Benchmarking Big Data Applications: A Review

Sayaka Akioka
School of Interdisciplinary Mathematical Sciences
Meiji University
Tokyo, Japan 164–8525
Email: akioka@meiji.ac.jp

***Abstract*—Big data applications have become one of the non-negligible applications in recent years. These big data applications are supposed to investigate gigantic amount of data from various data sources from several points of view, uncover new findings, and then deliver totally new values. As big data applications handle extremely huge amount of data compared with conventional applications, there is a high, and increasing demand for the computational environment, which accelerates and scales out big data applications. The serious problem here, however, is that the behaviors, or characteristics of big data applications are not clearly defined yet. The appropriate modeling, and benchmarking are indispensable for the development, or design of the adequate computational environment targeting on big data applications. This paper primarily intends to provide a comprehensive survey on modeling, and benchmarking big data applications. We overview, and compare HiBench Benchmark Suite, CloudSuite, and BigDataBench as existing benchmarks for big data applications. We also introduce a couple of projects on modeling big data applications as well.**

Keywords*—*big data; benchmark; stream mining.

I. INTRODUCTION

Big data applications have become one of the non-negligible applications in recent years. These big data applications are supposed to investigate gigantic amount of data from various data sources from several points of view, uncover new findings, and then deliver totally new values. As big data applications handle extremely huge amount of data compared with conventional applications, there is a high, and increasing demand for the computational environment, which accelerates and scales out big data applications. The serious problem here, however, is that the behaviors, or characteristics of big data applications are not clearly defined yet. There is no established model for big data applications right now.

High performance computing community has been investigating data intensive applications, which analyze huge amount of data as well. Raicu et al. pointed out that data intensive applications, and big data applications are fundamentally different from the viewpoint of data access patterns [1]. Therefore, the strategies for speed-up of data intensive applications, and big data applications have to be radically different. Many data intensive applications often reuse input data, and the primary strategy of the speed-up is locating the data close to the target CPUs. Big data applications, however, rarely reuse input data, and this strategy for data intensive applications does not work in many cases. Modern computational environment has been and is evolving mainly for speed-up of benchmarks, such as LINPACK [2], or SPEC [3]. These benchmarks are relatively scalable according to the number of CPUs. Big data applications are not scalable to the contrary, and the current computational environment is not necessarily ideal for big data applications.

This paper primarily intends to provide a comprehensive survey on modeling, and benchmarking big data applications. The appropriate modeling, and benchmarking are indispensable for the development, or design of the adequate computational environment targeting on big data applications. The rest of this paper is organized as follows. Section 2 introduces major benchmark suites for big data applications. Section 3 provides brief review on modeling of a stream mining application, which is one kind of the big data applications. In Section 4, we discuss the current status, and possible future directions on modeling, and benchmarking for big data applications. Section 5 concludes this paper.

II. BENCHMARK SUITE

This section reviews big data benchmark suites in chronological order according to publication dates of the corresponding benchmark suites.

A. HiBench Benchmark Suite

Huang et al. proposed HiBench benchmark suite in 2010 [4]. Huang et al. designed HiBench as a realistic, and comprehensive benchmark suite for Hadoop [5], which is one of the most popular implementations of MapReduce model [6]. HiBench includes both synthetic micro benchmarks, and real-world applications. Before HiBench, GridMix [7], Hive performance benchmarks [8], TeraSort [9], or DFSIO, which is contained in Hadoop source code for benchmarking Hadoop file systems, is the primary option for Hadoop benchmark. Although a benchmark suite is supposed to cover diverse characteristics of target applications, however, Huang et al. pointed out that none of the benchmark suites before HiBench fulfills the requirement for the three reasons;

1) Those benchmark suites require less computations compared to real world Hadoop applications.
2) The data access patterns modeled by those benchmark suites do not assume data access outside MapReduce, such as temporary files on local disks.
3) Some of those benchmark suites focus on more traditional data analysis containing random access using index, or join using partitioning key.

HiBench consists of three micro benchmarks, two Web search related tasks, two machine learning implementations, and one benchmark for Hadoop filesystem (HDFS Benchmark). Table I lists names of the benchmarks. Here, the three micro benchmarks, which are Sort, WordCount, and TeraSort,

 ISBN: 978-1-61208-461-9

TABLE I. CONTENTS OF HIBENCH

Micro Benchmarks	Sort WordCount TeraSort
Web Search	Nutch Indexing PageRank
Machine Learning	Bayesian Classification K-means Clustering
HDFS Benchmark	EnhancedDFSIO

TABLE II. CONTENTS OF CLOUDSUITE

Scale-out Workloads	
Data Serving	Cassandra
MapReduce	Bayesian Classification
Media Streaming	Darwin Streaming Server
SAT Solver	Klee SAT Solver
Web Frontend	a frontend of a web-based social event calendar
Web Search	Nutch
Tradisional Workloads	
CPU	PARSEC, SPEC CINT2006
memory	PARSEC, SPEC CINT2006
Web Frontend	SPECweb09
DBMS	TPC-C, TPC-E
Web Backend	MySQL

are from Hadoop distribution, and these micro benchmarks are widely used for years. In Web search workloads, Nutch Indexing is from Nutch open source search engine [10], and PageRank is a PageRank algorithm implementation [11] from SmartFrog [12]. Two of machine learning workloads are from Mahout [13], which is an open source machine learning library for Hadoop. Bayesian Classification is an implementation of the trainer of Nave Bayes [14]. K-means Clustering is an implementation of K-means clustering algorithm [15]. EnhancedDFSIO captures fine-grained transitions of throughputs of Hadoop file system, while the original DFSIO provides only average numbers.

B. CloudSuite

Ferdman et al. proposed CloudSuite, which is a benchmark suite for big data applications [16]. Table II lists names of workloads included in CloudSuite.

CloudSuite consists of scale-out workloads, and traditional workloads. The scale-out workloads consists of workloads assuming big data applications; Data Serving, MapReduce, Media Streaming, SAT Solver, Web Frontend, and Web Search. Data Serving workload is for NoSQL systems, which is widely utilized popular web applications as the backing store. Cassandra [17] is one of the major implementations of NoSQL. MapReduce workload is for MapReduce model applications, and the implementation is from Bayesian classification from Mahout, which is the same for Bayesian Classification workload in HiBench. Media Streaming workload intends streaming services such as YouTube. Darwin Streaming Server is one of the open media streaming servers [18]. SAT Solver workload represents SAT solvers, which is one of the major supercomputing applications. Klee SAT Solver is a major component of the Cloud9 parallel symbolic execution engine [19]. Web Frontend workload intends the frontend of web applications, which often share the basic functionalities. Web Frontend workload benchmarks the frontend of a web-based social event calendar, and the workload runs Nginx [20] with a built-in PHP module, and APC PHP opcode cache. Web Search workload assumes indexing task of web search engines. CloudSuite includes Nutch as Web Search workload, and this is the same to HiBench Web Search workload. Traditional workloads in CloudSuite consists of well-known benchmarks in each domain, such as PARSEC [21], SPEC [3], and TPC [22].

Ferdman et al. found common characteristics among their workloads;

1) Workloads are scattered across a large number of machines, and each split portion with its data in charge typically fits into the local memory.
2) One workload often consists of a large number of completely independent requests, and these requests do not share any state.
3) Each workload has some special design for cloud computing environment, such as provision for disappearing computational nodes.
4) Inter-machine connectivity is utilized mainly for high-level task management, or coordination.

Through these observations with CloudSuite, Ferdman et al. conjectured that there is a large mismatch between the requirements of big data applications, and predominant processors. They also conjectured that the gap between the requirements, and processor architecture is widening. Here are the conclusions from their study;

1) Big data applications on modern processors suffer from high instruction-cache miss rates.
2) Big data applications have less instruction-level, and memory-level parallelism, and modern out-of-order cores do not provide enough benefits to big data applications.
3) Big data applications handle too huge input data to exceed the capacity of on-chip caches.
4) Big data applications do not require on-chip, and off-chip high bandwidth, which modern processors provide.

Although Ferdman et al. pointed out big data applications behave differently from traditional applications, they did not reveal the reasons for this difference in their work. Therefore, Yasin et al. gave the detailed analysis in [23], and they concluded that big data applications suffer from overheads related to managing the data rather than accessing the data.

Yasin et al. focused on Bayesian Classification from CloudSuite, and they ran the code on Hadoop. Then, they investigated the process from the system level, the application level, and the architecture level (threefold analysis). Their findings are as follows.

1) JVM has a major impact at the system level.
2) There is much room for code optimization at the application level.
3) Hash index lookup is a key limiter, and the optimization at microarchitecture implementation level improves overall performance significantly.
4) Bottlenecks in big data applications are fairly distributed compared to the bottlenecks in traditional applications.

Here, 1) and 2) are notable findings. That is, 1) indicates that careful choice of JVM has a possibility of direct speedup

 ISBN: 978-1-61208-461-9

TABLE III. THE DATASETS BIGDATABENCH INCLUDES.

Datasets	Data Structures
Wikipedia Entries	unstructured
Amazon Movie Reviews	semi-structured
Google Web Graph	unstructured (directed graph)
Facebook Social Graph	unstructured (undirected graph)
E-commerce Transaction Data	structured
ProfSearch Person Resumes	semi-structured

for big data applications. Additionally, 2) represents elimination of inefficient application code in big data applications simply gives a speed up. Actually, [23] demonstrated that one optimization in the main loop gave 50% speedup. Yasin et al. also pointed out that the programming style of big data applications is another obstacle for JVM and CPU exploitation.

C. BigDataBench

Wang et al. proposed BigDataBench in 2014 [24]. Wang et al. advocated BigDataBench provides the better coverage on benchmarking big data applications compared to traditional benchmarks including HiBench, and CloudSuite for the four reasons as follows.

1) BigDataBenchmark includes not only broad application scenarios, but also diverse real-world datasets.
2) BigDataBenchmark is more data centric, and needs to fulfill " 4V "(Volume, Variety, Velocity, and Veracity).
3) Workloads in BigDataBenchmark reflect diversity of the real-world big data applications.
4) BigDataBenchmark covers major infrastructures for big data applications including Hadoop, and Hive.

Table III lists real-world datasets, which BigDataBenchmark includes. (Table III is taken from [24], and the latest version has more datasets [25].) Wang et al. argue that these datasets cover diverse of data types, data sources, and application domains. As shown on Table III, While HiBench, and CloudSuite contains one unstructured text dataset respectively, BigDataBench includes one unstructured text dataset, one semi-structured text dataset, two unstructured graph datasets, one structured table dataset, and one semi-structured table dataset. Besides these datasets, BigDataBench also provides Big Data Generator Suite (BDGS), which generates synthetic structured, semi-structured, or unstructured texts, graphs, or tables.

Table IV lists workloads included in BigDataBench. (Table IV is taken from [24], and the latest version has more workloads [25].) BigDataBench workloads are chosen considering combinations of application scenarios (micro benchmarks, basic store operations, relational query, search engine, social network, and e-commerce), application types (online service, real-time analytics, and offline analytics), data types (structured, semi-structured, unstructured), data sources (text, graph, and table), and big data software stacks (Hadoop, Spark, MPI, Cassandra, and more). HiBench targets only on Hadoop, MapReduce, and Hive for big data software stacks. Wang et al. pointed out CloudSuite failed to fulfill the diversity in both datasets, and workloads.

Through experiments with BigDataBench, Wang et al. obtained the three major conclusions as follows.

1) The intensity of Flowting point instructions in BigDataBench is two orders of magnitude lower than the intensity in the traditional benchmarks such as PARSEC, or SPEC. On the other hand, the average ratio of integer instructions to floating point instructions in big data Applications is around two orders of magnitude higher than the average ratio in traditional benchmarks, with the similar intensities of integer instructions in both big data applications, and traditional benchmarks.
2) The volume of input data has significant impact over performance of big data applications.
3) L1 cache MPKI of big data applications are higher than those of traditional applications as pointed out in experiments with CloudSuite (in Section-sec:CloudSuite), while Wang et al. found that L3 caches are effective for the big data applications listed in BigDataBench.

Jia et al. conducted more detailed characterizations with BigDataBench [26]. The summary of their findings is as follows.

1) Software stack, such as Hadoop, Spark, and Cassandra, have much significant impact over the performance of big data applications than the difference of algorithms does. Therefore, software stacks should be included in benchmark suites for big data applications.
2) L3 cache miss rate, instruction fetch stalls, data TLB behaviors, and snoop responses are the four major microarchitectural level metrics to differentiate behaviors of Hadoop-based applications from those of Spark-based applications.
3) Jia et al. employed clustring technique to categorize workloads in BigDataBench, and proposed ways to extract seven workloads as the projection of the original workloads.

III. MODELING BIG DATA APPLICATIONS

We have reviewed major benchmarks for big data applications in Section II. Here, we point out workloads in those benchmarks always require any data storage, no matter traditional database systems, or NoSQL style systems. Actually, there are two kinds of applications in big data applications. One kind is the applications we reviewed in Section II. The other kind is stream mining, which analyzes data in a lined chronological order on the fly (without saving, or revisiting the original data). There is no major benchmark specific for stream minings, and no characteristics of stream minings is unveiled yet. In this section, we briefly overview researches on modeling stream mining algorithms as a first step for the establishment of benchmark suites.

A. Data Access Pattern Analysis

Akioka et al. proposed a stream mining model with the special focus on data dependencies [27]. The figures in this section are borrowed from [27]. Figure 1 illustrates the overall model of stream mining algorithms. In Figure 1, a stream mining algorithm consists of two parts, stream processing part, and query processing part. While the query processing

TABLE IV. THE WORKLOADS BIGDATABENCH INCLUDES.

Application Scenarios	Application Type	Workloads	Data Types	Data Source	Software Stacks
Micro Benchmarks	offline analytics	Sort	unstructured	text	Hadoop, Spark, MPI
		Grep			
		WordCount			
		BFS		graph	
Basic Datastore Operations	online service	Read	semi-structured	table	Hbase, Cassandra, MongoDB, MySQL
		Write			
		Scan			
Relational Query	realtime analytics	Select QUery	structured	table	Impala, MySQL, Hive, Shark
		Aggregate Query			
		Join Query			
Search Engine	online services	Nutch Server	unstructured	text	Hadoop
	offline analytics	Index			
		PageRank		graph	Hadoop, Spark, MPI
Social Network	online services	Olio Server	unstructured	graph	Apache+MySQL
	offline analytics	Kmeans			Hadoop, Spark, MPI
		Connected Components (CC)			
E-commerce	online services	Rubis Server	structured	table	Apache+JBoss+MySQL
	offline analytics	Collaborative Filtering (CF)	semi-structured	text	Hadoop, Spark, MPI
		Naive Bayes			

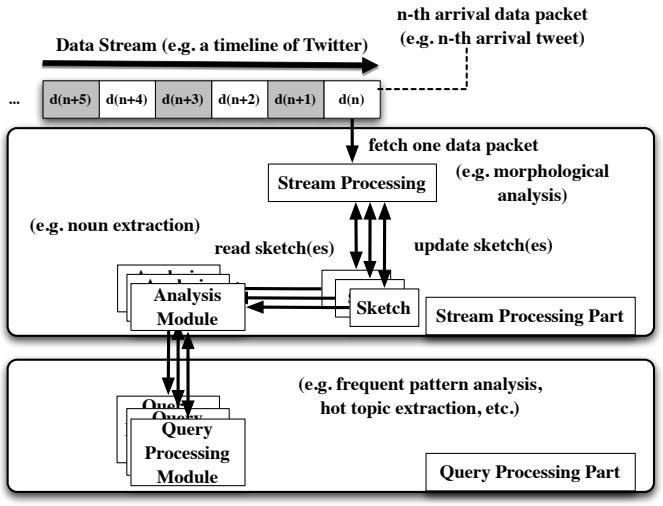

Fig. 1. A model of stream mining algorithms [27].

part is offline analysis, the stream processing part has a huge impact over the performance of the corresponding stream mining application. Once the stream processing part failed to process incoming data on the fly (before the next data unit arrives), the analysis keep losing subsequent data until the stream processing part catches up as there is no buffering space for incoming data in stream mining. In the stream processing part, a sketch is a small memory space similar to a cache. The stream processing module writes the results of preprocessing into the sketch(es), and the analysis module reads from the sketch(es) for the further processing.

Therefore, focusing on stream processing part in Figure 1, Figure 2 illustrates data dependencies between two processes analyzing data units in line, and data dependencies inside the process. The left top flow represents the stream processing part of the precending process, and the right bottom flow represents the stream processing part of the successive process. Each flow consists of six stages; read from sketches, read from input, stream processing, update sketches, read from sketches, and analysis. An arrow represents a control flow, and a dashed arrow represents a data dependencies. In Figure 2, there are three data dependencies in total.

B. Three Layer Model and its Extension

Junghans et al. proposed a three-layer model, which is another model for stream mining algorithm [28]. The figure in this section is borrowed from [28]. The shaded part in Figure 3 illustrates the original three-layer model. The whole picture of Figure 3 illustrates the extended version of three-layer model. The extended part in the model is to optimize the influential parameters in stream mining algorithms for the relaxed resource requirements, or the better quality of the mining results.

The flow of the whole process in the original three-layer model is as follows. First, the filter component filters incoming data stream by sampling, or load shedding. Secondly, the online mining component analyzes the original incoming data stream, or the filtered substream. Thirdly, the results of the online mining component will be stored in the synopsis, which is the second layer of the three-layer model. Here, synopsis indicates sketches, windows, or other dedicated data structures such as a pattern tree. Finally, the offline mining component answers user queries by accessing information stored in the synopsis. Therefore, the offline mining component does not need to fulfill the one pass requirement of stream mining.

The flow in the extension of the three-layer model is as follows. The resource monitoring, and the observation assessment component collect information about the current system state. Based on the monitoring by the resource monitoring, and the observation assessment, the parameters are decided whether they should be updated, or not. Then, the new parameters are set, and the stream mining algorithm run with the updated parameters.

C. Multiple Data Streams

Wu et al. advocated that the existing studies on stream mining assume only one data stream, and proposed formal definition of mining multiple data streams for the better practicality [29]. Wu et al. defined multiple data streams as a set of data flows generated by corresponding sources, and satisfy the following properties:

- continuous, one-pass, sequential, and self-functional,
- distributed, asynchronous, and non-blocking, and

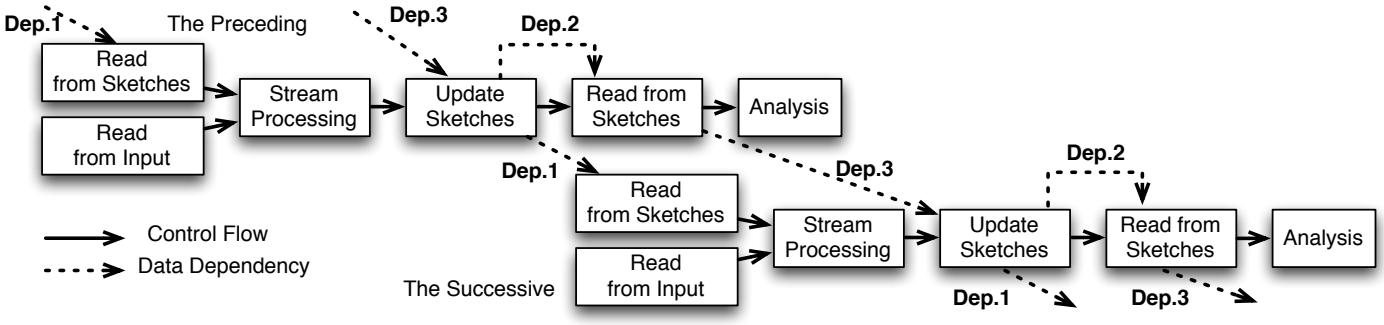

Fig. 2. Data dependencies of the stream processing part in two processes in line [27].

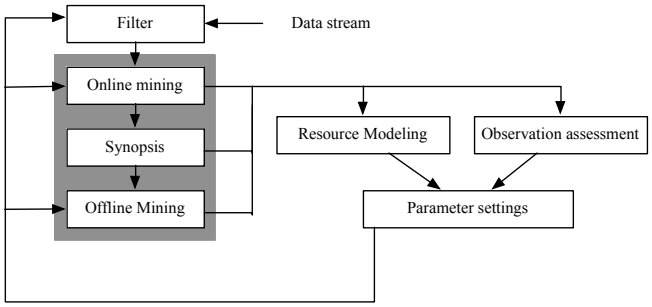

Fig. 3. Extended three-layer model [28].

- diversified, and autonomous.

According to [29], multiple data stream mining should be approached in a separate way from the way for single data stream mining for the following three reasons.

1) Multiple data streams are from many local data sources independently, and these data sources are not be able to process more than simple data preconditioning, or to save all the generated data either.
2) Multiple data stream mining is supposed to analyze across more than one data stream, instead of one single data stream.
3) Multiple data streams are not appropriate to be modeled as one single huge data stream with different attributes as each data is not under uniform timestamp cariteria, sampling rate, or privacy control policy.

In the multiple data stream model, a quadruple of the form (s, t, f, v) represents each data in a data flow, where s is the identification of the place, t is the time, or sequence number identifying the event, f is a function over the data, and v is a value vector of the output. An event refers whether data generation, or any other data processing. Each flow is a set of the quadruples, and fulfills the following properties.

- Each source specifies a single function to generate a single flow.
- If any pair of events, e_1 and e_2, occur at the same source, these two events have the same function invocation, and then the value t of e_1 is smaller than the value t of e_2, e_1 is identified as the event which occured before $e2$.
- For any pair of events, e_1 and e_2, that occur at the different sources, there is no function, or rule between e_1, and $e2$.

Similarly, flows can have some additional properties as follows.

- Homogeneous or heterogeneous: a pair of flows is said to be homegeneous (or heterogeneous) if the respective sources at which the two flows generate specify the same (or different) function(s), which are checked in terms of initial conditions, and output domain.
- Relational: a pair of flows, indicated by f_1, and f_2, is said to be relational if the value vectors of f_1 and value vectors of f_2 satisfy some relationship r.

IV. DISCUSSION

Primarily, the motivation for this paper is to get a quick overview of the current status of benchmarking, and modeling of big data applications. We reviewed several benchmark suites for a certain type of big data applications, and then we also overviewed several studies on modeling of another type of big data applications, which those current benchmark suites do not cover yet. Here are some points we have learned through the survey:

- No single project succeeded to establish a solid model, or solid models for big data applications yet. No single domain of big data applications is clearly characterized, either.
- Although each project has different findings for the details, the common finding is that the current CPU architecture is not suitable for acceleration of big data applications. The speed-up of big data applications seems likely to end up challenging the architecture community.
- Some projects pointed out that some methodologies of software implementations, or some coding styles block speed-up of big data applications. Of course, the way of software development changes according to the trend, or CPU architectures of the moment. The clear thing here is, however, the way of software development, and what CPU architecture community looks at are not meeting well.

 ISBN: 978-1-61208-461-9

- The software layer of big data applications is indispensable, and fat, and then this software layer is another obstacle for clarification of the behaviors of big data applications.

V. CONCLUSIONS

This paper surveyed the major benchmark suites for big data applications, and some projects on modeling of stream mining applications. Although there are several benchmark suites for big data applications as we reviewed, the community have not got the established benchmark suite(s), or model(s) yet. For the serious speed-up of big data applications, CPU artchitecture community, and big data application community move closer to share what each community is looking at.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI, Grant Number 15K21423, and 90308271.

REFERENCES

[1] I. Raicu, I. T. Foster, Y. Zhao, P. Little, C. M. Moretti, A. Chaudhary, and D. Thain, "The quest for scalable support of data-intensive workloads in distributed systems," in *Proc. the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009.

[2] J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users Guide*, SIAM, 1979.

[3] S. P. E. Corporation. Spec benchmarks. Retrieved: 2016-03-08. [Online]. Available: http://www.spec.org/benchmarks.html

[4] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Proc. of 2010 IEEE 26th International Conference on Data Engineering Workshops(ICDEW)*, 2010.

[5] T. A. S. Foundation. Hadoop. Retrieved: 2016-03-08. [Online]. Available: https://hadoop.apache.org

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. the 6th Conference on Symposium on Operating Systems Design and Implementation (OSDI'04)*, 2014.

[7] T. A. S. Foundation. Gridmix. Retrieved: 2016-03-08. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/gridmix.html

[8] Y. Jia and Z. Shao. Hive performance benchmarks. Retrieved: 2016-03-08. [Online]. Available: https://issues.apache.org/jira/browse/HIVE-396

[9] T. A. S. Foundation. Package org.apache.hadoop.examples.terasort. Retrieved: 2016-03-08. [Online]. Available: https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html

[10] ——. Nutch. Retrieved: 2016-03-08. [Online]. Available: http://nutch.apache.org

[11] P. Castagna. Having fun with pagerank and mapreduce. Retrieved: 2016-03-08. [Online]. Available: http://static.last.fm/johan/huguk-20090414/paolo_castagna-pagerank.pdf

[12] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft, "The smartfrog configuration management framework," in *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, 2009.

[13] T. A. S. Foundation. Mahout. Retrieved: 2016-03-08. [Online]. Available: http://static.last.fm/johan/huguk-20090414/paolo_castagna-pagerank.pdf

[14] A. Mahout. Naive bayes. Retrieved: 2016-03-08. [Online]. Available: https://mahout.apache.org/users/classification/bayesian.html

[15] ——. k-means clustering. Retrieved: 2016-03-08. [Online]. Available: https://mahout.apache.org/users/clustering/k-means-clustering.html

[16] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proc. of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII)*, 2012.

[17] T. A. S. Foundation. cassandra. Retrieved: 2016-03-08. [Online]. Available: http://cassandra.apache.org

[18] M. O. Forge. Darwin streaming server. Retrieved: 2016-03-08. [Online]. Available: http://dss.macosforge.org

[19] D. S. laboratory at EPFL. Cloud9. Retrieved: 2016-03-08. [Online]. Available: http://cloud9.epfl.ch

[20] nginx. nginx. Retrieved: 2016-03-08. [Online]. Available: http://nginx.org/en/

[21] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, 2011.

[22] TPC. Tpc benchmarks. Retrieved: 2016-03-08. [Online]. Available: http://www.tpc.org/information/benchmarks.asp

[23] A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive analysis of the data analytics workloads in cloudsuite," in *Proc. 2014 IEEE International Symposium on Workload Characterization (IISWC2014)*, 2014.

[24] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. ZHang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "Bigdatabench: a big data benchmark suite from internet services," in *Proc. 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[25] C. A. o. S. ICT. Bigdatabench—a big data benchmark suite. Retrieved: 2016-03-08. [Online]. Available: http://prof.ict.ac.cn/BigDataBench/#Benchmarks

[26] Z. Jia, J. Zhan, L. Wang, R. Han, S. A. McKee, Q. Yang, C. Luo, and J. Li, "Characterizing and subsetting big data workloads," in *Proc. 2014 IEEE International Symposium on Workload Characterization (IISWC2014)*, 2014.

[27] S. Akioka, H. Yamana, and Y. Muraoka, "Data access pattern analysis on stream mining algorithms for cloud computation," in *Proc. the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2010)*, 2010.

[28] C. Junghans, M. Karnstedt, and M. Gertz, "Quality-driven resource-adaptive data stream mining," in *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 1, 2011.

[29] W. Wu and L. Gruenwald, "Research issues in mining multiple data streams," in *Proc. the First International Workshop on Novel Data Stream Pattern Mining Techniques (StreamKDD'10)*, 2010.

 ISBN: 978-1-61208-461-9