, stream computations and SQL query processing. It takes into account different metrics like execution time, data process rate, shuffle data size, resource consumption and input/output data size. Although little work can be found regarding benchmarks specifically oriented to Flink, some proposals adapt existing ones to its new programming paradigm. That is the case of [42], which uses BigBench to compare Flink and Hive [43], showing that the former can achieve time savings of about 80%.

Not all Big Data benchmarking tools are focused on evaluating data processing systems. For example, the AM-

PLab benchmark [44] is focused on evaluating data warehousing solutions such as Hive, Tez [45], Shark [46] and Impala [47]. AMPLab uses HiBench to generate the data and performs the evaluation by means of a benchmark that includes scans, aggregations, joins and user-defined functions. The Yahoo! Cloud Serving Benchmark (YCSB) [48] aims to evaluate different NoSQL databases like HBase [11], Cassandra [18], MongoDB [49], Redis [50], Memcached [51], and many others. YCSB currently includes 6 different workloads, providing an input dataset generator and a configurable workload executor.

In this paper, we use our Big Data Evaluator (BDEv) tool [52], based on our previous work presented in [53]. BDEv is a powerful tool to perform evaluations of different Big Data frameworks on a cluster. By using a set of configuration files, BDEv launches the framework daemons, generates the input datasets and executes the workloads in a fully automatic way, generating an output report that includes the execution time of the workloads, resource utilization results and their associated graphs. BDEv supports up to 9 different Big Data frameworks, and 12 workloads including typical benchmarks such as TeraSort, PageRank and K-Means.

### *2.3. Performance studies of Big Data processing frameworks*

The Apache Hadoop framework has dominated the world of Big Data over the last decade, and thus its performance has been thoroughly addressed by a wide range of papers [54, 55, 56, 57]. However, recent works focus on in-memory processing frameworks due to the better flexibility and performance they provide. That is the reason why Spark is compared with Hadoop in [58], taking into account performance and resource utilization. The results show that Spark can reduce the execution time by 60% and 80% for CPU-bound and iterative benchmarks, respectively. However, Hadoop is 50% faster than Spark for I/O-bound benchmarks such as Sort. Meanwhile, another work [59] claims that frameworks like Twister [60] or paradigms like the Message Passing Interface (MPI) can provide better performance than Spark for iterative algorithms. By comparing Hadoop, HaLoop [61], Twister, Spark and an MPI library, the authors conclude that Hadoop obtains the worst performance results. Although Spark does not provide the best performance according to [59], it proves to be the most appropriate option for developing Big Data algorithms in a flexible way. This is because Twister does not support HDFS, which is indispensable for storing big datasets, whereas MPI is not a feasible option for developing and maintaining Big Data applications as it does not abstract data distribution, task parallelization and inter-process communications. A similar conclusion is reached in [62], which compares the performance of Spark and MPI for evolutionary algorithms.

Nowadays, Flink attracts increasing interest when evaluating Big Data frameworks, usually being compared with Spark. All the works that compare Flink with Spark conclude that the performance they provide is highly dependent on the workload executed. That is the case of [63], which compares Spark and Flink using typical benchmarks like WordCount, K-Means, PageRank and relational queries. The results show that Flink outperforms Spark except in the case of the most computationally intensive workloads (e.g. WordCount). Another work [64] analyzes the performance of Flink and Spark, configuring Spark both with the default and the optimized Kryo serializers. The paper uses three different genomic applications for evaluating the frameworks: Histogram, Map and Join. Flink shows better

performance in Histogram and Map, while Spark gets the best results for Join.

In order to provide some insight into the differences between Spark and Flink, some internal design characteristics are addressed in [65], identifying a set of configuration parameters that have a major influence on the execution time and scalability of these frameworks: task parallelism, shuffle configuration, memory management and data serialization. The benchmarks are also analyzed to identify the data operators they use. The main conclusion is that Spark is 40% faster than Flink for large-graph processing, while Flink is 33% faster than Spark for single-iteration and small-graph workloads. Further evaluations are conducted in [66] but using updated versions of these frameworks, showing that Spark provides better results and stability in general. However, some new features introduced by Flink can accelerate iterative algorithms, like the use of delta iterations in PageRank, which allows us to reduce the execution time by 70% compared with Spark. The authors also take into account other important parameters of the system: HDFS block size, input data size, interconnection network and thread configuration.

As can be seen, the evaluation of Spark and Flink is gaining attention, not only in terms of performance but also taking into account usability, configuration parameters and resource utilization. The performance obtained by these frameworks is highly dependent not only on the characteristics of the workload, but also on the particular version being evaluated (both are active projects that are continuously evolving). Furthermore, the suitability of the workloads that are usually executed in these works has been discussed in [67], proposing a new evaluation methodology that takes into account the input data size and the characteristics of the data model. Note that all the previous works have focused their evaluations on the batch processing capabilities of Spark and Flink. Other recent works have also assessed their stream processing capabilities [68, 69, 70], comparing them with other stream-only technologies such as Storm or Samza.

#### 2.4. Energy efficiency

The energy efficiency of Big Data frameworks has been addressed by previous works under different points of view, studying the main factors that can impact energy consumption and, more recently, developing new proposals in order to decrease it. These works can be classified into three different groups, depending on the method used to get the energy measurements.

The first group is composed by works that estimate the power values by using an energy model. These models usually take into account the power specifications of the underlying node and the utilization of system resources like the CPU. One clear example is [71], an evaluation performed in the Microsoft Azure cloud that uses a model based on the CPU load of the virtual machines to estimate power consumption. The results, which include experiments with a Hadoop cluster, show that heterogeneity of cloud instances harms energy efficiency. This problem is addressed in [72] by developing a new self-adaptive task assignment approach that uses an ant colony algorithm to improve the performance and energy efficiency of MapReduce jobs in heterogeneous Hadoop clusters. The authors modify Hadoop to implement a new scheduling algorithm, obtaining 17% of energy savings compared to the default scheduler. The power measurements obtained in [72] are estimated by using a simple model based on CPU resource utilization and



the power consumption of the machine in idle state. More examples of power estimation techniques are included in [73], a survey of different power consumption models for CPUs, virtual machines and servers.

In the second group, power values are obtained by means of an external power meter that is directly connected to the nodes. This is the case of [74], which analyzes the performance and power consumption of several deployment configurations of a Hadoop cluster. The results show that separating data and compute services involves lower energy efficiency than collocating them, and that the power consumption profiles are heavily application-specific. In the experiments, the power metrics were provided by APC Power Distribution Units (PDUs). A similar PDU is used to demonstrate that the energy-aware MapReduce scheduling algorithm proposed in [75] can consume 40% less energy on average. Another work that analyzes the energy efficiency of Hadoop [76] uses a power meter to measure the power consumption of the whole system. The paper identifies four factors that affect the energy efficiency of Hadoop: CPU intensiveness, I/O intensiveness, HDFS replication factor and HDFS block size, giving recommendations related to each of them. Another performance study [77] compares the use of Hadoop on “small” ARM nodes with “big” Intel Xeon ones, concluding that I/O-intensive workloads are more energy efficient in Xeon nodes, while CPU-intensive ones are more efficient on ARM nodes. In this work, power values are recorded by using a Yokogawa power monitor connected to the main electric input line of the system. In [78], “big” Intel Xeon nodes are compared with “small” Intel Atom ones using a Watts Up Pro power meter. The results show that Xeon nodes perform more efficiently as the input data size increases. The energy consumption of mobile devices can also be measured by using power monitors. In [79], which analyzes the energy efficiency of Big Data stream mobile applications, the batteries are sampled by using a power monitor to measure the energy consumed during 3G/WiFi communications.

The last group of works uses a software interface to access energy counters provided by some CPU vendors. Some of these counters can be accessed by monitoring tools like the Intel data center manager, used in [80] to analyze the energy efficiency of Hadoop on an HPC cluster. This work also proposes the use of the  $ED^2P$  metric to evaluate the performance-energy efficiency of Hadoop. Vendor-specific technologies like the HPE integrated Lights-Out (iLO), consisting of a Baseboard Management Controller (BMC) accessible through a REST interface, also allow us to obtain power measurements of the node. HPE iLO has been used in [54] to analyze the energy efficiency of different flavors of Hadoop on an HPC cluster, concluding that accelerating the completion of the workloads by using faster interconnects (e.g. InfiniBand) or disks (e.g. SSD) can significantly reduce the energy consumed. However, the most popular way of accessing these energy counters is using power management interfaces that are provided by CPU vendors, which can be used to monitor power in a wide range of modern CPUs. While AMD delivers the Application Power Management (APM) interface, Intel provides the Running Average Power Limit (RAPL) interface [81]. The accuracy of RAPL has been tested in [82], proving that the values it provides can be very useful to characterize the power consumption of an application. RAPL has also been used in [83] to evaluate the energy efficiency of graph-processing engines such as Giraph and the Spark GraphX library. This work shows that GraphX is able to consume 42% less energy than Giraph thanks to the use of in-memory RDDs, although it suffers from memory problems that do not appear in Giraph. In [84], RAPL is also used to get power consumption values to compare the horizontal and

vertical scalability of a Spark cluster, showing that vertical scalability provides better performance per watt.

### 2.5. *Microarchitectural characterization*

Most of the previous works that evaluate data processing systems generally focus on execution time and resource utilization as the only metrics for analyzing their performance, while only some of them also take into account energy efficiency. However, there are few works that try to further explore the results obtained in their evaluations by considering other important factors. One interesting example of such metrics is the evaluation of Big Data systems in terms of their microarchitectural performance, by collecting the hardware counters provided by modern CPUs. For instance, available counters allow us to obtain the number of CPU cycles, cache references and branch mispredictions. Note that depending on the CPU model there are different kinds of counters, even across a same vendor (i.e. the availability of these counters is highly CPU-dependent).

In [85], the characterization of Big Data benchmarks aims to identify redundancies in benchmark suites, selecting some representative subsets of HiBench and BigDataBench workloads in order to avoid repetitive results. To do so, the authors execute several benchmarks with Hadoop calculating instructions per cycle, cache miss and branch misprediction ratios, and off-chip bandwidth utilization using the Oprofile tool [86]. Then, they perform a principal component analysis and a hierarchical clustering algorithm to determine which benchmarks are redundant. A similar study is performed in [87], but widening the range of microarchitectural-level metrics that are analyzed and also using other frameworks apart from Hadoop like Spark, Hive and Shark. In this case, the benchmarks are selected from BigDataBench and the metrics are collected using Perf [88]. The work [78] cited in the previous subsection also uses performance counters to compare Intel Xeon and Intel Atom nodes, obtaining the values by means of the Intel VTune performance profiling tool.

Nowadays, the increasing use of memory intensive data analytics is motivating the appearance of new studies that characterize the performance of in-memory frameworks. Intel VTune is employed in [89] to study the CPU and memory intensiveness of several Spark workloads, revealing that the latency of memory accesses is the main performance bottleneck. Another work [90] proposes the effective cache hit ratio, which aims to be more representative than the cache hit ratio when explaining the relationship between the number of cache hits and the execution times in Spark. The effective cache hit ratio only takes a reference to a cache line as a hit when the dependencies of such a line are also located in the cache. Moreover, the authors demonstrate the relevance of their proposal by implementing a new Spark memory manager that handles cache lines and their dependencies as blocks. The results show that their approach speeds up data-parallel jobs by up to 37%.

Note that none of the works cited in the previous subsections provide a tool that allows us to record energy consumption and performance counters to characterize different frameworks and benchmarks in a portable way. In this work, we expand our BDEv tool with new features that enable us to execute customizable workloads and automatically calculate these metrics and represent their associated results.

Table 2: Summary of works evaluating Big Data frameworks

Work	Evaluated metrics				Frameworks			Avail.
	Perf.	Resources	Energy	Microarch.	Hadoop	Spark	Flink	
[58]	✓	✓			✓(2.4.0)	✓(1.3.0)		
[64]	✓					✓(1.3.1)	✓(0.9.0)	
[65]	✓	✓				✓(1.5.3)	✓(0.10.2)	
[66]	✓				✓(2.7.2)	✓(1.6.1)	✓(1.0.2)	✓
[71]			✓(Model)		✓			
[74]	✓		✓(PDUs)		✓(0.20)			
[83]			✓(RAPL)		✓(0.20)	✓(1.4.1)		
[85]				✓(Oprofile)	✓(1.0.3)			
[87]				✓(Perf)	✓(1.0.2)	✓(0.8.1)		
[89]	✓	✓		✓(VTune)		✓		
This work	✓	✓	✓(RAPL)	✓(Oprofile)	✓(2.7.3)	✓(2.2.0)	✓(1.3.2)	✓

## 2.6. Summary

This section has provided an in-depth survey regarding the state of the art in benchmarking Big Data processing frameworks by presenting around 50 works that address this topic. As a summary, Table 2 includes a group of selected works according to their relevance and similarity to this one. The table shows the metrics evaluated in each work: performance, resource utilization, energy efficiency and microarchitectural characterization. It also includes which Big Data frameworks are evaluated and their version (if indicated). The last column shows if there is any publicly available tool to perform the experiments. Finally, the last row includes the metrics and frameworks addressed in this paper in order to provide a direct comparison with the other works.

## 3. BDEv 3.0: goals and design

Basically, BDEv [52] is an evaluation tool for in-depth analysis of Big Data processing frameworks in a portable way. This section first describes its main features (Section 3.1) and overall design (Section 3.2), while proposing new extensions to the available metrics (Section 3.3). Finally, the main use cases targeted by BDEv are also addressed (Section 3.4).

### 3.1. BDEv features

BDEv is based on a set of features defined by its objectives. These features are:

*Automation of experiments.* BDEv is able to carry out the experiments without any interaction from the user. Once the evaluation parameters are defined, BDEv performs the entire experiment cycle in an automatic way, including the setting of the frameworks, the generation of the input datasets and the execution of the workloads.

*Leveraging of system resources.* The configuration of the frameworks is automatically set by detecting the resources available in the system, like the number of CPU cores or the memory size. Nevertheless, users can change any of these parameters to fit their specific needs. BDEv also allows us to configure the frameworks to make use of resources that are typically available in High Performance Computing (HPC) systems, like the IP over InfiniBand (IPoIB) interface.

*Multi-metric evaluation.* The outcome of the experiments includes the output of the workloads, the execution time and resource utilization statistics (e.g. CPU, disk, network). Using that information, the user is able to analyze the behavior of the frameworks under several perspectives. This paper proposes two further metrics: energy efficiency and microarchitectural characterization (see Section 3.3).

*Flexibility.* BDEv can evaluate Big Data systems in different ways, adapting itself to the particular needs of the user. Therefore, it provides a wide set of experimental options to determine the aspects (e.g. configuration parameters, frameworks, workloads) that are evaluated in each experiment.

*Portability.* BDEv aims to be easily executed in different kinds of systems. This involves the use of some system-dependent configuration parameters, which can be defined by the user, as well as the awareness of the environment where the experiments are being run (e.g. adaptability to using job schedulers in HPC systems).

*Error and timeout detection.* In some cases, errors or exceptions can occur during the experiments. If they are not detected, they can lead to incorrect measurements. BDEv analyzes the output of the workloads to check for errors, avoiding the use of erroneous executions for the final results. Users can also configure a timeout threshold, so if a workload exceeds this value the execution is aborted and its results are discarded.

*Easy collection of results.* The information obtained by the different metrics is analyzed to extract a summary that includes statistical results and automatically generated graphs. This eases the comparisons made by users.

## 3.2. BDEv design

This section describes the main subsystems and components of BDEv, shown in Figure 1.

### 3.2.1. Experiment

The *Experiment* package contains the components related to the general behavior of BDEv. The *Workflow manager* determines the operations required to carry out the evaluation by using the experimental parameters provided by the *Configuration manager*. Then, it uses the *Framework launcher* and the *Workload runner* to schedule framework- and workload-related operations, respectively. When the evaluation finishes, the *Result reporter* creates the summary of experimental results, generating related graphs by using the *Graph generator*.

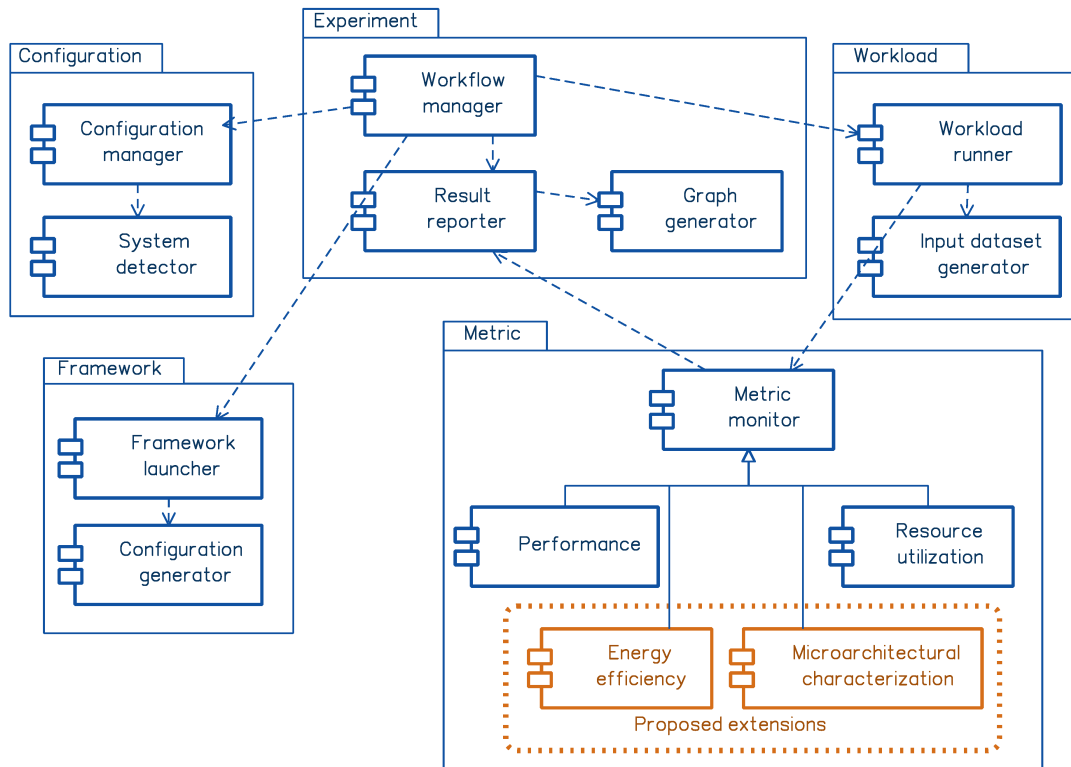


Figure 1: BDEv design overview

### 3.2.2. Configuration

The *Configuration* package contains the components that determine the parameters to use in the evaluation. The *Configuration manager* reads the parameters that have been defined by the user in the configuration directory. These parameters are related to different aspects, like system characteristics (e.g. directory to store the temporary files, network interfaces to use), HDFS configuration (e.g. block size) and other framework-specific parameters (e.g. number of workers per node, sort buffer size). They also include the experiment parameters defined by the user, such as the cluster size, frameworks and workloads to evaluate. If any system-related parameter is not set, the *System detector* determines the default value by analyzing the system, like the available memory size or the number of CPU cores.

### 3.2.3. Framework

The components of the *Framework* package control the correct configuration and launching of the frameworks to be evaluated. The *Framework launcher* manages the deployment of the underlying daemons needed to run each framework (e.g. NameNode/DataNode for HDFS), stopping them once the evaluation has finished. Before the launch, the configuration directory of the framework is set up by using a *Configuration generator* that is specific for each framework and uses the parameters previously defined by the *Configuration manager*.

Table 3 contains the frameworks currently supported in BDEv 3.0, their versions and the network interconnects

Table 3: Frameworks supported in BDEv

Framework	Version	Network interconnects
Hadoop	1.2.1 / 2.7.3	GbE / IPoIB
Hadoop-UDA	1.2.1 / 2.7.3	IPoIB & RDMA
RDMA-Hadoop	0.9.9	(GbE / IPoIB) & RDMA
RDMA-Hadoop-2	1.1.0 / 1.2.0	(GbE / IPoIB) & RDMA
Spark	1.6.3 / 2.2.0	GbE / IPoIB
RDMA-Spark	0.9.3 / 0.9.4	(GbE / IPoIB) & RDMA
Flink	1.2.1 / 1.3.2	GbE / IPoIB
DataMPI	0.6.0	GbE / IPoIB
Flame-MR	0.10.0 / 1.0	GbE / IPoIB

they can use. Apart from Hadoop, Spark and Flink, BDEv also supports some modifications of these frameworks that use RDMA communications, like Hadoop-UDA [91], RDMA-Hadoop [12] and RDMA-Spark [92]. Furthermore, other frameworks completely implemented from scratch like DataMPI [93] and Flame-MR [13] are also included. To our knowledge, no other evaluation tool includes support for as many frameworks as BDEv.

### 3.2.4. Workload

The components related to the execution of the workloads are contained in the *Workload* package. First, the *Input dataset generator* builds up the datasets required for their execution. Next, the *Workload runner* executes the selected workloads, using the *Metric monitor* components to record the different metrics.

BDEv supports different benchmark types, including standard micro-benchmarks, graph algorithms, machine learning workloads and SQL queries. It also allows executing interactive and batch user-defined commands. The input data generators are specific to each benchmark type, but the user can also define its own input generator. Table 4 contains the benchmarks currently included in BDEv and their corresponding input dataset generators.

### 3.2.5. Metric

This package contains the monitors that extract the evaluation metrics configured by the user. The *Metric monitor* launches the monitoring processes when a workload starts, stopping them when it finishes. Then, it communicates with the *Result reporter* and the *Graph generator* to create the reports and graphs, respectively, associated with the recorded data.

Each subcomponent of the *Metric monitor* specializes on a specific metric. The *Performance* metric records the execution time of the workloads, taking also into account the results obtained using different cluster sizes to provide scalability results. *Resource utilization* monitors system resources (e.g. CPU, disk, network), obtaining results for each individual node and the average values across the cluster.

Table 4: Benchmarks supported in BDEv 3.0

Workload type	Benchmark	Input Generator
Micro-benchmark	TestDFSIO	-
	WordCount	RandomTextWriter
	Grep	
	Sort	
TeraSort	TeraGen	
Graph processing	ConComp PageRank	DataGen
Machine Learning	Bayes K-Means	DataGen GenKMeansDataset
SQL queries	Aggregation Join Scan	DataGen
User	Command	Provided by the user

As mentioned before, BDEv 3.0 aims to enable holistic evaluations of Big Data frameworks by providing multiple evaluation metrics. In order to achieve this goal, we propose two new extensions to the *Metric* package (detailed in the next section) that can provide meaningful insight on the behavior of the frameworks. The *Energy efficiency* monitor records the energy and power being consumed in the cluster, while the *Microarchitectural characterization* monitor records the processor hardware events occurred during the execution.

### 3.3. Proposed extensions

This section addresses the implementation of the new metrics in BDEv 3.0, detailing the energy efficiency monitoring in Section 3.3.1 and the microarchitectural characterization in Section 3.3.2.

#### 3.3.1. Energy efficiency

Section 2.4 described how energy efficiency is usually assessed in Big Data evaluations: using an energy model, a physical power meter or a software interface. In BDEv, we have chosen the latter alternative as it provides more accurate power information than using a model, also ensuring the portability across several systems, which is not possible with power meters. We have used the RAPL interface that is available in all Intel processors from the Sandy Bridge microarchitecture onwards. We plan to extend this support to AMD processors in the near future by using the APM interface.

Basically, RAPL provides an easy way to keep the power of the processor within a user-defined limit. The internal circuitry of the CPU can estimate the current energy consumption in a highly accurate way, providing these results to the user via Model-Specific Registers (MSRs). These results are directly associated with components of the Intel RAPL architecture, depicted in Figure 2. A package identifies the energy consumption measurements of each CPU

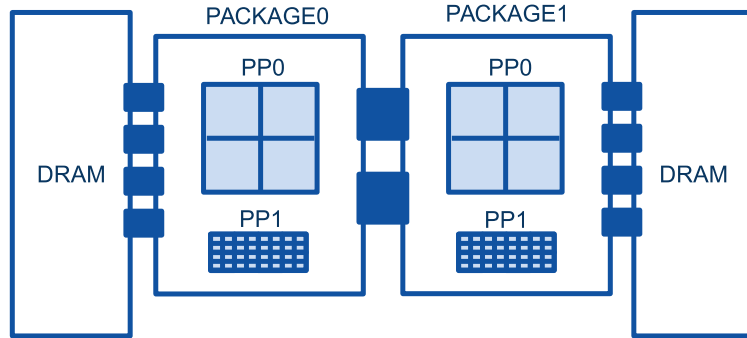


Figure 2: Overview of the Intel RAPL architecture for a dual-socket system

socket (i.e. each processor). Inside each package, separated energy values are provided for the processing cores, labeled as PP0 (Power Plane 0), and the integrated graphic card (if any), labeled as PP1. These energy values always have the following relationship:  $PP0 + PP1 \leq \text{package}$ . RAPL also provides independent values for the memory modules that are associated to each package, labeled as DRAM. Note that the figure shows the architecture of a dual-socket system, each socket providing a quad-core processor and a graphic card. Depending on the particular processor microarchitecture, the measurements of some of these components may not be available.

In order to implement the *Energy efficiency* monitor in BDEv, we have adapted, extended and integrated an existing tool [94] that accesses RAPL counters using the Performance API (PAPI) interface [95]. Amongst many other features, PAPI provides a standard interface for accessing hardware counters and its RAPL component allows us to obtain energy consumption values. The monitor detects all RAPL-related events available in a node and records the energy values using a configurable time interval, storing them to the corresponding output files. Note that RAPL only provides energy consumption values, and so the power consumption is calculated by BDEv based on the energy consumed in each time interval.

*Energy efficiency* monitors are launched when a workload is executed, as previously described in Section 3.2.5. When the execution is finished, the *Graph generator*, using the information elaborated by the *Result reporter*, builds the time series graphs corresponding to the recorded values for each node. The total energy consumed by all the nodes of the cluster and the average power consumption per node are also calculated, and the corresponding graphs are automatically generated for both metrics. Furthermore, additional graphs that show the  $ED^2P$  metric mentioned in Section 2.4 and used in the evaluation performed in Section 5.1 are also generated.

### 3.3.2. Microarchitectural characterization

The microarchitectural characterization of Big Data frameworks can provide useful insight on the data operations (e.g. map, join) that can be limiting the performance of the workloads. As mentioned in Section 2.5, current processors provide access to a set of hardware performance counters that can allow us to perform a fine-grained analysis in terms of several microarchitectural characteristics. These counters can detect and count certain microarchitectural



events from several hardware sources such as the CPU pipeline or the different cache levels. Such events can help to characterize the interaction between the applications and the hardware, thus providing a more precise picture of the hardware resource utilization. In fact, existing projects like Spark Tungsten [96] are specifically focused on microarchitectural aspects (e.g. cache-aware computation) to improve overall performance. BDEv allows us to accelerate this kind of developments by automating the extraction of microarchitectural metrics.

The *Microarchitectural characterization* monitor is able to collect microarchitecture-level performance data about the underlying hardware by keeping count of the events that happen in each node during the execution of a workload. An example of such events can be *LLC\_MISSES*, which obtains the number of misses in the last level cache. We have chosen Oprofile [86] as our base profiling tool, as it is able to provide hardware counter values for the entire system where it is running. Using such a performance profiling tool is less intrusive as it does not require manual instrumentation in the source code as needed when using certain APIs. The monitor included in BDEv is easy to configure by indicating the specific events to be recorded. Once the user sets these parameters by modifying the appropriate configuration files, the *Configuration manager* provides the corresponding values to the monitor.

The microarchitectural monitor operates as described in Section 3.2.5, starting the monitoring when a workload begins. The output results include the values of each event counter for each computing node and the total sum of the events occurred in the cluster during the execution. Furthermore, the summary graphs generated at the end of the experiments gather the values related to each event for each framework and workload, easing the comparison between experiments.

### 3.4. Targeted use cases

The use of BDEv 3.0 is not restricted to a single scenario. Several kinds of users can benefit from its features in a different way, including developers, system administrators and end users. This section provides more details about the usefulness of BDEv in each use case.

*Developers.* The development cycle of a Big Data framework or application generally includes several tests that serve to check the correct functioning of the solution. BDEv can automate this task by defining one or more experimental cases that process a certain dataset to get an expected result, detecting errors or timeouts during the execution. Moreover, the standard output of the workload can be analyzed to verify if the obtained result agrees with the expected one. Similarly, sometimes a component of a framework or an application is modified to optimize a certain metric, like the utilization of some resource (e.g. disk, network) or a microarchitectural metric (e.g. data cache locality). Using BDEv, the developer can: (1) identify the component to be optimized by analyzing the results from the evaluation metrics provided by BDEv; (2) once the optimized version has been implemented, BDEv can compare it with previous versions by stressing the targeted component and obtaining new metric results. For instance, developers can analyze the impact of load balancing issues on the power profile of a framework by defining several experiments that present distinct distributions, from balanced to skewed.

*System administrators.* The configuration of a Big Data framework for a specific infrastructure involves the definition of a large set of parameters. This can turn into an overwhelming task, considering that each parameter may have different possibilities that affect the behavior of the framework. In some cases, selecting an appropriate setting for a parameter involves an experimental evaluation of the different values. BDEv can ease this task by automatically evaluating those configurations, previously defined by the system administrator. As BDEv configurations can be defined separately (i.e. by using several configuration directories), the total set of possibilities can be established a priori, carrying out the experiments straightforwardly without needing user interaction. Apart from memory and CPU settings, BDEv also considers the configuration of multiple disks and the use of high performance resources like the IPoIB interface, allowing us to determine the impact of their utilization on the different metrics.

*End users.* Big Data users often come from many different research fields, sometimes without a thoughtful understanding of the insights of Big Data frameworks or the kind of workloads more suited for each one. Hence, the use of an evaluation tool like BDEv can ease the task of selecting the most suitable implementation of a workload that is available for several Big Data frameworks (e.g. K-Means for Hadoop, Spark and Flink). The selection of the framework will depend on the particular metric (e.g. performance, energy consumption) that the user may want to optimize. BDEv can also benefit those users that need to adjust the configuration parameters of a framework in order to optimize an application according to a certain metric (e.g. adjusting the HDFS block size to improve performance).

#### **4. Evaluation of Big Data processing frameworks with BDEv: Experimental configuration**

Next, we will present representative examples that demonstrate the usefulness of BDEv to extract meaningful insights from benchmarking Big Data workloads. BDEv is able to employ the new features explained in Section 3.3 along with its previous ones to assess the performance, resource utilization, energy efficiency and microarchitecture-level behavior of current Big Data frameworks. In order to show a practical example of such analysis, several experiments have been conducted in an HPC system. This section first describes the experimental testbed, including the hardware and software characteristics of the system. Then, the frameworks and workloads being evaluated are detailed in Sections 4.2 and 4.3, respectively.

##### *4.1. System configuration*

The experiments have been executed in the Grid'5000 infrastructure [97], using a 16-node cluster that corresponds with 1 master and 15 slaves. The main hardware and software characteristics of the nodes are described in Table 5. Basically, each node provides 2 Intel Haswell-based processors with 8 physical cores each (i.e. 16 cores per node), 126 GB of memory and 2 disks of 558 GB each.

##### *4.2. Evaluated frameworks*

The evaluation has focused on popular state-of-the-art in-memory processing frameworks: Spark and Flink. Hadoop, the most widespread batch processing framework, has also been analyzed to use its results as a baseline.

Table 5: Node characteristics

Hardware configuration	
CPU (Microarchitecture)	2 × Intel Xeon E5-2630 v3 (Haswell)
CPU Speed (Turbo)	2.40 GHz (3.20 GHz)
Physical/Virtual cores	16/32
Cache size L1 / L2 / L3	16 × 32 KB / 16 × 256 KB / 20 MB
Memory	126 GB DDR4 2133 MHz
Disk	2 × 558 GB HDD
Network	4 × 10 Gbps Ethernet
Software configuration	
OS version	Debian Jessie 8.5
Kernel	3.16.0-4
Java	Oracle JDK 1.8.0_121

Table 6: Configuration of the frameworks

Hadoop		Spark		Flink	
Mapper/Reducer heap size	3.4 GB	Executor heap size	109 GB	TaskManager heap size	109 GB
Mappers per node	16	Workers per node	1	TaskManagers per node	1
Reducers per node	16	Worker cores	32	TaskManager cores	32
Shuffle parallel copies	20	Default parallelism	480	Network buffers per node	20480
IO sort MB	852 MB			Parallelism	480
IO sort spill percent	80%			IO sort spill percent	80%
HDFS					
		HDFS block size	512 MB		
		Replication factor	1		

The experiments have been carried out using the last stable versions as of July 2017 (Hadoop 2.7.3, Spark 2.2.0 and Flink 1.3.2). Hence, this work also provides an up-to-date comparison that includes the latest enhancements for each framework. Their configuration has been set according to their corresponding user guides and taking into account the characteristics of the underlying system (e.g. number of CPU cores, memory size). The most important parameters of the resulting configuration are shown in Table 6. This table also shows the main parameters for HDFS, which has been used to store input and output datasets.

#### 4.3. Workloads

The workloads that have been assessed are common batch processing benchmarks selected to represent different use cases: sorting (TeraSort), machine learning (K-Means), graph analysis (Connected Components) and web search indexing (PageRank). The characteristics of the workloads are shown in Table 7, including the size of the input dataset and the data generator, as well as the origin of the source code of the benchmark for each framework. As there can be several implementations for the same algorithm and framework (e.g. PageRank), we have chosen the one that shows the best performance. Furthermore, the size of the datasets has been adjusted to keep the execution times into

Table 7: Benchmark characteristics

Benchmark	Input dataset		Implementation source		
	Size	Generator	Hadoop	Spark	Flink
TeraSort	300 GB	TeraGen	Hadoop examples	Adapted from [98]	Adapted from [98]
K-Means	39 GB (N=900M, K=5)	GenKMeansDataset	Mahout	MLlib	Adapted from example
ConComp	20 GB (30M pages)	DataGen	Pegasus	GraphX	Gelly
PageRank	20 GB (30M pages)	DataGen	Pegasus	Adapted from example	Adapted from example

reasonable ranges. Finally, the iterative benchmarks have been executed until convergence (i.e. until reaching a final solution). Further information about the workloads is hereby provided.

*TeraSort*. A standard I/O-bound benchmark that sorts 100 byte-sized key-value tuples. This workload assess the shuffle and sort capabilities of the frameworks. The reference implementation is provided by Hadoop, along with the corresponding input data generator (TeraGen). Spark and Flink do not provide any official implementation in their distributions. So, we have adapted the implementations provided in [98], which are compliant with the Hadoop one.

*K-Means*. An iterative clustering algorithm that classifies an input set of N samples into K clusters. The Apache Mahout project [9] provides the implementation for Hadoop and an input data generator (GenKMeansDataset). Spark uses its built-in machine learning library MLlib for running this algorithm, but there is no such counterpart in Flink-ML, and so its implementation has been adapted from the example provided by Flink.

*Connected Components (ConComp)*. An iterative graph algorithm that calculates the subnets of elements that are interconnected. It is included in Pegasus [99], a graph mining system for Hadoop. Both graph-oriented libraries of Spark and Flink, GraphX and Gelly, respectively, contain an implementation of this algorithm. The input dataset is generated by using the DataGen tool included in HiBench.

*PageRank*. An iterative graph algorithm that obtains a ranking of the elements of a graph, taking into account the number and quality of the links to each one. It uses the same input data generator as Connected Components. Pegasus also provides this algorithm for Hadoop, while the implementations for Spark and Flink have been adapted from their examples. Although there are implementations available in GraphX and Gelly, these versions did not improve the performance of the examples, and so they have not been used in the experiments.

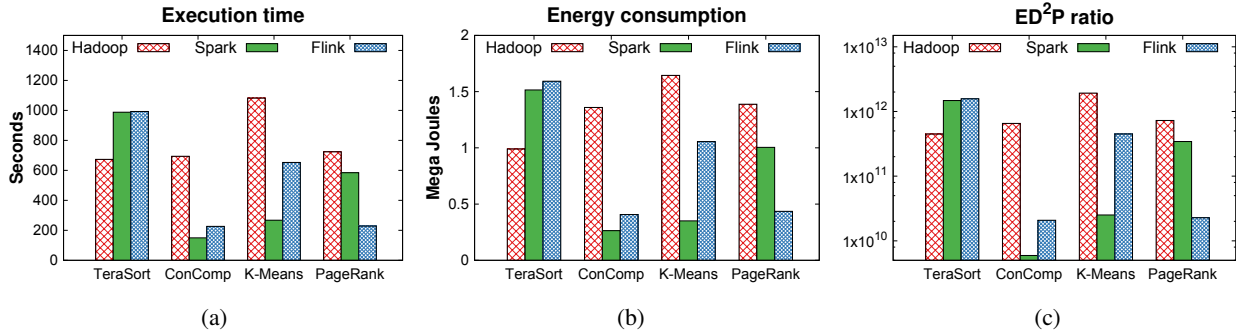


Figure 3: Execution time, energy consumption and  $ED^2P$  ratio results (lower is better)

## 5. Experimental results

Once the configuration of the experiments has been defined, the experimental results from the evaluation of Hadoop, Spark and Flink with BDEv 3.0 are presented. Section 5.1 analyzes the frameworks in terms of their execution time and energy efficiency, while Section 5.2 assesses their power consumption and resource utilization. Finally, Section 5.3 characterizes the frameworks according to their microarchitecture-level performance.

All the metrics shown in the graphs belong to the same base experiment, which corresponds to the one that obtained the median execution time among a set of 10 measurements. Note that the observed standard deviations were not significant and so they are not shown for clarity purposes. In order to ensure the same conditions for each framework and workload, the operating system buffer cache of the nodes has been erased before executing each experiment.

### 5.1. Performance and energy efficiency

Figure 3a shows the performance of the frameworks in terms of execution time, while Figure 3b presents their energy efficiency in terms of total energy consumed. Both metrics have been used to calculate the corresponding  $ED^2P$  ratio, displayed in Figure 3c using a logarithmic scale. The  $ED^2P$  metric was first proposed in [100] and indicates the performance-energy efficiency of a workload:

$$ED^2P = Energy\ Consumed \times (Execution\ Time)^2 \quad (1)$$

As can be seen, Hadoop is the best framework for TeraSort, outperforming Spark and Flink by 32% (see Figure 3a), which shows off the great data-sorting capabilities of the MapReduce model. The energy reductions provided by Hadoop are slightly higher (see Figure 3b): 35% and 38% lower energy consumption than Spark and Flink, respectively. This is explained by the fact that Hadoop shows lower CPU utilization than the other frameworks for TeraSort, thus leading to lower energy consumption (as will be analyzed in Figures 4d, 4e and 4f). As expected, in-memory frameworks clearly outperform Hadoop by a large margin when running iterative workloads. For instance, Spark is

34% faster than Flink and 78% faster than Hadoop for Connected Components, providing similar percentages of energy reduction according to Figure 3b. Regarding K-Means and PageRank, the best performers are Spark and Flink, respectively. Note also that the  $ED^2P$  metric reveals that the best performer for K-Means and PageRank provides a 10x improvement over the second one. This indicates that the results of the best framework are considered far better when energy consumption is taken into account. Finally, it is worth mentioning the variability of the energy consumption for different workloads. While the execution times of Spark and Flink are generally proportional to the energy consumed with a constant correlation, Hadoop presents higher consumption for PageRank and Connected Components even when their runtimes are similar to TeraSort. This demonstrates that Hadoop is clearly less energy efficient when running iterative workloads that are not I/O-bound.

## 5.2. Power consumption and resource utilization

As mentioned in Section 1, analyzing the usage of system resources can provide a better understanding of both the processing frameworks and the workloads, being of great interest to correlate these system metrics with power consumption, as can be done with BDEv 3.0. We have selected three workloads for this analysis: TeraSort, K-Means and PageRank. Results for Connected Components have not been included as its power profile in this system is very similar to PageRank. The power values shown in the next graphs only include the consumption of the entire package and the DRAM component (see Figure 2), as this CPU (Intel Haswell) has neither an integrated graphic card nor does it provide the separate consumption of the processing cores. For simplicity purposes, the power consumption of each node is calculated by summing the values of its two CPUs.

Figures 4, 5 and 6 present the power consumption, CPU and disk utilization for TeraSort, K-Means and PageRank, respectively. Regarding TeraSort, its I/O-bound nature can be clearly identified by analyzing the disk utilization graphs. However, the CPU graphs show isolated peak values for all the frameworks, especially at the end of the shuffle phase (around the second 200 for Hadoop and Spark, and 350 for Flink). As expected, those CPU peaks are correlated with the peak values in the power consumption graphs. It is interesting to note that Hadoop is not only the fastest framework for TeraSort, but also the one that shows a more stable power consumption, aside from two short peaks at the beginning of the execution and during the shuffle phase (see Figure 4a). Although the runtimes of Flink and Spark are very similar, they present quite different power, CPU and disk profiles. In fact, Flink shows higher CPU usages (see Figure 4f), averaging 70% CPU usage during the shuffle phase, while Hadoop and Spark never go above 30% aside from very short peaks. Hadoop presents the lowest CPU utilization, especially during the reduce phase (just after the shuffle phase), which explains its great energy savings mentioned before in the analysis of Figure 3b. Finally, it is easy to observe that Hadoop and Spark present a clear disk bottleneck during approximately the first half of the execution (see Figures 4g and 4h), while Flink also shows this bottleneck but especially during the computation of the reduce phase (see Figure 4i). This fact proves that the underlying dataflow implemented by Flink to process the input dataset is quite different to the one used by Hadoop and Spark. Note that, as mentioned in Section 2.1, Hadoop and Spark are based on batch processing engines, while Flink relies on a stream processing architecture that treats

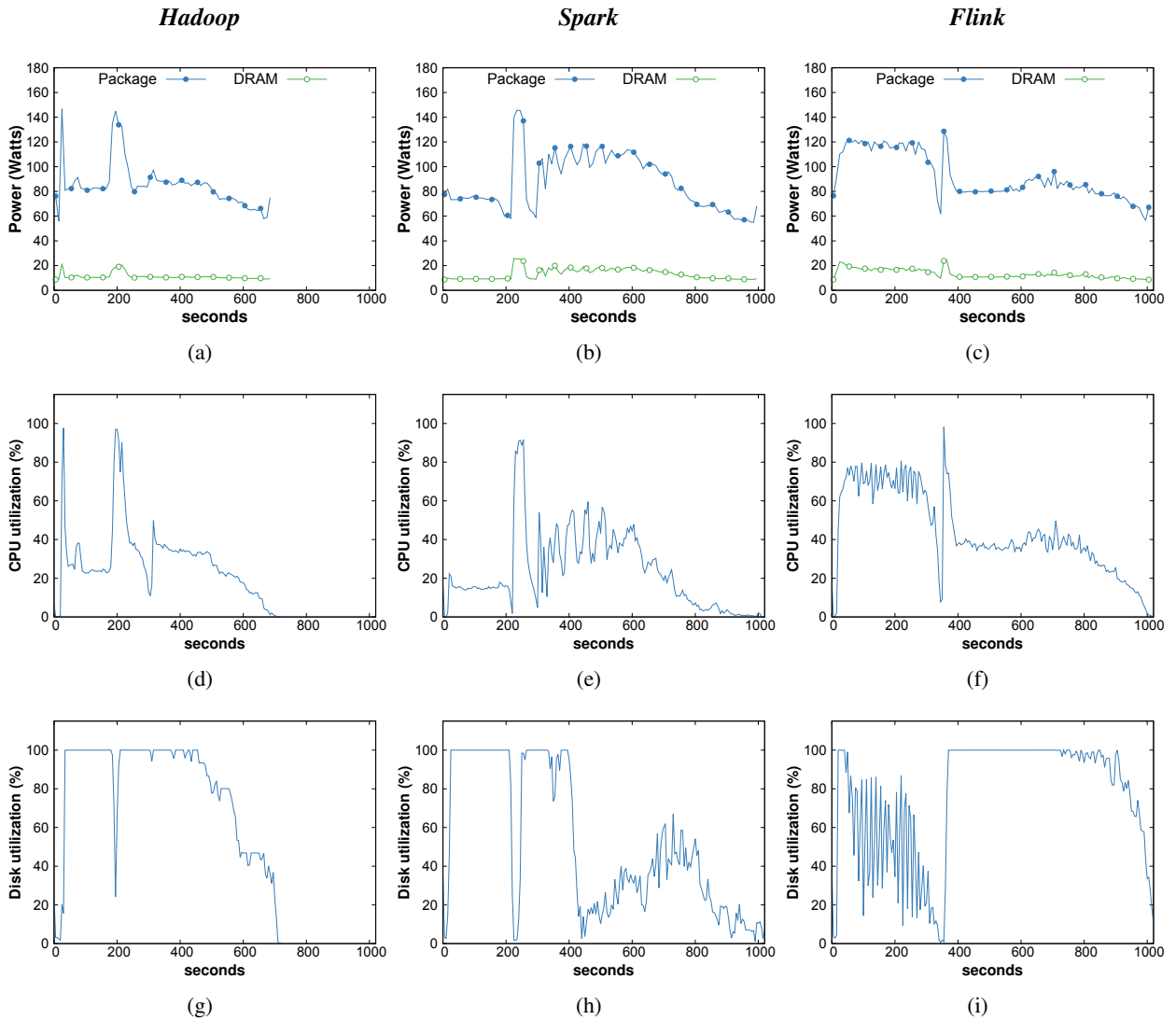


Figure 4: Average power consumption and resource utilization per node for TeraSort

batch workloads as a special case of streaming computations.

A similar analysis can be conducted for K-Means and PageRank (Figures 5 and 6). Their iterative nature can be seen in the cyclical behavior of both power consumption and CPU utilization. This is especially clear for Hadoop and Flink in K-Means (Figures 5d and 5f), and for Hadoop and Spark in PageRank (Figures 6d and 6e). Regarding the power consumption of K-Means, Spark shows considerably lower power values (below 90 watts) than Flink (see Figures 5b and 5c), which correlates with the aforementioned 10x improvement pointed out by the  $ED^2P$  ratio. The analysis of the system resource utilization illustrates that the disk access pattern for K-Means presents certain similarities across all the frameworks, showing noticeable disk I/O at the beginning and end of the execution, which mainly corresponds with reading/writing the input/output data (see Figures 5g, 5h and 5i). However, almost no disk activity is observed during the data processing, discarding any disk bottleneck in K-Means. This is mostly due to

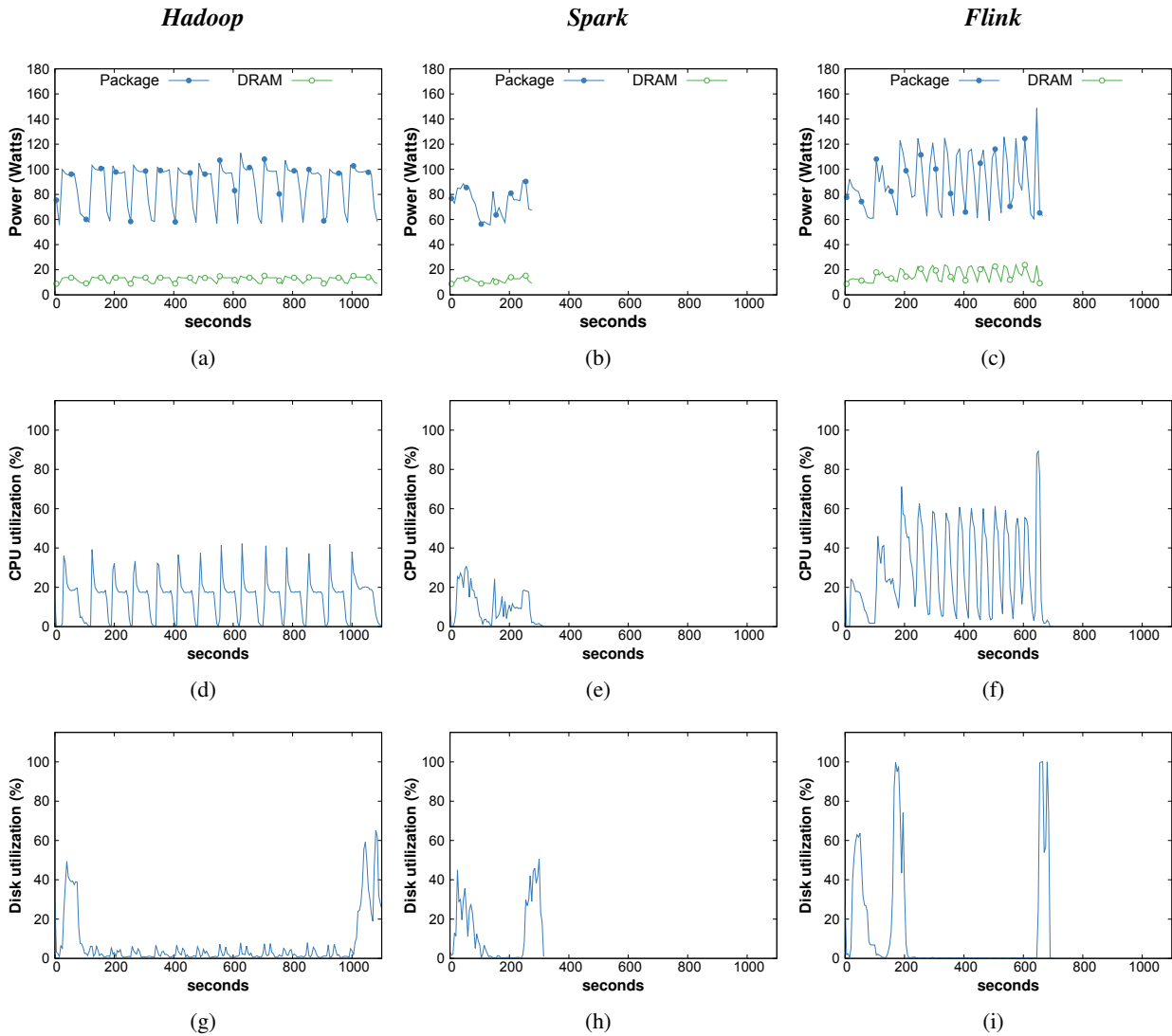


Figure 5: Average power consumption and resource utilization per node for K-Means

the strong iterative nature of this algorithm and the caching of intermediate results. Spark and Flink perform this caching by using in-memory data structures, while Hadoop takes advantage of the underlying buffer cache provided by the operating system that stores the data before writing to disk. In this case, it seems that the amount of data being processed can be cached entirely in memory. Regarding PageRank (Figure 6), Hadoop and Spark present high to medium disk utilization, respectively, as shown in Figures 6g and 6h. This is especially relevant when compared with Flink, which shows nearly zero disk I/O except an initial short peak to read the input dataset (see Figure 6i). This means that the PageRank implementations of Hadoop and Spark are generating a higher amount of intermediate data than the Flink counterpart, thus causing higher disk I/O and runtime. Nevertheless, it can be concluded that no disk bottleneck occurs for Spark and Flink. Finally, Hadoop is not only the worst performer when running PageRank, but also presents the highest power consumption, CPU and disk utilization by far. This confirms Hadoop as the



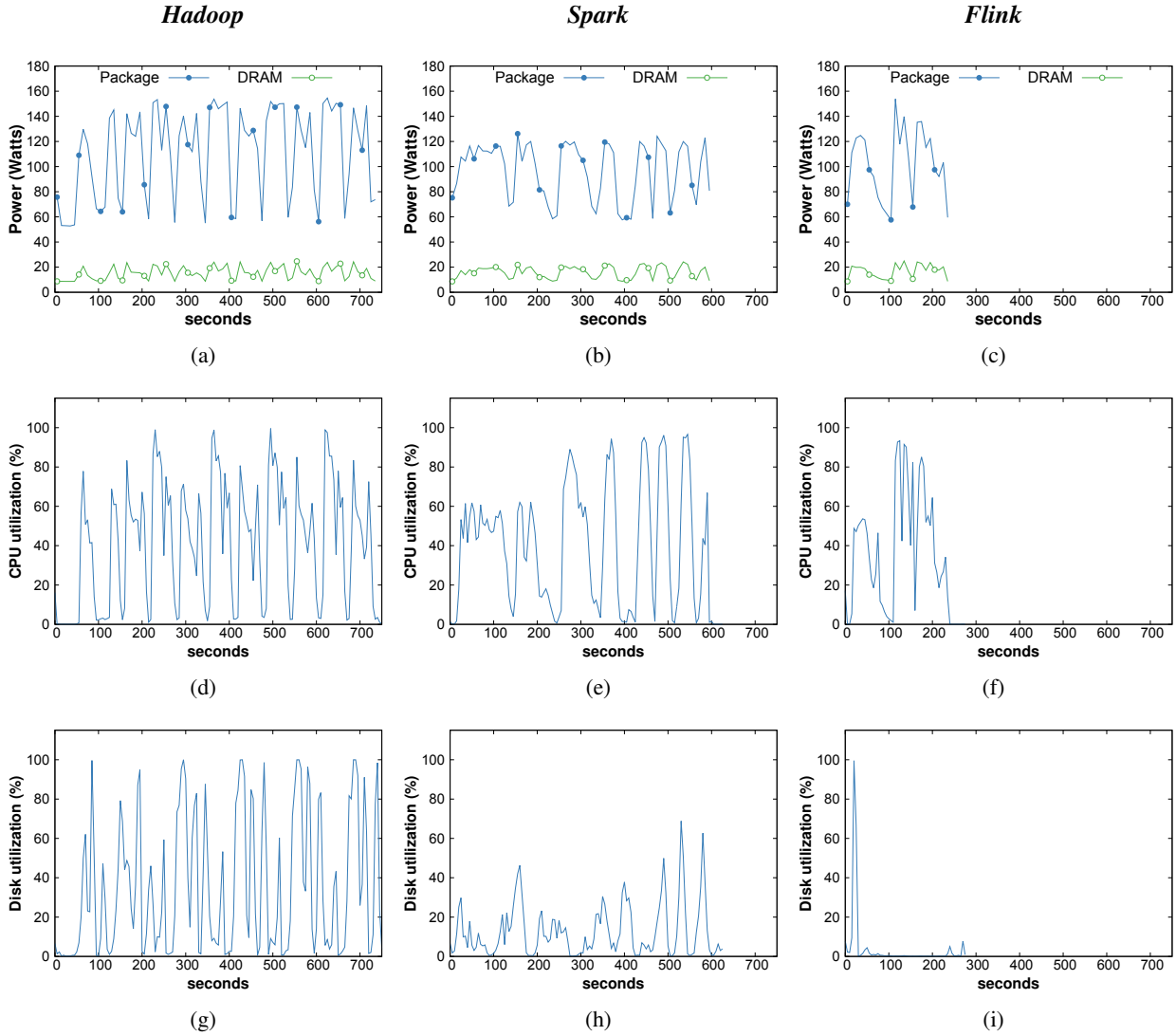


Figure 6: Average power consumption and resource utilization per node for PageRank

least energy efficient framework when executing iterative workloads, as mentioned before when analyzing the energy results of Figure 3b.

### 5.3. Microarchitecture-level metrics

The microarchitectural characterization has been carried out by measuring 10 hardware performance counters to calculate 5 different microarchitecture-level metrics: Instructions Per Cycle (IPC), branch misprediction ratio and L1/L2/L3 miss ratios. The L1 total miss ratio is not supported by this CPU, and thus the L1 miss ratio only refers to load instructions. Figure 7 displays the results obtained for these metrics, calculated over the sum of the values recorded in all nodes of the cluster.

Regarding IPC (see Figure 7a), Hadoop outperforms both Flink and Spark for all the workloads, while Spark

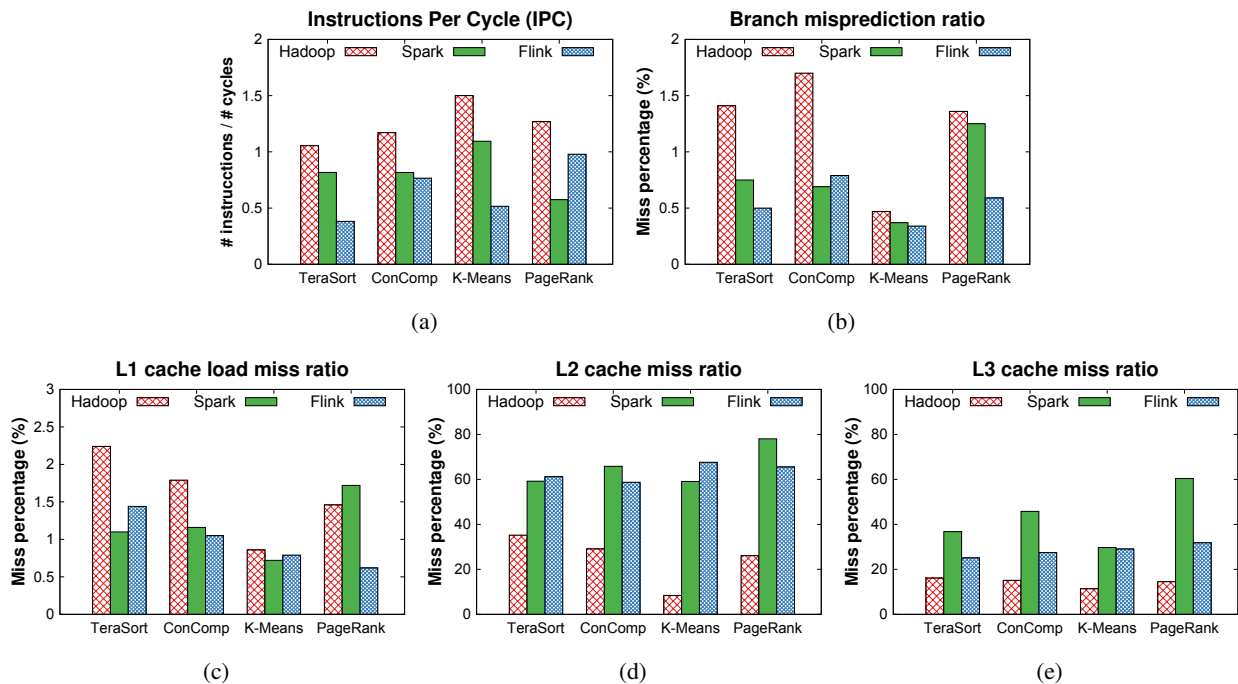


Figure 7: Microarchitecture-level metrics results

obtains higher values than Flink except for PageRank. The higher IPC obtained by Hadoop does not mean that it is actually performing better, as it only translates into more work being done per clock cycle. It is important to note that the efficiency indicated by the IPC metric may not be directly related with the execution time, as the high-level source code used to implement the frameworks (Java or Scala depending on the framework) and the workload itself are quite different (i.e. the machine code executed in the end by the CPU can vary completely). For the rest of metrics, lower values generally translate into better performance. The branch misprediction ratios displayed in Figure 7b show that Hadoop presents higher values than Spark and Flink. In this case, a higher value can directly affect performance by decreasing the throughput of the CPU pipeline, as much more useless instructions are first being issued to the execution pipeline and then discarded when the branch prediction turns out to be wrong. Flink and Spark do not present any prevalence one over the other, aside from the PageRank implementation of Flink, that shows a noticeable lower value than Spark (highly correlated with the execution times shown in Figure 3a). Although a higher branch misprediction ratio may initially be related to a higher IPC value due to the additional instructions that are executed, the IPC shown in the graph is actually calculated using the INS\_RETIRED event. This event only counts those instructions that end up being needed by the program flow. So, all those other instructions executed by the CPU in a speculative way are not taken into account, such as the ones that are issued when predicting a branch.

The miss ratios for the three cache levels available in this system (L1, L2 and L3) are shown in Figures 7c, 7d and 7e, respectively. For the first level, there is no clear difference between the frameworks, as this cache turns out to be highly efficient obtaining very low ratios overall (below 2.5%). As data is generally read in a sequential way

when processed by these frameworks, most of these accesses hit the L1 cache. For the L2 cache, Hadoop presents lower miss ratios than Spark and Flink, with a wide difference in K-Means (86% and 88% lower, respectively). For Spark and Flink there is no clear prevalent solution, as results are rather similar and depend on the workload. The higher L2 miss ratios of Spark and Flink means that, although they have shorter execution times for all the workloads except for TeraSort, their memory accesses seem to present less data locality. The same behavior can be observed for the L3 cache, as Hadoop shows the lowest values. Furthermore, both Hadoop and Flink have rather constant L3 miss ratios across workloads, with values under 20% and 30%, respectively. Meanwhile, Spark shows higher and more variable ratios, ranging from 30% to 60%. PageRank presents the worst scenario with a considerably higher ratio than Hadoop and Flink. Note that this may be related to the PageRank execution time obtained by Spark, which is significantly worse than Flink as shown in Figure 3a. The low variability of the L3 miss ratios of Hadoop and Flink can be explained by the larger size of this last cache level compared with the other ones. This fact mitigates any difference that the implementation of the workload may present for each framework.

The analysis performed in this section has shown the utility of using microarchitecture-level metrics to evaluate the execution of Big Data workloads. Other scenarios where the use of these metrics can be useful are: 1) comparing the behavior of different implementations of the same algorithm using a particular Big Data framework; 2) analyzing the impact of different configuration parameters of a framework; and 3) improving a specific microarchitectural metric when optimizing applications and frameworks.

## 6. Conclusions and future work

Current Big Data applications are becoming increasingly larger and even more resource consuming. However, the performance metrics generally used to characterize Big Data processing frameworks, such as total execution time or average CPU utilization, are often simplistic. Although these metrics can be useful to provide a broad picture, the large scale of Big Data applications is demanding the consideration of more insightful metrics, like energy consumption or low-level microarchitecture events, to adequately assess the performance of both the frameworks and workloads. However, only some recent state-of-the-art evaluations focus on energy efficiency, while very little work takes into account microarchitecture-level aspects. Furthermore, none of them provides reliable, portable and publicly available tools to carry out the evaluations on different systems.

This work has first presented an overview of the state of the art regarding the evaluation of distributed data processing frameworks for Big Data analytics, highlighting the need of more advanced evaluation tools to enable the assessment of energy efficiency and microarchitecture-level characterization. Next, we have extended the design of our Big Data Evaluator (BDEv) tool to allow the utilization of such metrics in multiple use cases, made by supporting Intel RAPL measurements and Oprofile-based monitoring. As these utilities are available in a wide range of systems, the portability of the tool is ensured. Finally, BDEv has been used to evaluate representative data processing frameworks (Hadoop, Spark and Flink) in terms of performance, resource utilization, energy efficiency and CPU events.

The analysis of the results provided by BDEv demonstrate its usefulness to fully understand the behavior of these frameworks when processing large datasets.

The presented extensions of BDEv turn it into a holistic evaluation tool that is publicly available at <http://bdev.des.udc.es>. Future work looks forward to further extending the energy measurements to consider other resources such as disk and network. We also plan to include support for other data processing paradigms, like streaming computations.

## Acknowledgments

This work was fully supported by the Ministry of Economy, Industry and Competitiveness of Spain (Project TIN2016-75845-P, AEI/FEDER/EU) and by the FPU Program of the Ministry of Education (grants FPU14/02805 and FPU15/03381). The experiments presented in this paper were carried out using the Grid'5000 testbed, supported by Inria, CNRS, RENATER and several French Universities.

## References

- [1] Apache Hadoop, <http://hadoop.apache.org/>, [Last visited: February 2018].
- [2] M. Zaharia, et al., Apache Spark: a unified engine for Big Data processing, *Communications of the ACM* 59 (11) (2016) 56–65.
- [3] P. X. Gao, A. R. Curtis, B. Wong, S. Keshav, It's not easy being green, *ACM SIGCOMM Computer Communication Review* 42 (4) (2012) 211–222.
- [4] A. Fernández, et al., Big Data with cloud computing: an insight on the computing environment, MapReduce, and programming frameworks, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4 (5) (2014) 380–409.
- [5] X. Jin, B. W. Wah, X. Cheng, Y. Wang, Significance and challenges of Big Data research, *Big Data Research* 2 (2) (2015) 59–64.
- [6] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, *Communications of the ACM* 51 (1) (2008) 107–113.
- [7] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop Distributed File System, in: *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'2010)*, Incline Village, NV, USA, 2010, pp. 1–10.
- [8] V. K. Vavilapalli, et al., Apache Hadoop YARN: yet another resource negotiator, in: *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC'13)*, Santa Clara, CA, USA, 2013, pp. 5:1–5:16.
- [9] Apache Mahout: scalable machine learning and data mining, <http://mahout.apache.org/>, [Last visited: February 2018].
- [10] C. Avery, Giraph: large-scale graph processing infrastructure on Hadoop, in: *Proceedings of the 2011 Hadoop Summit*, Vol. 11, Santa Clara, CA, USA, 2011, pp. 5–9.
- [11] Apache HBase: Hadoop distributed Big Data store, <https://hbase.apache.org/>, [Last visited: February 2018].
- [12] M. Wasi-Ur-Rahman, et al., High-performance RDMA-based design of Hadoop MapReduce over InfiniBand, in: *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW'13)*, Boston, MA, USA, 2013, pp. 1908–1917.
- [13] J. Veiga, R. R. Expósito, G. L. Taboada, J. Touriño, Flame-MR: an event-driven architecture for MapReduce applications, *Future Generation Computer Systems* 65 (2016) 46–56.
- [14] M. H. Iqbal, T. R. Soomro, Big Data analysis: Apache Storm perspective, *International Journal of Computer Trends and Technology* 19 (1) (2015) 9–14.
- [15] S. Kulkarni, et al., Twitter Heron: stream processing at scale, in: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, Melbourne, Australia, 2015, pp. 239–250.

- [16] S. A. Noghahi, et al., Samza: stateful scalable stream processing at LinkedIn, *Proceedings of the VLDB Endowment* 10 (12) (2017) 1634–1645.
- [17] J. Kreps, et al., Kafka: a distributed messaging system for log processing, in: *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB II)*, Athens, Greece, 2011, pp. 1–7.
- [18] Apache Cassandra: distributed NoSQL database, <http://cassandra.apache.org/>, [Last visited: February 2018].
- [19] M. Rostanski, K. Grochla, A. Seman, Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ, in: *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS'14)*, Warsaw, Poland, 2014, pp. 879–884.
- [20] P. Hintjens, *ZeroMQ: messaging for many applications*, O'Reilly Media, Inc., 2013.
- [21] B. Hindman, et al., Mesos: a platform for fine-grained resource sharing in the data center, in: *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*, Boston, MA, USA, 2011, pp. 295–308.
- [22] Apache Aurora, Mesos framework for long-running services and cron jobs, <http://aurora.apache.org/>, [Last visited: February 2018].
- [23] D. Merkel, Docker: lightweight Linux containers for consistent development and deployment, *Linux Journal* (239) (2014) 76–91.
- [24] Apache Flink: scalable batch and stream data processing, <http://flink.apache.org/>, [Last visited: February 2018].
- [25] Apache Apex: enterprise-grade unified stream and batch processing engine, <https://apex.apache.org/>, [Last visited: February 2018].
- [26] M. Zaharia, et al., Resilient Distributed Datasets: a fault-tolerant abstraction for in-memory cluster computing, in: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, San Jose, CA, USA, 2012, pp. 15–28.
- [27] Spark MLlib: Apache Spark's scalable machine learning library, <http://spark.apache.org/mllib/>, [Last visited: February 2018].
- [28] Spark GraphX: Apache Spark's API for graphs and graph-parallel computation, <http://spark.apache.org/graphx/>, [Last visited: February 2018].
- [29] A. Alexandrov, et al., The Stratosphere platform for Big Data analytics, *The VLDB Journal* 23 (6) (2014) 939–964.
- [30] FlinkML: machine learning library for Flink, <https://github.com/apache/flink/tree/master/flink-libraries/flink-ml>, [Last visited: February 2018].
- [31] Flink Gelly: Apache Flink's graph-processing API and library, <https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/libs/gelly/index.html>, [Last visited: February 2018].
- [32] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang, The HiBench benchmark suite: characterization of the MapReduce-based data analysis, in: *Proceedings of the 26th IEEE International Conference on Data Engineering Workshops (ICDEW'10)*, Long Beach, CA, USA, 2010, pp. 41–51.
- [33] L. Wang, et al., BigDataBench: a Big Data benchmark suite from Internet services, in: *Proceedings of the 20th IEEE International Symposium on High-Performance Computer Architecture (HPCA'14)*, Orlando, FL, USA, 2014, pp. 488–499.
- [34] Z. Jia, L. Wang, J. Zhan, L. Zhang, C. Luo, Characterizing data analysis workloads in data centers, in: *Proceedings of the 2013 IEEE International Symposium on Workload Characterization (IISWC'13)*, Portland, OR, USA, 2013, pp. 66–76.
- [35] C. Luo, et al., Cloudrank-D: benchmarking and ranking cloud computing systems for data processing applications, *Frontiers of Computer Science* 6 (4) (2012) 347–362.
- [36] A. Sangroya, D. Serrano, S. Bouchenak, MRBS: towards dependability benchmarking for Hadoop MapReduce, in: *Proceedings of the 18th International Euro-Par Conference on Parallel Processing Workshops (Euro-Par'12)*, Rhodes Island, Greece, 2012, pp. 3–12.
- [37] GridMix: a benchmark for Hadoop clusters, <https://hadoop.apache.org/docs/stable1/gridmix.html>, [Last visited: February 2018].
- [38] PigMix: queries for testing Pig performance, <https://cwiki.apache.org/confluence/display/PIG/PigMix>, [Last visited: February 2018].
- [39] K. Kim, et al., MRBench: a benchmark for MapReduce framework, in: *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, Melbourne, Australia, 2008, pp. 11–18.
- [40] A. Ghazal, 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 (SIGMOD'13), New York, NY, USA, 2013, pp. 1197–1208.

- [41] M. Li, J. Tan, Y. Wang, L. Zhang, V. Salapura, SparkBench: a Spark benchmarking suite characterizing large-scale in-memory data analytics, *Cluster Computing* 20 (3) (2017) 2575–2589.
- [42] S. Bergamaschi, L. Gagliardelli, G. Simonini, S. Zhu, BigBench workload executed by using Apache Flink, *Procedia Manufacturing* 11 (2017) 695–702.
- [43] A. Thusoo, et al., Hive - a Petabyte scale data warehouse using Hadoop, in: *Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE 2010)*, Long Beach, CA, USA, 2010, pp. 996–1005.
- [44] AMPLab: data warehouse benchmark, <https://amplab.cs.berkeley.edu/benchmark/>, [Last visited: February 2018].
- [45] B. Saha, et al., Apache Tez: a unifying framework for modeling and building data processing applications, in: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, Melbourne, Australia, 2015, pp. 1357–1369.
- [46] R. S. Xin, et al., Shark: SQL and rich analytics at scale, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, New York, NY, USA, 2013, pp. 13–24.
- [47] M. Kornacker, et al., Impala: a modern, open-source SQL engine for Hadoop, in: *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR'15)*, Asilomar, CA, USA, 2015, pp. 28:1–28:10.
- [48] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: *Proceedings of the 1st ACM Symposium on Cloud computing (SoCC'10)*, Indianapolis, IN, USA, 2010, pp. 143–154.
- [49] MongoDB: documented-oriented NoSQL database, <https://www.mongodb.com/>, [Last visited: February 2018].
- [50] Redis: open-source in-memory database, <https://redis.io/>, [Last visited: February 2018].
- [51] B. Fitzpatrick, Distributed caching with Memcached, *Linux Journal* (124) (2004) 72–76.
- [52] BDEv: Big Data Evaluator tool, <http://bdev.des.udc.es>, [Last visited: February 2018].
- [53] J. Veiga, R. R. Expósito, G. L. Taboada, J. Touriño, MREv: an automatic MapReduce Evaluation tool for Big Data workloads, in: *Proceedings of the International Conference on Computational Science (ICCS'15)*, Reykjavík, Iceland, 2015, pp. 80–89.
- [54] J. Veiga, R. R. Expósito, G. L. Taboada, J. Touriño, Analysis and evaluation of MapReduce solutions on an HPC cluster, *Computers & Electrical Engineering* 50 (2016) 200–216.
- [55] Z. Fadika, M. Govindaraju, R. Canon, L. Ramakrishnan, Evaluating Hadoop for data-intensive scientific operations, in: *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, Honolulu, HI, USA, 2012, pp. 67–74.
- [56] E. Feller, L. Ramakrishnan, C. Morin, On the performance and energy efficiency of Hadoop deployment models, in: *Proceedings of the 2013 IEEE International Conference on Big Data (IEEE BigData 2013)*, Santa Clara, CA, USA, 2013, pp. 131–136.
- [57] E. Dede, Z. Fadika, M. Govindaraju, L. Ramakrishnan, Benchmarking MapReduce implementations under different application scenarios, *Future Generation Computer Systems* 36 (2014) 389–399.
- [58] J. Shi, et al., Clash of the titans: MapReduce vs. Spark for large scale data analytics, *Proceedings of the VLDB Endowment* 8 (13) (2015) 2110–2121.
- [59] P. Jakovits, S. N. Srirama, Evaluating MapReduce frameworks for iterative scientific computing applications, in: *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS'14)*, Bologna, Italy, 2014, pp. 226–233.
- [60] J. Ekanayake, et al., Twister: a runtime for iterative MapReduce, in: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'2010)*, Chicago, IL, USA, 2010, pp. 810–818.
- [61] Y. Bu, B. Howe, M. Balazinska, M. D. Ernst, HaLoop: efficient iterative data processing on large clusters, *Proceedings of the VLDB Endowment* 3 (1) (2010) 285–296.
- [62] P. González, et al., Using the cloud for parameter estimation problems: comparing Spark vs MPI with a case-study, in: *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2017)*, Madrid, Spain, 2017, pp. 797–806.
- [63] N. Spangenberg, M. Roth, B. Franczyk, Evaluating new approaches of Big Data analytics frameworks, in: *Proceedings of the 18th International Conference on Business Information Systems (BIS'15)*, Poznań, Poland, 2015, pp. 28–37.
- [64] M. Bertoni, S. Ceri, A. Kaitoua, P. Pinoli, Evaluating cloud frameworks on genomic applications, in: *Proceedings of the 2015 IEEE*

International Conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, 2015, pp. 193–202.

- [65] O.-C. Marcu, A. Costan, G. Antoniu, M. S. Pérez-Hernández, Spark versus Flink: understanding performance in Big Data analytics frameworks, in: Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER'16), Taipei, Taiwan, 2016, pp. 433–442.
- [66] J. Veiga, R. R. Expósito, X. C. Pardo, G. L. Taboada, J. Touriño, Performance evaluation of Big Data frameworks for large-scale data analytics, in: Proceedings of the 2016 IEEE International Conference on Big Data (IEEE BigData 2016), Washington, DC, USA, 2016, pp. 424–431.
- [67] C. Boden, A. Spina, T. Rabl, V. Markl, Benchmarking data flow systems for scalable machine learning, in: Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR'17), Chicago, IL, USA, 2017, pp. 5:1–5:10.
- [68] J. Samosir, M. Indrawan-Santiago, P. D. Haghighi, An evaluation of data stream processing systems for data driven applications, in: Proceedings of the International Conference on Computational Science (ICCS'16), San Diego, CA, USA, 2016, pp. 439–449.
- [69] S. Chintapalli, et al., Benchmarking streaming computation engines: Storm, Flink and Spark streaming, in: Proceedings of the 1st IEEE Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM'16), Chicago, IL, USA, 2016, pp. 1789–1792.
- [70] S. Qian, G. Wu, J. Huang, T. Das, Benchmarking modern distributed streaming platforms, in: Proceedings of the 2016 IEEE International Conference on Industrial Technology (ICIT 2016), Taipei, Taiwan, 2016, pp. 592–598.
- [71] C. Negru, M. Mocanu, V. Cristea, S. Sotiriadis, N. Bessis, Analysis of power consumption in heterogeneous virtual machine environments, *Soft Computing* 21 (16) (2017) 4531–4542.
- [72] D. Cheng, P. Lama, C. Jiang, X. Zhou, Towards energy efficiency in heterogeneous Hadoop clusters by adaptive task assignment, in: Proceedings of the 35th IEEE International Conference on Distributed Computing Systems (ICDCS'15), Columbus, OH, USA, 2015, pp. 359–368.
- [73] C. Mobius, W. Dargie, A. Schill, Power consumption estimation models for processors, virtual machines, and servers, *IEEE Transactions on Parallel and Distributed Systems* 25 (6) (2014) 1600–1614.
- [74] E. Feller, L. Ramakrishnan, C. Morin, Performance and energy efficiency of Big Data applications in cloud environments: a Hadoop case study, *Journal of Parallel and Distributed Computing* 79 (2015) 80–89.
- [75] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, W. Shi, Energy-aware scheduling of MapReduce jobs for Big Data applications, *IEEE Transactions on Parallel and Distributed Systems* 26 (10) (2015) 2720–2733.
- [76] B. Feng, J. Lu, Y. Zhou, N. Yang, Energy efficiency for MapReduce workloads: an in-depth study, in: Proceedings of the 23rd Australasian Database Conference (ADC'12), Melbourne, Australia, 2012, pp. 61–70.
- [77] D. Loghin, B. M. Tudor, H. Zhang, B. C. Ooi, Y. M. Teo, A performance study of Big Data on small nodes, *Proceedings of the VLDB Endowment* 8 (7) (2015) 762–773.
- [78] M. Malik, S. Rafatirah, A. Sasan, H. Homayoun, System and architecture level characterization of Big Data applications on big and little core server architectures, in: Proceedings of the 2015 IEEE International Conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, 2015, pp. 85–94.
- [79] E. Baccarelli, et al., Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for Big Data stream mobile computing: review, challenges, and a case study, *IEEE Network* 30 (2) (2016) 54–61.
- [80] N. Tiwari, S. Sarkar, U. Bellur, M. Indrawan, An empirical study of Hadoop's energy efficiency on a HPC cluster, in: Proceedings of the International Conference on Computational Science (ICCS'14), Cairns, Australia, 2014, pp. 62–72.
- [81] H. David, E. Gorbato, U. R. Hanebutte, R. Khanna, C. Le, RAPL: memory power estimation and capping, in: Proceedings of the 2010 International Symposium on Low-Power Electronics and Design (ISLPED'10), Austin, TX, USA, 2010, pp. 189–194.
- [82] S. Desrochers, C. Paradis, V. M. Weaver, A validation of DRAM RAPL power measurements, in: Proceedings of the 2nd International Symposium on Memory Systems (MEMSYS'16), Alexandria, VA, USA, 2016, pp. 455–470.
- [83] K. N. Khan, M. A. Hoque, T. Niemi, Z. Ou, J. K. Nurminen, Energy efficiency of large scale graph processing platforms, in: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016), Heidelberg, Germany, 2016, pp.

1287–1294.

- [84] T. Yoo, M. Yim, I. Jeong, Y. Lee, S.-T. Chun, Performance evaluation of in-memory computing on scale-up and scale-out cluster, in: Proceedings of the 8th International Conference on Ubiquitous and Future Networks (ICUFN 2016), Vienna, Austria, 2016, pp. 456–461.
- [85] W. Xiong, et al., A characterization of Big Data benchmarks, in: Proceedings of the 2013 IEEE International Conference on Big Data (IEEE BigData 2013), Silicon Valley, CA, USA, 2013, pp. 118–125.
- [86] Oprofile: a system profiler for Linux, <http://oprofile.sourceforge.net>, [Last visited: February 2018].
- [87] Z. Jia, et al., Characterizing and subsetting Big Data workloads, in: Proceedings of the 2014 IEEE International Symposium on Workload Characterization (IISWC'14), Raleigh, NC, USA, 2014, pp. 191–201.
- [88] V. M. Weaver, Linux perf\_event features and overhead, in: Proceedings of the 2nd International Workshop on Performance Analysis of Workload Optimized Systems (FastPath'13), Austin, TX, USA, 2013.
- [89] A. J. Awan, M. Brorsson, V. Vlassov, E. Ayguadé, Performance characterization of in-memory data analytics on a modern cloud server, in: Proceedings of the 5th IEEE International Conference on Big Data and Cloud Computing (BDCloud 2015), Dalian, China, 2015, pp. 1–8.
- [90] Y. Yu, W. Wang, J. Zhang, K. B. Letaief, LERC: coordinated cache management for data-parallel systems, arXiv preprint arXiv:1708.07941 (2017).
- [91] Y. Wang, X. Que, W. Yu, D. Goldenberg, D. Sehgal, Hadoop acceleration through network levitated merge, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), Seattle, WA, USA, 2011, pp. 57:1–57:10.
- [92] X. Lu, D. Shankar, S. Gugnani, D. K. Panda, High-performance design of Apache Spark with RDMA and its benefits on various workloads, in: Proceedings of the 2016 IEEE International Conference on Big Data (IEEE BigData 2016), Washington, DC, USA, 2016, pp. 253–262.
- [93] X. Lu, F. Liang, B. Wang, L. Zha, Z. Xu, DataMPI: extending MPI to Hadoop-like Big Data computing, in: Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS'14), Phoenix, AZ, USA, 2014, pp. 829–838.
- [94] RAPL read tool, <https://github.com/LPD-EPFL/raplread>, [Last visited: February 2018].
- [95] V. M. Weaver, et al., Measuring energy and power with PAPI, in: Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW'12), Pittsburgh, PA, USA, 2012, pp. 262–268.
- [96] Project Tungsten, <https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html>, [Last visited: February 2018].
- [97] Grid'5000: large-scale resource provisioning network, <https://www.grid5000.fr>, [Last visited: February 2018].
- [98] TeraSort for Apache Spark and Flink, <https://github.com/eastcirclek/terasort>, [Last visited: February 2018].
- [99] U. Kang, C. E. Tsourakakis, C. Faloutsos, PEGASUS: a peta-scale graph mining system - implementation and observations, in: Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09), Miami, FL, USA, 2009, pp. 229–238.
- [100] A. J. Martin, Towards an energy complexity of computation, Information Processing Letters 77 (2–4) (2001) 181–187.