Deciphering Big Data Stacks: An Overview of Big Data Tools

Tomislav Lipic[1], Karolj Skala[1], Enis Afgan*[1,2]

[1]Centre for Informatics and Computing
Rudjer Boskovic Institute, RBI
Zagreb, Croatia
{tlipic, skala}@irb.hr

[2]Department of Biology
Johns Hopkins University
Baltimore, MD, USA
enis.afgan@jhu.edu

Abstract— **With its ability to ingest, process, and decipher an abundance of incoming data, the Big Data is considered by many a cornerstone of future research and development. However, the large number of available tools and the overlap between those are impeding their technological potential. In this paper, we present a systematic grouping of the available tools and present a network of dependencies among those with the aim of composing individual tools into functional software stacks required to perform Big Data analyses.**

Keywords—Big Data; distributed systems; Cloud computing; data processing platform trends; interdependency

I. Introduction

With the multi-V model [1] used to categorize Big Data, the task of effectively addressing Big Data analysis needs to balance a combination of intricate issues, including data curation, management and privacy, as well as resource provisioning and management. Further complicating the situation is that each Big Data analysis project requires a unique yet composite environment driven by the project's goals. One increasingly complicated facet of the environment is the required software stack - deploying the appropriate stack is progressively becoming a technologically challenging undertaking.

Simultaneous to the development of the Big Data software stack challenge, the availability of Cloud computing platforms has spread beyond specialized commercial offerings into widespread installations at universities, institutes, and the industry [2]. This adoption was fueled by the demonstrated increase in available infrastructure flexibility, functionality, and service scalability. That alone, however, was not sufficient to sustain the wide-adoption of the Cloud; in addition, a well-defined programmatic access (i.e., API) and clear term-of-service (i.e., pricing/allocation) were a key that helped transform otherwise abstract technical terms (e.g., virtual machines, volumes, objects) into accessible functionality (e.g., scalable web applications), enabling the cloud-transformation.

Conceptually, the Cloud provides many features suitable for deploying Big Data stacks and has been demonstrated as a viable platform for deploying certain Big Data solutions [3]. The Cloud allows information integration from Big Data sources by offering capacity to store and process the data, enabling actionable insights to be derived. However, the process of utilizing Big Data solutions is still complex and time consuming, mostly left in the hands of adept data scientists and engineers [4]. Namely, in order to take advantage of the benefits that Big Data solutions offer, applications need to be (re)designed and (re)developed using appropriate concepts and technologies, making them Big Data-aware. Combined with the required computing infrastructure, Big Data requires a significant commitment on behalf of an organization, a group, or an individual.

To ease this transition, this paper sheds light on the tools used to deliver a software stack required to perform Big Data analysis. It does so by providing a systematic characterization of the current tools in form of a tool catalog, describing tools' capabilities and offering a functional comparison among those. It then presents a network of deployment dependencies among the described tools. By providing a concise description of the components used to deliver Big Data analysis platforms, we believe the paper can contribute towards enabling a BigData-transformation, comparable to the cloud-transformation, and support adoption of the available technologies. In future work, we will leverage this information to enable automated composition and deployment of the available technologies into full-featured stacks to deliver functional Big Data analysis platforms in the Cloud.

II. TECHNOLOGIES FOR BIG DATA CHALLENGES

The field of Big Data is focusing on the challenges that the available data is imposing and aims to produce actionable results based on the data. Inspired by the diversity of inputs and the multiplicity of conceivable outcomes, the number of technological options covering the Big Data space has exploded. One way to navigate through this space is to follow a trajectory of the developed technologies as they address different Big Data challenges. Four such trajectories have emerged and can be identified as recognizable *trends*: (1) batch processing, (2) query processing, (3) low-latency query processing, and (4) continuous processing. These trends represent a grouping of technologies, embodied by tools with similar purpose.

A. *Batch Processing*

Batch processing can be seen as the 'traditional' Big Data - it revolves around the MapReduce paradigm [5] where an input file is split into a number of smaller input files (the map step), an algorithm is executed in parallel batch mode on each of the input files across possibly many commodity computers, and the outputs from each step are then combined into a single output (the reduce step). The Apache Hadoop project implements this paradigm; this open source implementation has been adopted and modified to a various degree by a number of derivative distributions, most notably Cloudera CDH, Hortonworks Data Platform (HDP), MapR M series, and Amazon Elastic MapReduce (EMR). Utilizing the Hadoop framework requires an application to be developed using the MapReduce paradigm and is typically recommended when the input data is large (terabytes or larger) and diverse. Hadoop is accompanied with its own file system, Hadoop Distributed File System (HDFS) [6], which provides storage and data redundancy on top of commodity hardware. HDFS is not a POSIX compliant file system and any data being operated on via Hadoop is required to be imported into HDFS. Upon completion of the data analysis, the data is exported onto a POSIX file system to be accessed by other tools.

B. *Query Processing*

MapReduce provides key features for addressing the analysis and storage challenges of large volumes of data. However, it requires programmatic access to the data. By providing higher-level interfaces instead, access to data would be democratized, allowing users not versed in programming could access the data directly. As a first step in this direction, a query language was devised and implemented as part of Apache Pig [7], which translates high-level custom queries into MapReduce jobs. Additionally, Apache Hive [8] was developed that takes SQL-like queries and translates those into MapReduce jobs. Unlike adopting the raw MapReduce approach that requires significant adoption effort, these solutions allow the existing investment in SQL to be leveraged in the Big Data context.

The usage of NoSQL databases is another approach for storing and querying Big Data. Instead of just storing the data directly in HDFS and retrieving it via MapReduce jobs, the data is stored in a database and retrieved via high-level queries [9]. These databases provide non-relational, distributed, and horizontally scalable storage solutions while hiding away the details of replicas and table sharding otherwise necessary when storing data at large scale. At the same time, they provide direct access to the data in form of simpler, query-based interfaces. NoSQL databases are roughly categorized into the following four classes [10], [11]: document stores, key-value stores, columnar, and graph databases. Tools such as CouchDB (document), Riak (key-value), HBase (column), and Neo4j (graph) provide this horizontal storage scalability and implement the query interface yielding the Big Data stores more accessible (see Han et al. [12] for a comprehensive survey and *http://nosql-database.org/* for an up-to-date list of NoSQL databases).

C. *Low-latency Query Processing*

For certain use cases (e.g., interactive visual analytics), it may be desirable to analyze data in near real-time (i.e., where processing lasts from seconds up to a minute). Batch processing is simply too slow for this type of data analysis and modified query engines have been devised that allow the computation time to be reduced up to 100 times [13]. The improved responsiveness is achieved by performing computation in memory [14] or by mapping SQL-like queries onto columnar data layout through multi-level execution trees without translating them into MapReduce jobs [15]. Although conceptually similar, the low-latency querying is not intended to replace batch and query processing within the Hadoop framework but to complement it by being used to analyze the outputs of a traditional Hadoop job or to prototype computations. Tools such as Cloudera Impala [16] and Apache Drill [17] implement these paradigms and are themselves based on Google's internal Dremel system [15]. Apache Drill also supports data sources other than HDFS, namely NoSQL databases while Impala can query data stored in HBase.

D. *Continuous Processing*

Complementing the real-time capabilities of the low-latency queries is continuous data processing, or processing data streams. For applications such as online machine learning or real-time applications there is a need to process unbound streams of data and convert it into desired form. Projects such as Apache Storm [18], Apache S4 [19], and Apache Samza [20], or managed platform solutions such as AWS Kinesis [21], address this challenge by providing programmable interfaces for operating on countless tuples, or events, of incoming data. These applications allow one to define operations on tuples that transform incoming data as needed (e.g., transform raw sensor data into a precursor for emergency situations or stock market data into trading trends). Individual transformations can be composed into a topology, which becomes an execution workflow capable of addressing arbitrarily complex transformations. It is worth noting that unlike MapReduce jobs and queries that have a start and an end, data streams processing runs continuously. Underlying the data streams analysis layer is an effective data delivery system. Projects such as Apache Kafka [22] or Apache Flume [23] act as messengers that provide a robust and scalable mechanism for the data to be published and consumed by a data stream processing framework.

E. *Crossing the Trends*

In addition to the projects that function in the context of any single trend, there are efforts that sit at a cross-section of multiple trends. Apache Spark [24] is an example of such project; it supports low-latency queries as well as processing data streams. The project implements a distributed memory abstraction, called Resilient Distributed Datasets, that allows computations to be performed in memory on large clusters in a fault-tolerant manner [14]. Similarly, the Stratosphere project [25] builds on the work brought about by Spark by also implementing an in-memory alternative to the traditional MapReduce. Unlike Spark with its Resilient Distributed Datasets implementation, Stratosphere implements a programming model called Parallel Contracts as part of its execution engine Nephele [26]. This is realized by extending the MapReduce paradigm by adding additional data processing operators, for example join, union, and iterate. In combination with the map and reduce operators, Stratosphere allows complex data-flow graphs to be composed as more comprehensive data workflows.

Tools described above that perform data processing require and rely on resources provisioned and managed by cluster resource managers, such as Apache YARN [27] or Apache Mesos [28]. These resource managers are able to dynamically allocate necessary resources for a specific tool via resource isolation, which allows for Hadoop, MPI, and other environments to readily utilize the same infrastructure. This increases infrastructure flexibility because the same set of physical resources is easily repurposed based on need.

F. Domain-specific Applications and Libraries

The described trends identify a layer of data processing infrastructure. Alone, this layer acts as a set of processing engines suitable for handling different types of stored data. On top of these engines are domain-specific libraries and applications that provide higher-level interfaces for interacting with the engines. This layered structure is visualized in *Fig. 1.* while Table 1 provides a reasonably comprehensive coverage of the currently available applications and libraries, grouped by their function and intended here as a user reference.

TABLE I. FUNCTIONAL CLASSIFICATION OF EXISTING HIGHER-LEVEL BIG DATA APPLICATIONS AND LIBRARIES

Functional classification	**Existing Big Data technologies**	**Function / Description**
High-level programming abstractions	Apache Pig, Apache DataFu, Cascading Lingual, Cascalog	Provide higher-level interface, for example SQL, for composing Hadoop data analysis jobs.
	Shark (for Spark), Trident (for Storm), Meteor, Sopremo and PonIC (for Stratosphere)	Facilitate easier and more efficient programming of complex data processing and analysis tasks in batch, iterative, and real-time manner.
Big Data-aware machine-learning toolkits	Apache Mahout (for Hadoop), MLlib (for Spark), Cascading Pattern, GraphLab framework, Yahoo SAMOA	Allow machine learning algorithms to be more easily utilized in the context of the MapReduce paradigm, individually tailored for different types of input data and/or processing type.
Graph processing systems	Apache Giraph (for Hadoop), Bagel (for Spark), Stratosphere Spargel, GraphX (for Spark), Pegasus, Aurelius Faunus, GraphLab PowerGraph	A range of tools covering generic graph based computations, complex networks, and interactive graph computations.
	Stinger, Neo4j, Aurelius Titan	Storage facilities for graph-based tools and data structures.
Data ingestion and scheduling systems	Apache Sqoop, Apache Chukwa, Apache Flume	Act as orchestrator frameworks by facilitating bulk data transfers between Hadoop and structured data stores, log collection, or data streams into central data stores.
	Apache Falcon, Apache Oozie	Handle data processing and management and workflow scheduling, including data discovery, process orchestration and lifecycle management.
Systems management solutions	Apache Hue	Web user interface providing browsers for HDFS files, MapReduce jobs, and a range of higher-level applications (e.g., HBase, Hive, Pig, Sqoop).
	Apache Ambari, Apache Helix, Apache Whirr, Cask Coopr	Cluster managers used for provisioning, managing, and monitoring applications, services, and cloud resources.
Benchmarking and testing applications	Berkeley Big Data benchmark, BigBench, BigDataBench,, Big Data Top 100, Apache Bigtop	Used for statistical workload collection and generation or testing Hadoop-related projects.

III. STACK DEPLOYMENT DEPENDENCY GRAPH

To make them useful for complex analyses, the available technologies often need to be assembled into composite software stacks. The schematic provided in *Fig. 1.* depicts an abstract stack, mapping the technologies discussed thus far onto the appropriate stack layers. The Cloud offers an opportunistic platform for quickly and flexibly composing the available technologies into the required stack with exact tools. Doing so, however, requires detailed understanding of the tools' *functions* and *deployment requirements*.

The previous section provides descriptions of the tools' function while this section provides insight regarding the tools' deployment requirements. Discovering tool deployment requirements is, unfortunately, not a clear-cut task. *Fig. 2.* demonstrates the complexity of this effort by depicting a tool deployment interdependency graph for Big Data tools. This graph has been constructed by manually examining the available Big Data tools and technologies and identifying explicit links between them. Constructing such a graph for an even more comprehensive set of tools in an automatic or semi-automatic manner would itself be a Big Data analysis problem.

The interdependency graph represents an ordering mechanism among the tools when deploying them. For example, if one wants to use Pig Latin language, they would need Pig runtime environment, which would in turn require Hadoop, which would in turn require either YARN and consequently HDFS or HBase and HDFS. Along with the absolute requirements on dependent tools, some tools also have optional dependencies (indicated by faint arrows). Optional dependencies either enhance or modify functionality of the tool to make it suitable for more tasks. Pig can, for example, utilize Oozie as a workflow scheduling engine to enhance execution of workflows run via Pig.

In addition to providing an effective overview of the current Big Data tools' dependencies, the network provides

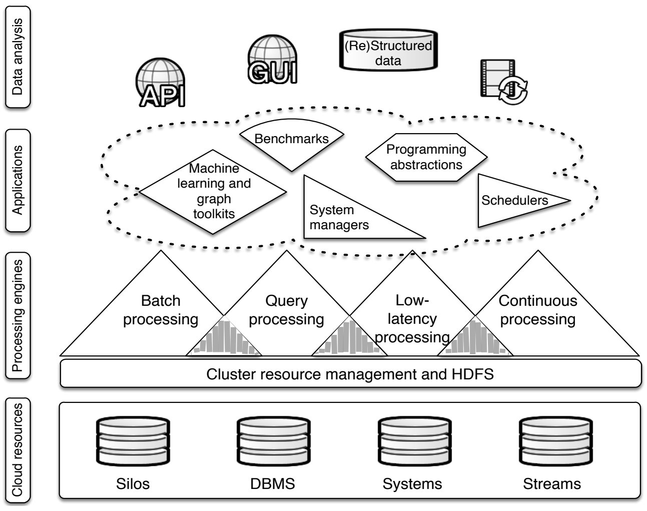

Fig. 1. A layered view of an abstract Big Data analysis stack. Shaded areas indicate functional overlap between the processing trends.

insights into the correlation of the available tools. Inspecting the network, several locus points are visible (indicated by the size of a node). The size of the nodes is proportional to the number of incoming dependencies a node (i.e., tool) has and thus the largest nodes correspond to a set of core tools required most often for functional Big Data stacks. Most notably, this is HDFS and cluster resource managers (e.g., YARN) that act as common denominators in the stack. Color-based locus points (e.g., Hadoop, Spark, Stratosphere) represent alternative technologies with comparable functionality. The choice of technology in this case is based on usage and is rooted in the type of input data being processed. The leaf nodes in the network represent unique or niche tools that fill a specific roll.

Also visible from the graph is that the majority of tools favor YARN as a central resource manager. Building on top of YARN are either tools that provide user-facing functions (i.e., leaf nodes) or a class of tools that act as higher-level interfaces to the resources for other tools (e.g., Spark, Hadoop). Tools

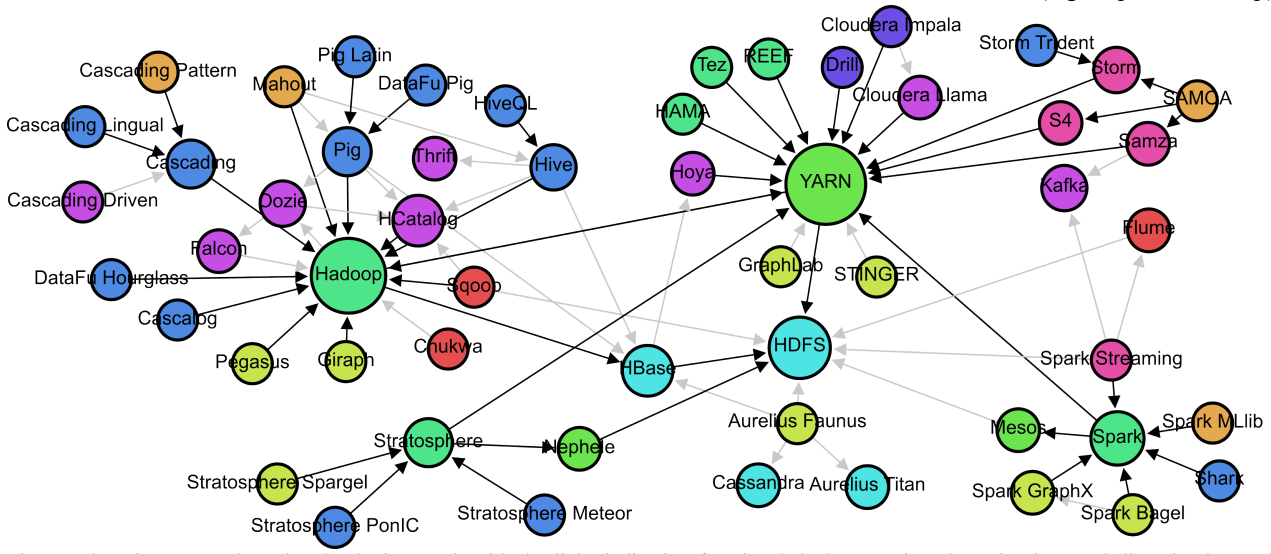

Fig. 1. The Big Data tools and technologies graph with the links indicating functional deployment interdependencies. Dark lines imply an absolute requirement while faint lines imply an optional requirement. Whether to fill the optional requirement is influenced by a desired functionality of the final stack so if the functionality is desired, the dependency needs to be filled. Colors of nodes indicate functional similarity of the tools and the size of the nodes indicate the degree of dependence other tools have on the given tool.

linking to those tools form groups that cover a range of domains (e.g., Spark-based tools covering a number of domains and are all linked to Spark). This is in contrast to thinking that those tools would focus only on a single problem domain. Hence we used color throughout the graph to indicate notion of similar problem domain for the tools. For example, Giraph and GraphX are alternative tools for solving similar problems.

IV. CONCLUSIONS

Instigated only a few years ago by a novel algorithm parallelization model, MapReduce, and an open source implementation of the same, Hadoop, the Big Data field was born. The field has quickly evolved into one of the major movements driving today's research and economy. Fueled by the many-V's of data, the Big Data field has exploded with a myriad of options ranging from technical solutions, commercial offerings, to conceptual best practices. Trying to adopt or leverage the field of Big Data feels like walking through a minefield of terms, unclear or even conflicting statements, and implicitly defined requirements. Simultaneously, the influx of data is mandating use of available technologies in a wide spectrum of analyses: from financial stock exchanges to monitoring consumer sentiment, to analyzing state of running systems.

In response to this turbulent state of the Big Data landscape, in this paper, we summarized the current technological state in the space of Big Data. We have done this by compiling a catalog of the most recognized and notable Big Data tools and technologies and then grouped them based on their function. Next, based on this classification of the real-world state, we have devised a graph of their deployment interdependencies. This graph allows different layers of the Big Data stack to be defined, which in turn enables dependencies between individual tools to be identified. Having the ability to identify the layers and the dependencies between individual tools demystifies much of the Big Data landscape and enables functional systems composed of multiple tools to be more easily built.

We believe that the focus of future work in the space of Big Data should be on democratizing access to the available tools by offering functional solutions rather than independent technologies. This will foster development of tuned workflows and pipelines, and thus tangible value, instead of mere capacity. As part of ongoing work (e.g., [4]), we will be leveraging the information available in the devised graph to enable custom and autonomous deployment of Big Data stacks on Cloud resources.

ACKNOWLEDGMENT

This work was supported in part by FP7-PEOPLE programme grant 277144 (AIS-DC).

REFERENCES

[1] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. S. Netto, and R. Buyya, "Big data computing and clouds: Trends and future directions," *J. Parallel Distrib. Comput.*, Aug. 2014.

[2] S. Pandey and S. Nepal, "Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond," *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1774–1776, 2013.

[3] D. Talia, "Clouds for Scalable Big Data Analytics," *Computer (Long. Beach. Calif).*, vol. 46, no. 5, pp. 98–101, May 2013.

[4] L. Forer, T. Lipic, S. Schonherr, H. Weisensteiner, D. Davidovic, F. Kronenberg, and E. Afgan, "Delivering bioinformatics MapReduce applications in the cloud," in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014, pp. 373–377.

[5] J. Dean and S. Ghemawat, "MapReduce : Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 1–13, 2008.

[6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010*, 2010.

[7] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-so-foreign Language for Data Processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008, pp. 1099–1110.

[8] A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010, pp. 996–1005.

[9] C. Fay, D. Jeffrey, G. Sanjay, C. H. Wilson, A. W. Deborah, B. Mike, C. Tushar, F. Andrew, and E. G. Robert, "Bigtable: A Distributed Storage System for Structured Data," *OSDI*, 2006.

[10] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record*, vol. 39, no. 4. p. 12, 2011.

[11] S. Weber and C. Strauch, "NoSQL Databases," *Lect. Notes Stuttgart Media*, pp. 1–8, 2010.

[12] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, 2011, pp. 363–366.

[13] S. Shenker, I. Stoica, M. Zaharia, and R. Xin, "Shark: SQL and Rich Analytics at Scale," *Proc. 2013 ACM SIGMOD Int. Conf. Manag. Data*, pp. 13–24, 2013.

[14] M. Zaharia, M. Chowdhury, T. Das, and A. Dave, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," *NSDI'12 Proc. 9th USENIX Conf. Networked Syst. Des. Implement.*, pp. 2–2, 2012.

[15] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel : Interactive Analysis of Web-Scale Datasets," *Proc. VLDB Endow. 3*, vol. 1–2, pp. 330–339, 2010.

[16] "Cloudera Impala." [Online]. Available: http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.

[17] M. Hausenblas and J. Nadeau, "Apache Drill: Interactive Ad-Hoc Analysis at Scale," Jun. 2013.

[18] "Storm: Distributed and fault-tolerant realtime computation." [Online]. Available: http://storm.incubator.apache.org/.

[19] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," *2010 IEEE Int. Conf. Data Min. Work.*, pp. 170–177, 2010.

[20] "Apache Samza." [Online]. Available: http://samza.incubator.apache.org/.

[21] "Amazon Kinesis." [Online]. Available: http://aws.amazon.com/kinesis/.

[22] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," *Proc. NetDB*, 2011.

[23] "Apache Flume." [Online]. Available: http://flume.apache.org/.

[24] "Spark." [Online]. Available: https://spark.incubator.apache.org/.

[25] "Stratosphere Project." [Online]. Available: http://stratosphere.eu/.

[26] A. Alexandrov, M. Heimel, V. Markl, D. Battré, F. Hueske, E. Nijkamp, S. Ewen, O. Kao, and D. Warneke, "Massively parallel data analysis with PACTs on Nephele," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 1625–1628, Sep. 2010.

[27] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwali, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN : Yet Another Resource Negotiator," in *ACM Symposium on Cloud Computing*, 2013, p. 16.

[28] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," *Proc. 8th USENIX Conf. Networked Syst. Des. Implement.*, p. 22, 2011.