Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

A Comprehensive View of Hadoop Research - A Systematic Literature Review

Ivanilton Polato[a,b,*], Reginaldo Ré[b], Alfredo Goldman[a], Fabio Kon[a]

[a]*Department of Computer Science, University of São Paulo, São Paulo, Brazil*
[b]*Department of Computer Science, Federal University of Technology - Paraná, Campo Mourão, Brazil*

Abstract

[Context:] In recent years, the valuable knowledge that can be retrieved from petabyte scale datasets – known as Big Data – led to the development of solutions to process information based on parallel and distributed computing. Lately, Apache Hadoop has attracted strong attention due to its applicability to Big Data processing. **[Problem:]** The support of Hadoop by the research community has provided the development of new features to the framework. Recently, the number of publications in journals and conferences about Hadoop has increased consistently, which makes it difficult for researchers to comprehend the full body of research and areas that require further investigation. **[Solution:]** We conducted a systematic literature review to assess research contributions to Apache Hadoop. Our objective was to identify gaps, providing motivation for new research, and outline collaborations to Apache Hadoop and its ecosystem, classifying and quantifying the main topics addressed in the literature. **[Results:]** Our analysis led to some relevant conclusions: many interesting solutions developed in the studies were never incorporated into the framework; most publications lack sufficient formal documentation of the experiments conducted by authors, hindering their reproducibility; finally, the systematic review presented in this paper demonstrates that Hadoop has evolved into a solid platform to process large datasets, but we were able to spot promising areas and suggest topics for future research within the framework.

Keywords: Systematic Literature Review, Apache Hadoop, MapReduce, HDFS, Survey

1. Introduction

One of the largest technological challenges in software systems research today is to provide mechanisms for storage, manipulation, and information retrieval on large amounts of data. Web services and social media produce together an impressive amount of data, reaching the scale of petabytes daily (Facebook, 2012). These data may contain valuable information, which sometimes is not properly explored by existing systems. Most of this data is stored in a non-structured manner, using different languages and formats, which, in many cases, are incompatible (Bakshi, 2012; Stonebraker et al., 2010).

Take, for instance, Facebook, which initially used relational database management systems (DBMS) to store its data. Due to the increasingly large volume of information generated on a daily basis (from a 15TB dataset in 2007 to a 700TB dataset in 2010) (Thusoo et al., 2010), the use of such infrastructure became impracticable. Specially because, most of its data is unstructured, consisting of logs, posts, photos, and pictures. One of the Facebook's largest cluster holds more than 100 PB of data, processing more than 60,000 queries a day (Facebook, 2012). Having achieved in September 2012 more than 1 billion active users, Facebook may be considered one of the largest and most valuable social network.

Companies holding large amounts of user data started to be evaluated not just by their applications but also by their datasets, specially the information that can be retrieved from them. Big companies like Google, Facebook and Yahoo! have an aggregate value not only for their provided services but also for the huge amount of information kept. This information can be used for numerous future applications, which may allow, for example, personalized relationships with users.

The "Big Data" (Zikopoulos and Eaton, 2011; White, 2012) term is used to refer to a collection of large datasets that may not be processed using traditional database management tools. Some of the challenges involved when dealing with Big Data goes beyond processing, starting by storage and, later, analysis. Concerning data analysis and Big Data, the need for infrastructures capable of processing large amounts of data, within an acceptable time and on constrained resources, is a significant problem. Plausible solutions make use of parallel and distributed computing. This model of computation has demonstrated to be essential nowadays to extract relevant information from Big Data. Such processing is accomplished using clusters and grids, which use, generally, commodity hardware to aggregate computational capacity at a relatively low cost.

Although parallel and distributed computing may be one of the most promising solutions to store and manipulate Big Data, some of its characteristics may inhibit its use by common users. Data dependency and integrity, cluster load balancing and task scheduling are major concerns when dealing with parallel and distributed computing. Adding the possibility of an almost certain machine failure, the use of these concepts becomes non-trivial to inexperienced programmers. Several frameworks have been released to abstract these characteristics and provide

[*]Corresponding author. Telephone/Fax: +55 44 3518 1449
Email address: ipolato@utfpr.edu.br (Ivanilton Polato)

high level solutions to end users (DeWitt et al., 2008; Battré et al., 2010; Malewicz et al., 2010; Isard et al., 2007); some of them were built over programming paradigms, such as MPI and MapReduce.

The MapReduce programming paradigm, now highly used in the context of Big Data, is not new. One of the first uses of this paradigm was on the LISP programming language. It relies basically on two functions, Map and Reduce. The first generates maps based on a given user defined function and the second groups Map outputs together to compute an answer. The paradigm is very useful when dealing with batch programs where data is manipulated in a sequential way. Recently the MapReduce paradigm attracted attention because of its applicability to parallel computing. Google's MapReduce composed initially of the GFS distributed filesystem (Ghemawat et al., 2003) and the implementation of MapReduce (Dean and Ghemawat, 2004, 2008), brought to the fore the use of the simple and consolidated functions Map and Reduce in parallel and distributed computing using Java and C++ libraries. This approach feeds the Reduce function with the Map function results. This enables parallelism since partitioned portions of data may be fed into different instances of Map tasks throughout the cluster. The results are gathered, used as inputs to the Reduce instances, and the computation is accomplished. The great novelty here is that the approach hides from users a lot of the complexity of parallelization and distribution. Users can focus on the functionality of their programs and the framework abstracts the complexity and controls the infrastructure.

Based on this novel approach, Doug Cutting, an employee of Yahoo! at the time, and Mike Cafarella, a professor at University of Michigan, developed Hadoop, later called the Apache Hadoop framework. It is an open source implementation of the Google's MapReduce approach. It uses the same idea from Google's: hiding complexity from users allowing them to focus on programming. Mostly known by its MapReduce implementation, Apache Hadoop also has an ecosystem composed of several applications ranging from data warehousing to a data flow oriented programming language. The Apache Hadoop framework provides solutions to store, manipulate and extract information from Big Data in several ways. The framework has evolved over the last few years and promotes data integrity, replication, scalability, and failure recover in a transparent and easy-to-use way. All these factors have made Apache Hadoop very popular both in academia and in industry.

The early adoption of Hadoop by the research community has provided rapid evolution and development of new features to the framework. Over the last five years, the framework received numerous contributions from researchers, most of them published worldwide in journals and conferences. The large number of publications makes it difficult for researchers to find specific topics or areas that still need further investigation, resulting in a need for an unbiased analysis of the scientific and technological research results obtained in this field in the past years. Thus, this article is mainly directed to:

- Researchers and graduate students willing to carry out new research around Hadoop, which can use the current paper as a guide to which areas have been covered in past research and which areas deserve more attention;
- Practitioners, and users interested in knowing what are the major technological features incorporated into Hadoop and how they evolved over time;
- Researchers interested in having an overall panorama of the research that has been carried out around the field.

This work is also an opportunity to map publications and classify them into categories, which indicates possible promising areas in which contributions are needed. In this paper we present the results of a systematic literature review on Hadoop research. Our objectives include reviewing the literature on the framework to find out which topics have been researched over the last few years. We want to discover which areas have received improvements and solutions from academic research and which areas still have room for advancements. Thus, our main contributions are:

- A systematic literature review pointing out which studies have directly contributed to the Apache Hadoop;
- A taxonomy of the selected studies to allow an in-depth analysis of the more active areas regarding the framework;
- Pointing out the main research areas that still need and have room for improvements around Hadoop.

The rest of the paper is organized as follows. We start presenting our research method and protocol used to develop this systematic review. Section 3 presents the characterization of the selected studies. In Section 4, we classify the selected studies describing and presenting the contributions of each one to the Apache Hadoop project. We discuss the findings in Section 5 and analyze the results to point out hot and cold zones in the project. Related papers are presented in Section 6 and finally, in Section 7 we present our conclusions and discuss opportunities for future research.

2. Research Method

Systematic reviews provide a way to execute in-depth unbiased literature reviews, aggregating scientific value to its results. The objective of a systematic review is to present a correct assessment regarding a research topic through the application of a reliable, rigorous, and auditable methodology, for example as proposed by Kitchenham and Charters (2007). The systematic review starts with the definition of a protocol that will guide the progress of the research, including research questions and methods used during the revision process. According to those authors, the protocol must include:

- The research questions that the study aims to respond;
- Strategies used in the searches for the primary studies like search strings and selected digital libraries, journals, and conferences;

- Inclusion and exclusion criteria for primary studies;
- Quality assessment procedure for selected studies;
- Strategy for data extraction and subsequent synthesis of extracted data.

In this section, we present the protocol we used in our Apache Hadoop systematic review. The main objective of this review was to elucidate both fundamental aspects and contributions to Apache Hadoop, including its programming paradigm, MapReduce, its storage file system, the Hadoop Distributed File System (HDFS), and the rest of its ecosystem.

2.1. Objectives and research questions

Parallel and distributed computing currently has a fundamental role in data processing and information extraction of large datasets. Over the last years, commodity hardware became part of clusters, since the x86 platform cope with the need of having an overall better cost/performance ratio, while decreasing maintenance cost.

Apache Hadoop is a framework developed to take advantage of this approach, using such commodity clusters for storage, processing and manipulation of large amounts of data. The framework was designed over the MapReduce paradigm and uses the HDFS as a storage file system. Hadoop presents key characteristics when performing parallel and distributed computing, such as data integrity, availability, scalability, exception handling, and failure recovery. Even more, these features are presented in a transparent manner to the user, which represents a novel approach to newcomers.

In this context, the objective of this work is to answer the question: What are the main topics of investigation related to Apache Hadoop and its programming paradigm, MapReduce? Although Hadoop is a fairly adopted platform for distributed processing of large datasets, it still has room for improvements and we want to discover what are these areas. To achieve our goal, it is necessary to evaluate the aspects addressed in the studies of the academic community. This is our Research Question 1:

> RQ1: What are the main research topics and aspects covered by publications concerning the Apache Hadoop framework and the MapReduce paradigm?

Our second objective is to organize the selected studies in a structured taxonomy. Thus, our Research Question 2 must also be answered during our research.

> RQ2: How to organize the main aspects covered by the recent studies about Apache Hadoop in a taxonomy?

This question aims to classify the recent studies according to their collaboration with the Apache Hadoop framework, and to answer RQ2 we could subdivide the question into three items that will help the classification of the selected studies and the formalization of the taxonomy:

- RQ2.1: How to create categories from the selected studies to classify them into a taxonomy?
- RQ2.2: What kind of validation is performed in each study? Simulation, analytical model, and experimentation?
- RQ2.3: The proposed approaches take into account whether the problem being solved is application/data-specific or more generic?

2.2. Search strategies

Having the research questions established, the search strategies and search string were defined. We also defined the search scope and the consulted digital libraries.

Research Keywords. We identified the keywords based on the research questions. There was no need to consider synonyms to the keywords since the names were predefined based on the names defined by the project developers. The search terms identified were "Hadoop" and "MapReduce" written exactly as defined by the developers in the Apache Hadoop website (http://hadoop.apache.org).

Search Strings. The search strings were built based on the research questions using the selected keywords. Sometimes search strings have to be adapted according to the specific needs of digital libraries, but this was not necessary in our case. The search string used to obtain the initial results of this review was "(Hadoop) OR (MapReduce)". This type of expression, more generic, may be used in almost all digital libraries.

Sources. The criteria used to select sources of studies were: must have web search mechanism; search mechanisms must allow customized searches by title and abstract; full articles must be available for download using available contracts between our university and the digital library; importance and relevance of sources. With the search string defined, we chose the following digital libraries as sources:

- ACM Digital Library
- IEEEXplore Digital Library
- ScienceDirect

2.3. Selection of studies

Studies selected for further analysis on this systematic review must be published as full papers in Journals or in Conferences indexed by the selected digital libraries. After obtaining the results from the digital libraries, all studies have to be analyzed individually to confirm the relevance in the context of our review. This way, we composed a group of primary papers that were reviewed during our research. To select or discard studies, inclusion and exclusion criteria were defined as follows. Table 1 shows the stages involved in the selection of papers.

Inclusion criteria. The inclusion of a work is made based on its relevance to the research questions. First, we analyzed title, keywords and abstract of the studies obtained from the initial search on the libraries. The selected studies from this stage

Stage 1	Apply the search query to all the sources, gathering the results.
Stage 2	Exclude invalid and duplicated papers.
Stage 3	Apply inclusion/exclusion criteria to the papers titles, keywords, and abstracts.
Stage 4	Apply inclusion/exclusion criteria to introductions and conclusions.
Stage 5	Review the selected studies applying, when necessary, inclusion/exclusion criteria to the text.

Table 1: Studies selection stages

were analyzed again, this time by its introduction and conclusions. If at any time one of the inclusion criteria was broken, the study was discarded.

The inclusion criteria were the following:

1. Studies dealing primarily with concepts related to the development of solutions for the Apache Hadoop MapReduce platform and its ecosystem;
2. Studies having Hadoop/MapReduce as a final objective (as an end) and not as a means (infrastructure) to accomplish computations for other areas;
3. Studies that present complementary approaches that extend Hadoop/MapReduce and/or its ecosystem; and,
4. Studies published in Journals and/or Conferences.

Exclusion criteria. Exclusion of studies was made by the analysis of title, keywords, abstract, and later by introduction and conclusions when necessary, observing the following criteria:

1. Studies published in Workshops, since they are very numerous and normally represent less mature work that later, after refinement, are enhanced and published in conferences and journals;
2. Studies that make use of Hadoop as an infrastructure platform to obtain results without modifying or proposing new solutions directly related to Hadoop;
3. Studies that do not cite Hadoop although citing MapReduce, i.e., other MapReduce frameworks;
4. Studies that present complementary approaches that do not extend Hadoop/MapReduce and/or its ecosystem;
5. Studies that do not have the complete full text available at the source;
6. Studies that do not answer or are irrelevant to the research questions;
7. Repeated studies that were published in more than one source;
8. Short papers, talks, demonstrations, or tutorials;
9. Similar studies published by the same authors. In this case, the most recent or most complete one was considered.

3. Characterization of the Selected Studies

This systematic review follows a protocol developed based on Kitchenham and Charters's methodology (2007). This section presents the characterization of the selected studies according to our protocol.

Source	**Query Results**	**Stage 2**	**Stage 3**	**Stage 4**
ACM	634	347	70	38
IEEE	966	630	94	63
ScienceDirect	84	15	9	5

Table 2: Number of selected papers per source.

Selected studies were obtained from three digital libraries: IEEEXplore, ACM, and ScienceDirect. Searches were conducted in the beginning of 2013 and later updated in June, 2014; in both cases, the search was limited to papers published until December 31, 2013. Table 2 shows the numbers of studies selected from each digital library. Our final selection resulted in 38 studies from the ACM Digital Library, 63 from the IEEE Xplore Digital Library, and 5 from the ScienceDirect Digital Library, totaling 106 studies that were reviewed.

Paper Type	**Journal**		**Conference**	
	Initial	Selected	Initial	Selected
ACM	105	10	344	28
IEEE	37	2	763	61
ScienceDirect	84	5	0	0
Total	226	17	1107	89

Table 3: Studies by type.

Table 3 presents the results separated by source and type of publication. Note that our protocol focused on selecting only studies published in Journals and Conferences. We also have limited the initial search to studies published since 2008 since Hadoop became a major project at the Apache Software Foundation in 2008. The difference on totals from Table 2 is explained by the Workshop papers, discarded by our research protocol. Table 4 shows the year distribution of the selected papers.

Publication	**Journal**			**Conference**	
Year	ACM	IEEE	SD	ACM	IEEE
2013	0	1	5	2	3
2012	4	0	0	12	17
2011	1	1	0	8	23
2010	3	0	0	4	15
2009	2	0	0	1	3
2008	0	0	0	1	0

Table 4: Studies by year.

With the selection stage complete, each study was analyzed and reviewed, being also classified into one or more categories of our taxonomy, presented in the next section.

4. Contributions to Apache Hadoop and its Ecosystem

Apache Hadoop is best known for being the open source implementation of Google's MapReduce computing model. Being an open source solution, Hadoop attracted the attention of both the academic and industrial communities. Yet, the major reasons for its wide adoption are related to its intrinsic characteristics, namely scalability, fault tolerance, and the ability to perform parallel and distributed computations on low-end computing clusters. This section presents the papers selected via our research protocol, which are somehow related to these Hadoop characteristics.

One of the objectives of this systematic literature review is to propose a taxonomy and classify the related research literature into categories. Our taxonomy creation methodology was based on the project structure proposed by the Apache Hadoop developers. After careful review of the selected papers, we developed our proposed taxonomy, in which we classified the studies into four major categories related to the Hadoop framework architecture: MapReduce, Storage, Ecosystem, and Miscellaneous. The first category, MapReduce, includes studies that develop some solution involving the paradigm and its associated concepts, like scheduling, data flow, and resource allocation. The Data Storage & Manipulation category includes studies comprising HDFS, gathering studies involving storage, replication, indexing, random access, queries, research involving DBMS infrastructure, Cloud Computing, and Cloud Storage. The third category includes studies containing new approaches to the Hadoop Ecosystem, including papers related to both existing and new components created to cope with specific classes of problems. Finally, the last category received studies that did not fit well in the other three categories, including research involving GPGPU, cluster energy management, data security, and cryptography. Our taxonomy was then structured as shown in Figure 1.

Each study was placed into one or more categories, since most of the selected papers involve one or more concepts of Hadoop. These intersecting areas are specially represented in the projection graphs presented in Section 5, which show a relationship among several studies from different categories. Now, we present the main contributions observed in the selected studies, classified according to our taxonomy. Note that some categories were grouped here to promote a better organization and comprehension in this section.

Each of the following subsections presents the concepts and issues involved in key areas, as well the published approaches with enhancements to Apache Hadoop. Readers interested in the details of specific aspects of Hadoop research shall read the corresponding subsections below. Readers interested in having only a general overview of Hadoop research may skip now to the discussion in Section 5.

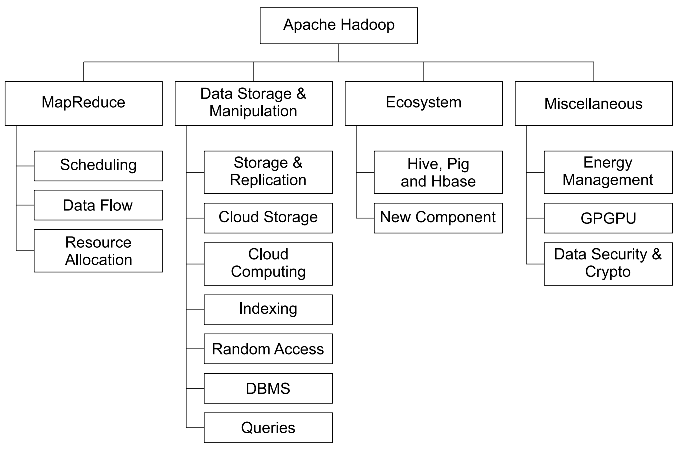

Figure 1: Taxonomy hierarchical organization.

4.1. Scheduling

Scheduling, alongside with storage, is one of the most critical aspects of highly-concurrent parallel systems and was a topic addressed by a large number of the studies we analyzed. Originally, Hadoop comes with three schedulers: FIFO, Fair, and Capacity. The default Hadoop scheduler is the FIFO, where jobs are loaded into a queue and scheduled to execute according to their order in the queue. The Fair Scheduler was originally developed by Facebook and later released to the Hadoop community. It gives a fair share of the cluster capacity over time, thus its name. Jobs are put into allocated pools, which receive a guaranteed minimum number of Map and Reduce slots. Using preemption, the scheduler manages the cluster to assure a fair share of resources to each job. The third scheduler, named Capacity Scheduler, was developed by Yahoo! to guarantee a fair allocation of computational resources to a large number of cluster users. To do so, it uses queues with a configurable number of task (Map or Reduce) slots. Resources are shared among queues according to its priority. Free resources from empty queues are temporarily allocated to other queues. While FIFO was the first and most simple scheduler implemented, the Fair and the Capacity scheduler were created to address the problem of managing multiple users sharing the same resources. We now present schedulers and new approaches designed to enhance the performance or to solve untreated classes of problems in Hadoop.

Regarding the original schedulers, Tan et al. (2012a) propose analytical models for FIFO and Fair schedulers, based on extensive measurements and source code investigations. For a class of heavy-tailed Map service time distributions, authors derive the distribution tail of the job processing delay under the three schedulers. Analytically analyzing the delays under different schedulers for MapReduce, the authors discovered an interesting starvation problem with the widely used Fair Scheduler due to its greedy approach to launch Reduce tasks. To address this issue, the Coupling Scheduler was designed and implemented. This scheduler couples the progresses of Mappers and Reducers and jointly optimize the placements for both of them. This mitigates the starvation problem and improves the overall data locality. Tao et al. (2011) propose an improved FAIR scheduling algorithm, which takes into account job characteristics and data locality. This scheduler kills tasks to free slots for new users. It adopts different policies for I/O- and CPU-bound jobs based on data locality. Results demonstrate that the improved version decreases both data transfer and the execution time of jobs. Similarly, Nguyen et al. (2012) propose a hybrid scheduler based on dynamic priority aimed to reduce the latency for variable length concurrent jobs. This algorithm is designed for data intensive workloads and tries to maintain data locality during job execution.

Other concern regarding schedulers is the heterogeneity of the hardware within clusters, which is discussed by some authors. The LATE (Longest Approximate Time to End) scheduler proposed by Zaharia et al. (2008) addresses the problem of speculative execution of straggling tasks while still concerned with performance. Zaharia et al. (2010) also developed another technique named *delay scheduling*, which intends to address the problem of data locality while keeping fairness during task execution. In the work of Zhao et al. (2012), a new job scheduling based on the Fair scheduler is presented. The scheduler takes into account job characteristics and, similarly to the Fair Scheduler, allocates them into queues. The scheduler manages three queues: one for large jobs, another one for small jobs, and a third one for priority jobs. The authors claim that, compared to the Fair scheduler, tasks of large jobs could be executed with higher parallelism; short jobs will not starve and long jobs can still finish in a reasonable time; and, higher priority jobs that come up may be computed as soon as possible. Ibrahim et al. (2012) developed a scheduling algorithm called Maestro to alleviate the non-local Map tasks execution problem that relies on replica-aware execution of Map tasks. To accomplish this, Maestro keeps track of the chunks and replica locations, along with the number of other chunks hosted by each node. This way, Maestro can schedule Map tasks with low impact on other nodes' local Map tasks execution by calculating the probabilities of executing all the hosted chunks locally.

Lei et al. (2011) propose a novel approach, CREST (Combination Re-Execution Scheduling Technology), which can achieve the optimal running time for speculative Map tasks and decrease the response time of MapReduce jobs. To mitigate the impact of straggler tasks, it is common to run a speculative copy of the straggler task. The main idea is that re-executing a combination of tasks on a group of cluster nodes may progress faster than directly speculating the straggler task on a target node, due to data locality. The evaluation conducted demonstrates that CREST can Reduce the running time of speculative Map tasks by 70% on the best cases and 50% on average, compared to LATE.

Kumar et al. (2012) propose a context-aware scheduler. The proposed algorithm uses the existing heterogeneity of most clusters and the workload mix, proposing optimizations for jobs using the same dataset. Although still in a simulation stage, this approach seeks performance gains by using the best of each node on the cluster. The design is based on two key insights. First, a large percentage of MapReduce jobs are run periodically and roughly have the same characteristics regarding CPU, network, and disk requirements. Second, the nodes in a Hadoop cluster become heterogeneous over time due to failures, when newer nodes replace old ones. The proposed scheduler is designed to tackle this, taking into account job characteristics and the available resources within cluster nodes. The scheduler uses then three steps to accomplish its objective: classify jobs as CPU or I/O bound; classify nodes as Computational or I/O good; Map the tasks of a job with different demands to the nodes that can fulfill the demands. Chen et al. (2010) propose another approach for heterogeneous environments. The SAMR (Self-Adaptive MapReduce) scheduling algorithm improves MapReduce by saving execution time and system resources. On MapReduce, slow tasks prolong the execution time of an entire job. In heterogeneous clusters, nodes require different times to accomplish the same tasks due to their differences, such as computation capacities, communication, architectures, memory, and power. The scheduler uses historical information of each cluster node to tune parameters and discover slow tasks. This way, the scheduler is able to classify certain nodes as slow, launching backup tasks using the data locality principle.

A different approach to heterogeneity is presented by Rasooli and Down (2011), which propose a Hadoop scheduling algorithm that uses system information such as estimated job arrival rates and mean job execution times to make scheduling decisions. Using system information, the algorithm classifies jobs into classes and matches them with available resources. Priority, minimum share required, and fair share of users are also considered when making a scheduling decision. The proposed algorithm is dynamic and updates its decisions based on changes in these parameters. Another approach was proposed by You et al. (2011), a Load-Aware scheduler (LA scheduler) used in heterogeneous environments with dynamic loading. The scheduler consists of a data collection module, which gathers the system-level information of the TaskTrackers periodically, and a task assignment module, which makes scheduling decisions according to the information previously collected.

Polo et al. (2010) implemented an application-centric task scheduler to predict the performance of concurrent MapReduce jobs dynamically and adjust resource allocation for them. It uses estimates of individual job completion times given a particular resource allocation. The scheduler tries to maximize each job's chance of meeting its performance goal by dynamically estimating the completion time for each MapReduce job. It benefits from MapReduce jobs composition of a large number of tasks, which is known in advance, during the job initialization phase, and from the job progression that can be observed during runtime. Similarly, Tang et al. (2012) propose an algorithm to satisfy the users job deadline requirements in the cloud environment. The MTSD (MapReduce Task Scheduling for Deadline) algorithm uses the data locality and cluster heterogeneity information to classify nodes and improve the Map tasks data locality. Authors also present a task execution model based on the node classification algorithm. Differently from Polo *et al.*, the authors consider Map and Reduce tasks differently, since their execution times are not correlated, and could not be accurate to compute average task execution time using Map and Reduce tasks together.

Ahmad et al. (2012) propose an implementation called Tarazu, comprising a suite of optimizations to improve MapReduce performance on heterogeneous clusters. The proposed optimizations are a communication-aware load balancing scheme of Map computations across the nodes, a communication-aware scheduling of Map computations to avoid bursty network traffic, and a predictive load balancing of Reduce computations across the nodes. Finally, the authors also propose online measurement-based heuristics to estimate the information needed for making application- and cluster-aware decisions. Focusing on the same problem, Wang et al. (2012) propose Predator, an experience guided configuration optimizer, which uses a MapReduce cost model to estimate jobs execution times and classifies MapReduce parameters into groups by their different tunable levels, using the most adequate to achieve better performance.

Concerning not only heterogeneous environments but also heterogeneous MapReduce workloads, Tian et al. (2009) propose another dynamic scheduler integrated to Hadoop. The main goal is to improve the hardware utilization rate when different MapReduce workloads run on the cluster. Originally, Hadoop schedulers are not aware of differences among MapReduce workloads, e.g., CPU-bound or I/O-bound workloads. The proposed scheduler studies as a workload prediction mechanism, distributing workloads into three queues: CPU-bound, I/O-bound, and Wait-queue. The authors proposed an analytical model to calculate and classify the workloads at runtime. First, new jobs are put into the waiting queue. Next, the scheduler assigns one Map task to every TaskTracker for predicting the job type. An analytical model is used to classify the job and allocate it to the other two queues to finish its execution. The triple-queue scheduler can balance the usage of both CPU and disk I/O, improving Hadoop throughput by about 30% under heterogeneous workloads.

Data locality as a performance issue has also been recently studied in Hadoop. He et al. (2011a) propose a new scheduler with the premise that local Map tasks are always preferred over non-local Map tasks, no matter which job a task belongs to. A locality marker is used to classify nodes and to ensure each node has a fair chance to grab its local tasks. To accomplish this, the scheduler relaxes the job order for task scheduling. Doing so, it improves performance by avoiding data transfer in Map tasks, which may degrade job execution performance. This scheduler also tries to guarantee, besides high data locality, high cluster utilization. This is achieved by minimizing data transfer so nodes can maximize the use of its resources for computation. Zhang et al. (2011c) propose a scheduling method called next-k-node scheduling (NKS) that improves the data locality of Map tasks. The scheduler calculates a probability for each Map task in a job, generating low probabilities for tasks that have its input data stored on the next node. Following, the method preferentially schedules the task with the highest probability. Thus, it reserves tasks with lower probabilities to the nodes holding their input data, improving data locality.

Scheduling Map tasks on a Hadoop instance deployed in a heterogeneous environment may degrade system performance. This happens because the schedulers may not be able to Reduce the occurrence of Map tasks not scheduled to the nodes storing the input data. Zhang et al. (2011b) propose a data-locality-aware scheduling method to address this problem. After receiving a compute request from an idle node, the method preferably schedules the task whose input data is stored on this requesting node. If there are no such tasks, it selects the task whose input data is nearest to the requesting node. The scheduler also makes a decision on whether to reserve a certain task for the node storing its input data or scheduling it on a requesting node by transferring its input data on the fly. Although the former decision may improve data locality, it may incur on runtime overhead, e.g., waiting time. This overhead may also occur when adopting the latter decision, when the transmission time for copying the input data to the requesting node may overcome the waiting time.

The Hadoop original schedulers neither exploit data locality nor addresses partitioning skew (the case where the computational load is unbalanced among Map and/or Reduce tasks, generally causing performance degradation) present in some MapReduce applications. Hammoud and Sakr (2011) present another approach discussing the data locality problem. It deals specifically with Reduce tasks. The Reduce phase scheduling is modified to become aware of partitions, locations, and size, to decrease network traffic. The scheduler, named Locality-Aware Reduce Task Scheduler (LARTS), uses a practical strategy that leverages network locations and sizes of partitions to exploit data locality. LARTS attempts to schedule Reducers as close as possible to their maximum amount of input data and conservatively switches to a relaxation strategy seeking a balance among scheduling delay, scheduling skew, system utilization, and parallelism. The work of Zhang et al. (2012a) also deals with the Reduce tasks data locality. The authors propose a two-phase execution engine of Reduce tasks to cope with massive remote data access delays that may degrade system performance. The degradation is related to massive remote I/O operations to copy the intermediate results of Map tasks. In the first phase, the engine selects the nodes to run Reduce tasks and then informs the selected nodes to prefetch intermediate results for Reduce tasks. In the second phase, the selected nodes allocate computing and memory resources to execute the Reduce tasks.

Hammoud et al. (2012) propose another approach named Center-of-Gravity Reduce Scheduler (CoGRS). The work designs a locality-aware, skew-aware Reduce task scheduler for saving MapReduce network traffic. The proposed scheduler attempts to schedule every Reduce task at its center-of-gravity node determined by the network locations. By scheduling Reducers at their center-of-gravity nodes, they argue for decreased network traffic, which may possibly allow more MapReduce jobs to co-exist on the same system.

Seo et al. (2009) present a prefetching and a pre-shuffling scheme that can improve the overall performance in shared MapReduce computation environments. The prefetching scheme exploits data locality, while pre-shuffling is designed to reduce the network overhead required to shuffle key-value pairs. The proposed prefetching scheme is subdivided in two types. First, intra-block prefetching, which prefetches data within a single block while performing a complex computation. Second, inter-block prefetching runs in the block level, by prefetching an entire block replica to a local rack. In the pre-shuffling scheme, the task scheduler looks into the input splits of the Map phase and predicts how to partition the key-value pairs considering the Reducer locations. The expected data are assigned to a Map task near the future Reducer before the execution of the Mapper. The proposed schemes are implemented in HPMR (High Performance MapReduce Engine), as a plug-in type component for Hadoop.

Task splitting is another relevant issue around scheduling.

Guo et al. (2011) propose a mechanism to dynamically split and consolidate tasks to cope with load balancing and break through the concurrency limit resulting from fixed task granularity. The default strategy of Map operation organization in Hadoop is that each Map task processes key-value pairs contained in one block. The sizes of key-value pairs may vary so that the numbers of key-value pairs stored in different blocks may differ. The proposed task splitting tackles these problems. For single-job scheduling, Aggressive Scheduling with Prior Knowledge and Aggressive Scheduling were proposed for both cases, first with prior knowledge and then without it. For multi-job scheduling, the authors present the Overlapped Shortest-Job-First Scheduling, which invokes the basic short-job-first scheduling algorithm periodically and schedules all waiting jobs in each cycle. Combined with a task splitting mechanism, it gives an optimal average job turnaround time if tasks are arbitrarily splittable.

Finally, Inter-job parallelism may also be interesting to enhance MapReduce performance, although its data flow was not originally designed to do it. All intermediate results belong to the jobs that have created and used them. One relatively common situation happens when multiple jobs access the same file. Nykiel et al. (2010) propose a module named MRShare to approach this problem. MRShare transforms a batch of queries into a new batch that is executed more efficiently, by merging jobs into groups and evaluating each group as a single query. Merging multiple jobs allows the entire batch of jobs to be analyzed to maximize the degree of resource sharing, minimizing resource consumption. Another approach is proposed by Shi et al. (2011). The Shared Scan Scheduler, named S^3 and developed as a plugin to Hadoop, shares the scanning of a common file for multiple jobs. The scheduler is designed to deal with jobs arriving at different times, processing them as early as possible. This is the main difference between the two approaches: MRShare operates before the job execution starts and S^3 processes jobs at different times. Both approaches can enhance the performance of Hadoop when dealing with single data being used by multiple jobs.

4.2. Data Flow

The data processing strategy employed by MapReduce consists of two primitive functions: Map and Reduce. Behind this simple abstraction is a single fixed data flow. A MapReduce job is divided into Map and Reduce tasks, and assigned to idle slots of workers according to these two stages. Thus, there are two types of workers, Mappers to Map tasks and Reducers to Reduce tasks. In the beginning of the Map phase, the input data is loaded into HDFS. To ensure fault tolerance, the input data are partitioned into equal sized blocks and replicated according to a replication factor. Each block will be processed by a Mapper, resulting in intermediate outputs, which are locally sorted, optionally combined from key-value pairs sharing the same key, and stored in local disks of the Mappers. The Reduce phase starts as soon as there are enough Map outputs to start a Reduce task. In this moment, the scheduler assigns Reduce tasks to workers. The data transfer is performed by each Reducer that pulls and shuffles intermediate results using a one-to-one shuffling strategy. Reducers are responsible to read intermediate results and merge them to group all values with the same keys. Subsequently, each Reducer applies Reduce to the intermediate values considering these keys to produce the final output that is stored in HDFS.

Leo and Zanetti (2010) implemented a solution to make Hadoop available to Python programmers called Pydoop. A Python package based on CPython provides an API for MapReduce and HDFS. This works as an alternative to Hadoop Streaming or Jython. Hadoop Streaming uses a communication protocol to execute a Python script as the Mapper or Reducer via the standard input and output. Therefore, it cannot process arbitrary data streams, and the user directly controls only the Map and Reduce parts, except for HDFS operations.

There are some papers that alter the Hadoop MapReduce data flow to improve performance (Wang et al., 2011; Vernica et al., 2012; Ho et al., 2011; Ibrahim et al., 2010; Kwon et al., 2012; Verma et al., 2011, 2012; Zhu and Chen, 2011; Lin et al., 2010; Ahmad et al., 2013) or to augment features to meet specific requirements (Elnikety et al., 2011; Li et al., 2011; Bu et al., 2012; Elteir et al., 2010; Grover and Carey, 2012; Laptev et al., 2012; Bhatotia et al., 2011; Zhang et al., 2011d). Wang et al. (2011) propose Hadoop-A, an acceleration framework that optimizes the efficiency of Hadoop using plugin components for fast data movement. It addresses the performance issues in multiple ways. First, the authors developed a novel merge algorithm that avoids multiple rounds of disk accesses to merge the same intermediate data segments from Map tasks. Second, the original architecture of Hadoop ensures the correctness of the two-phase MapReduce protocol by forbidding Reduce tasks to start before all intermediate data have been merged together. This results in a serialization barrier that significantly delays the Reduce phase. This barrier is broken by a full redesigned pipeline of shuffle, merge, and Reduce phases for Reduce tasks. In this pipeline, Map tasks map data splits as soon as they can. Finally, they propose a novel algorithm that merges data without the use of disks and enables data movement via RDMA (Remote Direct Memory Access). Using these techniques, Hadoop-A is capable of increasing the throughput of Hadoop and reduce the CPU utilization.

Similarly, the proposal of Vernica et al. (2012) also describes solutions to improve Hadoop's performance. Differently from Wang *et al.*, who overlap the Shuffle, Merge, and Reduce phases, Vernica *et al.* focus on the interaction of Mappers, introducing an asynchronous communication channel between Mappers. In the current implementation of Hadoop, Mappers are completely independent. Using a transactional distributed meta-data store (DMDS), Mappers can post metadata about their state and check the state of all other Mappers. The authors argue that this "situation-aware Mappers" (SAMs) make Hadoop more flexible and adaptive, since optimizations can be done based on the Mappers global state. SAMs are used in a number of adaptive techniques: Adaptive Mappers (AM) dynamically control the checkpoint interval, Adaptive Combiners (AC) use best-effort hash-based aggregation of Map outputs, Adaptive Sampling (AS) uses some early Map outputs to produce a global sample of their keys, and Adaptive Partitioning (AP) dynamically partitions Map outputs based on the sample.

The solution of Wang *et al.* focuses on tasks executed by Reducers while the work of Vernica *et al.* focuses on tasks executed by Mappers. Similarly, Ahmad et al. (2013) proposed MaRCO (MapReduce with communication overlap), which is directed to the overlapping of the Shuffle with the Reduce computation. The original Hadoop data flow was modified allowing the operation of Reduce tasks on partial data. MaRCO breaks Reduce into many smaller invocations on partial data from some map tasks, and a final reducing step re-reduces all the partial reduce outputs to produce the final output. Lin et al. (2013) have proposed an overlapping model between map and shuffle phases. The approach is based on two complementary scheduling algorithms called MaxSRPT and SplitSRPT. MaxSRPT minimizes the average response time of the queue, while SplitSRPT addresses the poor performance of MasSRPT when jobs are more unbalanced. Moreover, this study presents an analytical model proving that the problem of minimizing response time in the proposed model is strongly NP-hard. Yet on the data flow modifications Mohamed and Marchand-Maillet (2013) have proposed to change the Hadoop data flow by using MPI to overlap Map and Reduce phases. Thus, Map and Reduce phases are executed in a concurrent parallel manner by exchanging partial intermediate data through a pipeline provided by MPI. In the proposed model Map and Shuffle phases are merged and work as a single phase. The authors have also proposed a scheduler to improve the performance of the prototype. The work of Xie et al. (2013) uses a preshuffling approach to reduce the network overload imposed by shuffle-intensive applications. To accomplish this, a push model using in-memory buffer and a 2-stage pipeline in the preshuffling scheme to exchange partial data between map and reduce tasks are implemented.

Ho et al. (2011) and Ibrahim et al. (2010) concentrate their efforts on improving performance by changing the data flow in the transition between Mappers and Reducers. Originally, Hadoop employs an all-to-all communication model between Mappers and Reducers. This strategy may result in saturation of network bandwidth during the shuffle phase. This problem is known as the Reducers Placement Problem (RPP). Ho *et al.* modeled the traffic in a multiple-racks environment. Two algorithms and an analytical method were proposed as a solution to the RPP. The approach uses optimization techniques to formulate the problem. They developed a greedy algorithm to find the optimal solution for the problem. Ibrahim *et al.* address the problem of how to efficiently partition the intermediate keys to decrease the amount of shuffled data. This guarantees fair distribution of the Reducers' inputs, improving the overall performance. The correct partition of the intermediate key may also solve the RPP. The locality-aware and fairness-aware key partitioning (LEEN) algorithm was developed to decrease partitioning skew, reducing also data transfer while balancing the data distribution among nodes. LEEN improves the data locality of the MapReduce execution efficiency with the use of an asynchronous Map and Reduce scheme.

On the other hand, instead of considering the lack of flexibility of data flow as the main problem, Zhu and Chen (2011) and Kwon et al. (2012) change the data flow to solve problems that degrades the Hadoop performance. Zhu and Chen propose two mechanisms to cope with the problem of detection of the failed worker. The proposed Adaptive Interval tries to configure dynamically the expiration time, which is adaptive on the job size. In turn, the Reputation-based Detector tries to evaluate the reputation of each worker. Once the reputation of a worker is lower than a threshold, the worker will be considered as a failed worker. The skew problem is characterized when the Reduce phase cannot start until straggling Map tasks have been completed. Kwon *et al.* present a system called SkewTune, which mitigates skew due to an uneven distribution of data between Map and Reduce phases. It also mitigates skew due to some subsets of the data taking longer to process than others.

Lin et al. (2010) proposed a framework called MOON to improve Hadoop performance introducing several modifications. MOON uses a hybrid resource architecture that comprises a set of dedicated reliable computers to overcome higher rates of node unavailability in volunteer computing systems, and adaptive task and data scheduling algorithms to offer reliable MapReduce services. The main problems tackled by MOON are the prohibitively high replication cost of HDFS to provide reliable storage in volatile systems, the lack of replication of intermediate outputs produced by Map tasks resulting in task reexecution, and the inability of schedulers to handle suspended or interrupted tasks on volunteer computing systems.

Hadoop was not originally developed to support iterative computing, which represents an important class of applications. Machine learning, among other data analysis algorithms and techniques, makes use of iterative computing to obtain results. One example is the Apache Mahout library, which is meant to develop scalable machine learning applications using collaborative filtering, recommenders, clustering, and others. Mahout is able to promote iterative computing by grouping together a series of chained jobs to obtain the results. The results of each job are fed into the next chained job until final results are obtained. In the MapReduce paradigm, each iteration must wait until the previous one finishes completely and have its output entirely written and committed to the underlying file system. Elnikety et al. (2011) proposed iHadoop. It modifies the data flow techniques and task scheduling to make them aware of the nature of iterative computations. The framework tries to achieve better performance by executing iterations asynchronously, where an iteration starts before its preceding iteration finishes. This way, outputs of iterations (as they progress) are fed to the following ones, allowing processing their data concurrently.

In the same manner, Liang et al. (2011) propose a Hadoop extension called Dacoop to cope with data-iterative applications. However, Dacoop uses cache mechanisms to treat the repeatedly processing of data shared among jobs. Dacoop extends the MapReduce programming interface by introducing the shared memory-based data cache mechanism and caching the data on the file split level, providing the extended programming interface with the explicit specification of the repeatedly processed data files and the data caching management model, and adopting a data-caching-aware task scheduling, which schedules tasks following cache-level and disk-level data locality.

Similarly, the HaLoop approach (Bu et al., 2012) proposes a new system that modifies Hadoop to support large-scale, iterative data analysis applications. Although built on top of Hadoop, HaLoop proposes a new loop-aware task scheduler. This, together with loop-invariant data caching, improves the performance of iterative jobs. MapReduce frameworks generally are capable of performing large-scale data processing in a single pass and are not suited to iterative programming. Two known examples that may obtain better performances using such approaches are page ranking and social graph analysis.

Bhatotia et al. (2011) propose a MapReduce framework called Incoop to process datasets that evolve over time. The work of Bhatotia *et al.* changes the Reduce phase of the Hadoop data flow introducing a contraction phase, more precisely altering the Combiner functions. This phase is responsible for controlling the granularity of tasks, dividing large tasks into subtasks to reuse them appropriately during iterations. However, different from other approaches, Bhatotia *et al.* do not make use of cache mechanisms between processing iterations; instead, Incoop runs a daemon process on the NameNode machine that acts as a memoization server that stores intermediate results. Another key point on the work of Bhatotia *et al.* is Inc-HDFS, a modified version of HDFS that identifies similarities in input data of consecutive job runs splitting the input into chunks based on their content instead of using a fixed-size.

Zhang et al. (2011d) propose a framework called iMapReduce that uses Hadoop to process structured data iteratively. iMapReduce tackles some Hadoop problems to process iterative computation: the waste of resources to create, schedule, and destroy jobs that perform constant functions in each iteration; the performance penalty to load and shuffle immutable data that remains the same through the iterations; and the serial execution of jobs in each iteration, resulting in synchronism in Map and Reduce tasks. iMapReduce introduces the concept of persistent tasks to cope with the problem of waste of resources, avoiding unnecessary creation, scheduling, and destruction of tasks. This mechanism also avoids repeatedly data load and shuffle operations between iterations. To break the synchronism and allow the execution of Map tasks as soon as possible, iMapReduce implements a persistent socket connection, keeping alive communication between tasks to store transient data.

Notwithstanding the importance of improving the performance of Hadoop, some studies change the data flow to meet specific requirements. Elteir et al. (2010) propose modifications to start the Reduce phase before the end of the Map phase, to cope with a specific class of problems called recursively reducible MapReduce jobs. Problems of this category do not impose synchronization on the processing of data. The authors present two different approaches to cope with the problem by changing the data flow in the Reduce phase. First, the Reduce phase is executed hierarchically after a number of Map tasks have been completed. In the second approach, a predefined number of Reduce tasks incrementally process records collected from Map tasks. Grover and Carey (2012) and Laptev et al. (2012) focus on sampling issues of workflows. Grover and Carey's approach provides ways to sample a massive dataset to produce a fixed-size sample whose contents satisfy a given predicate. The proposed model allows data to be incrementally processed. This gives the job the ability to control its growth. Laptev *et al.* propose the incremental computation of early results for arbitrary workflows estimating the degree of accuracy achieved in the computation. The proposal provides approximate results based on samples in advanced analytical applications on very massive datasets with the objective of satisfying time and resource constraints.

Finally, Verma et al. (2011) propose a framework called ARIA that, given a job completion deadline, is capable of dynamically allocating the appropriate amount of resources to the job so that it meets the required Service Level Objective (SLO). ARIA builds a job profile from a job that is routinely executed on a new dataset. Using the profile and the SLO-based scheduler, ARIA can estimate the amount of resources required for job completion and it determines job ordering and the amount of resources to allocate to meet the job deadlines. Verma et al. (2012) evolved ARIA to enhance workload management decision in jobs with deadlines. That evolution includes three complementary mechanisms: an ordering policy for the jobs in the processing queue based on the EDF policy (Earliest Deadline First); a mechanism for allocating a tailored number of Map and Reduce slots to each job with a completion time requirement; and a mechanism for allocating and deallocating spare resources. The authors implemented a deadline-based scheduler which integrates all the three mechanisms.

4.3. Storage & Replication

The Hadoop Distributed File System is the block storage layer that Hadoop uses to keep its files. HDFS was designed to hold very large datasets reliably using data replication (Shvachko et al., 2010). This allows HDFS to stream large amounts of data to user applications in a reasonable time. Its architecture is composed of two main entities: NameNode and DataNodes, which work in a master-slave fashion. NameNode is responsible for keeping the metadata about what and where the files are stored in the file system. DataNodes are responsible for storing the data itself. HDFS works as a single-writer, multiple-reader file system. When a client opens a file for writing, it is granted a lease for the file and no other client can write to the file until the operation is complete. Additionally, after the file is closed, the bytes written cannot be altered or removed except that new data can be added to the file by reopening the file for append. HDFS have received several contributions that implements enhancements so that it can be used in different type of approaches in MapReduce computations.

A particular study that contributes in different ways to the framework was developed specifically to improve Hadoop's performance. The work of Jiang et al. (2010) presents a MapReduce performance study, using Hadoop as basis to tackle its performance bottlenecks. Jiang *et al.* propose known alternative methods as solutions to tuning MapReduce performance. They enhanced the way a reader retrieves data from the storage system with a direct I/O support, which outperforms streaming I/O by 10%. They implemented a simple range-indexing scheme for sorted files, improving the Hadoop performance. Finally, the authors also proposed an optimization to the HDFS to deal with small files; HDFS may have loss of performance when dealing with a large group of small files due to its strategy of keeping all metadata in the master node memory. This approach allows DataNodes to save some metadata of small files in their memory, improving performance when dealing with small files. Similarly, Shafer et al. (2010) analyze the HDFS performance bottlenecks under concurrent workloads. The authors claim that HDFS performance can be improved using application-level I/O scheduling and still preserve the portability. Authors also explore solutions like pre-allocating file space on disk, adding pipelining and prefetching to both task scheduling and HDFS clients, and modifying or eliminating the local file system as a way to improve HDFS performance. But, since portability is a project premise in Hadoop, some of these changes may not be fully convenient, because of the portability reduction they may cause.

Being designed to store and keep large files/datasets, Dong et al. (2010) propose a novel approach to improve the efficiency of storing and accessing small files on HDFS. In this approach, characteristics of file correlations and access locality of small files. The approach is addressed to solve the small file problems of a specific resource sharing system, to store and share courseware majorly in the form of presentation files and video clips. Performance is also affected by file correlations for data placement, and without a prefetching mechanism for reads, a considerable overhead may be generated.

Focusing on the same problem of correlated data placement, Eltabakh et al. (2011) propose CoHadoop, a extension of Hadoop that allows applications to control where data are stored. CoHadoop addresses Hadoop's lack of ability to collocate related data on the same set of nodes. The approach is designed such that the strong fault tolerance properties of Hadoop are retained. Colocation can be used to improve the efficiency of many operations, including indexing, grouping, aggregation, columnar storage, and joins. Additionally, the authors propose efficient Map-only algorithms that exploit collocated data partitions. Exploiting data placement on HDFS, Xie et al. (2010) propose a new strategy to HDFS running on a heterogeneous cluster. The approach focuses on distributing a large dataset to the nodes according to the computing capacity of each one. Two algorithms were implemented and incorporated into HDFS. The first one initially distributes the file blocks from a dataset to the cluster nodes. The second data placement algorithm is used to solve data skew problems, reorganizing the file blocks distribution along the cluster.

Hadoop may work using a replacement file system, when changes in HDFS are not possible or practicable to be made. Mikami et al. (2011) proposes the use of a new file system named GFarm. It is POSIX compliant and uses data locality, which makes it suitable to be used on Hadoop. GFarm can also be used to run MPI applications. Thus, the same data used on MapReduce applications can be used on different applications such as POSIX compliant and MPI applications. This would not be possible when using HDFS without generating extra copies of these data. To integrate GFarm into Hadoop, a plugin named Hadoop-GFarm was designed, implemented, and tested with MapReduce applications.

Concerned with Byzantine fault-tolerance, Costa et al. (2013) implemented a new Hadoop version incorporating Byzantine fault-tolerance to MapReduce. Initially, the approach runs $f + 1$ map tasks, being f the maximum number of faulty replicas. This was the minimum number of replicas the authors reached by considering the expected low probability of arbitrary faults. The model also achieves better performance by using speculative execution of reduce tasks. Although the resources used practically doubles in this approach, this cost may be acceptable for a large number of applications handling critical data.

The use of Solid State Disks (SSDs) as a storage solution is increasing as the cost/MB is decreasing. The use of these technologies on clusters is such a recent trend in industry that the first solutions regarding its use on MapReduce clusters start to surface. Most of the research up to date tends to analyze whether MapReduce can benefits in terms of performance when deploying HDFS on SSDs. The work of Jeon et al. (2013) analyzes the Flash Translation Layer (FTL) – the core engine for the SSDs – to understand the endurance implications of such technologies on Hadoop MapReduce workloads. As a result, the research presents the behavior of SSD for Hadoop-based workloads including wear-leveling details, garbage collection, translation and block/page mappings. The research of Kang et al. (2013) explores the benefits and limitations of in-storage processing on SSDs (the execution of applications on processors in the storage controller). This approach benefits from characteristics such as high performance on concurrent random writes, powerful processors, memory, and multiple I/O channels to flash memory provided by the SSDs, enabling in-storage processing with small hardware changes. The authors implemented a model (Smart SSD model) that uses an object-based protocol for low-level communication with the host, extending the Hadoop MapReduce framework to support a Smart SSD. Experiments shows benefits such as increase of performance and reduce of total energy consumption.

4.4. Cloud Computing

Although designed to be applied on distributed environments, Hadoop was not originally conceived to work using Cloud Computing concepts. In this sense, some approaches try to cope with these needs. Regarding Cloud Computing management, Kousiouris et al. (2011) design a data management system to enable Storage as a Service transparently using Hadoop. To do so, the authors developed a new file system, ODFS (OPTIMIS Data Manager and Distributed File System), using HDFS as a base. The file system adds RESTful interfaces to expose critical functionalities from Hadoop as services and extends HDFS. These components are used to enrich security, logging, and data analysis features. Components also provide data access compatibility between federated Clouds. The main contribution resides in the fact that ODFS enables suitable interfaces with the external infrastructure providers to launch storage VMs and extend the distributed file system, creating a transparent cloud federation to the user. Kondikoppa et al. (2012) designed and implemented a network-aware scheduler to be used on federated clusters, improving the map tasks scheduling in such environment and, consequently, achieving better performance. Using the previously presented GFarm file system, Wang et al. (2013) have proposed G-Hadoop, a MapReduce framework that enables large-scale distributed computing across multiple clusters. It replaces HFDS for the Gfarm file system and schedules tasks across multiple nodes of multiple clusters controlled by different organizations.

To provide a Byzantine Fault Tolerance (BFT) model into Hadoop, Clement et al. (2009) propose UpRight-Zookeeper and UpRight-HDFS using the Zookeeper (a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services) and HDFS open source code bases. Zookeeper was modified, adding authenticators to messages and to send/receive them to/from the right quorums of nodes. These modifications allow Zookeeper to protect its clusters against a wider range of faults. Concerning HDFS, it was modified to remove its single points of failure. UpRight-HDFS also provides end-to-end BFT against faulty clients, DataNodes, and NameNodes.

Data transfer between nodes in a Cloud Computing environment may consume large bandwidth and time. The use of such resources represents an elevation of cost to users. An approach to solve these problems, HadoopRSync, is presented by Zhang et al. (2011a). Such approach uses an incremental update derived from the original RSync, which minimizes the data transferred by transmitting only an encoding of the changes to the file instead of the whole file itself. An upload algorithm to update files on Hadoop and a download algorithm to renew files on user side composes HadoopRsync. The asymmetric scheme is employed because file synchronization in both directions have different characteristics. This approach seems to be suitable for file hosting services using HDFS, which can lower the bandwidth usage and speed up file updates. Following the cloud data transfer problem and concerned with the cloud intermediate data problem, Ko et al. (2010) implement a new storage system (ISS - Intermediate Storage System) that implements three techniques designed to store cloud intermediate data while maintaining performance. In a summarized view, the three techniques consist of an asynchronous replication of intermediate data, a replication of selective intermediate data, and, finally, the exploitation of bandwidth heterogeneity of datacenter topology applied to the cloud environment. The system replicates intermediate data from Map and Reduce outputs preventing re-execution of multi-stage applications.

Regarding cloud storage and data availability, QDFS (Guang-hua et al., 2011) is an HDFS replacement that employs a backup policy based on recovery volumes and a quality-aware DataNode selection strategy to store files. The idea is to develop a reliable file system under unstable network environments. HDFS assumes that each DataNode have analogous storage characteristics and all DataNodes are connected within a reliable network. Therefore, to store data blocks, HDFS select DataNodes that have enough storage space to ensure that all DataNodes have balanced workload. The QDFS approach selects DataNodes based on their quality of service (QoS), which is calculated using characteristics such as transfer bandwidth, availability of service, and free storage space. QDFS enforces a data redundancy policy based on recovery volumes, reducing the used storage space. It also evaluates the QoS of a DataNode dynamically, making it more suitable for dynamic network environments.

Yet on availability and replication, Wei et al. (2010) present a cost-effective dynamic replication management scheme referred to as CDRM to provide cost-effective availability and improve performance and load balancing of cloud storage. The work addresses two issues: how many minimal replicas should be kept in the system to satisfy availability requirements and how to place these replicas to effectively distribute workloads over the data nodes. A novel model is proposed to capture the relationship between availability and number of replicas, and the replica placement is based on capacity and blocking probability of data nodes. By adjusting the replica label and location according to workload changes and node capacity, CDRM can dynamically redistribute workloads among data nodes in a heterogeneous cloud.

Resource virtualization is a strong trend industry. The work of Mao et al. (2011) proposes a system named EHAD (Elastic Hadoop Auto-Deployer) that integrates and shares resources among users on a Hadoop cluster. Virtualization technologies are used to gather physical resources and allocate virtual machines to users. The authors designed a request handler to deploy multithreaded virtual machine nodes for users. Cluster resources such as computing power, storage, and network composes a resource pool. This pool is allocated for user virtual machines on demand. The created environment automatically deploys the Hadoop framework according to the needs described on user requests. This is a similar approach to Hadoop On Demand (HOD), which is a system for provisioning and managing independent Hadoop MapReduce and Hadoop Distributed File System (HDFS) instances on a shared cluster environment. The main difference relies on the fact that the EHAD approach claims to create, deploy, and destroy Hadoop environments automatically and on demand.

A similar approach is presented in the work of Mandal et al. (2011). The difference here is the use of Hadoop clusters across multiple cloud domains. Hadoop clusters are created on-demand and are composed of virtual machines from different available clouds. The concept of virtual pools of heterogeneous resources like computing power, storage, and networks orchestrated through a common interface is also used here. This way, cloud-integrated Hadoop workflows read and write data repositories co-located with cloud sites out in the network. The prototype was developed using a cloud framework called Open Resource Control Architecture (ORCA), which is responsible for managing available resources. The prototype is capable of providing sets of resources from multiple cloud and network domains and runs Hadoop automatically in these resources. This approach may also be considered similar to the Hadoop On Demand. The difference here is the use of multiple cloud domains, instead of a single local cluster.

AROMA (Lama and Zhou, 2012) is another approach to automated allocation of heterogeneous Cloud resources. The system also configures Hadoop parameters for achieving quality of service goals while minimizing the incurred cost. AROMA addresses the challenge of provisioning ad-hoc jobs that have performance deadlines in Clouds through a novel two-phase machine learning and optimization framework.

Data transfer in virtualized environments is addressed in the work of Lee et al. (2013), which propose an adaptive data transfer algorithm in virtual MapReduce clusters. The algorithm is capable of moving files across different virtual machines located at the same physical machine without any network transfers between virtual machines. The work is specially useful to transfer the output of each map task to the appropriate reducer without using network bandwidth, which reduces overall job completion time.

Park et al. (2012) propose an important contribution to virtualization. They developed a dynamic virtual machine reconfiguration technique (DRR - Dynamic Resource Reconfiguration) for distributed data-intensive platforms on virtualized cloud environments running Hadoop. The dynamic reconfiguration of virtual machines improves the overall job throughput by improving the input data locality of a virtual MapReduce cluster. DRR temporarily increases the number of cores to virtual machines to run local tasks. It also schedules tasks based on data locality and adjusts the computational capability of the virtual nodes to accommodate the scheduled tasks. The idea is that different resource requirements by different tasks or jobs may cause each virtual node to under-utilize its resources. With the virtual machine reconfiguration, each node can be adjusted to provide only the necessary amount of resources demanded from the node.

Related to Cloud Computing, the approach of He et al. (2012) runs Hadoop MapReduce distributed on the Open Science Grid, which uses multiple clusters geographically distributed. Hadoop on the Grid claims to provide elastic MapReduce environment on opportunistic resources available. The goal is to improve Hadoop fault tolerance for wide area data analysis by mapping the distributed data centers to virtual racks, transparent to MapReduce applications.

4.5. DBMS, Indexing, Queries, and Random Access

Although HDFS shows good performance on scalability, fault tolerance, high throughput, and flexibility to handle unstructured data, it presents drawbacks in some contexts when compared with Data Base Management Systems (DBMS) and other approaches. Several studies intend to improve Hadoop's performance altering or replacing HDFS with other solutions as presented before. Some approaches intend to develop indexing, query processing, hybrid solutions using DBMS and structured data processing inside Hadoop. Further, most studies present advanced solutions comprising more than one of these strategies as follows.

Liao et al. (2010) and Dittrich et al. (2012) propose improve Hadoop's performance by means of a complete new indexing manner, while An et al. (2010) and Dittrich et al. (2010) suggest alter existing indexing mechanisms as their strategy. Liao et al. (2010) propose the use of built-in hierarchical indexing to support complex type queries in HDFS. The method of hierarchical structures is applied to both B-tree, R-tree and their variants to optimize queries. Several enhancements of index structure with respect to node size, buffer strategy, and query processing were developed using properties of HDFS. Dittrich et al. (2012) propose an enhancement of HDFS and Hadoop MapReduce that improves runtimes of several classes of MapReduce jobs. The approach, named HAIL (Hadoop Aggressive Indexing Library), changes the upload pipeline of HDFS in order to create different clustered indexes on each data block replica. HAIL keeps the existing physical replicas of an HDFS block in different sorted orders and with different clustered indexes. Hence, for a default replication factor of three at least three different sort orders and indexes are available. Thus, the likelihood to find a suitable index increases, improving the runtimes for workloads.

In the system proposed by An et al. (2010), a global index access mechanism was adopted. The B+-tree index data is stored in HDFS and distributed across all the nodes. Thus, the index access is parallelized following the MapReduce execution style. Part of the work of Dittrich et al. (2010) comprises a Trojan index used to co-partition the data at load time. This indexing technique is the used solution to integrate indexing capability in a non-invasive and DBMS-independent manner.

Indeed, Hadoop++ (Dittrich et al., 2010) concentrates on query processing, particularly query execution plan. The proposed system boosts task performance without changing the Hadoop. To reach this goal, ten UDFs (User Defined Functions) were injected into Hadoop source code, affecting it from inside. Thus, Hadoop++ do a hard-coded query processing pipeline explicit and represent it as a DB-style physical query execution plan.

The HDFS has been designed originally to support sequential queries. Zhou et al. (2012a) propose an approach to enable random queries to the file system. Accessing small data on HDFS may cause unnecessary data transfer, once the size of the packet may be bigger than the data packet being sent. The approach presents a data transfer policy that support both sequential and random access. It uses a dynamic method to set the size of data packet. If a random access is requested, the data packet is set to be equal to or less than the required data size. Otherwise the default data packet size will be used. Similarly to Zhou *et al.* Buck et al. (2011) propose a query processing solution to compute a specific type of data. A plugin to Hadoop, named SciHadoop, is intended to process scientific structured data using a simple query language. Thus, the query processing in SciHadoop is expressed entirely at the level of scientific data models.

Iu and Zwaenepoel (2010) propose a query optimizer called HadoopToSQL to work especially when only a subset of the data should be processed. HadoopToSQL is a hybrid system that seeks to improve Hadoop performance by transforming MapReduce queries to use the indexing, aggregation and grouping features provided by DBMS. The authors proposes two algorithms that generate SQL code from MapReduce queries: one algorithm can extract input set restrictions from MapReduce queries; and, the other can translate entire MapReduce queries into equivalent SQL queries. The HadoopToSQL is able to take a compiled MapReduce program generated by the Java compiler and analyze it to find ways to run it efficiently on an

SQL database.

A Hadoop-DBMS hybrid system is presented by Abouzeid et al. (2009). This work completely replaces HDFS by a parallel DBMS. HadoopDB approaches parallel databases in performance and efficiency, yet still yields the scalability, fault tolerance, and flexibility of Hadoop. The basic idea behind HadoopDB is to use MapReduce as the communication layer above multiple nodes running single-node DBMS instances. Queries are expressed in SQL, translated into MapReduce by extending existing tools, and as much work as possible is pushed into the higher performing single-node databases. An et al. (2010) propose a new Hadoop-DBMS hybrid communication layer as well. In that work, DBMS engines are read-only execution layer into Hadoop. In this approach the DBMS is used to provide efficient operators while the HDFS is responsible for managing the data providing a fault tolerance environment in the data layer. The modified database engine is able to process data from HDFS file at the block level, which makes it suitable to the MapReduce paradigm.

Similarly, Bajda-Pawlikowski et al. (2011) also propose a Hadoop-DBMS hybrid system, in addition to addressing query processing. The work offers extensions in HadoopDB to process efficiently data warehousing queries. The authors discuss more advanced execution plans where some joins access multiple database tables within the Map phase of a single job. After repartitioning on the join key, related records are sent to the Reduce phase where the actual join is computed.

In some ways, the work of Abouzeid et al. (2009) deals on processing of structured data. As well as that research, the partially outlined work of Buck et al. (2011) and the approach of Kaldewey et al. (2012) also deal with structured data processing. The proposal of Buck *et al.* is intended to process array-based queries. The array-based model used by SciHadoop is defined by two properties: the shape of an array, which is given by the length along each of its dimensions, and the corner point of an array, which defines the array's position within a larger space. SciHadoop was designed to accept queries expressed in a query language developed by the authors. The plugin implemented modifies the standard task scheduler to function at the level of scientific data models using arrays, rather than low-level byte streams, used regularly by the HDFS.

In turn, the Kaldewey et al. (2012) objective is to cope with structured datasets that fit a star schema. The research prototype for structured data processing can achieve performance improvements over existing solutions, without any changes to the Hadoop implementation. This prototype, called Clydesdale, inherit the fault-tolerance, elasticity, and scalability properties of MapReduce. This is also of significant practical value since it allows to run Clydesdale on future versions of Hadoop without having to re-compile and re-test Hadoop with a set of custom changes. Clydesdale achieves this through a novel synthesis of several techniques from the database literature and carefully adapting them to the Hadoop environment.

4.6. The Hadoop Ecosystem: Hive, Pig, HBase

The MapReduce paradigm is not suitable for all problems involving large datasets. The four modules that currently compose the Apache Hadoop core (Hadoop Common, HDFS, MapReduce and YARN) are well suitable for unstructured data. However, when processing common old-fashioned structured data, Hadoop may suffer from performance issues since the framework was not originally developed to process these data. To overcome these problems, several Apache Hadoop related projects have been developed over the past years. For example: Pig, a high-level data-flow language and execution framework for parallel computation; Mahout, a scalable machine learning and data mining library; Hive, a data warehouse infrastructure that provides data summarization and ad hoc querying; and, HBase, a scalable distributed database inspired on Google BigTable that supports structured data storage for large tables.

Konishetty et al. (2012) propose the implementation of the Set data structure and operations of union, intersection, and difference in a scalable manner on top of Hadoop HBase. The work presents optimizations for three Set operations and also limitations on implementing this data structure in the Hadoop ecosystem. Zhang and De Sterck (2010) propose CloudBATCH, a new Hadoop component that enables it to function as a traditional batch job queuing system with enhanced functionality for cluster resource management. The approach allows the cluster management using only Hadoop to discover hybrid computing needs involving both MapReduce and legacy applications. CloudBATCH runs on top of HBase and includes a set of tables used for storing resource management information, globally accessible across all nodes to manage metadata for jobs and resources.

Among the Apache Hadoop ecosystem projects, Pig (Gates et al., 2009) has received relevant contributions recently. Pig is a high-level data flow system that fills the existent gap between SQL and MapReduce. Pig offers SQL-style high-level data manipulation constructs, such as filter and join, which can be assembled in an explicit data flow and interleaved with custom Map- and Reduce-style functions or executables. Pig programs are compiled into sequences of MapReduce jobs to be executed. Pig uses Pig Latin (Olston et al., 2008), a language that combines the best of both worlds: high-level declarative querying in the spirit of SQL and low-level procedural programming using MapReduce. Pig compiles Pig Latin into physical plans that are executed over Hadoop.

A Pig Latin program is a sequence of steps, much like in a programming language, each of which carries out a single data transformation. Writing a Pig Latin program is similar to specifying a query execution plan. Tanimura et al. (2010) propose an extension to Pig. The approach implements a RDF data processing framework built on top of Hadoop. Pig has received additional extensions to support RDF data processing, providing a scalable architecture, a set of data processing tools and a general query optimization framework. These enhancements allow users to perform efficient integrated data processing using RDF data.

Since Pig brought support to query like languages to Hadoop, the reuse of intermediate results may be an interesting technique to save processing time and enhance performance on similar jobs. Elghandour and Aboulnaga (2012) present ReStore, an extension to Pig that enables it to manage the storage and reuse of intermediate results of the MapReduce workflows executed in the Pig data analysis system. This maximizes data reuse opportunities between MapReduce jobs which are independently executed by the system. ReStore matches input workflows of MapReduce jobs with previously executed jobs and rewrites these workflows to reuse the stored results of the positive matches. According to Elghandour and Aboulnaga, even though ReStore has been developed as an extension to Pig, its techniques may be applied in any data flow system that generates workflows of MapReduce jobs for input queries such as Hive (Thusoo et al., 2009) and Jaql (Beyer et al., 2011).

Similarly, Olston et al. (2011) propose a workflow manager called Nova, which pushes continually-arriving data through graphs of Pig programs being executed on Hadoop clusters. Nova is like data stream managers in its support for statefull incremental processing, but unlike them, it deals with data in large batches using disk-based processing. The proposal of a workflow manager enables key scheduling and data handling capabilities such as: continuous processing; independent scheduling, where different portions of a workflow may be scheduled at different times/rates; and, cross-module optimization in which a workflow manager can identify and exploit certain optimization opportunities, e.g., common input being consumed by two different workflows. It is different from ReStore because Nova is supposed to deal with different workflows that may use common input data, while ReStore is supposed to keep intermediate results from workflows executions to future reuse.

Zhang et al. (2012b) present a performance modeling framework for Pig programs. The idea is to suggest a solution for two common problems regarding Cloud Computing and Pig programs. First, estimating the completion time of such programs as a function of allocated resources. Second, estimating the amount of resources (number of task slots) to complete a Pig program in a given deadline. The approach forces Pig to use the optimal schedule of its concurrent jobs, the authors were able to eliminate the existing non-determinism in Pig program execution of concurrent jobs, achieving better performance predictions.

Finally, another important concern when dealing with structured data inside Hadoop is data placement, as mentioned before. Hadoop components such as Hive and Pig rely on the HDFS to store its data and cannot directly control this storage. Based on conventional data placement structures, an approach named RCFile (Record Columnar File) (He et al., 2011b) presents a solution to this problem. Since RCFile is developed and integrated to Hive, it stores tables by horizontally partitioning them into multiple row groups. Following, each group is vertically partitioned so that each column is stored independently. Columns may also be compressed and grouped according to the needs. RCFile promotes four desirable requirements for data placement in MapReduce environments: fast data loading, fast query processing, efficient storage space utilization, and adaptivity to dynamic workload patterns enhancing Hive performance.

4.7. Energy Management

The reduction in cost of hardware enabled corporations to increase the number of data centers and machines. Consequently, energy consumption has become a vital issue regarding costs of data storing and its processing. Several papers discuss the cluster energy management problem generically. Regarding Apache Hadoop clusters, Li et al. (2011) propose an algorithm for maximizing throughput of a rack of machines running a MapReduce workload, subject to a total power budget. The main idea is to optimize the trade-off between job completion time and power consumed. The novelty in the approach relies on the accounting for thermally-induced variations in machine power consumption. The algorithm minimizes job completion time (or equivalently, maximizes the computational capacity of a cluster) for any given power budget.

GreenHadoop, proposed by Goiri et al. (2012), is an Apache Hadoop variant for data centers powered by photovoltaic solar (green energy) arrays and electrical grid (brown energy) as a backup. The objective is to investigate how to manage the computational workload to match the green energy supply in small/medium data centers running data-processing frameworks. However, scheduling the energy consumption of MapReduce jobs is challenging because they do not specify the number of servers to use, their run times, or their energy needs. Moreover, power-managing servers should guarantee that the data to be accessed by the jobs remain available. GreenHadoop seeks to maximize the green energy consumption of the MapReduce workload, or equivalently to minimize its brown energy consumption. GreenHadoop predicts the amount of solar energy that is likely to be available in the future, using historical data and weather forecasts. By using these predictions, it may then decide to delay some (low-priority) jobs to wait for available green energy, but always within their time bounds. If brown energy must be used to avoid bound violations, it schedules the jobs at times when brown energy is cheaper, while also managing the cost of peak brown power consumption.

Another approach to energy management in clusters is the GreenHDFS (Kaushik et al., 2010, 2011). Instead of dealing with the MapReduce component of Apache Hadoop, it deals with the HDFS. GreenHDFS partitions cluster servers into *Hot* zones, used for frequently accessed files, and *Cold* zones, for rarely used files. This approach enables energy saving by putting *Cold* zones servers to sleep. To do so, a migration policy moves files between zones accordingly. Initially, this policy was reactive and used historical data to move files between zones. This approach was improved creating a predictive file zone placement, which defines the initial placement of a file, and then uses a predictive file migration policy. This approach uses supervised machine learning to train its file attribute component and to manage changes between zones.

GreenPipe, presented by Mao et al. (2012), provides a specific solution to bioinformatics, but its main objective is related to energy consumption problems. GreenPipe is a MapReduce-enabled high-throughput workflow system for applications of bioinformatics, which defines a XML based workflow and executes it on Hadoop. The workflow execution is divided in two modes. In the first one, called physical mode, the XML workflow is translated into MapReduce jobs and launched on a physical Hadoop cluster. The second mode, called virtual mode, works with virtualization, obtaining virtual machines from IaaS (Infrastructure as a Service) platforms and running Hadoop jobs in the VM cluster. Authors also address the optimizations of the planning and job priority, and energy efficiency by introducing a power-aware scheduling algorithm in the workflow engine. The scheduler tries to allocate VM resources based on the power consumed by the applications. It also tries to group similar energy trace VMs in a physical machine, reducing the energy consumption without sacrificing the application performance.

4.8. GPGPU

GPGPU (General-purpose computing on graphic processing units) (Owens et al., 2008) is a technique to use commodity GPUs to perform general purpose computing in applications traditionally handled by CPUs. CUDA and OpenCL are examples of integrated development environments that may be used to such purposes. Shirahata et al. (2010) presents a hybrid Map task scheduling technique for GPU-based heterogeneous computer clusters. The developed job scheduler assigns the Map tasks onto CPU cores and GPU devices in order to minimize the overall MapReduce job execution time. The scheduler uses profiles collected from dynamic monitoring of Map task's behavior to decide where to schedule a task. Jobs containing tasks that have data parallelism may be suitable for GPU execution, while tasks containing many branches or synchronizations are not recommended to be executed using this approach. The model uses Hadoop Pipes, a C++ interface to Hadoop MapReduce. Hadoop Pipes uses sockets as the channel over which the TaskTracker communicates with the process running the C++-based Map and Reduce functions. Its use is justified because other native methods such as Hadoop Streaming and Java Native Interface may introduce significant overhead, representing loss of performance.

Xin and Li (2012) have successfully demonstrated that the use of OpenCL in commodity Hadoop clusters may outperform regular clusters significantly. Their approach concerns to both data- and compute-intensive applications. Grossman et al. (2013) also integrated OpenCL into Hadoop to enable the use of heterogeneous processors, such as the CPU and GPU combination. The extension supports the execution of user-written Java kernels on heterogeneous devices, optimizes communication through asynchronous transfers and dedicated I/O threads. Authors claim that the approach can achieve nearly 3x overall speedup for some specific Hadoop MapReduce applications.

Another approach proposed by Fang et al. (2011) is named Mars, a MapReduce runtime system accelerated with GPUs to improve Hadoop performance. Mars runs on NVIDIA GPUs (MarsCUDA), AMD GPUs (MarsBrook) as well as multicore CPUs (MarsCPU). Mars was integrated into Hadoop, being called MarsHadoop. In this scenario, each machine in a network can utilize its GPU with MarsCUDA or MarsBrook in addition to its CPU with the original Hadoop. By using both the GPU and the CPU, GPU-only performance was improved by 40 percent for some applications tested by the authors. Finally, Tan et al. (2012b) introduce a new framework capable of using both GPU and CPU processing elements collaboratively in MapReduce jobs. The framework named Pamar (Processing Element Aware MapReduce) was designed to clusters having asymmetry in GPGPU/CPU node configurations. It automatically detects the type of processing elements available on each node. Pamar also scans for the processing elements requirements of submitted jobs. The authors also implemented a scheduler called HPE (Heterogeneous Processing Element) that uses the job requirements to make scheduling decisions. After the integration into Hadoop, the framework has demonstrated improvement in job queue completion time.

4.9. Data Security & Cryptography

In the cloud computing era, data security, privacy, and integrity became important features to be supported by frameworks and service providers. In Apache Hadoop, data is stored according to user accounts. This mechanism may not be secure enough when dealing with data spread across multiple datacenter and cloud providers. Some studies addresses these problems, presenting solutions to enhance both Hadoop data security and confidentiality. Wei et al. (2009) design a model that enhances the basic MapReduce framework with a set of security components. In open systems, MapReduce faces a data processing service integrity problem since service providers may come from different administration domains that are not always trustworthy. The approach is meant to provide a service integrity assurance framework for MapReduce, replicating some tasks and assigning them to different Mappers/Reducers. This is achieved by using a decentralized replication-based integrity verification scheme for running MapReduce in open systems. It uses a set of security properties such as non-repudiation and resilience to DoS attacks and replay attacks while maintaining the data processing efficiency of MapReduce.

Another approach proposed by Lin et al. (2012) addresses the data confidentiality issue in Hadoop by enhancing HDFS. Two hybrid encryption schemes were implemented and integrated into HDFS to achieve data confidentiality: the HDFS-RSA, which uses the RSA encryption and AES and the HDFS-Pairing, which uses a pairing-based encryption scheme and AES. As expected, both schemes introduce overhead on reading and writing operations. The biggest overhead is on writing, because the encrypting process is accomplished in two parts. Reading operations also have an overhead, although it is much lower and acceptable than the writing overhead.

Regarding cloud security, Shen et al. (2011) focus on the problem of data migration between public and private clouds. The authors discuss the potential threats that may occur when migrating data between clouds. Based on these threats, a model was developed to secure the data transfer. The model uses SSL, tickets and data encryption. A module named MDM (Migration Decision Module) was developed and used in the HDFS. Results and validation show that, although secure, the encryption process generates a considerable overhead in the transfer process, which may reach up to nine times the time cost without using the MDM. Cloud security is also addressed by Zhou et al. (2013), but instead of using encryption, it uses the principle of sensitive data splitting, where sensitive data is kept in trusted private cloud, while insensitive data may be moved to public cloud. The system called Prometheus is able to properly work on the semi-honest cloud model as well as on the malicious cloud model.

Khaled et al. (2010) propose a token-based access control system for RDF Data in a cloud implemented using Hadoop. One of the most efficient ways to handle RDF data is to store it in cloud computers. However, access control is a major concern for cloud-resident RDF data. The proposed mechanism defines six types of access levels and one enforcement strategy for the resulting access control policies. The enforcement strategy is implemented at three levels: Query Rewriting (preprocessing phase), Embedded Enforcement (MapReduce execution phase), and Post-processing Enforcement (data display phase). This way, users are granted tokens based on their business needs and authorization levels previously determined.

Concerning problems with data integrity in the cloud, Nguyen and Shi (2010) propose a model called Opera (Open Reputation Model) that employs reputation-based trust management to improve Hadoop computational efficiency. In general, the existing methods rely on a single-value reputation to capture the differences between the Service Providers in terms of properties such as performance or availability. The approach tracks node trust as a vector of efficiency-related considerations, such as node downtime and failure frequency. Users are allowed to query the reputation vector of any registered component. This model is associated to the use of the proposed scheduler, which may improve the performance of jobs and reduce the number of failed/killed tasks.

While the Opera approach is intended to achieve better performance on Hadoop, malicious behaviors (e.g., falsified computation) are not considered, making it unsuitable for protection against data integrity attacks. Khan and Hamlen (2012) present a solution to this problem. Hatman (Hadoop Trust Manager) considers a reputation-based trust management approach to detect integrity violation in Hadoop clouds. It augments Hadoop NameNodes with reputation-based trust management of their slave DataNodes. NameNodes maintain a trust matrix that keeps trust and reputation information used to dispatch jobs to DataNodes. To obtain high scalability, all trust management computations are formulated as distributed cloud computations. This approach increases the computing power and improves the data integrity of cloud computations.

5. Discussion

In this section, we discuss the results and findings of our analysis and classification of the selected studies. First, we introduce an overview of the Hadoop evolution, presenting its main features that have changed over time. Following, we discuss an overview of the selected studies and some interactions among the taxonomy categories, presented further ahead.

5.1. *Hadoop evolution*

Since its initial release, Hadoop changed constantly and considerably in 59 releases – the last one was version 2.2.0 on October 15, 2013 – over six years of development. Besides bug fixes, present in each one of these releases, new features and modules were developed and incorporated, consolidating the framework as the main solution to process large amounts of data using the MapReduce paradigm. Some releases deserve special attention. From the initial releases, versions 0.20.x were the first ones considered more stable, which could be used on production environments. The first one was released in April 2009, and this branch was one of the longest active branches in the history of Hadoop. This release missed and important feature present on subsequent releases: support for file appending on HDFS. A couple years later, versions 0.23.x were released. Albeit being considered alpha-quality releases, they included two important features to present Hadoop versions: HDFS federation and the initial release of Hadoop NextGen MapReduce, also known as YARN.

In 2011, after years of development, Hadoop reached the 1.0.0 version. This release included webHDFS and security features on HBase, besides performance enhancements. Being a cut from the 0.20.x branch, it did not include some bug fixes and features incorporated between 0.20.x and 0.23.x releases, which missed some important security features. This was fixed on later 1.x releases.

On May 2012, Hadoop 2.0.x was first released. The 2.0.0-alpha included significant major features over the 1.x series. Derived from the 0.23.x branch, the most important features included were the HDFS HA (High Availability), YARN and HDFS federation, besides some performance tuning. This version is considered as a consolidated release from the 0.23.x branch.

These two branches, 1.x and 2.x, are the major branches on Hadoop releases and have significant differences that deserve some observations. The first one was the introduction of the YARN, or Nextgen MapReduce (initially present on 0.23.x releases). YARN proposes a separation between two major functions of the JobTracker: resource management and job scheduling/monitoring. In 2.x releases, YARN is a distributed application management framework while Hadoop MapReduce remains as a pure distributed computation framework. Another inclusion was the HDFS HA (Oriani and Garcia, 2012), which promotes the high availability of HDFS. Prior to the 2.0.0 release, the NameNode was a single point of failure in an HDFS cluster. A failure on the NameNode machine would make the entire HDFS cluster unavailable. The HDFS High Availability feature provides the option of running two redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby. Finally, the third feature is the HDFS Federation, which supports multiple Namenodes in a HDFS file system.

Theoretically, although still supported in versions 2.x, the distance between version 1.x tends to increase and, in the future, backwards compatibility in version 2.x will no longer be supported. This is reasonable, since the separation between resource management and job/task allocation benefits the cluster

Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

infrastructure in terms of use and flexibility. As shown next, we have been able to identify several interrelated studies, which evolved over the last five years using Apache Hadoop and its ecosystem.

5.2. Overview and Studies Interaction

Scheduling is considered crucial to Hadoop performance. In this sense, the selected papers that were allocated in the scheduling, data flow, and resource allocation categories are majorly concerned with this issue. Some works propose multi-queue schedulers to improve performance (Zhao et al., 2012; Tian et al., 2009; Kumar et al., 2012). Other authors use different approaches to achieve it, such as the data locality aware schedulers (Zaharia et al., 2010; He et al., 2011a; Zhang et al., 2011b; Hammoud and Sakr, 2011; Zhang et al., 2012a; Hammoud et al., 2012; Ibrahim et al., 2012; Zhang et al., 2011c; You et al., 2011; Tao et al., 2011), which are concerned with the correct allocation and placement of Map and Reduce tasks. Performance problems may also be tackled by using historical data from cluster nodes, which allows, e.g., the speculative execution of MapReduce tasks (Zaharia et al., 2008; Lei et al., 2011; Rasooli and Down, 2011; Chen et al., 2010). Although concerned with performance, some papers present solutions covering important correlated areas, for example, Cloud Computing resource allocation (Zhang et al., 2011b; Hammoud et al., 2012; Tang et al., 2012; Zhang et al., 2012b; Park et al., 2012), which reflects directly on Hadoop performance in such environments. Thus, scheduling in heterogeneous clusters is also an important topic addressed by some studies (Tian et al., 2009; Ahmad et al., 2012; Kumar et al., 2012). Ultimately, some approaches develop mechanisms of reuse of intermediate data among MapReduce jobs using common datasets (Kumar et al., 2012; Nykiel et al., 2010; Shi et al., 2011).

As discussed earlier, changes in data flow, manipulation, and resource allocation are mainly made to address performance or to meet specific requirements. When dealing with improving performance, some studies break the synchronization barrier between phases of the Map and Reduce stages (Wang et al., 2011; Ibrahim et al., 2010). While Wang et al. (2011) propose a full pipeline to overlap the shuffle, merge and Reduce phases, Ibrahim et al. (2010) focus on data locality, altering only the partitioning of intermediate keys on the Map phase. Although Vernica et al. (2012) do not directly alter the synchronism of the phases, they introduce a communication channel among Mappers, breaking its isolated execution. Besides Ibrahim et al. (2010), other studies deal with data locality (Kwon et al., 2012; Zhu and Chen, 2011; Ho et al., 2011). Kwon et al. (2012) propose a skew mitigation approach to UDOs (User Defined Operations), which resplits input data when failed workers are detected. Failed workers are detected on Zhu and Chen's (2011) studies by two mechanisms called adaptive interval and reputation-based detector. In contrast, Ho et al. (2011) tackle the RPP (Reducer Placement Problem) problem caused by all-to-all communication between Mappers and Reducers. This work proposes algorithms to place Reducers in correct racks to minimize the saturation of network bandwidth.

The proposal of Verma et al. (2011) is slightly different from the aforementioned, since they suggest a framework named ARIA that can allocate the appropriate amount of resources to execute a job, meeting a soft deadline, which is routinely executed on a new dataset. To do this, the framework uses a scheduler based on earliest deadline first policy. Verma et al. (2012) evolve ideas used in ARIA to propose a better solution based on other mechanisms. On the other hand, Lin et al. (2010) propose changes in data flow and resource allocation to accommodate reliable nodes, categorized as dedicated or volatile, which store reliable or temporary files. Furthermore, the authors implement schedulers particularly to that scenario. Although Hadoop was not originally developed to support iterative computing, some efforts have been made to give it this capability (Elnikety et al., 2011; Liang et al., 2011; Bu et al., 2012; Bhatotia et al., 2011; Zhang et al., 2011d). Except for the proposals presented by Bhatotia et al. (2011) and Zhang et al. (2011d), research studies presented by Elnikety et al. (2011), Liang et al. (2011) and, Bu et al. (2012) use different cache levels to treat the repeatedly processing and schedulers which are loop or cache-aware. Other papers (Elteir et al., 2010; Grover and Carey, 2012; Laptev et al., 2012) deal with very specific problems, such as recursively reducible jobs, early results, and sampling issues.

Storage is another intersection point in selected papers concerning the framework. HDFS was modified to increase performance, as for the case of its I/O mode (Jiang et al., 2010; Zhang et al., 2011a; Shafer et al., 2010), solving data placement problems (Xie et al., 2010), or adapting it to deal with small files (Jiang et al., 2010; Dong et al., 2010) since HDFS was not originally designed to store such files. Some works replace the original HDFS with a more suitable solution for specific compatibility problems (Mikami et al., 2011; Kousiouris et al., 2011) or to support areas such as Cloud Computing (Kousiouris et al., 2011; Guang-hua et al., 2011). Cloud Computing has received considerable attention in research studies trying to improve performance and resource usage (Zhang et al., 2011a; Wei et al., 2010; Mao et al., 2011; Mandal et al., 2011; Ko et al., 2010) while using Hadoop.

In order to improve Hadoop performance, several approaches make modifications in how data is indexed, recovered by query processing, and stored. Indeed, most studies suggest the use of more than one strategy to boost Hadoop performance. Liao et al. (2010) and Dittrich et al. (2012) propose a completely new indexing technique, while An et al. (2010) and Dittrich et al. (2010) propose changes in Hadoop's original indexing. Liao et al. (2010) and Bajda-Pawlikowski et al. (2011) change query processing in order to cope with complex type queries and warehouse queries. Other studies approach query processing in different ways, like query optimization (Dittrich et al., 2010; Iu and Zwaenepoel, 2010), query languages (Abouzeid et al., 2009; Buck et al., 2011), and random queries (Zhou et al., 2012a). Some of these works (An et al., 2010; Iu and Zwaenepoel, 2010; Abouzeid et al., 2009; Bajda-Pawlikowski et al., 2011) use hybrid solutions (DBMS based) to support indexing methods and modifications in query processing. Finally, two studies focus on structured data (Buck et al., 2011;

Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

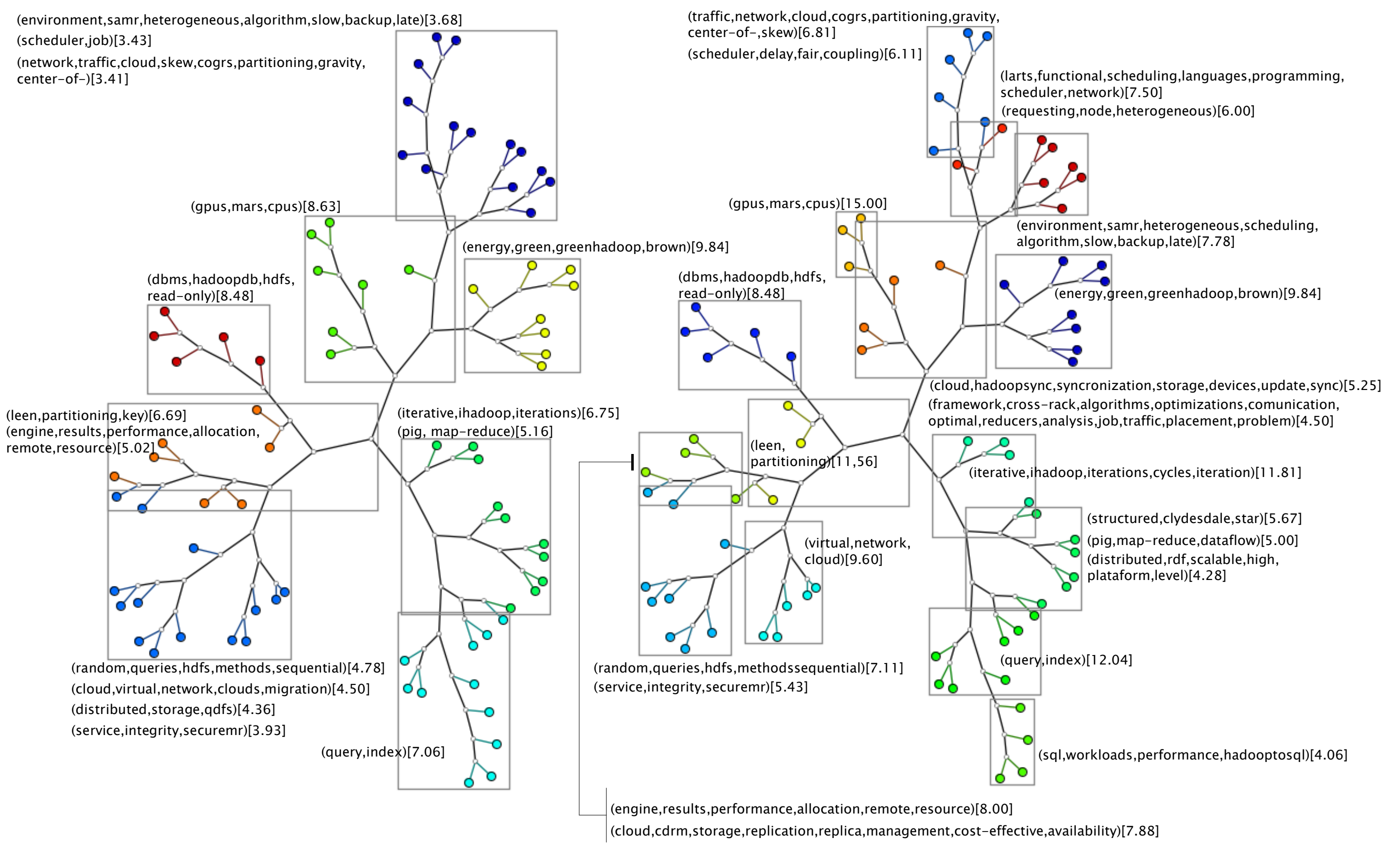
(environment,samr,heterogeneous,algorithm,slow,backup,late)[3.68]
(scheduler,job)[3.43]
(network,traffic,cloud,skew,cogrs,partitioning,gravity, center-of-)[3.41]
(gpus,mars,cpus)[8.63]
(energy,green,greenhadoop,brown)[9.84]
(dbms,hadoopdb,hdfs, read-only)[8.48]
(leen,partitioning,key)[6.69]
(engine,results,performance,allocation, remote,resource)[5.02]
(iterative,ihadoop,iterations)[6.75]
(pig, map-reduce)[5.16]
(random,queries,hdfs,methods,sequential)[4.78]
(cloud,virtual,network,clouds,migration)[4.50]
(distributed,storage,qdfs)[4.36]
(service,integrity,securemr)[3.93]
(query,index)[7.06]
(traffic,network,cloud,cogrs,partitioning,gravity, center-of-,skew)[6.81]
(scheduler,delay,fair,coupling)[6.11]
(larts,functional,scheduling,languages,programming, scheduler,network)[7.50]
(requesting,node,heterogeneous)[6.00]
(gpus,mars,cpus)[15.00]
(environment,samr,heterogeneous,scheduling, algorithm,slow,backup,late)[7.78]
(energy,green,greenhadoop,brown)[9.84]
(dbms,hadoopdb,hdfs, read-only)[8.48]
(cloud,hadoopsync,syncronization,storage,devices,update,sync)[5.25]
(framework,cross-rack,algorithms,optimizations,comunication, optimal,reducers,analysis,job,traffic,placement,problem)[4.50]
(leen, partitioning)[11,56]
(iterative,ihadoop,iterations,cycles,iteration)[11.81]
(virtual,network, cloud)[9.60]
(structured,clydesdale,star)[5.67]
(pig,map-reduce,dataflow)[5.00]
(distributed,rdf,scalable,high, plataform,level)[4.28]
(random,queries,hdfs,methodssequential)[7.11]
(service,integrity,securemr)[5.43]
(query,index)[12.04]
(sql,workloads,performance,hadooptosql)[4.06]
(engine,results,performance,allocation,remote,resource)[8.00]
(cloud,cdrm,storage,replication,replica,management,cost-effective,availability)[7.88]

Figure 2: Comparison of two clusterization steps using the Neighborhood Join projection technique.

Kaldewey et al., 2012).

The Apache Hadoop ecosystem is composed of parallel projects that intend to provide support to specific areas in which the MapReduce paradigm would not be suitable or would have performance issues. The most known ones are Pig, Hive, and HBase. Pig – a high-level data flow language that runs on top of MapReduce – received contributions that enabled the support to RDF data (Tanimura et al., 2010), the reuse of intermediate processing results (Elghandour and Aboulnaga, 2012), and continuous streaming of newly arrived data into Pig programs that are already in execution (Olston et al., 2011). Pig, Hive, and HBase have received many other contributions, since they are used as infrastructure for several research projects around Scheduling, DBMS, Indexing, Random Data Access, Storage, and Cloud Computing.

Some areas, although not directly related to the framework, developed several contributions to Apache Hadoop, including themes such as energy management, the use of GPUs and data integrity/security. Energy management has demonstrated to be an important research topic as the number of data centers has increased consistently over the last few years. The demand for electric power to such computing facilities has been addressed in some studies, including the use of green energy (Goiri et al., 2012; Kaushik et al., 2010, 2011), energy efficiency (Mao et al., 2012), and power budget directed data processing (Li et al., 2011). In the case of GPU use, papers addressed the increase on performance (Xin and Li, 2012; Fang et al., 2011; Tan et al., 2012b) by using the GPU shipped as part of x86 hardware nodes compounding commodity clusters. Finally, researchers developed approaches concerning data security, privacy, and integrity within the framework (Wei et al., 2009; Lin et al., 2012), some of them were specifically designed to be used with Cloud Computing technologies (Shen et al., 2011; Khaled et al., 2010; Nguyen and Shi, 2010; Khan and Hamlen, 2012).

5.3. *Taxonomy*

We placed all studies into one or more categories of our taxonomy. These categories were created based on the Apache Hadoop project and considering the context of the selected papers. The framework is divided into subprojects, which initially guided early versions of our taxonomy. Key areas such as scheduling, resource allocation, and storage were first selected. When the analysis of the selected studies began, new categories were created in our taxonomy based on the new topics found. The final set of categories and their subdivision were presented in the previous section.

Another way to analyze the grouping of studies is to use specific visualization tools. We created a corpus using title, keywords, and abstract from the selected studies and we used a tool called Projection Explorer (PEx) (Paulovich et al., 2007; Cuadros et al., 2007) to create a set of visualizations for our work. PEx is able to create and explore visual representations

Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

Taxonomy Category	Subcategory	Number of Studies	Percent
MapReduce	Scheduling	42	40%
	Resource Allocation	40	38%
	Data flow	28	26%
Data Storage & Manipulation	Storage & Replication	18	17%
	Cloud Computing	17	16%
	Queries	16	15%
	Cloud Storage	8	8%
	DBMS	8	8%
	Indexing	7	7%
	Random Access	3	3%
Ecosystem	New Component	23	22%
	Hive. Pig. Hbase	12	11%
Miscellaneous	Data Security & Crypto	7	7%
	Energy Management	6	6%
	GPGPU	4	4%

Table 5: Research topics addressed in selected studies

of document collections. Each document is represented as a circle in a plane and the distance between circles in subtrees represents their similarity. The closer the circles are in the same subtree, the similar the documents are. By using this tools we were able to group documents into categories and to visualize the results as shown in Figure 2. The figure shows two clusterization steps. In the first step, represented in the subgraph on the left, we used 8 clusters. In the second step, represented in the subgraph on the right, we used 15 clusters. We understand from the figure that with more clusters, there are more terms that group the articles into specific branches, which shows the correlation among works. The use of such visualizations helps to confirm studies placement into categories. Also, the results were close to our manual classification. To generate the visualizations, we conducted a series of experiments, varying the number of groups, and the best results were achieved using eight groups, the number of subsections previously presented in Section 4.

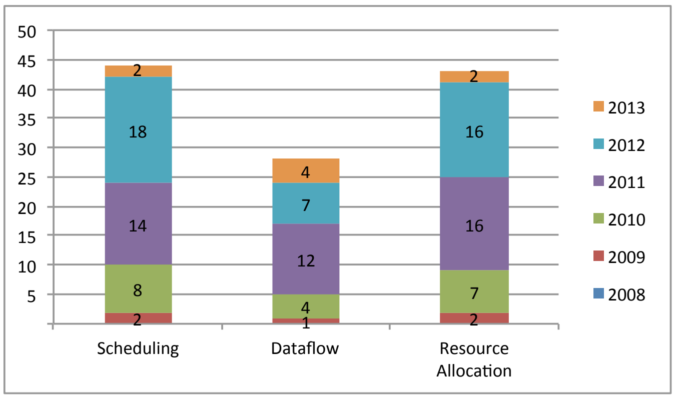

Figure 3: Number of Studies by Year - MapReduce Category

5.4. *Results and Findings*

Our taxonomy was created to help answer our first research question, RQ1: "What are the main research topics and aspects covered by publications concerning the Apache Hadoop framework and the MapReduce paradigm?". We were able to classify the studies and their contributions into categories. This classification gives a visualization of which areas are well explored in Hadoop. Table 5 and Figure 3 show that the MapReduce paradigm within Hadoop aggregates more contributions to the framework. Scheduling and data flow, and consequently, resource allocation, have major roles on Hadoop performance. In fact, all the papers in this category are concerned with performance improvements in MapReduce applications. Also, most proposals changing the MapReduce data flow are related to achieving better application runtimes. This is accomplished by breaking existing paradigm enforced barriers, such as the isolation between the Map and Reduce phases. Yet, on the MapReduce category, some papers are not concerned with performance. By altering the MapReduce data flow, some approaches extend the applicability of Apache Hadoop to support, e.g., iterative applications, originally not suitable to use the framework.

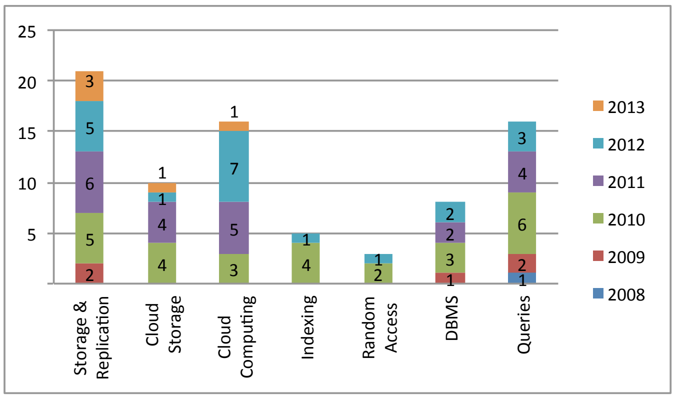

Figure 4: Number of Studies by Year - Data Storage & Manipulation Category

We can also see in Table 5 and in Figure 4 that research studies enhancing HDFS and approaches to create or develop query strategies are relatively common. At least one out of five papers in this review cover data storage and manipulation. The HDFS file system was developed to have high throughput storing large datasets. Most of the papers in the area alter HDFS to store other types of datasets, e.g., small files such as RDF files, or scattered datasets, in a more efficient way. Additionally, works enhancing or developing new techniques to query data on HDFS are present. This is reflected in the Ecosystem category (Figure 5), since several approaches are developed as new components being placed on top of Hadoop. Although most of these approaches do not result in significant changes in Hadoop, some contributions in the DBMS area demand several changes to the framework, making it difficult to use them with a new Hadoop version as soon as it is released. We may still see some categories with few contributions, e.g., indexing and random access, which have partial support from Apache Hadoop and have received contributions to solve specific classes of problems. Finally, recent research areas, such as green computing, energy management, GPGPU, Cloud Computing, and Cloud Storage, are promising but few works were published involving Hadoop

(Figure 6). Works intersecting these areas may be more likely to be developed, since the framework became more stable along the last two years and the MapReduce paradigm was rethought as a support to unprecedented areas. They have a direct impact on Table 5, where most studies have been placed into more than one category.

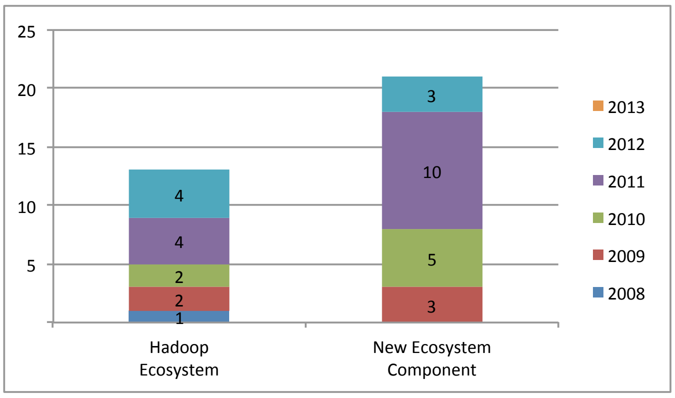

Figure 5: Number of Studies by Year - Ecosystem Category

With the selected papers organized into the taxonomy, we were able to show more clearly how the proposals were validated and what were real contributions to the Apache Hadoop. The validation methods used in the work proposed in the selected papers are discussed in RQ2.2 "What kind of validation is performed in each study? Simulation, analytical model, experimentation?". Tables A.6, A.7, A.8 in Appendix A shows the validation techniques used in each topic of our taxonomy.

Source code contributions to the Apache Hadoop and its ecosystem are proofs of a concept that validates the proposals. Hence, we consider that implementation is the most valuable form of validation. Also, the proposals, including the ones containing implementations, can be additionally validated by experiments, simulations and, more formally, using analytical models. Indeed, we can conclude that most studies (88%) use more than one form of validation.

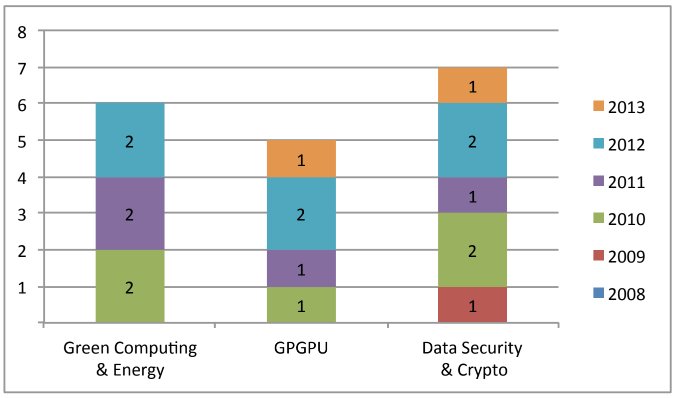

Figure 6: Number of Studies by Year - Miscellaneous Category

As described in 2.2, our search strategies prioritize studies presenting concrete contributions to the Apache Hadoop, i.e., source code contributions. Only 5% of the papers do not contain any implementation (Guo et al., 2011; Kaushik et al., 2011; Lei et al., 2011; Rasooli and Down, 2011; Jeon et al., 2013). Even without implementation into Hadoop, these works were included because they present other important contributions. Guo et al. (2011), Lei et al. (2011), and Rasooli and Down (2011) propose schedulers and validate their proposals using simulations. Kaushik et al. (2011) present new ecosystem components. It is interesting to notice that these studies are not present in the column "Implementation" of the Table A.6 and Table A.7 because they do not contribute directly to Hadoop. Thus, the other 95% of the works contain source code contributions to Hadoop. In contrast, 8% of the proposals only contain implementation, without any other form of validation (Olston et al., 2008; Wei et al., 2009; Zhang and De Sterck, 2010; Kousiouris et al., 2011; Shi et al., 2011). The majority of the implementations is originated from academic works. Some of these proposals intend to provide source code and evolve the Hadoop project. Others use the implementation only as proof of concept. In contrast, few papers aim to solve specific industry problems.

Experimentation is the second validation technique most used by researchers to evaluate the proposals (82%). In fact, the combination of experimentation and implementation is the most used, and was observed in 80% of the papers. It shows the attention employed by researchers to validate the proposed implementations. However, it is important to highlight some problems encountered during this systematic literature review. Many of the experiments conducted by the researchers cannot be fully reproduced, since we did not identify any formal experiment explicitly described. For some research studies, even more troubling, datasets and/or benchmarks used in experiments are unavailable to other researchers. The absence of a package and/or dataset may precludes the quality assessment of the original experiments and their replication. For example, experiments containing very specific benchmarks and/or datasets, or experiments inadequately designed may introduce biases or even influence the results obtained. A real example of this problem can be noted comparing the proposals of Wang et al. (2011) and Ho et al. (2011). The proposal of Wang et al. (2011) tries to solve the problem caused by repetitive merges and disk access in the Reduce phase using a network-levitated merge algorithm. However, they do not discuss the Reducer Replacement Problem (RPP), caused by an increase of network traffic. In turn, Ho et al. (2011) attack the RPP problem minimizing the network traffic, but they do not mention the impact on disk access. In both cases, the experiments conducted do not refer to threats of validity.

Finally, analytical models and simulations were used to validate few studies, 25% and 13% respectively. Analytical models are a formal way to validate the studies and are generally used in conjunction with implementation. Only 3 out of 18 studies presenting analytical models do not contain implementation (Rasooli and Down, 2011; Guo et al., 2011; Ho et al., 2011). Proposals regarding scheduling, data flow, resource allocation, data storage, and replication were mostly the ones validated by analytical models. On the other hand, simulation is more used in papers involving scheduling. Surely, scheduling is an appro-

Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

priate subject to be validated both with analytical models and simulation, since it is a widely studied topic in Computer Science and well understood from a theoretical point of view. In fact, 7 out of 9 papers presenting simulations were related to scheduling proposals which are also concerned with resource allocation. As well as analytical models, simulations were generally used in conjunction with another validation technique. Just three proposals were validated exclusively via simulation.

Another way of viewing the proposals is to consider the focus of the provided solution. This is addressed by RQ2.3 "The proposed approaches take into account whether the problem being solved is application/data-specific or more generic?". Specific solutions are proposed by 21% of the studies while 79% are general solutions. Specific and general solutions encompass indifferently all topics in our taxonomy. Similarly, specific and general solutions were indistinctly validated by the techniques aforementioned. However, we can point out that all specific solutions were validated at least via an implementation.

6. Related Work

Although some surveys about the MapReduce paradigm may be found in the literature, up to this date, no other systematic literature review specifically about the Apache Hadoop framework was found. Concerning parallel data processing using MapReduce, Lee et al. (2012) present a survey focused on this paradigm. The authors present the MapReduce architecture, its advantages and pitfalls. This survey is not specifically focused on Hadoop, but on the paradigm itself. Therefore, other approaches such as Dryad (Isard et al., 2007), Nephele/PACTs (Battré et al., 2010) and Clustera (DeWitt et al., 2008) are considered in the MapReduce context. The work of Lee *et al.* lists improvements and variants made on these frameworks in terms of high-level languages, data flow, schema, I/O optimizations, scheduling, performance, energy issues, and hybrid systems.

Other surveys approach specific areas in Hadoop, such as scheduling. Yoo and Sim (2011) review a collection of approaches for scheduling jobs for MapReduce considering scheduling issues such as locality, synchronization overhead, and fairness constraints. The work presents two main schedulers, including Hadoop approaches, comparing their features, strengths, and weaknesses. Rao and Reddy (2011) present a similar approach. Dealing with scheduling, the authors present improvements and guidelines on how to improve it in Hadoop on Cloud Environments.

7. Conclusion, Research Opportunities, and Future Work

The Apache Hadoop framework has been widely adopted by both the industry and research communities. A proof of this is the number of publications about the framework, which amounts to a very large number over the past five years. We conducted a systematic literature review as a mean to map the contributions made by several authors to Hadoop. From more than 1500 papers, we have selected 106 that contributed directly to the project. We classified the papers into a taxonomy, which helped to observe areas that are well explored, as well as more recent topics. Hadoop performance, scheduling, data flow modifications, and resource allocation management are the topics that have received the majority of contributions.

After analyzing the selected papers, we were able to draw important conclusions regarding Hadoop development. First of all, we have noticed that the number of publications have increased constantly over the last years, but the main focus on the papers have changed. Since our major concern is the analysis of studies that have Hadoop as an end/final objective, according to the second inclusion criteria, we have selected papers with contributions to the framework. According to our research, the number of publications seems to have reached its peak in 2012, and probably will not experience another peak in the future. That may be explained for several reasons:

- Apache Hadoop has overcome its initial "unstable" phase. The framework have grown solid and stable. HDFS is a reality used to store large files. Performance issues were addressed. MapReduce was consolidated as a distributed computing paradigm.

- Hadoop early adoption by several big companies (Yahoo!, Facebook, Ebay, Amazon, Adobe, among many others) drew attention of many IT companies. Due to its key concept of parallel and distributed abstraction, Hadoop was widely adopted to process data using the MapReduce paradigm. The major Apache Hadoop premise that "users should be writing their programs and obtaining results from their data" became a reality. Aside from configuring correctly the cluster (which, actually is not an easy task), users should not be concerned with task distribution, load balancing, and failures.

- The growth on Hadoop adoption have consolidated because users were able to understand what Apache Hadoop MapReduce is designed for. Applying MapReduce to the appropriate classes of problems yields the expected performance and results.

This arises from the effort made by academia and industry, which have, over the last 6 years, developed a solid solution to process Big Data. The last three years were the apex of Hadoop development, incorporating new projects to its ecosystem, upgrading its architecture, and reaching stability. That is one of the reasons we have seen the number of studies in our research increase constantly from 2010 up to 2012 and decreasing in 2013. But we noticed that from 2012 on, papers are focused on using Hadoop as a consolidated infrastructure to solve existent problems, without significant modifications to the framework; a proof of that is the large number of solutions produced over the last three years that can be applied to different problems.

Another consideration is that Cloud Computing platforms have also consolidated and provided correct support for Hadoop and its ecosystem. On the one hand, projects such as Savanna from OpenStack and Apache Mesos are providing support for Hadoop in the Cloud. OpenStack's Savanna provides a means to provision Hadoop clusters on top of OpenStack according to the users needs, while Apache Mesos provides resource isolation, sharing them across distributed applications of frameworks, which enables the possibility to run, e.g., Apache Hadoop and MPI on the same cluster, bringing flexibility to the infrastructure. Both projects are open source and are evolving rapidly. On the other hand, platforms such as the well-established Amazon Elastic MapReduce, and products from several other companies provide ready-to-go platforms (Hortonworks, EMC, Cloudera, MapR, and many others), bringing to reality the possibility of Hadoop over Cloud Computing tailored to user's needs and budget.

With this in mind, it is worth to notice that several research efforts conducted by academics do not become direct contributions to Hadoop. Even tough, Apache Hadoop managed to become a stable platform. In general, approaches developed by companies related to the Apache Hadoop project, such as Yahoo! and Facebook are incorporated into the project at some time. On the other hand, although some approaches mentioned in this study could be released as an extra ecosystem component, this step most of the times is not accomplished by researchers. Being an open source framework, the Apache Hadoop community should try to reduce this gap with academic researchers. Solutions described in several of the papers we analyzed could be useful if provided as alternative approaches to different classes of problems in Apache Hadoop. At this point, we can affirm that more generic solutions have a better chance to be incorporated into the Hadoop project than solutions to specific problems. Another point that deserves attention is that the separation between Hadoop 1.x and 2.x in terms of architectural changes may have impact on this generic/specific solutions statement, e.g., the separation between resource management and scheduling in Hadoop 2.x, may have effects on some studies that were concerned with both topics, such as (Zhang et al., 2012b; Nguyen and Shi, 2010; Verma et al., 2011). Most of the scheduling works were developed on Hadoop 0.20.x, before the introduction of MapReduce NextGen, which affects its applicability. Even so, we were able to observe that most studies that were tied to a particular version of Hadoop evolved their works, either implementing new versions or developing new approaches/components that could take a better advantage of the new Hadoop architecture.

A second point to be observed is the absence of formal documentation in the experiments conducted by the authors. This is a problem that was already described in some publications concerning the quality of the research that we were able to confirm. We observed a focus in almost all studies on the infrastructure used in the experiments. Data used on experiments sometimes are not clearly described and, most of the times, are not publicly available. This could be considered a threat to validity of many works, since experiments cannot be fully reproduced. This problem could be solved by using concepts such as open data or data provenance. In this case, approaches proposed by Zhou et al. (2012b) and Akoush et al. (2013) are bringing data provenance to the Hadoop platform. Another concern are the benchmarks used in experiments. The absence of publicly available data for experiments, could lead to the use of specific biased datasets, which could favor specific approaches. But, as different areas require different kinds of benchmarks, they would have to be developed according to the area of application, e.g., scheduling, resource allocation, file system, and network communication. Even with the development of new tools, authors should register their experiments in a formal way and make them available for further evolution of their research, using adequate data provenance.

Finally, this review was also conducted trying to find out promising areas for research in the MapReduce paradigm, specially in the Hadoop framework. The storage and cloud computing areas have consistently raised the number of publications and new improvements have already been achieved. Although we observed a great evolution, this area is likely to be further explored to enhance the performance in specific areas, such as the HDFS file system. Low-latency times for random reads are still a challenge, since the distributed file system was not originally developed with these features in focus. Another interesting area is the flash storage. These technologies are being introduced into clusters and grids, but their full potential is not explored yet. The analysis of workloads and behavior of Hadoop MapReduce are beginning to be explored and can bring another rush of performance to the framework. Linked to these technologies is the green computing area, which could definitely benefit from the low-power consumption from solid state disks. We did not find a high number of publications in the green computing field. This may represent a gap in the area that needs further research, or because research being conducted are in an initial stage and were not published yet. With new storage technologies becoming cheaper, the intersection among storage, cloud, and green computing will probably deserve further exploration and development, presenting new challenges to the research community.

8. Acknowledgements

This work was partially funded by Fundação Araucária and NAPSoL-PRP/USP.

References

Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proceedings of the VLDB Endowment 2009;2(1):922–33.

Ahmad F, Chakradhar ST, Raghunathan A, Vijaykumar TN. Tarazu: optimizing MapReduce on heterogeneous clusters. SIGARCH Computer Architecture News 2012;40(1):61–74.

Ahmad F, Lee S, Thottethodi M, Vijaykumar T. Mapreduce with communication overlap (marco). Journal of Parallel and Distributed Computing 2013;73(5):608 –20.

Akoush S, Sohan R, Hopper A. Hadoopprov: Towards provenance as a first class citizen in mapreduce. In: Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance. Berkeley, CA, USA: USENIX Association; TaPP'13; 2013. p. 1–4.

An M, Wang Y, Wang W, Sun N. Integrating DBMSs as a read-only execution layer into Hadoop. In: International Conference on Parallel and Distributed Computing, Applications and Technologies. 2010. p. 17–26.

Bajda-Pawlikowski K, Abadi DJ, Silberschatz A, Paulson E. Efficient processing of data warehousing queries in a split execution environment. In:

Proceedings of the International Conference on Management of Data. New York, NY, USA: ACM; 2011. p. 1165–76.

Bakshi K. Considerations for big data: Architecture and approach. In: Aerospace Conference. IEEE; 2012. p. 1–7.

Battré D, Ewen S, Hueske F, Kao O, Markl V, Warneke D. Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In: Proceedings of the 1st Symposium on Cloud Computing. New York, NY, USA: ACM; 2010. p. 119–30.

Beyer KS, Ercegovac V, Gemulla R, Balmin A, Eltabakh MY, Kanne CC, Özcan F, Shekita EJ. Jaql: A scripting language for large scale semistructured data analysis. Proceedings of the VLDB Endowment 2011;4(12):1272–83.

Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquin R. Incoop: MapReduce for incremental computations. In: Proceedings of the 2nd Symposium on Cloud Computing. New York, NY, USA: ACM; volume 7; 2011. p. 1–14.

Bu Y, Howe B, Balazinska M, Ernst MD. The HaLoop approach to large-scale iterative data analysis. The VLDB Journal 2012;21(2):169–90.

Buck JB, Watkins N, LeFevre J, Ioannidou K, Maltzahn C, Polyzotis N, Brandt S. SciHadoop: array-based query processing in Hadoop. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: ACM; volume 66; 2011. p. 1–11.

Chen Q, Zhang D, Guo M, Deng Q, Guo S. SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: 10th International Conference on Computer and Information Technology. IEEE; 2010. p. 2736–43.

Clement A, Kapritsos M, Lee S, Wang Y, Alvisi L, Dahlin M, Riche T. Upright cluster services. In: Proceedings of the 22nd symposium on Operating systems principles. New York, NY, USA: ACM; 2009. p. 277–90. ACM.

Costa P, Pasin M, Bessani A, Correia M. On the performance of byzantine fault-tolerant mapreduce. Dependable and Secure Computing, IEEE Transactions on 2013;10(5):301–13.

Cuadros A, Paulovich F, Minghim R, Telles G. Point placement by phylogenetic trees and its application to visual analysis of document collections. In: Symposium on Visual Analytics Science and Technology. IEEE; 2007. p. 99–106. PeX.

Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th Conference on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association; volume 6; 2004. p. 137–50.

Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM 2008;51(1):107–13.

DeWitt DJ, Paulson E, Robinson E, Naughton J, Royalty J, Shankar S, Krioukov A. Clustera: an integrated computation and data management system. Proceedings of the VLDB Endowment 2008;1(1):28–41.

Dittrich J, Quiané-Ruiz JA, Jindal A, Kargin Y, Setty V, Schad J. Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). Proceedings of the VLDB Endowment 2010;3(1-2):515–29.

Dittrich J, Quiané-Ruiz JA, Richter S, Schuh S, Jindal A, Schad J. Only aggressive elephants are fast elephants. Proceedings of the VLDB Endowment 2012;5(11):1591–602.

Dong B, Qiu J, Zheng Q, Zhong X, Li J, Li Y. A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by PowerPoint files. In: International Conference on Services Computing. IEEE; 2010. p. 65–72.

Elghandour I, Aboulnaga A. ReStore: reusing results of MapReduce jobs. Proceedings of the VLDB Endowment 2012;5(6):586–97.

Elnikety E, Elsayed T, Ramadan H. iHadoop: Asynchronous iterations for MapReduce. In: Third International Conference on Cloud Computing Technology and Science. IEEE; 2011. p. 81–90.

Eltabakh MY, Tian Y, Özcan F, Gemulla R, Krettek A, McPherson J. CoHadoop: flexible data placement and its exploitation in Hadoop. Proceedings of the VLDB Endowment 2011;4(9):575–85.

Elteir M, Lin H, chun Feng W. Enhancing MapReduce via asynchronous data processing. In: 16th International Conference on Parallel and Distributed Systems. IEEE; 2010. p. 397–405.

Facebook . Under the hood: Scheduling MapReduce jobs more efficiently with Corona. 2012. URL: `http://fb.me/Engineering/`.

Fang W, He B, Luo Q, Govindaraju N. Mars: Accelerating MapReduce with graphics processors. Parallel and Distributed Systems, IEEE Transactions on 2011;22(4):608–20.

Gates AF, Natkovich O, Chopra S, Kamath P, Narayanamurthy SM, Olston C, Reed B, Srinivasan S, Srivastava U. Building a high-level dataflow system on top of Map-Reduce: the Pig experience. Proceedings of the VLDB Endowment 2009;2(2):1414–25.

Ghemawat S, Gobioff H, Leung ST. The Google File System. ACM SIGOPS Operating Systems Review 2003;37(5):29–43.

Goiri In, Le K, Nguyen TD, Guitart J, Torres J, Bianchini R. GreenHadoop: leveraging green energy in data-processing frameworks. In: Proceedings of the 7th European Conference on Computer Systems. New York, NY, USA: ACM; 2012. p. 57–70.

Grossman M, Breternitz M, Sarkar V. Hadoopcl: Mapreduce on distributed heterogeneous platforms through seamless integration of hadoop and opencl. In: Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International. 2013. p. 1918–27.

Grover R, Carey M. Extending Map-Reduce for efficient predicate-based sampling. In: 28th International Conference on Data Engineering. IEEE; 2012. p. 486–97.

Guang-hua S, Jun-na C, Bo-wei Y, Yao Z. QDFS: A quality-aware distributed file storage service based on hdfs. In: International Conference on Computer Science and Automation Engineering. IEEE; volume 2; 2011. p. 203–7.

Guo Z, Pierce M, Fox G, Zhou M. Automatic task re-organization in MapReduce. In: International Conference on Cluster Computing. IEEE; 2011. p. 335–43.

Hammoud M, Rehman M, Sakr M. Center-of-Gravity reduce task scheduling to lower MapReduce network traffic. In: International Conference on Cloud Computing. IEEE; 2012. p. 49–58.

Hammoud M, Sakr M. Locality-aware reduce task scheduling for MapReduce. In: Third International Conference on Cloud Computing Technology and Science. IEEE; 2011. p. 570–6.

He C, Lu Y, Swanson D. Matchmaking: A new MapReduce scheduling technique. In: Third International Conference on Cloud Computing Technology and Science. IEEE; 2011a. p. 40–7.

He C, Weitzel D, Swanson D, Lu Y. Hog: Distributed hadoop mapreduce on the grid. In: High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:. 2012. p. 1276–83.

He Y, Lee R, Huai Y, Shao Z, Jain N, Zhang X, Xu Z. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In: 27th International Conference on Data Engineering. IEEE; 2011b. p. 1199–208.

Ho LY, Wu JJ, Liu P. Optimal algorithms for cross-rack communication optimization in MapReduce framework. In: International Conference on Cloud Computing. IEEE; 2011. p. 420–7.

Ibrahim S, Jin H, Lu L, He B, Antoniu G, Wu S. Maestro: Replica-aware map scheduling for MapReduce. In: 12th International Symposium on Cluster, Cloud and Grid Computing. IEEE/ACM; 2012. p. 435–42.

Ibrahim S, Jin H, Lu L, Wu S, He B, Qi L. LEEN: Locality/fairness-aware key partitioning for MapReduce in the cloud. In: Second International Conference on Cloud Computing Technology and Science. 2010. p. 17–24.

Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Operating Systems Review 2007;41(3):59–72.

Iu MY, Zwaenepoel W. HadoopToSQL: a MapReduce query optimizer. In: Proceedings of the 5th European conference on Computer systems. New York, NY, USA: ACM; 2010. p. 251–64.

Jeon H, El Maghraoui K, Kandiraju GB. Investigating hybrid ssd ftl schemes for hadoop workloads. In: Proceedings of the ACM International Conference on Computing Frontiers. New York, NY, USA: ACM; CF '13; 2013. p. 20:1–20:10.

Jiang D, Ooi BC, Shi L, Wu S. The performance of MapReduce: an in-depth study. Proceedings of the VLDB Endowment 2010;3(1-2):472–83.

Kaldewey T, Shekita EJ, Tata S. Clydesdale: structured data processing on MapReduce. In: Proceedings of the 15th International Conference on Extending Database Technology. New York, NY, USA: ACM; 2012. p. 15–25.

Kang Y, suk Kee Y, Miller E, Park C. Enabling cost-effective data processing with smart ssd. In: Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on. 2013. p. 1–12.

Kaushik R, Abdelzaher T, Egashira R, Nahrstedt K. Predictive data and energy management in GreenHDFS. In: International Green Computing Conference and Workshops. 2011. p. 1–9.

Kaushik R, Bhandarkar M, Nahrstedt K. Evaluation and analysis of GreenHDFS: A self-adaptive, energy-conserving variant of the Hadoop Distributed File System. In: Second International Conference on Cloud Computing Technology and Science. IEEE; 2010. p. 274–87.

Khaled A, Husain M, Khan L, Hamlen K, Thuraisingham B. A token-based access control system for RDF data in the clouds. In: Second International Conference on Cloud Computing Technology and Science. 2010. p. 104–11.

Khan S, Hamlen K. Hatman: Intra-cloud trust management for Hadoop. In: International Conference on Cloud Computing. IEEE; 2012. p. 494–501.

Kitchenham B, Charters S. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001; Keele University and Durham University Joint Report; 2007.

Ko SY, Hoque I, Cho B, Gupta I. Making cloud intermediate data fault-tolerant. In: Proceedings of the 1st Symposium on Cloud Computing. New York, NY, USA: ACM; 2010. p. 181–92. ACM.

Kondikoppa P, Chiu CH, Cui C, Xue L, Park SJ. Network-aware scheduling of mapreduce framework ondistributed clusters over high speed networks. In: Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit. New York, NY, USA: ACM; FederatedClouds '12; 2012. p. 39–44. Scheduling, Cloud Computing.

Konishetty VK, Kumar KA, Voruganti K, Rao GVP. Implementation and evaluation of scalable data structure over HBase. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. New York, NY, USA: ACM; 2012. p. 1010–8.

Kousiouris G, Vafiadis G, Varvarigou T. A front-end, Hadoop-based data management service for efficient federated clouds. In: Third International Conference on Cloud Computing Technology and Science. IEEE; 2011. p. 511–6.

Kumar KA, Konishetty VK, Voruganti K, Rao GVP. CASH: Context Aware Scheduler for Hadoop. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. New York, NY, USA: ACM; 2012. p. 52–61.

Kwon Y, Balazinska M, Howe B, Rolia J. SkewTune: mitigating skew in mapreduce applications. In: Proceedings of the International Conference on Management of Data. New York, NY, USA: ACM; 2012. p. 25–36.

Lama P, Zhou X. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In: Proceedings of the 9th International Conference on Autonomic Computing. New York, NY, USA: ACM; ICAC '12; 2012. p. 63–72. Cloud Computing, Resource Allocation.

Laptev N, Zeng K, Zaniolo C. Early accurate results for advanced analytics on MapReduce. Proceedings of the VLDB Endowment 2012;5(10):1028–39.

Lee K, Nam Y, Kim T, Park S. An adaptive data transfer algorithm using block device reconfiguration in virtual mapreduce clusters. In: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference. New York, NY, USA: ACM; CAC '13; 2013. p. 1–8.

Lee KH, Lee YJ, Choi H, Chung YD, Moon B. Parallel data processing with MapReduce: a survey. SIGMOD Record 2012;40(4):11–20.

Lei L, Wo T, Hu C. CREST: Towards fast speculation of straggler tasks in MapReduce. In: 8th International Conference on e-Business Engineering. IEEE; 2011. p. 311–6.

Leo S, Zanetti G. Pydoop: a Python MapReduce and HDFS API for Hadoop. In: Proceedings of the 19th International Symposium on High Performance Distributed Computing. New York, NY, USA: ACM; 2010. p. 819–25.

Li S, Abdelzaher T, Yuan M. TAPA: Temperature aware power allocation in data center with Map-Reduce. In: International Green Computing Conference and Workshops. 2011. p. 1–8.

Liang Y, Li G, Wang L, Hu Y. Dacoop: Accelerating data-iterative applications on map/reduce cluster. In: 12th International Conference on Parallel and Distributed Computing, Applications and Technologies. 2011. p. 207–14.

Liao H, Han J, Fang J. Multi-dimensional index on Hadoop Distributed File System. In: 5th International Conference on Networking, Architecture and Storage. IEEE; 2010. p. 240–9.

Lin H, Ma X, Archuleta J, Feng Wc, Gardner M, Zhang Z. MOON: MapReduce On Opportunistic eNvironments. In: Proceedings of the 19th International Symposium on High Performance Distributed Computing. New York, NY, USA: ACM; 2010. p. 95–106.

Lin HY, Shen ST, Tzeng WG, Lin BS. Toward data confidentiality via integrating hybrid encryption schemes and Hadoop Distributed File System. In: 26th International Conference on Advanced Information Networking and Applications. IEEE; 2012. p. 740–7.

Lin M, Zhang L, Wierman A, Tan J. Joint optimization of overlapping phases in mapreduce. Performance Evaluation 2013;70(10):720 –35. Proceedings of {IFIP} Performance 2013 Conference.

Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: a system for large-scale graph processing. In: Proceedings of the International Conference on Management of Data. New York, NY, USA: ACM; 2010. p. 135–46.

Mandal A, Xin Y, Baldine I, Ruth P, Heerman C, Chase J, Orlikowski V, Yumerefendi A. Provisioning and evaluating multi-domain networked clouds for Hadoop-based applications. In: Third International Conference on Cloud Computing Technology and Science. IEEE; 2011. p. 690–7.

Mao H, Zhang Z, Zhao B, Xiao L, Ruan L. Towards deploying elastic Hadoop in the cloud. In: International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. 2011. p. 476–82.

Mao Y, Wu W, Zhang H, Luo L. GreenPipe: A Hadoop based workflow system on energy-efficient clouds. In: 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum. IEEE; 2012. p. 2211–9. IEEE.

Mikami S, Ohta K, Tatebe O. Using the Gfarm File System as a POSIX compatible storage platform for Hadoop MapReduce applications. In: Proceedings of the 12th International Conference on Grid Computing. Washington, DC, USA: IEEE/ACM; 2011. p. 181–9.

Mohamed H, Marchand-Maillet S. Mro-mpi: Mapreduce overlapping using {MPI} and an optimized data exchange policy. Parallel Computing 2013;39(12):851 –66. Programming models, systems software and tools for High-End Computing.

Nguyen P, Simon T, Halem M, Chapman D, Le Q. A hybrid scheduling algorithm for data intensive workloads in a mapreduce environment. In: Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing. Washington, DC, USA: IEEE Computer Society; UCC '12; 2012. p. 161–7. ACM, Scheduling.

Nguyen T, Shi W. Improving resource efficiency in data centers using reputation-based resource selection. In: International Green Computing Conference. 2010. p. 389–96.

Nykiel T, Potamias M, Mishra C, Kollios G, Koudas N. MRShare: sharing across multiple queries in MapReduce. Proceedings of the VLDB Endowment 2010;3(1-2):494–505.

Olston C, Chiou G, Chitnis L, Liu F, Han Y, Larsson M, Neumann A, Rao VB, Sankarasubramanian V, Seth S, Tian C, ZiCornell T, Wang X. Nova: continuous Pig/Hadoop workflows. In: Proceedings of the International Conference on Management of Data. New York, NY, USA: ACM; 2011. p. 1081–90.

Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig Latin: a not-so-foreign language for data processing. In: Proceedings of the International Conference on Management of Data. New York, NY, USA: ACM; 2008. p. 1099–110.

Oriani A, Garcia I. From backup to hot standby: High availability for hdfs. In: Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on. 2012. p. 131–40.

Owens J, Houston M, Luebke D, Green S, Stone J, Phillips J. GPU Computing. Proceedings of the IEEE 2008;96(5):879–99.

Park J, Lee D, Kim B, Huh J, Maeng S. Locality-aware dynamic VM reconfiguration on MapReduce clouds. In: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing. New York, NY, USA: ACM; 2012. p. 27–36. ACM.

Paulovich FV, Oliveira MCF, Minghim R. The projection explorer: A flexible tool for projection-based multidimensional visualization. In: Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing. Belo Horizonte, Brazil: IEEE; 2007. p. 27–36. PeX.

Polo J, Carrera D, Becerra Y, Torres J, Ayguade and E, Steinder M, Whalley I. Performance-driven task co-scheduling for MapReduce environments. In: Network Operations and Management Symposium. IEEE; 2010. p. 373–80. IEEE.

Rao B, Reddy DL. Survey on improved scheduling in Hadoop MapReduce in cloud environments. International Journal of Computer Applications 2011;34(9):29–33.

Rasooli A, Down DG. An adaptive scheduling algorithm for dynamic heterogeneous Hadoop systems. In: Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research. Riverton, NJ, USA: IBM Corp.; 2011. p. 30–44.

Seo S, Jang I, Woo K, Kim I, Kim JS, Maeng S. HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment. In: International Conference on Cluster Computing and Workshops. IEEE; 2009. p. 1–8.

Shafer J, Rixner S, Cox A. The Hadoop Distributed Filesystem: Balancing

Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

portability and performance. In: International Symposium on Performance Analysis of Systems Software. IEEE; 2010. p. 122–33. IEEE.

Shen Q, Zhang L, Yang X, Yang Y, Wu Z, Zhang Y. SecDM: Securing data migration between cloud storage systems. In: 9th International Conference on Dependable, Autonomic and Secure Computing. IEEE; 2011. p. 636–41.

Shi L, Li X, Tan KL. S3: An efficient Shared Scan Scheduler on MapReduce framework. In: International Conference on Parallel Processing. 2011. p. 325–34.

Shirahata K, Sato H, Matsuoka S. Hybrid map task scheduling for GPU-based heterogeneous clusters. In: Second International Conference on Cloud Computing Technology and Science. 2010. p. 733–40.

Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop Distributed File System. In: Proceedings of the 26th Symposium on Mass Storage Systems and Technologies. Washington, DC, USA: IEEE; 2010. p. 1–10.

Stonebraker M, Abadi D, DeWitt DJ, Madden S, Paulson E, Pavlo A, Rasin A. MapReduce and parallel DBMSs: friends or foes? Communications of the ACM 2010;53(1):64–71.

Tan J, Meng X, Zhang L. Delay tails in MapReduce scheduling. In: Proceedings of the 12th Joint International Conference on Measurement and Modeling of Computer Systems. New York, NY, USA: ACM; 2012a. p. 5–16.

Tan YS, Lee BS, He B, Campbell R. A Map-Reduce based framework for heterogeneous processing element cluster environments. In: 12th International Symposium on Cluster, Cloud and Grid Computing. IEEE/ACM; 2012b. p. 57–64.

Tang Z, Zhou J, Li K, Li R. MTSD: A task scheduling algorithm for MapReduce base on deadline constraints. In: 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum. IEEE; 2012. p. 2012–8. IEEE.

Tanimura Y, Matono A, Lynden S, Kojima I. Extensions to the Pig data processing platform for scalable RDF data processing using Hadoop. In: 26th International Conference on Data Engineering Workshops. IEEE; 2010. p. 251–6.

Tao Y, Zhang Q, Shi L, Chen P. Job scheduling optimization for multi-user MapReduce clusters. In: 4th International Symposium on Parallel Architectures, Algorithms and Programming. 2011. p. 213–7. IEEE.

Thusoo A, Sarma J, Jain N, Shao Z, Chakka P, Zhang N, Antony S, Liu H, Murthy R. Hive - a petabyte scale data warehouse using Hadoop. In: 26th International Conference on Data Engineering. IEEE; 2010. p. 996–1005.

Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R. Hive: a warehousing solution over a Map-Reduce framework. Proceedings of the VLDB Endowment 2009;2(2):1626–9.

Tian C, Zhou H, He Y, Zha L. A dynamic MapReduce scheduler for heterogeneous workloads. In: 8th International Conference on Grid and Cooperative Computing. 2009. p. 218–24.

Verma A, Cherkasova L, Campbell RH. ARIA: automatic resource inference and allocation for mapreduce environments. In: Proceedings of the 8th International Conference on Autonomic Computing. New York, NY, USA: ACM; 2011. p. 235–44.

Verma A, Cherkasova L, Kumar V, Campbell R. Deadline-based workload management for MapReduce environments: Pieces of the performance puzzle. In: Network Operations and Management Symposium. IEEE; 2012. p. 900–5.

Vernica R, Balmin A, Beyer KS, Ercegovac V. Adaptive MapReduce using situation-aware mappers. In: Proceedings of the 15th International Conference on Extending Database Technology. New York, NY, USA: ACM; 2012. p. 420–31.

Wang K, Lin X, Tang W. Predator an experience guided configuration optimizer for hadoop mapreduce. In: Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. 2012. p. 419–26.

Wang L, Tao J, Ranjan R, Marten H, Streit A, Chen J, Chen D. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. Future Generation Computer Systems 2013;29(3):739 –50. Special Section: Recent Developments in High Performance Computing and Security.

Wang Y, Que X, Yu W, Goldenberg D, Sehgal D. Hadoop acceleration through network levitated merge. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: ACM; volume 57; 2011. p. 1–10.

Wei Q, Veeravalli B, Gong B, Zeng L, Feng D. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In: International Conference on Cluster Computing. IEEE; 2010. p. 188–96.

Wei W, Du J, Yu T, Gu X. SecureMR: A service integrity assurance framework for MapReduce. In: Annual Computer Security Applications Conference. 2009. p. 73–82.

White T. Hadoop: The Definitive Guide. 3rd ed. O'Reilly Media, Inc., 2012.

Xie J, Tian Y, Yin S, Zhang J, Ruan X, Qin X. Adaptive preshuffling in hadoop clusters. Procedia Computer Science 2013;18(0):2458 –67. 2013 International Conference on Computational Science.

Xie J, Yin S, Ruan X, Ding Z, Tian Y, Majors J, Manzanares A, Qin X. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In: International Symposium on Parallel Distributed Processing, Workshops and Phd Forum. IEEE; 2010. p. 1–9. IEEE.

Xin M, Li H. An implementation of GPU accelerated MapReduce: Using Hadoop with OpenCL for data- and compute-intensive jobs. In: International Joint Conference on Service Sciences. 2012. p. 6–11.

Yoo D, Sim KM. A comparative review of job scheduling for MapReduce. In: International Conference on Cloud Computing and Intelligence Systems. IEEE; 2011. p. 353–8.

You HH, Yang CC, Huang JL. A load-aware scheduler for MapReduce framework in heterogeneous cloud environments. In: Proceedings of the Symposium on Applied Computing. New York, NY, USA: ACM; 2011. p. 127–32. ACM.

Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European Conference on Computer Systems. New York, NY, USA: ACM; 2010. p. 265–78.

Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I. Improving MapReduce performance in heterogeneous environments. In: Proceedings of the 8th Conference on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association; volume 8; 2008. p. 29–42.

Zhang C, De Sterck H. CloudBATCH: A batch job queuing system on clouds with Hadoop and HBase. In: Second International Conference on Cloud Computing Technology and Science. 2010. p. 368–75.

Zhang J, Yu X, Li Y, Lin L. HadoopRsync. In: International Conference on Cloud and Service Computing. 2011a. p. 166–73.

Zhang X, Feng Y, Feng S, Fan J, Ming Z. An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In: Proceedings of the International Conference on Cloud and Service Computing. Washington, DC, USA: IEEE; 2011b. p. 235–42.

Zhang X, Wang G, Yang Z, Ding Y. A two-phase execution engine of reduce tasks in Hadoop MapReduce. In: International Conference on Systems and Informatics. 2012a. p. 858–64.

Zhang X, Zhong Z, Feng S, Tu B, Fan J. Improving data locality of MapReduce by scheduling in homogeneous computing environments. In: 9th International Symposium on Parallel and Distributed Processing with Applications. IEEE; 2011c. p. 120–6. IEEE.

Zhang Y, Gao Q, Gao L, Wang C. iMapReduce: A distributed computing framework for iterative computation. In: International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. IEEE; 2011d. p. 1112–21. IEEE.

Zhang Z, Cherkasova L, Verma A, Loo BT. Optimizing completion time and resource provisioning of Pig programs. In: 12th International Symposium on Cluster, Cloud and Grid Computing. IEEE/ACM; 2012b. p. 811–6.

Zhao Y, Wang W, Meng D, Lv Y, Zhang S, Li J. TDWS: A job scheduling algorithm based on MapReduce. In: 7th International Conference on Networking, Architecture and Storage. IEEE; 2012. p. 313–9.

Zhou W, Han J, Zhang Z, Dai J. Dynamic random access for Hadoop Distributed File System. In: 32nd International Conference on Distributed Computing Systems Workshops. 2012a. p. 17–22.

Zhou W, Mapara S, Ren Y, Li Y, Haeberlen A, Ives Z, Loo BT, Sherr M. Distributed time-aware provenance. Proc VLDB Endow 2012b;6(2):49–60.

Zhou Z, Zhang H, Du X, Li P, Yu X. Prometheus: Privacy-aware data retrieval on hybrid cloud. In: INFOCOM, 2013 Proceedings IEEE. 2013. p. 2643–51.

Zhu H, Chen H. Adaptive failure detection via heartbeat under Hadoop. In: Asia-Pacific Services Computing Conference. IEEE; 2011. p. 231–8.

Zikopoulos P, Eaton C. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. Mcgraw-hill, 2011. URL: `http://books.google.com.br/books?id=Plcg_9NJ_fUC`.

Appendix A. Validation Techniques Used in the Selected Studies

	Implementation	Experiments
Scheduling	Tian et al. (2009); Zhang et al. (2012a); Vernica et al. (2012); Zhang et al. (2011b); Verma et al. (2011); Kumar et al. (2012); Hammoud et al. (2012); Zhang and De Sterck (2010); Liang et al. (2011); Tan et al. (2012a); Goiri et al. (2012); Seo et al. (2009); Shirahata et al. (2010); Elnikety et al. (2011); Nguyen and Shi (2010); Ibrahim et al. (2010); Hammoud and Sakr (2011); He et al. (2011a); Nykiel et al. (2010); Shi et al. (2011); Chen et al. (2010); Ahmad et al. (2012); Zhao et al. (2012); Bu et al. (2012); Bhatotia et al. (2011); Lin et al. (2010); Verma et al. (2012); Ibrahim et al. (2012); Mao et al. (2012); Polo et al. (2010); Tan et al. (2012b); Tang et al. (2012); Tao et al. (2011); You et al. (2011); Zhang et al. (2011c, 2012b); Kondikoppa et al. (2012); Nguyen et al. (2012); Wang et al. (2012); Lin et al. (2013); Mohamed and Marchand-Maillet (2013)	Tian et al. (2009); Vernica et al. (2012); Zhang et al. (2011b); Verma et al. (2011); Guo et al. (2011); Kumar et al. (2012); Hammoud et al. (2012); Liang et al. (2011); Tan et al. (2012a); Goiri et al. (2012); Seo et al. (2009); Shirahata et al. (2010); Elnikety et al. (2011); Nguyen and Shi (2010); Ibrahim et al. (2010); Hammoud and Sakr (2011); Nykiel et al. (2010); Ahmad et al. (2012); Zhao et al. (2012); Bu et al. (2012); Bhatotia et al. (2011); Lin et al. (2010); Verma et al. (2012); Ibrahim et al. (2012); Mao et al. (2012); Polo et al. (2010); Tan et al. (2012b); Tang et al. (2012); Tao et al. (2011); You et al. (2011); Zhang et al. (2011c, 2012b); Kondikoppa et al. (2012); Nguyen et al. (2012); Wang et al. (2012); Lin et al. (2013); Mohamed and Marchand-Maillet (2013)
Data flow	Zhu and Chen (2011); Vernica et al. (2012); Liang et al. (2011); Tan et al. (2012a); Laptev et al. (2012); Elteir et al. (2010); Grover and Carey (2012); Wang et al. (2011); Seo et al. (2009); Shirahata et al. (2010); Elnikety et al. (2011); Ibrahim et al. (2010); Hammoud and Sakr (2011); He et al. (2011a); Olston et al. (2011); Ho et al. (2011); Shi et al. (2011); Buck et al. (2011); Kwon et al. (2012); Bu et al. (2012); Bhatotia et al. (2011); Lin et al. (2010); Verma et al. (2012); Zhang et al. (2011d); Ahmad et al. (2013); Lin et al. (2013); Mohamed and Marchand-Maillet (2013); Xie et al. (2013)	Zhu and Chen (2011); Vernica et al. (2012); Liang et al. (2011); Tan et al. (2012a); Laptev et al. (2012); Elteir et al. (2010); Grover and Carey (2012); Wang et al. (2011); Seo et al. (2009); Shirahata et al. (2010); Elnikety et al. (2011); Ibrahim et al. (2010); Hammoud and Sakr (2011); Olston et al. (2011); Ho et al. (2011); Buck et al. (2011); Kwon et al. (2012); Bu et al. (2012); Bhatotia et al. (2011); Lin et al. (2010); Verma et al. (2012); Zhang et al. (2011d); Ahmad et al. (2013); Lin et al. (2013); Mohamed and Marchand-Maillet (2013); Xie et al. (2013)
Resource Allocation	Tian et al. (2009); Zhang et al. (2012a); Zhu and Chen (2011); Zhang et al. (2011b); Verma et al. (2011); Kumar et al. (2012); Hammoud et al. (2012); Zhang and De Sterck (2010); Liang et al. (2011); Grover and Carey (2012); Goiri et al. (2012); Seo et al. (2009); Shirahata et al. (2010); Nguyen and Shi (2010); Hammoud and Sakr (2011); He et al. (2011a); Nykiel et al. (2010); Mandal et al. (2011); Chen et al. (2010); Li et al. (2011); Ahmad et al. (2012); Zhao et al. (2012); Mao et al. (2011); Lin et al. (2010); Verma et al. (2012); Ibrahim et al. (2012); Mao et al. (2012); Park et al. (2012); Polo et al. (2010); Tan et al. (2012b); Tang et al. (2012); Tao et al. (2011); You et al. (2011); Zhang et al. (2011c, 2012b, 2011d); Costa et al. (2013); Kondikoppa et al. (2012); Lama and Zhou (2012); Wang et al. (2013)	Tian et al. (2009); Zhu and Chen (2011); Zhang et al. (2011b); Verma et al. (2011); Guo et al. (2011); Kumar et al. (2012); Hammoud et al. (2012); Liang et al. (2011); Grover and Carey (2012); Goiri et al. (2012); Seo et al. (2009); Shirahata et al. (2010); Nguyen and Shi (2010); Hammoud and Sakr (2011); Nykiel et al. (2010); Mandal et al. (2011); Ahmad et al. (2012); Zhao et al. (2012); Mao et al. (2011); Lin et al. (2010); Verma et al. (2012); Ibrahim et al. (2012); Mao et al. (2012); Park et al. (2012); Polo et al. (2010); Tan et al. (2012b); Tang et al. (2012); Tao et al. (2011); You et al. (2011); Zhang et al. (2011c, 2012b, 2011d); Costa et al. (2013); Kondikoppa et al. (2012); Lama and Zhou (2012); Wang et al. (2013)
Storage & Replication	Wei et al. (2010); Eltabakh et al. (2011); Zhou et al. (2012a); Bajda-Pawlikowski et al. (2011); Goiri et al. (2012); Dittrich et al. (2012); Guang-hua et al. (2011); He et al. (2011b); Wei et al. (2009); Lin et al. (2012); Mikami et al. (2011); Verma et al. (2012); Clement et al. (2009); Ko et al. (2010); Shafer et al. (2010); Xie et al. (2010); Costa et al. (2013); Kang et al. (2013)	Dong et al. (2010); Wei et al. (2010); Eltabakh et al. (2011); Zhou et al. (2012a); Bajda-Pawlikowski et al. (2011); Goiri et al. (2012); Dittrich et al. (2012); Guang-hua et al. (2011); He et al. (2011b); Lin et al. (2012); Mikami et al. (2011); Verma et al. (2012); Clement et al. (2009); Ko et al. (2010); Shafer et al. (2010); Xie et al. (2010); Costa et al. (2013); Jeon et al. (2013); Kang et al. (2013)
Cloud Storage	Kousiouris et al. (2011); Wei et al. (2010); Zhang and De Sterck (2010); Zhang et al. (2011a); Guang-hua et al. (2011); Shen et al. (2011); Dong et al. (2010); Ko et al. (2010); He et al. (2012); Lee et al. (2013)	Wei et al. (2010); Zhang et al. (2011a); Guang-hua et al. (2011); Dong et al. (2010); Ko et al. (2010); He et al. (2012); Lee et al. (2013)
Cloud Computing	Khaled et al. (2010); Zhang et al. (2011b); Hammoud et al. (2012); Zhang et al. (2011a); Khan and Hamlen (2012); Nguyen and Shi (2010); Mandal et al. (2011); Guang-hua et al. (2011); Shen et al. (2011); Ko et al. (2010); Mao et al. (2012); Park et al. (2012); Tang et al. (2012); Zhang et al. (2012b); He et al. (2012); Lama and Zhou (2012); Zhou et al. (2013); Wang et al. (2013); Ahmad et al. (2013)	Zhang et al. (2011b); Hammoud et al. (2012); Zhang et al. (2011a); Khan and Hamlen (2012); Nguyen and Shi (2010); Mandal et al. (2011); Guang-hua et al. (2011); Ko et al. (2010); Mao et al. (2012); Park et al. (2012); Tang et al. (2012); Zhang et al. (2012b); He et al. (2012); Lama and Zhou (2012); Zhou et al. (2013); Wang et al. (2013); Ahmad et al. (2013)
Indexing	Dong et al. (2010); Dittrich et al. (2010); An et al. (2010); Liao et al. (2010); Dittrich et al. (2012)	Dong et al. (2010); Dittrich et al. (2010); An et al. (2010); Liao et al. (2010); Dittrich et al. (2012)
Random Access	Dong et al. (2010); Zhou et al. (2012a); Liao et al. (2010)	Dong et al. (2010); Zhou et al. (2012a); Liao et al. (2010)
DBMS	Kaldewey et al. (2012); Bajda-Pawlikowski et al. (2011); Tanimura et al. (2010); Dittrich et al. (2010); Abouzeid et al. (2009); Konishetty et al. (2012); An et al. (2010); He et al. (2011b)	Kaldewey et al. (2012); Bajda-Pawlikowski et al. (2011); Tanimura et al. (2010); Dittrich et al. (2010); Abouzeid et al. (2009); Konishetty et al. (2012); An et al. (2010); He et al. (2011b)

Table A.6: Studies with Implementation and/or Experiments (MapReduce and Data Storage & Manipulation Categories)

Journal of Network and Computer Applications. Volume 46, November 2014, Pages 1–25
http://dx.doi.org/10.1016/j.jnca.2014.07.022

	Implementation	Experiments
Queries	Vernica et al. (2012); Gates et al. (2009); Kaldewey et al. (2012); Bajda-Pawlikowski et al. (2011); Tanimura et al. (2010); Dittrich et al. (2010); Abouzeid et al. (2009); Iu and Zwaenepoel (2010); An et al. (2010); Nykiel et al. (2010); Liao et al. (2010); Olston et al. (2011); Dittrich et al. (2012); Olston et al. (2008); He et al. (2011b); Buck et al. (2011)	Vernica et al. (2012); Gates et al. (2009); Kaldewey et al. (2012); Bajda-Pawlikowski et al. (2011); Tanimura et al. (2010); Dittrich et al. (2010); Abouzeid et al. (2009); Iu and Zwaenepoel (2010); An et al. (2010); Nykiel et al. (2010); Liao et al. (2010); Olston et al. (2011); Dittrich et al. (2012); He et al. (2011b); Buck et al. (2011)
Hadoop Ecosystem	Kousiouris et al. (2011); Gates et al. (2009); Bajda-Pawlikowski et al. (2011); Grover and Carey (2012); Tanimura et al. (2010); Iu and Zwaenepoel (2010); Konishetty et al. (2012); Olston et al. (2011, 2008); He et al. (2011b); Elghandour and Aboulnaga (2012); Clement et al. (2009); Zhang et al. (2012b)	Gates et al. (2009); Bajda-Pawlikowski et al. (2011); Grover and Carey (2012); Tanimura et al. (2010); Iu and Zwaenepoel (2010); Konishetty et al. (2012); Olston et al. (2011); He et al. (2011b); Elghandour and Aboulnaga (2012); Clement et al. (2009); Zhang et al. (2012b)
New Ecosystem Component	Kaldewey et al. (2012); Zhou et al. (2012a); Bajda-Pawlikowski et al. (2011); Wang et al. (2011); Dittrich et al. (2010); Abouzeid et al. (2009); Nguyen and Shi (2010); An et al. (2010); Fang et al. (2011); Olston et al. (2011); Guang-hua et al. (2011); Elghandour and Aboulnaga (2012); Buck et al. (2011); Wei et al. (2009); Mao et al. (2011); Mikami et al. (2011); Bhatotia et al. (2011); Leo and Zanetti (2010); Lin et al. (2010); Clement et al. (2009); Zhang et al. (2011d)	Kaldewey et al. (2012); Zhou et al. (2012a); Bajda-Pawlikowski et al. (2011); Wang et al. (2011); Dittrich et al. (2010); Abouzeid et al. (2009); Nguyen and Shi (2010); An et al. (2010); Fang et al. (2011); Olston et al. (2011); Guang-hua et al. (2011); Elghandour and Aboulnaga (2012); Buck et al. (2011); Mao et al. (2011); Mikami et al. (2011); Bhatotia et al. (2011); Leo and Zanetti (2010); Lin et al. (2010); Clement et al. (2009); Zhang et al. (2011d)
Green Computing & Energy	Goiri et al. (2012); Shirahata et al. (2010); Nguyen and Shi (2010); Li et al. (2011); Mao et al. (2012)	Goiri et al. (2012); Shirahata et al. (2010); Nguyen and Shi (2010); Mao et al. (2012)
GPGPU	Xin and Li (2012); Shirahata et al. (2010); Fang et al. (2011); Tan et al. (2012b); Grossman et al. (2013)	Xin and Li (2012); Shirahata et al. (2010); Fang et al. (2011); Tan et al. (2012b); Grossman et al. (2013)
Data Security & Crypto	Khaled et al. (2010); Zhang and De Sterck (2010); Khan and Hamlen (2012); Shen et al. (2011); Wei et al. (2009); Lin et al. (2012); Zhou et al. (2013)	Khan and Hamlen (2012); Lin et al. (2012); Zhou et al. (2013)

Table A.7: Studies with Implementation and/or Experiments (Ecosystem and Miscellaneous Categories)

	Analytical Model	Simulation
Scheduling	Tian et al. (2009); Zhang et al. (2012a); Rasooli and Down (2011); Verma et al. (2011); Guo et al. (2011); Tan et al. (2012a); Nykiel et al. (2010); Ahmad et al. (2012); Zhao et al. (2012); Bhatotia et al. (2011); Ibrahim et al. (2012); Tang et al. (2012); Tao et al. (2011); Zhang et al. (2012b); Nguyen et al. (2012); Wang et al. (2012); Lin et al. (2013)	Rasooli and Down (2011); Verma et al. (2011); Guo et al. (2011); Kumar et al. (2012); Lei et al. (2011); He et al. (2011a); Chen et al. (2010); Kondikoppa et al. (2012); Nguyen et al. (2012); Lin et al. (2013)
Data flow	Zhu and Chen (2011); Tan et al. (2012a); Elteir et al. (2010); Ho et al. (2011); Kwon et al. (2012); Bhatotia et al. (2011); Zhang et al. (2011d); Lin et al. (2013)	He et al. (2011a); Lin et al. (2013)
Resource Allocation	Tian et al. (2009); Zhang et al. (2012a); Zhu and Chen (2011); Rasooli and Down (2011); Verma et al. (2011); Guo et al. (2011); Nykiel et al. (2010); Li et al. (2011); Ahmad et al. (2012); Zhao et al. (2012); Mao et al. (2011); Ibrahim et al. (2012); Tang et al. (2012); Tao et al. (2011); Zhang et al. (2011d); Costa et al. (2013); Lama and Zhou (2012)	Rasooli and Down (2011); Verma et al. (2011); Guo et al. (2011); Kumar et al. (2012); Lei et al. (2011); He et al. (2011a); Chen et al. (2010); Zhang et al. (2012b); Kondikoppa et al. (2012)
Storage & Replication	Wei et al. (2010); Eltabakh et al. (2011); Guang-hua et al. (2011); Lin et al. (2012); Costa et al. (2013)	Kaushik et al. (2011); Jeon et al. (2013)
Cloud Storage	Wei et al. (2010); Guang-hua et al. (2011); Lee et al. (2013)	Shen et al. (2011)
Cloud Computing	-	-
Indexing	-	-
Random Access	-	-
DBMS	-	-
Queries	Nykiel et al. (2010)	-
Hadoop Ecosystem	-	-
New Ecosystem Component	-	-
Green Computing & Energy	Li et al. (2011)	Kaushik et al. (2011)
GPGPU	-	-
Data Security & Crypto	Lin et al. (2012); Zhou et al. (2013)	Shen et al. (2011)

Table A.8: Studies with Analytical Models and/or Simulation