PERSPECTIVE ARTICLE

Testing MapReduce Programs: A Systematic Mapping Study

Jesús Morán* | **Claudio de la Riva** | **Javier Tuya**

Department of computing, University of Oviedo, Asturias, Spain

Correspondence
*Jesús Morán, Department of computing, University of Oviedo, Gijón, Spain. Email: moranjesus@uniovi.es

Present Address
Campus de Viesques, 33394, Gijón, Spain

Summary

Context: MapReduce is a processing model used in Big Data to facilitate the analysis of large data under a distributed architecture.
Objective: The aim of this study is to identify and categorise the state-of-the-art of software testing in MapReduce applications, determining trends and gaps.
Method: Systematic mapping study to discuss and classify according to international standards 54 relevant studies in relation to: reasons for testing, types of testing, quality characteristics, test activities, tools, roles, processes, test levels, and research validations.
Results: The principal reasons for testing MapReduce applications are performance issues, potential failures, issues related to the data, or to satisfy the agreements with efficient resources. The efforts are focused on performance and, to a lesser degree, on functionality. Performance testing is carried out through simulation and evaluation, whereas functional testing considers some program characteristics (such as specification and structure). Despite the type of testing, the majority of efforts are focused at the unit and integration test level of the specific MapReduce functions without considering other parts of the technology stack.
Conclusions: Researchers have both opportunities and challenges in performance and functional testing, and there is room to improve their research though the use of mature and standard validation methods.

KEYWORDS:
Software testing, Systematic mapping study, MapReduce, Big Data Engineering

1 | INTRODUCTION

Big Data or data-intensive programs are those that cannot run using traditional technology/techniques[1] and usually need novel approaches. *MapReduce* is one of the most important processing models used in *Big Data* based on the "divide and conquer" principle[2]. *MapReduce* programs run two functions in a distributed infrastructure; the *Map* function splits one problem into several subproblems (divide) and the *Reduce* function solves each subproblem (conquer). There are several technologies that execute and manage *MapReduce* programs such as *Spark*[3], *Flink*[4] and *Hadoop*[5], all widely implemented in industry[6]. It is necessary to ensure the quality of these programs, especially those employed in critical sectors like health or security, such as DNA alignment or for image processing in ballistics with *MapReduce*[7,8]. These new approaches to processing large data in general, and *MapReduce* in particular, have several characteristics that could have an impact on program quality, for example: (1) analysis of large quantities of data, (2) variety of the input information, (3) data without an apparent data model (schema-less), (4) program optimizations to obtain better performance, (5) implementation of the data models in each program (schema-on-read), (6) execution over heterogeneous infrastructure, and (7) automatic mechanisms to manage the resources (for example, scaling and fault tolerance).

There are several approaches to improve quality, and software testing is one of the most commonly used. According to the ISO/IEC/IEEE 29119-1:2013 standard[9], software testing aims to provide information relating to program quality and the potential impacts/risks of poor quality.

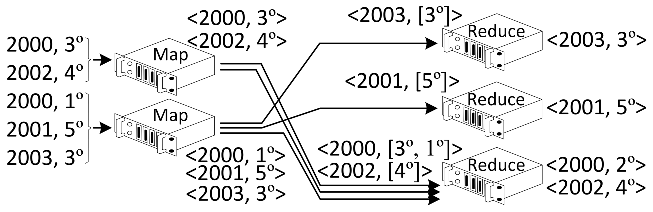

FIGURE 1 Example of the MapReduce program that calculates the average temperature per year.

Software testing research has evolved in recent years[10], but there are several challenges related to the testing of programs in cloud and adaptive architectures[11].

The adoption of and interest in these technologies/paradigms has increased over the last few years to the extent that several Fortune 1000 enterprises consider *Big Data* critical for business[12]. Despite the importance of these applications, some studies predicted that 60% of *Big Data* projects fail to go beyond piloting and would be abandoned in 2017[13]. There are several challenges and concerns: poor data quality[14,15], lack of technological skills[16,17], and various different technological issues such as complexity[18], maturity[19], operability[20] and technical problems[14]. Those problems complicate development and *MapReduce* applications could potentially be implemented with faults. Although software testing is one of the quality assurance techniques most frequently used to evaluate software products, there are not many studies relating to *MapReduce* applications. The contribution of this paper is an evaluation and characterization of the state-of-the-art of software testing in *MapReduce* applications through a *systematic mapping study*[21,22,23]. In this type of study, research questions are proposed and then answered based on relevant literature. The research questions are usually aimed at structuring the state-of-the-art[24] and in general are broader than in the *systematic literature reviews*[25]. The current research questions are: Why, What, How, By whom, Where and When is testing performed in *MapReduce* programs?

A *mapping study* by Sharma et al.[26] on *Big Data* and *Hadoop*[5] indicates that the number of papers has increased significantly in recent years. This interest in *Big Data* during the previous years could have evolved the state-of-the-art of software testing in the *MapReduce* programs. Another *mapping study* was undertaken in 2013 by Camargo et al.[27] on software testing in *MapReduce* programs. Their study analyses only 14 papers and the results are focused on what types of faults the *MapReduce* programs have, how to perform the tests, as well as the tools and the testing techniques used. In contrast to the aforementioned *mapping study*, this paper obtains more thorough results because of its deeper scope and different approach/motivation. The main differences between this *mapping study* and that of Camargo et al.[27] are: (1) broader research questions to analyze the software testing field in a more holistic way than ad-hoc or specific research questions, (2) broader and more generalized results obtained through the research questions, (3) relevant literature obtained through a large search involving more sources, (4) almost quadruple the number of papers analyzed in depth to improve the results, (5) deeper analysis of the papers based on several international standards in order to obtain accurate results, and (6) inclusion of recent research lines.

The paper continues as follows. Section 2 introduces *MapReduce* and describes the main challenges from the testing point of view. The research questions are proposed in Section 3 together with the systematic steps planned to answer them. The execution of these steps (conducting) is described in Section 4. The answers to research questions and other results are detailed in Section 5. These results are discussed in Section 6. The confidence of the results obtained from both planning and conducting is enumerated in Section 7. Finally, Section 8 contains the conclusions.

2 | MAPREDUCE PROCESSING MODEL

MapReduce programs[2] divide one problem into several subproblems that are executed in parallel over a large number of computers. The programs have two principal functions: (1) *Map*, that analyses parts of the input data and classifies them into subproblems, and (2) *Reduce*, that solves each of these subproblems. The data processed by these functions are handled internally in the form of <key, value> pairs. The 'key' is the identifier of each subproblem and the value contains information that the subproblem needs to solve. To illustrate *MapReduce*, let us imagine a program that calculates the average temperature per year. This problem could be divided into one subproblem per year, then each subproblem only solves the average temperature in one year. In this program, the 'key' is the year because it identifies each subproblem, wheras the 'value' is the temperature of this year because this information is needed to solve the subproblem. Figure 1 details a distributed execution of the program analyzing the years 2000-2003. Firstly, the *Map* function receives the data pertaining to years and temperatures and creates the <key, value> pairs with <year, temperature>. For example, <2000, 3°> means that 3° is needed to solve the subproblem that calculates the average temperature of 2000. Then the *Reduce* function receives from all *Maps* one year with all of its temperatures, and calculates the average. For example, if one *Reduce* function receives the data that in the year 2000 there were 3° and 1° temperatures, that is <2000, [3°, 1°]>, then the average temperature for the year is 2°.

The programs are executed by a framework that automatically manages the resource allocation, the re-execution of one part of the program in case of infrastructure failures, and the scheduling of all executions, among other mechanisms. The data analyzed could be stored in several distributed sources, such as non-relational databases and distributed file systems.

The integration of all of these technologies in the *MapReduce* program stack presents a challenge for developers and testers. Some technologies do not scale well, do not support indexing, or do not support ACID transactions, among others issues. Another challenge is the implementation of the data model in the program. *MapReduce* can analyze raw data without a data model (schema-less or unstructured) because the modeling of the data is codified in the program (schema-on-read). When considering the large data scale, it is difficult to establish a model for all data and there are several issues related to poor data quality, such as missing data, noise or incorrect data. Another problem is that new raw data are continuously generated and the data model could change over time, and then the program would need some changes.

The balance and the statistical properties of the data can also change over time and they can affect the program, especially if there are performance optimizations in the code based on data property assumptions. For example, suppose that in the program that analyzes the average temperature per year, the last two years contain 80% of the data. In this case there could be at least two issues: (1) performance problems if these two years are analyzed in the same computer, and (2) memory leaks or resource issues due to the high quantity of data analyzed by one computer. A further challenge is the type of processing implemented; originally *MapReduce* analyzed the data only in batches, but nowadays there are streaming or iterative approaches, among others. For example, the temperature sensors create streams of data, and so the calculation of the average temperature is more efficient using a streaming approach, but it is more difficult to implement and not all programs could be processed in this way. In some domains it is better to change the <key, value> approach to another that permits better modeling of the program, such as Pangool[28], that uses tuples, or more complex structures like graphs[29].

In the main framework of *MapReduce*, *Hadoop*, there are a lot of configuration parameters that could affect the execution in terms of resources, data replications and so on. More than 25 of these parameters are significant in terms of performance[30]. The developer does not know the resources available when the program is deployed because the cluster continuously changes (new resources adding to scale or infrastructure failures[31]), and this also makes the optimal configuration difficult. There are other advanced functionalities of *MapReduce* that could optimize the program, such as for example the *Combine* function. The problem is that if these functionalities are not well established there could be some side effects, such as incorrect output.

In *Big Data* there are also other testing issues related to the ethical use of data. Different security procedures and policies should be considered in *MapReduce* programs throughout the data lifecycle. For example, the analysis of some data could be forbidden in the next season due to agreements with the data provider or due to legal issues. In other cases, the data should be anonymized or encrypted, especially any sensitive data.

Several generic tools are used in the industry to test *MapReduce* programs, such as JUnit[32] with mocks. In order to facilitate the testing of *MapReduce* programs, MRUnit[33] runs the unit test cases without a cluster infrastructure. Another approach is MiniCluster[34] that simulates a cluster infrastructure in memory, or Herriot[35] that interacts with real infrastructure allowing finer-grained control, for example by the injection of computer failures that alter the execution of the program. There are different types of infrastructure failures that affect test execution and several tools simplify their injection such as AnarchyApe[36], ChaosMonkey[37] or Hadoop Injection Framework[38]. The remainder of this paper analyses and summarizes the efforts of the research studies that are focused on covering the issues related to testing *MapReduce* applications.

3 | PLANNING OF THE MAPPING STUDY

This *mapping study* aims to characterize the knowledge of software testing approaches for *MapReduce* programs through a study of the exisiting research literature. To avoid bias, the planning of the *mapping study* describes several tasks based on the guidelines from Kitchenham et al.[22]:

1. Formulation of the research questions (Subsection 3.1).
2. The search process to extract the significant literature (primary studies) to answer the research questions (Subsection 3.2).
3. Data extraction to obtain the relevant data from the literature (Subsection 3.3).
4. Data analysis to summarize, mix and put the data into context to answer the questions (Subsection 3.4).

These tasks are planned and then conducted independently as described in Figure 2. The execution (conducting) of the mapping study is summarized in Section 4.

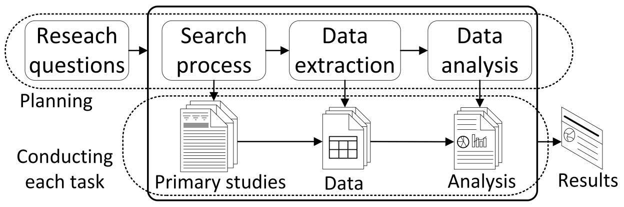

FIGURE 2 Steps of Systematic Mapping Study.

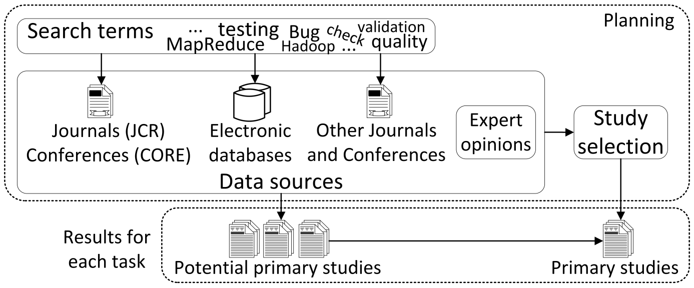

FIGURE 3 Search process to obtain the primary studies in the mapping study.

3.1 | Research Questions

The research questions are formulated to cover all of the information about software testing research in the context of *MapReduce* programs with different points of view. This work formulates the research questions based on the 5W+1H model[39,40], also known as the Kipling method[41]. This method is used in other *systematic reviews* of software engineering[42] and answers the questions: Why, What, How, By whom, Where and When. The research questions of this *mapping study* are:

RQ1. **Why** is testing performed in *MapReduce* programs?

RQ2. **What** testing is performed in *MapReduce* programs?

RQ3. **How** is testing performed in *MapReduce* programs?

RQ4. **By whom**, **where** and **when** is testing performed in *MapReduce* programs?

3.2 | Search Process

The *mapping study* answers the research questions by analyzing the series of studies that contain relevant information about these questions. These studies are called primary studies and are obtained through the tasks described in Figure 3. First, the search terms (set of several words/terms) related to software testing and *MapReduce* are searched for in different data sources (journals, conferences and electronic databases). The papers that match these searches together with other studies recommended by experts constitute the potential primary studies. Finally, these studies are filtered as part of study selection in order to obtain only the studies that contain information which answers the research questions. In the following subsections each of the planning steps is described in detail.

3.2.1 | Search Terms

The search terms are obtained from the three points of view proposed by Kitchenham et al.[22]: population, intervention and outcome. In this *mapping study* the population refers to the technologies and areas related to *MapReduce*, whereas the intervention and outcome refer to the software testing methods and the improvements obtained through software testing.

The search terms of this *mapping study* follow the chain "*MapReduce* technology related terms AND Quality related terms" where:

The *MapReduce* technology related terms correspond with population and are enumerated in Table 1 with synonyms. The selection of the search terms is difficult when technologies are relatively new because the terminology is not well-established[43]. The *Big Data* paradigm and the *MapReduce* processing model are surrounded by a lot of buzzwords like other fields such as Cloud computing. In order to obtain the maximum

TABLE 1 MapReduce technology related terms (population).

Technology	Terms and years of creation
Field	Big Data, Massive data, Large data
Data processing	Hadoop (2006)
-Batch	MapReduce (2004)
-Iterative	Spark (2013), Tez (2013), Stratosphere (2010), Dryad (2007), Flink (2014)
-Streaming	Storm (2011), S4 (2010), Samza (2013)
-Lambda	Lambdoop (2013), Summingbird (2013)
-BSP	Giraph (2013), Hama (2011)
-Interactive	Drill (2012), Impala (2012)
-MPI	Hamster (2011)
Testing	MRUnit (2009), Junit (1998), Mock, MiniMRCluster (2006), MiniYarnMRCluster (2012), Mini cluster (2007), QuerySurge (2011)
Security	Sentry (2013), Kerberos (2007), Knox (2013), Argus (2014)
Resource Manager	Yarn (2012), Corona (2012), Mesos (2009)
MapReduce abstraction	Pig (2008), Hive (2010), Jaql (2008), Pangool (2012), Cascading (2010), Crunch (2011), Mahout (2010), Data fu (2010)
Yarn frameworks	Twill (2013), Reef (2013), Spring (2013)
Yarn integration	Slider (2014), Hoya (2013)
Data integration	Flume (2010), Sqoop (2009), Scribe (2007), Chukwa (2009), Hiho (2010)
Workflow	Oozie (2010), Hamake (2010), Azkaban (2012), Luigi (2012)
Coordinator	Zookeeper (2008), Doozerd (2011), Serf (2013), Etcd (2013)
SDK	Hue (2010), HDInsight (2012), Hdt (2012)
Serialization	Sequence File (2006), Avro (2009), Thrift (2007), Protobuf (2008)
Cluster Management	Ambari (2011), StackIQ (2011), Whte elephant (2012), Ganglia (2007), Cloudera manager (2011), Hprof (2007), MRBench (2008), HiBench (2010), GridMix (2007), PUMA (2012), SWIM (2011)
File system	HDFS (2006), S3 (2006), Kafka (2011), GFS (2003), GPFS (2006), CFS (2013)
Other storage	HBase (2008), Parquet (2013), Accumulo (2008), Hcatalog (2011)
Cluster deployment	Big top (2011), Buildoop (2014), Whirr (2010)
Data Lifecycle	Falcon (2013)

relevant literature and avoid missing some primary studies due to buzzwords and jargon, a thorough search is performed considering the *MapReduce* and *Big Data* related technologies enumerated in Table 1.

Quality related terms correspond with the Quality (sub)characteristics of ISO/IEC 25010:2008-2011[44] and ISO/IEC 9126-1:2001[45] and their synonyms (outcome), together with other testing terms (intervention). Both are enumerated in Table 2.

This work plans a wide search with 9384 combinations of terms in the paper title, obtained by 92 *MapReduce* technology-related terms and 102 quality-related terms.

3.2.2 | Data Sources

The potential primary studies may be found in different data sources. This *mapping study* searches for the studies in the following data sources, grouped in four categories:

a) High-impact journals and conferences. The potential studies are obtained through DBLP[46] with the search terms in 31 JCR journals[47] and 53 CORE conferences[48] enumerated in Appendix A. The journals and conferences selected are related to the software testing or *Big Data*.

b) Electronic databases. The search terms are queried in IEEE Xplore[49], ACM Digital Library[50], Scopus[51], Ei Compendex[52] and ISI Web of Science[53], that are employed in other *mapping studies* of software testing[54] and recommended by Kitchenham et al.[55].

c) Other journals and conferences. Relatively new topics like *MapReduce* and *Big Data* are more likely to be published in specialized workshops/conferences[43]. The non-JCR journals and non-CORE conferences related to software testing or *Big Data* could be a good source of potential

TABLE 2 Quality related terms (outcome and intervention).

Quality characteristics	Terms
Functional suitability	Functionality, functional, suitability, suitable, correctness, correctable, accuracy, accurate, compliance, compliant, appropriateness, appropriate
Performance efficiency	Performance, performable, efficiency, efficient, time-behaviour, resource utilization
Compatibility	Compatibility, replaceability, replaceable, co-existence, interoperability, interoperable
Usability	Recognizability, recognizable, learnability, learnable, operability, operable, ease of use, helpfulness, helpful, attractiveness, attractive, attractivity, technical, accessibility, accessible
Reliability	Reliability, reliable, availability, available, fault tolerance, recoverability, recoverable
Security	Security, secure, safety, confidentiality, confidential, integrity, non-repudiation, accountability, accountable, authenticity, authenticable
Maintainability	Maintainability, maintainable, modularity, modular, reusability, reusable, analyzability, analyzable, changeability, changeable, modification, modifiable, stability, stable, testability, testable
Portability	Portability, portable, adaptability, adaptable, transferability, transferable, installability, installable, effective, effectiveness
Other terms	Testing, assert, assertion, check, checking, test, test case, validate, validation, verify, verification, bug, defect, fault, failure, error, quality, risk, evaluation

primary studies. This *mapping study* searches for studies through DBLP[46] with the search terms in the 33 journals and 49 conferences enumerated in Appendix B.

d) Expert opinions. The three previous categories involve a wide search of software testing studies about *MapReduce* programs, but there could still be relevant studies that would not be identified by this method. The opinion of authors with experience in software testing and *MapReduce*, together with the other related *mapping studies*[27] could provide potential primary studies as Kitchenham et al. suggest[56].

3.2.3 | Study Selection

Study selection is more difficult in *systematic mapping studies* than in *systematic reviews*[55]. Some potential primary studies obtained from the data sources might not contain information about software testing in the *MapReduce* programs. In this *mapping study* a series of filters selects only the studies that contain relevant information that answers the research questions. The potential primary studies that do not pass the filters are excluded, and the remainder make up the primary studies used to answer the research questions. The filters consist of the following exclusion criteria applied in the following order:

C1) Exclusion filter by year. A potential primary study is excluded when the publication year is before the *MapReduce* paper (2004) or before the creation of technologies/fields expressed in the search terms of Table 1.

C2) Exclusion filter by area. Potential primary studies are excluded when their research is not about Computer Science or Information systems.

C3) Exclusion filter by field. Potential primary studies are excluded when they do not contain *Big Data* information.

C4) Exclusion filter by topic. The final filter only includes the studies about software testing in the *MapReduce* programs; the remainder are excluded.

For example, the last filter excludes papers focused on software testing of the underlying technology such as the distributed system *Hadoop*, cloud computing, net or operative system, among others. Despite the normal execution of *MapReduce* programs depends on all these technologies, usually they are mature enough and the developer/tester is only focused on the *MapReduce* application. Some papers that have been excluded are intended to improve the performance of *Hadoop* through infrastructure failure forecasting[57] or to inject infrastructure failures in a distributed file system[58], among other examples that also do not test the *MapReduce* applications. Some other papers employ the *MapReduce* and *Big Data* capabilities to speed up testing in other *non-MapReduce* programs. For example,[59,60] are frameworks to perform unit testing and mutation testing in general programs taking advantage of the parallel capabilities of the *MapReduce* processing model.

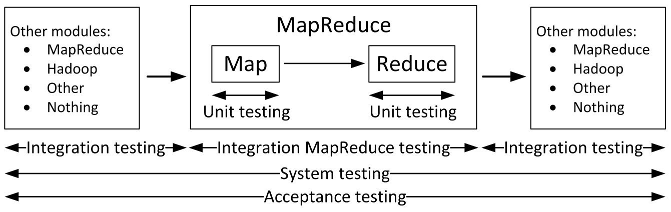

FIGURE 4 Test levels based on ITSQB and adapted to MapReduce.

3.3 | Data Extraction

The relevant information from the primary studies is extracted through a template divided in two parts. The first part is in general based on checklists of international standards related to the research questions, and the second part is focused on other data that could be interesting to analyze. The data extracted for answering the research questions are:

RQ1 "Why is testing performed in *MapReduce* programs?" Extraction of the arguments employed in the primary study to perform testing in *MapReduce* programs.

RQ2 "What testing is performed in *MapReduce* programs?" The data are extracted following two checklists that characterize the type of testing performed in each primary study: a checklist of the 31 ISO/IEC 25010:2011 Quality (sub)characteristics[44], and a checklist of the 17 ISO/IEC/IEEE 29119-4:2015 Quality-Related Types of Testing[61].

RQ3 "How is testing performed in *MapReduce* programs?" The data are extracted by following a checklist of the 11 ISO/IEC/IEEE 29119-1:2013 Annex A: Test activities[9], together with a checklist of test areas as follows: Testing specific to *MapReduce* programs, Testing not specific to *MapReduce* programs (other technologies/paradigms can be tested), Unclear and Not applicable. In addition, the following information about the tools used for testing is extracted: Does the study include the creation of a specific tool or use an existing tool? Is the tool based on another tool? Is the tool available? For example, if the tool is accessible via the Internet or with some type of open source license.

RQ4 "By Whom, where and when is testing performed in *MapReduce* programs?" The data are extracted following three checklists focused on the roles, the lifecycle and the test level. The first checklist contains the following roles: Manager, Analyst, Architect, Tester, Test manager, Test strategist, Other stakeholders, Unclear and Not applicable. These test roles are described in the ISO/IEC/IEEE 29119-1:2013 Annex E[9]. The second checklist contains the 6 ISO/IEC 12207:2008 Software Implementation lower level Processes[62] and the 11 ISO/IEC 12207:2008 System Context Technical processes[62]. The third checklist is based on ISTQB test levels[63] and adapted to *MapReduce* with two changes represented in Figure 4: (1) Unit testing is divided into "Unit testing in *Map* function" and "Unit testing in *Reduce* function", and (2) "Integration testing" is for the integration of the *MapReduce* program with other modules, whereas "Integration *MapReduce* testing" is for the integration between *Map* and *Reduce* functions.

Other data are extracted in the *mapping study* because they may be interesting when characterizing the results and obtaining new findings. These data are extracted in a checklist with the following information about the research validation of the studies:

a) The different types of validation summarized by Mary Shaw[64]: Analysis, Evaluation, Experience, Example, Persuasion and Blatant assertion.

b) Other characterizations of the research: Validation with external programs, Validation with own programs, Another type of validation, Without validation, Unclear, Other and Not applicable.

3.4 | Data Analysis

The data extracted from the primary studies are analyzed in order to answer the research questions. In empirical software engineering there are several methods[65] based on different approaches according to the type of data or research questions, among other things. In this *mapping study* the analysis is performed using (1) thematic analysis[66] to answer RQ1, and (2) meta-ethnography[67] for the remaining research questions. These methods are focused on qualitative data but analyze the data in a different way.

The thematic analysis method is selected to respond to RQ1 (Why is testing performed in *MapReduce* programs?) because it extracts a taxonomy of the reasons for testing from the primary studies. Then RQ1 is answered by a frequency analysis of these reasons for testing. This thematic analysis is performed with a grounded approach[68] that consists of the following steps:

1. Reading of the primary studies.
2. Extraction of the segments/phrases that include the reasons for testing.
3. Creating a group of labels for each previous segment/phrase based on the type of reason for testing.

4. Refining all labels several times until a few labels are obtained that compose a taxonomy of the reasons for testing.

5. Frequency analysis of the reasons for testing employed in the primary studies based on the previous taxonomy.

Meta-ethnography is selected to answer research questions RQ2 to RQ4 because it transforms the data from the primary studies into a more easily analyzable shared context. This method is employed in software engineering[69] and translates all primary studies on data under several facets that contain the checklists described in the data extraction (Section 3.3). Once the data are extracted from the primary studies in these checklists, the research questions are answered by a frequency analysis. This *mapping study* follows the 7 steps proposed by Noblit et al.[67]:

1. Getting started. The topic under analysis is software testing of the *MapReduce* programs and is well studied through *mapping study*.

2. Deciding what is relevant to the initial area of interest. All primary studies are important.

3. Reading the studies. The primary studies are read in order to extract the relevant data.

4. Determining how the studies are related. Primary studies could contain related concepts or very different concepts. The relationship between these concepts is established through the checklists of the data extraction of Section 3.3.

5. Translating the studies into one another. The primary studies are translated into relevant data according to the unified checklists of Section 3.3.

6. Synthesizing translations. This *mapping study* creates more general concepts by the answers of research questions. RQ2 is answered by a frequency analysis of their two checklists, whereas both RQ3 and RQ4 are answered through their three checklists described in Section 3.3.

7. Expressing the synthesis. The research questions are answered and discussed in Section 5 following the previous steps of the *mapping study*.

4 | CONDUCTING THE MAPPING STUDY

This section describes how each step of the *systematic mapping study* was conducted and how all problems were overcome. The planning of the systematic mapping study was refined by the three authors after several iterations.

Search terms: In the first instance, a small number of specific search terms such as *MapReduce* and *Big Data* were defined, but some relevant literature did not match with this search. For example, *Hadoop* is a distributed system that supports the execution of *MapReduce* programs and *non-MapReduce* programs, but there are several papers that use *Hadoop* and *MapReduce* words interchangeably. Other relevant papers do not include the word *MapReduce* in the title, but do contain other words related to the *MapReduce*/*Big Data* ecosystem like Hive, PIG or Spark. Finally, we refined the research method by adding more search terms in order to obtain the maximum amount of relevant literature.

Data sources: The data sources were also refined several times, especially the journals and conferences/workshops. Initially, we planned to analyze only the top journals and conferences such as ICSE. However, we observed that the relevant literature of software testing in *MapReduce* were not published at all in these journals and conferences. Finally, we added more journals and conferences/workshops that might contain relevant literature using both SEWORLD[70], DBLP[46] and our research experience. We added both JCR/CORE and non-JCR/non-CORE venues because a significant number of primary studies are published in this heterogeneity of venues, as we discuss in Section 5.

Study selection: For each data source, one author developed queries using the large number of search terms. This search was difficult to carry out because the software engineering search engines did not adequately support the *mapping studies* searches[71]. To avoid this problem, we created a program that splits the 9384 combinations of search terms in 2346 searches and simulates a human performing these requests. The potential primary studies were obtained over a period of approximately two months in order to avoid bans in the search engines due to a high number of requests. After some months we tried to use this program in another *mapping study*, but the program was obsolete due internal changes in the search engines. As other researchers have noted, we also observe that digital search engines are not well-suited to complex searches[55].

After two months of both automatic and manual searches in 2311 proceedings/volumes (624 from JCR/CORE venues and 1687 from non-JCR/non-CORE venues), in July 2016 we obtained more than 100000 studies represented in Figure 5. Then we removed those that were retrieved several times across different data sources, obtaining thereafter more than 70000 potential primary studies. The majority of these studies were clearly non-relevant for this *mapping study* because they were not focused on software testing in the *MapReduce* programs. Following some practices of other *systematic reviews* of both social science[72] and software engineering[73], those studies that were clearly non-relevant were filtered out by only one of the authors, whereas those studies that were potentially relevant were filtered in parallel by two of the authors. The first filter was applied by only one of the authors because it only excludes those studies that are either published before *MapReduce* or before the technology that matches the query. For example, there were several studies excluded in the first filter because despite the fact that they were retrieved by the words “testing” and “pig”, they were published before the Apache Pig technology (2008) was developed. These studies were usually concerned with testing pigs (the animals) rather than Pig (the software). The majority of studies could be excluded/selected after only reading the title, but in other cases the author needed to read the abstract or the whole paper, in particular when considering the last filters. After the first filter, there were still more than 14000 potential primary studies in consideration.

The second filter excludes those studies that are not related to either computer science or information systems. This filter was also applied by only one author because the studies excluded are clearly non-relevant, such as those about testing pigs (the animals) published after 2008. After the second filter was applied, there were still more than 1500 potential studies. The third filter excludes those studies that are not related to the *Big Data* field. This filter was applied by one author and excluded a few studies, some of which are about "cascading failures" in computer science models or databases that are clearly unrelated to the *Big Data* field. After applying the third filter, there remained more than 1300 potential primary studies.

The fourth filter obtains those studies focused on software testing in the *MapReduce* processing model. This filter and the selection of the primary studies were almost completely applied by two of the authors and the disagreements were discussed by all authors. In the first instance one of the authors excluded 334 studies that are non-relevant because are related to *Big Data Analytics*. The remaining studies, numbering 1043, were related to *Big Data Engineering* and were filtered independently by two of the authors until the primary studies to be used in this paper were obtained. Both authors agreed on 1002 studies: 50 of them passed the filter and were selected as primary studies, and the other 952 did not pass the filter. In contrast, both authors disagreed on 41 studies: one of the authors considered that 35 of them should pass the filter and be selected as primary studies, whereas the other author considered that the other 6 studies should also pass the filter and be selected as primary studies. Despite 96% agreement between both authors, we applied the Kappa coeffient to statistically measure the inter-rater agreement[74]. We obtained 0.69 as a Kappa coefficient with [0.60-0.78] as a 95% confidence interval. This is usually interpreted as substantial[74] or moderate[75] agreement between both authors during the selection of the primary studies. The 4% disagreement (representing 41 studies) were discussed and analyzed by the three authors until total agreement of the primary studies to be used in this paper was achieved. The majority of disagreements were caused by an initial incorrect definition of the *systematic mapping study* plan because one author considered that studies about software testing in Hadoop system should be considered as primary studies, and the other author did not. We refined the plan indicating that the primary studies are only those about software testing in the *MapReduce* processing model and not those about software testing in other technologies or frameworks that do not comprise *MapReduce*. Other disagreements happened because one of the authors did not consider those papers about software testing in *MapReduce* abstractions like Pig and Hive as primary studies. There were other disagreements, for example those papers that instead of testing are related to debugging. After all authors had discussed and resolved the disagreements, 65 studies passed the filter. Some of these studies are the continuation of the same research, such as a conference paper with an improvement published in a journal. The old versions of the studies were excluded keeping only the lastest study. There were several papers from the HP Labs team, but we considered that only three of them are considered primary studies because these studies were distinct from each other. As Section 5 discussed, one of them[76] is focused on obtaining the execution time with microbenchmarks, whereas the other[77] is focused on the cloud cluster using different techniques, and the final study[78] is focused on Pig queries. Finally, 54 unique studies were selected as primary studies.

Data extraction: In order to perform the data extraction, each one of these 54 primary studies were read at least once by two of the authors. Despite the guidelines from Kitchenham et al.[22] which suggest that at least two researchers extract the data independently, other researchers consider it practical that one author extracts the data and the other author checks the extraction[71]. This last practice is applied in software engineering by other *systematic reviews*[79] and we also extracted the data in similar way. One of the authors extracted the data from the primary studies, another author checked the extraction, and the doubts were discussed and resolved by the three authors.

Data analysis: Once the data were extracted, all authors discussed the interpretations and potential results. Then the three authors started to write the findings and the report.

The current *systematic mapping study* took a lot of time despite not being the first conducted by our research group. The time consumed is one of the main criticisms of *systematic reviews*[55]. We specifically expended more time in: (1) creation and execution of a program to support the high number of search queries, (2) selection of the primary studies from a large amount of literature, (3) extraction of the data from each primary study, and (4) refinement of the research method. We performed the *systematic mapping study* two times, initially in 2015 and finally updated with the literature of 2016.

5 | RESULTS

The results were obtained through the execution (conducting) of the *systematic mapping study* that answers the research questions. The primary studies are summarized in Subsection 5.1. From these, the data were extracted, and the analysis is developed in Subsection 5.2 answering the research questions. Other results that do not answer the research questions but remain relevant in characterizing the state-of-art of software testing in *MapReduce* applications are summarized in Subsection 5.3. Finally, the general results are discussed in Section 6.

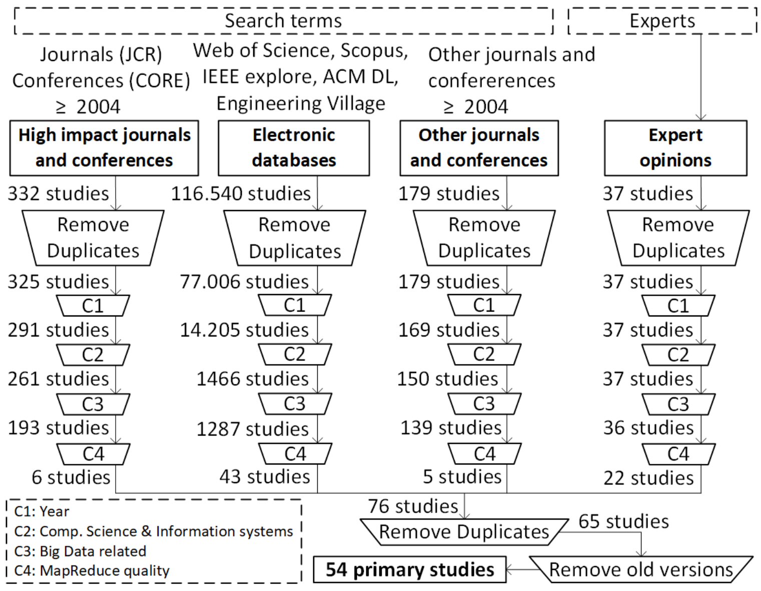

FIGURE 5 Study selection of the primary studies.

TABLE 3 Frequency of the primary studies over time.

Statistics	2010		2011		2012		2013		2014		2015		2016 until July	
Frequency	1	(2%)	7	(13%)	4	(7%)	19	(35%)	12	(22%)	10	(19%)	1	(2%)
Absolute frequency	1	(2%)	8	(15%)	12	(22%)	31	(57%)	43	(80%)	53	(98%)	54	(100%)

5.1 | Primary Studies

In this work, there are 54 primary studies that are derived from more than 70000 potential studies obtained though the search process detailed in Figure 5. These primary studies are detailed in Table C5 of Appendix C with the year of publication, type of contribution and a summary of their contents.

The *MapReduce* processing model was described in 2004, but the software testing efforts in this field according to the primary studies only started in 2010 with only 1 study and after six years and six months the number of primary studies had increased to 54. Table 3 summarizes the frequencies of these primary studies over time and reveals that the research efforts of the topic may have grown because after 2013 the attention increases.

The different types of validations employed in the research are summarized in Table 4. The majority of the studies validate their research through examples (41%) or experience (35%). In 76% of the studies, the validation is carried out by applying the testing research in a program(s), but in 11% of the primary studies the research is not validated.

Testing in *Big Data* has opened up new challenges[80], especially in the understanding of the data and its complex structures[81]. Gudipati et al.[82] establish a classification of testing in the *Big Data* field. This study includes the validation of the *MapReduce* process together with other non-functional characteristics like performance and failover. All of these characteristics are among the main challenges in *Big Data* testing[81]. In order to overcome these challenges though software testing, it is recommended to deploy a distributed environment like production, preferably in the cloud[82,83].

Software testing can be performed in different dimensions and some authors suggest addressing the three Vs of *Big Data* (Volume, Velocity and Variety). In the case of high Volume, it could be difficult to check whether the test case output is the output expected, and the use of automatic tools can be helpful[82]. In the case of Variety such as semi-structured or un-structured data, it can be helpful to transform them in a structured way[82]. To test the Velocity, it is recommended to design performance tests[82]. In addition to Volume, Variety and Velocity, other authors suggest considering the Veracity through data cleaning and normalization[83]. Those four Vs have an impact not only on the program execution, but also on the performance tests[84]. Zhenyu Liu[84] classifies the performance testing in *Big Data* as: (1) concurrent test (the impact of multiple users and

TABLE 4 Number of primary studies per type of validation.

			Number of studies			
Analysis					7 (13%)	54 (100%)
Evaluation					0 (0%)	
Experience					19 (35%)	
Example					22 (41%)	
Persuasion					0 (0%)	
Blatant assertion					6 (11%)	
With validation	**Over programs**	**External programs**	30 (56%)	41 (76%)	47 (87%)	54 (100%)
		Own programs	12 (22%)			
	Other validation		8 (15%)			
Without validation					6 (11%)	

applications in concurrency), (2) load testing (realistic data loads to analyze the response of the program), (3) stress test (testing under extreme data), and (4) capacity test (the analysis of the resources that can be used).

The majority of the primary study papers are focused on capacity and load testing. These studies are summarized in Section 5.1.1, whereas those primary studies that are more related to the functionality are described in Section 5.1.2.

5.1.1 | Performance testing and analysis

In the primary studies, performance analysis is mainly addressed by the simulation of program executions, or by evaluation of a performance prediction model. These prediction models characterize performance based on different kinds of input parameters. The model of Song et al.[85] predicts the execution time given some characteristics about both the input dataset, the program functionality and the programming cluster. In addition, other models obtain the execution time by also considering the file system[86]. The prediction models can have different goals beyond the execution time, for example the Yang et al. model[87] helps to obtain the values of the input parameters that achieve the best execution time. The tester varies the input parameters (the network or the locality of the data, among others) and then analyzes the impact in performance.

Performance can be predicted by using a stochastic approach, for example by Stochastic Petri Nets[88]. Another stochastic model[89] also considers the *MapReduce* tasks that are re-executed due to frequent failures. The performance of *MapReduce* and *Big Data* applications can also be evaluated through large scale stochastic models by Mean Field Analysis[90].

While some models predict performance by analyzing the execution time of several samples[91] or considering previous executions[76], other models consider some specific characteristics of the *MapReduce* execution. The Vianna et al. model[92] considers the influence over the performance of *MapReduce* tasks that are executed in parallel. The network is another issue that can cause bottlenecks in *MapReduce* programs and several models consider the network in order to predict the performance[93,94]. Others also consider the task failures and I/O congestion[95].

Together with the network, memory can cause performance issues, especially in iterative programs or those with high I/O operations. The performance of the shared-memory computation programs can be predicted with the Tanzil et al. model[96], whereas in those programs with Remote Direct Memory Access, the Wasi-ur-Rahman et al. model[97] can be used. Apache Spark[3] programs process the data using distributed memory abstraction and their performance can be predicted by a model that executes a sample of data[98].

The cluster that executes *MapReduce* programs can also influence performance, especially when this cluster is formed by a heterogenic infrastructure. In these clusters, the Zhang et al. model[77] predicts performance within the bounds of upper and lower execution time. Another model that can predict the performance in these clusters employs the machine learning technique Support Vector Machine[99]. There are several clusters deployed in the cloud to obtain several advantages in terms of elasticity and cost. For programs executed in these clusters, performance can be predicted modeling the systems with Layered Queueing Network[100]. In the case of I/O intensive programs in the cloud, performance can be predicted using a CART (Classification And Regression Tree) model[101]. When the programs executed in the public cloud have deadline requirements to satisfy, performance can be predicted with the Locally Weighted Linear Regression model considering the previous execution and the data executed in parallel[102]. For those programs that are not only executed in a public cloud, but in a hybrid cloud, their performance can be predicted with the Ohnaga et al. model[103].

Several frameworks transform queries into *MapReduce* jobs, such as Hive[104] and Pig[105]. The execution time of the Hive SQL-like queries may be forecast using multiple linear regression to predict the execution time of all the *MapReduce* jobs generated from these queries[106]. The multiple regression analysis can be also used to predict the execution time of the join queries in Pig programs[107]. In contrast, the Zhang et al. model[78] predicts the performance of Pig programs considering the previous executions.

In addition to the prediction models, the testers can simulate the execution of the programs to analyze their performance in a fine-grained way. As with the prediction models, the simulators also consider characteristics about the input dataset, the program functionality, the programming cluster and the file system[108]. The MRPerf simulator[109] considers the inter and intra rack interactions over network using ns-2, and can be combined with other simulators, such as DiskSim. The Chauhan et al. simulator[110] is based on MRPerf but including, among others elements, some random time due to operating system scheduling and network communication delays.

The execution time of *MapReduce* programs can also be obtained using the modeling language proposed by Barbierato et al[111]. The tester can also monitor the execution of *MapReduce* programs and test cases, obtaining charts to evaluate performance and potential bottlenecks[112]. Villalpando et al.[113] propose a model for the *Big Data* application establishing a relationship between performance and reliability measures based on the international standard of quality ISO/IEC 25010[44].

Despite there being several research lines concerned with predicting the execution time, there is no comprehensible comparison between them. In general, these studies are evaluated only with a few different case studies. The scientific contribution of these prediction models can be improved with empirical evaluation against other models using a standardized benchmark.

The main difference between these models is not just the technique/approach employed, but also the parameters used by the model. Different characteristics of the input dataset, program functionality, programming cluster and file system are considered as parameters, for example: size of data or number of <key, value> pairs (input dataset), complexity or overhead of Map (program functionality), number of CPU cores or racks (programming cluster), and number of HDFS replicas or the data transfer time for an HDFS block (file system).

There are a lot of different parameters, but there is no clear indication of which parameters have more influence on performance. The contribution of the performance prediction studies can be improved evaluating which parameters really influence performance and which do not. Then the prediction models can be designed with a more standardized subset of parameters that have a notorious influence on performance.

5.1.2 | Functional testing

Misconfiguration is one of the most common problems that lead to memory/performance issues in *MapReduce*[114]. However, according to the empirical study by Ren et al.[115], users rarely tune the configuration parameters that are related to performance. Users usually only turn the configuration parameters that are related to failures[115]. Another empirical study analyzes 200 production failures and determines that the majority of failures are related to the data, and only 1.5% are related to the performance (out of memory)[116]. In production there are several programs that do not finish their execution; Kavulya et al.[117] indicate that around 3% of programs have this problem, and a broader study indicates this percentage falls between 1.38% and 33.11%[115].

An analysis of 507 programs indicates at least 5 different kinds of faults caused by the non-deterministic execution of the *MapReduce*[118]. Camargo et al.[119] classify the specific faults of *MapReduce*, whereas Morán et al.[120] classify those caused by the non-determinism execution. Chen et al.[121] propose a formal approach to detect these faults caused by non-determinism. In contrast, Csallner et al.[122] employs symbolic execution to check the program under test. Another technique to detect the faults caused by non-determinism dynamically checks the properties of the program under test with random data[123]. One of the reasons for the non-deterministic execution is the tolerance of infrastructure failures. There are several studies that propose to inject infrastructure failures in the test case design[124]. Failure Scenario as a Service (FSaaS)[125] injects infrastructure failures into a cluster deployed in the cloud.

Several testing techniques are devised in order to generate test inputs aimed at detecting functional faults, such as those caused by non-deterministic execution or other semantic errors. The MRFlow testing technique[126] generates the test coverage items that can be used to generate test inputs based on the data-flow technique adapted to the *MapReduce* processing model. Another technique to generate data of the test cases employs a bacteriological algorithm aimed to kill some semantic mutants specific to *MapReduce* which varies both the number of the Reducers and the existence (or not) of the Combiner functionality[127]. In those *Big Data* ETL (Extract, Transform and Load) programs that integrate several technologies (*MapReduce*, Pig, Hive, among others), a subset of representative data for test can be obtained from the dataset through input space partition together with constraints[128]. In dataflow programs like Pig, the test inputs can be generated using dynamic-symbolic execution in the control-flow graph of the program[129].

Other kinds of checks can be performed in *MapReduce* programs. Dörre et al.[130] propose an automatic checker that statically detects incompatibilities between the types of the <key, value> pairs processed by *MapReduce* programs. Rabkin et al.[131] statically analyze the configuration parameters used by different frameworks, including Hadoop. The *MapReduce* developers and testers should analyze the configuration parameters used because 17% of Hadoop options are not documented and 6% are not used in the code. The main *Big Data* frameworks can be affected in the same way as Hadoop because these issues are common in open-source programs[131]. The correctness of *MapReduce* programs can also be verified formally through proofs modeling the specification as Coq functions[132].

TABLE 5 Number of primary studies per type of reason for testing.

Types of reasons	Number of papers	Number of formal reasons	Number of informal reasons	Total number of reasons
Performance related	30	5 (6%)	36 (43%)	41 (49%)
Failure related	11	3 (4%)	9 (11%)	12 (14%)
Improper use	2	3 (4%)	1 (1%)	4 (5%)
Data related	9	2 (2%)	8 (10%)	10 (12%)
Configuration related	3	2 (2%)	1 (1%)	3 (4%)
Time related	2	2 (2%)	0 (0%)	2 (2%)
Cost related	7	0 (0%)	7 (8%)	7 (8%)
Other	4	0 (0%)	4 (5%)	4 (5%)
		17 (20%)	66 (80%)	83 (100%)

5.2 | Analysis

The primary studies contain the answers to the research questions, but this information is hidden inside them. The analysis obtains valuable information in order to answer the research questions based on the data extracted from the primary studies. The data were extracted following the template defined in Subsection 3.3 and then analyzed by the methods described in Subsection 3.4. In the following subsections the primary studies are analyzed, classified and summarized in order to obtain the answer to each research question systematically.

5.2.1 | RQ1 Why is testing performed in MapReduce programs?

MapReduce programs are tested for several reasons. A model/taxonomy of these reasons were obtained by applying the meta-ethnography method to the primary studies, as described in Subsection 3.4. The reasons for testing obtained are:

- *Performance related*: issues derived from the performance goals, service level agreements, size of the data, performance under infrastructure failures and prediction/analysis/optimization of performance.
- *Failure related*: the specific faults of *MapReduce* programs and the number of programs that fail in production.
- *Improper use*: not all programs fit correctly in the *MapReduce* processing model.
- *Data related*: the challenges related to schema-less data and poor data quality.
- *Configuration related*: the misconfiguration of the infrastructure or program parameters may produce a failure.
- *Time related*: the programs may fail after a long time of resource usage.
- *Cost related*: testing can be carried out in order to reduce the cost of development, resource utilization and so on.
- *Other*: the reasons that do not fall in another category of the model/taxonomy of the reasons but do not constitute a new category of reasons.

For each of the above categories of reasons for testing, Table 5 indicates the number of primary studies that details these reasons. Note that a primary study can contain one or more reasons for testing. In Table 5, each reason for testing is also classified based on the degree of formality of the evidence in accordance with the following types: reasons with formal evidence and reasons with informal evidence.

Reason with formal evidence: the reason for testing is detailed in the primary studies empirically or with some rigorous evidence of this reason to test. For example, if one paper performs an extensive analysis of several programs and detects that testing is necessary because a lot of programs crash in production, this would be considered a reason with formal evidence.

Reason with informal evidence: the reason for testing is not clearly explained or not detailed in the primary studies due to the absence of rigorous analysis of the evidence for this reason to test. For example, if a paper indicates that the testing is necessary because the developers do not know how to configure the performance parameters of *MapReduce* programs, this would be considered a reason with informal evidence.

TABLE 6 Number of primary studies per type of performance-related reason.

Types of "performance related" reasons	Number of papers	Number of formal reasons	Number of informal reasons	Total number of reasons
Optimization/improvement of application performance	11	0 (0%)	11 (27%)	11 (27%)
Analysis of application performance	11	0 (0%)	11 (27%)	11 (27%)
Influence of infrastructure in application performance	4	3 (7%)	2 (5%)	5 (12%)
Influence of dataset in application performance	2	0 (0%)	2 (5%)	2 (5%)
Fulfill SLA or performance goals	10	2 (5%)	8 (20%)	10 (24%)
Other	2	0 (0%)	2 (5%)	2 (5%)
		5 (12%)	36 (88%)	41 (100%)

The most frequent type of reason for testing is "performance related", being described in 30 primary studies and representing 49% of the total reasons given in all primary studies, followed by "failure related" with 14%, "data related" with 12%, and "cost related" with 8% of the total number of reasons. Considering the formal evidence of the testing reasons, "performance related" is also the main reason in the primary studies with 6% of the total reasons (5 of formal evidence out of a total of 17 of formal evidence), followed by "failure related" and "improper use" with 4% of total reasons (3 of formal evidence out of a total of 17 of formal evidence).

In the model/taxonomy obtained through the analysis of the primary studies, the 41 "performance related" reasons for testing were sub-divided in the following sub-categories of reasons:

- *Optimization/improvement of application performance*: testing is aimed at the improvement of program performance.
- *Analysis of application performance*: understanding of performance to detect bottlenecks, among other issues.
- *Influence of the infrastructure on application performance*: whereas *MapReduce* applications can be designed without considering the infrastructure, program performance is influenced by the production infrastructure.
- *Influence of dataset on application performance*: in the same way that the infrastructure impacts performance, the dataset used in production also make an influence.
- *Fulfill SLA or performance goals*: the reason for testing the program is to fulfill service level agreements or other performance goals such as deadlines.
- *Other*: the reasons that do not fall in another sub-category of the model/taxonomy of the performance reasons but do not constitute a new sub-category of reasons.

For each one of the above sub-categories of testing that are related to performance, Table 6 indicates the number of the primary studies and their reasons for testing.

From the 41 "performance related" reasons for testing, the most frequent are focused on the analysis (27% of "performance related" reasons) and optimization of performance (27% of "performance related" reasons), followed by the fulfillment of performance goals (24% of "performance related" reasons). The remainder of reasons for testing related to performance analyze the influence of the infrastructure (12% of "performance related" reasons) and the dataset (5% of "performance related" reasons), followed by other issues (5% of "performance related" reasons).

Of all the reasons for testing the programs, only 20% are based on formal evidence, and the remaining 80% are based on informal evidence. Regardless of the formality of evidence, the reasons for testing *MapReduce* programs most frequently described in the primary studies include "performance related", especially for the analysis, optimization and fulfillment of performance goals. The least commonly cited reasons for testing are "time related", "configuration related", "improper use" and "other".

5.2.2 | RQ2 What testing is performed in MapReduce programs?

The planning of Subsection 3.4 proposes a meta-ethnography[67] to answer this research question. The data extracted from each primary study is categorized against two facets in order to answer RQ2:

a. Quality (sub)characteristics for each study according to the ISO/IEC 25010:2011[44] represented in Table 7.

TABLE 7 Number of primary studies per ISO/IEC 25010:2011 Quality (sub)characteristic.

			Number of studies		
ISO 25010:2011 System/software product quality	Functional suitability	Functional Completeness	2 (4%)	14 (26%)	46 (85%)
		Functional correctness	14 (26%)		
		Functional appropriateness	2 (4%)		
	Performance efficiency	Time-behaviour	32 (59%)	35 (65%)	
		Resource utilisation	14 (26%)		
		Capacity	1 (2%)		
	Reliability	Maturity	1 (2%)	3 (6%)	
		Availability	1 (2%)		
		Fault tolerance	1 (2%)		
		Recoverability	3 (6%)		
Other studies	Characterization studies			4 (7%)	8 (15%)
	Overview of testing			4 (7%)	

TABLE 8 Number of primary studies per ISO/IEC/IEEE 29119-4:2015 Quality-Related Type of Testing.

		Number of studies	
ISO/IEC/IEEE 29119-4:2015 Types of testing	Performance-Related Testing	32 (59%)	44 (81%)
	Functional Testing	12 (22%)	
	Backup/Recovery Testing	2 (4%)	
Other studies	Characterization studies	5 (9%)	10 (19%)
	Overview of testing	5 (9%)	

b. Quality-Related Types of Testing proposed in each study based on ISO/IEC/IEEE 29119-4:2015[61] and summarized in Table 8.

The majority of efforts are focused on "performance efficiency", accounting for 65% of the studies, then on "functional suitability" with 26% of the studies, and finally on "reliability" with 6% of the studies. Regarding the type of testing, 59% apply "performance-related testing", 22% employ "functional testing" and 4% use "backup/recovery testing".

The results obtained through the combination of both facets are more or less those expected: "performance-related testing" is related to "performance efficiency" characteristics, the "functional testing" to "functional suitability", and "backup/recovery testing" to "reliability".

5.2.3 | RQ3 How is testing performed in MapReduce programs?

This research question is answered through the meta-ethnography[67] proposed in Subsection 3.4. In order to answer RQ3, the primary studies were analyzed considering three facets:

a. Testing methods/techniques are summarized in Table 9 according to the test activities proposed in Annex A of ISO/IEC/IEEE 29119-1:2013[9].

b. Dependency between the primary studies and the *MapReduce* processing model is depicted in Table 10. This table describes whether the testing methods, techniques or studies are specific to *MapReduce* or could be applied to other paradigms/technologies.

c. Tools created or used in the primary studies to perform software testing are characterized in Table 11.

The majority of the papers (74%) focus on testing only the *MapReduce*-specific parts of the program. These programs have challenges related to performance issues and the correct operation of the program under parallel architecture. These issues among others are tested mainly by "evaluation", according to 48% of the studies and "simulation" in 17% of the studies. Other testing activities are used to a lesser degree, such as "structure based" in 7% of the studies or static analysis in 6% of the studies.

TABLE 9 Number of primary studies per ISO/IEC/IEEE 29119-1:2013 Test activity of Annex A.

					Number of studies		
ISO/IEC/ IEEE 29119-1:2013 Annex A: Test activities	V&V analysis		Evaluation	26 (48%)		29 (54%)	43 (80%)
			Simulation	9 (17%)			
	Testing	Dynamic testing	Structure based	4 (7%)	7 (13%)	12 (22%)	
			Specification based	1 (2%)			
			Experienced based	1 (2%)			
			Other	1 (2%)			
		Static testing	Static analysis	3 (6%)	5 (9%)		
			Other	3 (6%)			
	Formal methods		Model checking	1 (2%)		2 (4%)	
			Proof of correctness	1 (2%)			
Other studies	Characterization studies				6 (11%)	11 (20%)	
	Overview of testing				5 (9%)		

TABLE 10 Number of primary studies per test area covered.

		Number of studies	
Specific of MapReduce		40 (74%)	50 (93%)
Not specific of MapReduce		10 (19%)	
Other studies	Characterization studies		4 (7%)

TABLE 11 Number of primary studies per tool created in their research.

				Number of studies	
Tool created or used	Based on other	Tool available	3 (6%)	11 (20%)	19 (35%)
		Tool not available	8 (15%)		
	Not based on other tools	Tool available	2 (4%)	8 (15%)	
		Tool not available	6 (11%)		
No tool created or used					35 (65%)

More than half of the studies (65%) do not create or use testing tools in their research. There are in total 19 tools, where 11 are based on other software testing related tools, and only 5 are freely available on the Internet with an open source license.

5.2.4 | RQ4 By whom, where and when is testing performed in MapReduce programs?

The planning of the *mapping study* described in Section 3.4 proposes a meta-ethnography[67] to answer the research question through three facets:

a. The different roles that participate in the testing efforts of the *MapReduce* programs, described in Table 12.

b. Test levels summarized in Table 13 that contains a characterization of ISTQB test levels[63] adapted to the *MapReduce* processing model according to Figure 4.

c. The development cycle phase according to the Software Implementation lower level Processes and System Context Technical Processes of ISO/IEC 12207[62] described in Table 14.

TABLE 12 Number of primary studies per role.

		Number of studies	
Roles	**Tester**	45 (83%)	49 (91%)
	Developer	5 (9%)	
Other studies	**Characterization studies**		5 (9%)

TABLE 13 Number of primary studies per ISTQB Test level.

			Number of studies		
Levels of testing in ISTQB	**Unit testing**	**Unit testing Map**	16 (30%)	19 (35%)	44 (81%)
		Unit testing Reduce	19 (35%)		
	Integration MapReduce testing		35 (65%)		
	Integration testing		4 (7%)		
	System testing		2 (4%)		
	Acceptance testing		0 (0%)		
Other studies	**Characterization studies**			5 (9%)	10 (19%)
	Overview of testing			5 (9%)	

TABLE 14 Number of primary studies per ISO/IEC 12207:2008 Software Implementation lower level Process and System Context Technical process.

		Number of studies	
ISO/IEC 12207:2008 Software Implementation lower level Processes	**Software Construction Process**	3 (6%)	48 (89%)
	Software Qualification Testing Process	47 (87%)	
ISO/IEC 12207:2008 System Context Technical processes	**Implementation Process**	3 (6%)	48 (89%)
	System Qualification Testing Process	47 (87%)	
	Software Operation Process	1 (2%)	
Other studies	**Characterization studies**	5 (9%)	6 (11%)
	Overview of testing	1 (2%)	

As expected, the main player for testing the *MapReduce* programs is the tester, as per 83% of the studies, and then the developer according to 9% of the studies. Almost all primary studies (87%) describe testing efforts in the "Software/System Qualification Testing Process" compared with 6% which focus on "Software Construction or the Implementation Process". In these processes, the studies cover in more detail the specific *MapReduce* parts of the program (*Map* and *Reduce* functions) instead of the other parts. The majority of the research efforts in 65% of the studies focus on the integration testing between *Map* and *Reduce* functions, and then 35% of the studies cover unit testing at the *Map* or *Reduce* functions. To a lesser extent, the testing efforts are oriented towards the parts of the program that could not contain *MapReduce* functions: 7% of the studies consider integration testing between the *MapReduce* functions with other parts of the program, and 4% of the studies relate to testing the system. All testing levels are covered by the primary studies except for acceptance testing.

From these results, it appears that the fulfillment of the contract or user requirements tested in the acceptance testing level is not greatly affected by the existence of *MapReduce* functions in the system. Despite the fact that *Big Data* programs can contain a composite of several technologies/programs, testing research efforts focus on testing the *MapReduce* functions in isolation from the rest of the system. Few studies consider that a *Big Data* program can contain *MapReduce* functions together with other technologies. Regardless of the test level, the testing described in the primary studies is mainly performed in the Software/System Qualification Testing Process.

5.3 | Summary

The research questions of Subsection 3.1 were answered through the primary studies, data extraction and data analysis. A summary is presented below:

RQ1. Why is testing performed in *MapReduce* programs? There are at least seven reasons for testing the *MapReduce* programs. The most frequent reasons are based on performance issues (to analyze, optimize and fulfill performance goals), the existence of several or specific failures, the type and quality of the data processed by these programs, and testing to predict the resources required and efficiently select the resources to be used. To a lesser degree, the other reasons for testing are the improper use of the processing model or technology, program misconfiguration or failures after a long period of executions.

RQ2. What testing is performed in *MapReduce* programs? The majority of the research efforts in testing the *MapReduce* programs focus on the analysis of performance, and to a lesser extent the functional aspects of *MapReduce* programs.

RQ3. How is testing performed in *MapReduce* programs? Mainly by evaluation and simulation. In both cases testing is focused specifically on the *MapReduce* functions and does not consider other parts of the program. Several tools are used to perform testing, but few are available on the Internet.

RQ4. By whom, where and when is testing performed in *MapReduce* programs? Testing is mainly performed by the tester in the Software/System Qualification Testing Process and major efforts focus on the *MapReduce* program (unit and integration testing between *Map* and *Reduce* functions).

The analysis of several features about the primary studies reveals, in addition to the answers to the research questions, other findings which are analyzed below.

The relation between the reasons for testing the programs and the type of testing employed in each study is displayed in Figure 6. According to Table 8, 59% of the studies focus on performance testing (RQ2), which is very important because *MapReduce* applications analyze large quantities of data. From RQ1 the reasons for testing the programs are obtained and 58% of these reasons are related to performance (48 reasons of a total of 85 according to the left side of Figure 6). The reasons for performance testing and the number of studies that test performance are aligned. However, according to Table 8, the studies related to functionality only represent 22% of all studies even though 42% of the reasons for testing are related to functionality (35 reasons of a total of 85 according to the left side of Figure 6). There are more reasons for testing functionality than there are studies about functionality, which can indicate a challenge in the functionality testing to cover these reasons and improve the quality evaluation of the *MapReduce* applications.

The main test activities in RQ3 are evaluation (seen in 48% of the studies) and simulation (seen in 17%). These two activities are the most frequent because the majority of studies are focused on performance testing (59% according to RQ2). Figure 7 characterizes the test activities (RQ3) and test levels (RQ4) regarding different types of testing (RQ2). The test levels in each type of testing are more or less similar to the answer to RQ4: the principal efforts are at the integration testing level of *Map* and *Reduce* functions and to a lesser degree at unit level. However, the test activities are different depending on the type of testing: performance testing employs evaluation and simulation to predict the time of execution and resources required, but functionality testing performs a variety of different test activities considering specific characteristics of the *MapReduce* processing model (static testing, structure based, formal methods, experience based and specification based).

The majority of the studies are published in conferences (76%) and there are a few studies published in a high-impact journal (13%). Despite the fact that the number of research lines of testing in *MapReduce* is growing, the validation of these approaches is still simply through experience or case studies focusing on only a few programs, which are sometimes created by the researcher. According to Table 4, 11% of studies are not validated, 41% are validated with examples and 22% employ programs created by the researcher to validate their own work. The research contribution of testing papers can be improved using controlled experiments with a standard benchmark, especially when considering the performance prediction techniques that in general are not validated against other techniques. As noted in Section 5.1, performance prediction techniques employ a lot of different characteristics/parameters of the input dataset, program functionality, programming cluster and file system. In consequence, there is no clear intuition of which parameters have more influence in performance. The researchers can improve the testing techniques with both an accurate analysis of the parameters that have more impact in performance and rigorous experimentation using other testing techniques as a baseline.

This work analyzed 54 studies in detail, obtained through a wide search that resulted in 1377 *Big Data* studies by applying a filter (C4), in which only the studies that address software testing of *MapReduce* applications pass. Of these 1377 *Big Data* studies, 1043 are about *Big Data Engineering* and 334 about *Big Data Analytics*. Table 15 classifies the *Big Data Engineering* studies based on the research topic in order to characterize the research efforts. This classification reflects the research efforts to boost the *Big Data Engineering* field because 44.1% of the studies improve the technology, 18.31% analyze the technology through studies and surveys, 9.01% create new technologies to manage and analyze data, and 6.62%

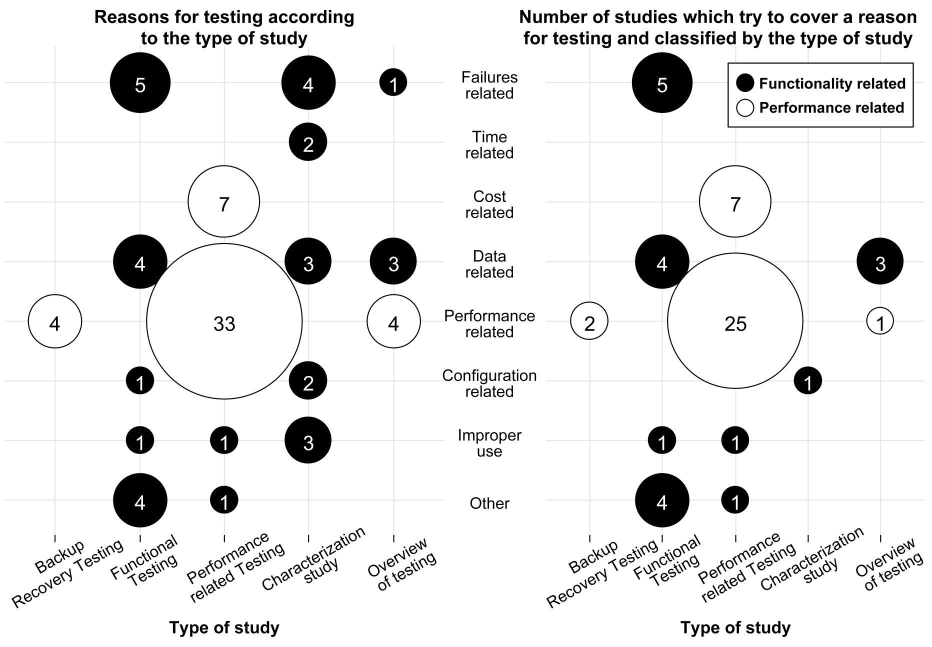

FIGURE 6 Number of reasons for testing and primary studies per type of study.

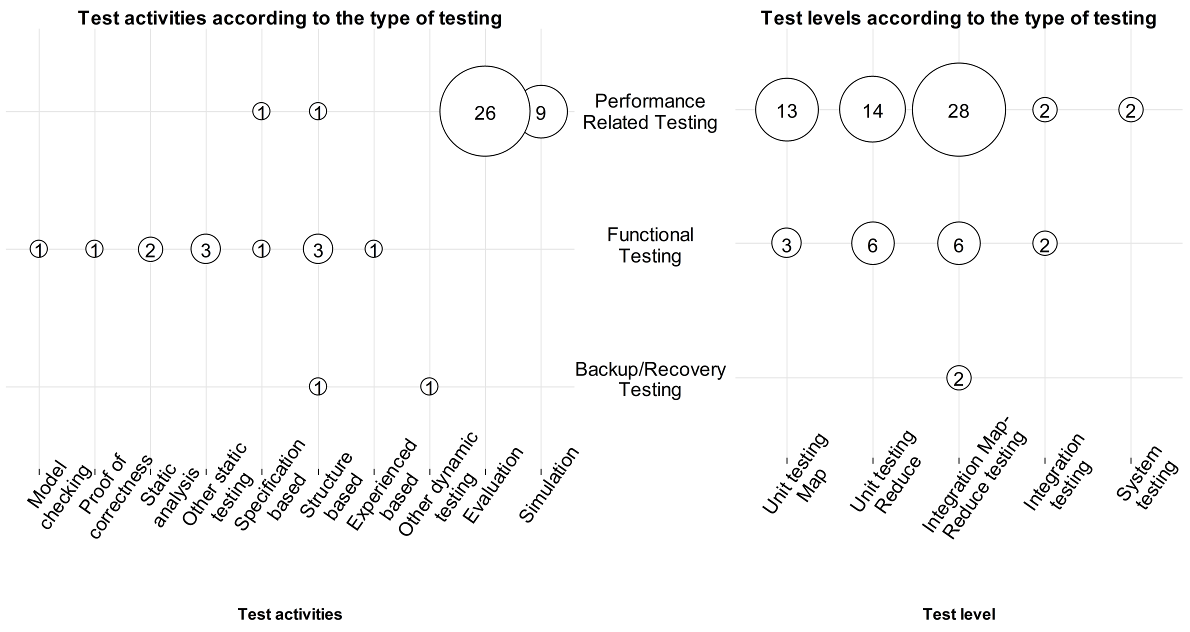

FIGURE 7 Number of primary studies per Test activity and test level according to the type of testing.

are focused on the state-of-the-art and challenges. Despite the challenges of testing in the *Big Data* area[81,83], there are few research lines which focus on testing *Big Data* programs in general and *MapReduce* programs in particular.

The most relevant findings of this *mapping study* are enumerated in Table 16 and discussed in the next Section.

TABLE 15 Number of Big Data Engineering studies in the last filter of the mapping study.

		Number of studies		
Improvements of technology	Performance	121 (11.60%)	460 (44.10%)	1043 (100%)
	Security	81 (7.77%)		
	Data acquisition, storage and extraction	45 (4.31%)		
	Fault tolerance and availability	42 (4.03%)		
	Energy	42 (4.03%)		
	Improvements outside of Hadoop	35 (3.36%)		
	Scheduling	34 (3.26%)		
	MapReduce model	14 (1.34%)		
	Different frameworks	10 (0.96%)		
	Other improvements	36 (3.45%)		
Studies/Surveys	General quality in Big Data	171 (16.40%)	191 (18.31%)	
	Other	20 (1.92%)		
Software testing	For MapReduce programs	64 (6.14%)	103 (9.88%)	
	For non-MapReduce programs	39 (3.74%)		
Big Data in the cloud		101 (9.68%)		
New frameworks	New Hadoop frameworks	85 (8.15%)	94 (9.01%)	
	Other new Frameworks	9 (0.86%)		
State-of-the-art and challenges		69 (6.62%)		
Debug		6 (0.58%)		
Other		19 (1.82%)		
Not applicable (Big Data Analytics)				334

6 | DISCUSSION OF RESULTS

This Section discusses the main findings obtained in the current *systematic mapping study* and enumerated in Table 16. Despite the recent interest in *Big Data* through several studies published to improve/study the underlying technology, few of them are focused on software testing [Finding 1]. Researchers not only have opportunities in software testing for *Big Data* programs, but also for *MapReduce* applications. Although *MapReduce* is one of the processing models most frequently used in *Big Data*, the programs are usually formed by the integration of a stack/pipe of different technologies. In contrast, the majority of research about software testing is only focused on the *Map/Reduce* code, without considering the code of other technologies of the *Big Data* stack [Finding 2]. The testing techniques are usually similar to those employed in general purpose software, and so the researchers should adapt other general testing research to *MapReduce* considering the specific characteristics of the processing model.

The majority of studies about software testing in *MapReduce* applications are focused on performance using verification and validation test activities such as simulation or evaluation [Finding 3]. These tests are usually done to predict/forecast/analyze performance through models that use several parameters characterizing both the program functionality, the programming cluster and the file system [Finding 4]. Since each of these models employs different heterogeneous parameters, then it is difficult to understand which are the ones that really affect performance, as well as the real weight/influence that these parameters have in performance. Performance testing research could be improved by means of both an analysis of the parameters used by other researchers, and rigorous experimentation using other models as a baseline.

According to the research lines, the main reason for testing *MapReduce* applications is performance [Finding 5]. Also the majority of the testing techniques for *MapReduce* applications are related to performance, as expected. The research lines also suggest that functionality is another of the relevant reasons to test *MapReduce* applications, but the actual number of functional testing techniques is low [Finding 6]. Researchers may have opportunities to devise new functional testing techniques considering the specific characteristics of *MapReduce* programs such as distributed execution and scalability, among others. The functional testing techniques of *MapReduce* programs involve different test activities, which include structure-based, static analysis and formal methods [Finding 7]. The researchers should adapt the dynamic/static/formal testing techniques of general-purpose software (data-flow, combinatorial or mutation testing, among others) to *MapReduce* considering the specific characteristics of the programming model.

TABLE 16 Findings of the Mapping study.

Id	Finding
1	Despite several studies that are aimed at both improving and studying the state-of-art of *Big Data* technology, there are in comparison few research lines focused on software testing of the *Big Data* programs [Subsection 5.3]
2	The majority of testing research in *MapReduce* applications is focused on either *Map* or *Reduce* or the integration of both, and cannot be applied to other processing models because they are specifically designed for *MapReduce* [Subsections 5.2.3 and 5.2.4]
3	The majority of research is about performance testing, and, to a lesser degree, functional testing [Subsection 5.2.2]. This research is about verification and validation analysis, and, to a lesser degree, about dynamic testing [Subsection 5.2.3]
4	The prediction/analysis models employed in performance testing use different numbers of heterogeneous parameters based not only on the *MapReduce* program functionality, but also on the cluster infrastructure, file system and data[Subsection 5.1.1]
5	The most frequent reasons for testing the *MapReduce* programs are based on performance issues (analyze, optimize and fulfill performance goals), existence of several and specific failures, the type and quality of the data processed by these programs, and testing to predict and efficiently select the resources [Subsection 5.2.1]
6	There are several rigorous reasons for testing the functionality of *MapReduce* applications, such as the percentage of programs that fail in production or the improper use of both functional semantics and data, but there are not many research efforts focused on this line of interest [Subsection 5.3]
7	Whereas performance testing is done by simulation and evaluation, functional testing employs different test activities, such as static testing and structure-based testing [Subsection 5.3]
8	As expected, testing research is focused on the software qualification process to help the tester [Subsection 5.2.4]
9	The majority of research neither creates nor uses a tool for testing *MapReduce* programs [Subsection 5.2.3]
10	Software testing research focused on *MapReduce* applications is usually published in conferences, and furthermore it is usually published without a strong validation, using only some case studies instead of rigorous empirical experiments [Subsection 5.3]

Regardless of performance or functionality, the majority of testing is aimed at helping the tester in the software qualification process [Finding 8] without tools [Finding 9]. The contributions of researchers could not only help the testers, but could also help the final users providing automatic tools to support the design of test cases, and monitoring tools to analyze failures produced at runtime in production.

The majority of studies about software testing in *MapReduce* applications are published in conferences and evaluated with some case studies [Finding 10]. Researchers could improve both visibility and quality by means of rigorous experiments based on a benchmark of *MapReduce* programs that can expose functional/performance failures, such as SWIM[133], GridMix[134], SparkBench[135], BigBench[136] or TPCx-BB[137].

7 | LIMITATIONS OF THE MAPPING STUDY

Despite the fact that both the planning and the execution (conducting) of this *mapping study* aimed to avoid bias, some limitations and researcher decision biases could exist[138].

- The results are limited by the academic context because the data sources are focused on the research field. Bias could be generated if the research papers do not represent the reality and motivations of software testing in *MapReduce* programs.
- Following some practices from social science[72] and software engineering[73], the selection of the primary studies was performed by one author for those papers that are clearly non-relevant. In contrast, two authors selected the primary studies independently from 1043 studies that had more chances to be relevant. Both authors agreed in 96% of studies and obtained a substantial/moderate agreement with 0.69 as a Kappa coefficient and [0.60-0.78] as 95% confidence interval.
- Despite the authors not finding quality problems in the primary studies, the quality of these studies was not formally evaluated. The same issue occurs in the majority of the *systematic mapping studies*[73] because quality assessment is usually not required[43].
- The data extraction was performed by one author and checked by another author. This practice is used in other *systematic reviews*[79] and some researchers consider it more practical than when data is extracted by several authors[71].
- Further bias occurs if some research questions cannot be properly answered through the checklist of the data extraction. In order to minimize this bias, the majority of these checklists are based on the international standards.

- Another less important potential bias could occur during the search process if some primary studies are not found with the search terms or expert opinions. In order to minimize bias, a thorough search is performed in several databases, journals, conferences and experts.

In order to avoid bias in the results, all steps are reviewed and some countermeasures are taken in research questions, the search process, data extraction and data analysis:

1. Research question: created by the Kipling method[41] instead of ad-hoc.
2. Search process:

   Search terms: the use of a large number of terms could improve the search process by obtaining more potential primary studies. Some authors encourage the use of several short queries instead of long queries[139]. This *mapping study* searches for a combination of 92 *MapReduce* related terms and 102 testing terms obtained from ISO/IEC 25010:2011 Quality (sub)characteristics[44] with synonyms obtained through Kitchenham et al.[22] points of view.

   Data sources: this study searched 5 electronic databases recommended by Kitchenham et al.[55] and 2311 proceedings/volumes related to software testing of *MapReduce* programs. The other data source taken into account is the opinionss of experts in the field in order to minimize the bias by adding primary studies that could not be found by the previous search.

   Study selection: this *mapping study* excludes non-relevant studies based on 4 filters. These filters were reviewed in order to obtain the relevant studies.
3. Data extraction: the majority of the data extracted are based on checklists, in some cases obtained from international standards and in others created or adapted to the *MapReduce* processing model.
4. Data analysis: the methods used in this work are employed in software engineering[65].

8 | CONCLUSIONS

The number of studies on software testing of *MapReduce* programs has increased during recent years. A characterization was carried out based on 54 research studies obtained from more than 70000 potential papers. The testing tasks in these programs are normally performed by the tester in the Software/System Qualification Testing Process due to a combination of the following 7 reasons: performance issues, potential failures, issues related to the data such as for example data quality, the reduction of the cost in resources, misconfigurations, improper use of the technology, time problems or other issues. These reasons for testing assume that both functional and performance testing are necessary, but the studies employ different approaches: functional testing considers different aspects of the program (such as specification and structure) while performance testing is more focused on simulation and evaluation. The current body of research focuses on performance testing, while there is a challenge in functional testing due to the importance of this research line and the lack of research efforts.

The main goal of performance testing in *MapReduce* studies is to predict the execution time and the resources required to efficiently execute the programs and satisfy the agreements. From the functionality point of view, the goal of the studies is to detect faults considering the specific characteristics of the *MapReduce* processing model. Regardless of the type of testing, the majority of efforts are specific for the *MapReduce* technology at unit and integration level of the *Map* and *Reduce* functions. This situation may indicate a challenge in the integration of *MapReduce* programs with other programs, especially other *Big Data* stack technologies.

The research into software testing in *MapReduce* programs is mainly validated with example programs. There is scope to evolve with better validations and thus improve the research impact. Despite the lack of maturity, several studies create tools to support testing, but few are available on the Internet for users or other researchers. In *Big Data* there are few research studies related to software testing in comparison to the number of research efforts focused on improving the technology, which indicates new opportunities in software testing of *Big Data* in general, and *MapReduce* in particular.

ACKNOWLEDGMENTS

This work was supported in part by projects TIN2016-76956-C3-1-R and TIN2013-46928-C3-1-R funded by the Spanish Ministry of Economy and Competitiveness, and GRUPIN14-007, funded by the Principality of Asturias (Spain) and ERDF funds.

How to cite this article: J. Morán, C. de la Riva, and J. Tuya, Testing MapReduce Programs: A Systematic Mapping Study, *J Softw Evol Proc*, *2018;VV:P–P* e2120. https://doi.org/10.1002/smr.2120.

APPENDIX

A HIGH-IMPACT JOURNALS AND CONFERENCES USED FOR THE MAPPING STUDY

TABLE A1 High impact journals for the mapping study

JCR journals	2015				2016			
	Rank	Impact factor	Number of citations	Number of papers	Rank	Impact factor	Number of citations	Number of papers
ACM Computing Surveys	Q1	5.24	4150	88	Q1	6.75	6629	76
ACM SIGPLAN Notices	Q1	0.49	3657	389	Q1	0.34	2541	378
ACM Transactions on Database Systems (ACM TODS)	Q2	0.63	969	19	Q2	1.52	1504	26
ACM Transactions on Information Systems (ACM TOIS)	Q2	0.98	1220	27	Q2	2.31	1790	33
ACM Transactions on Software Engineering and Methodology (ACM TOSEM)	Q3	1.51	700	21	Q2	2.52	1104	16
Computer Science and Information Systems (ComSIS)	Q4	0.62	265	64	Q4	0.84	392	47
Distributed and Parallel Databases	Q4	0.80	293	21	Q4	1.18	349	19
Distributed Computing	Q3	1.26	498	26	Q2	1.67	954	24
Empirical Software Engineering (ESE)	Q2	1.39	828	52	Q2	3.28	1453	68
The International Arab Journal of Information Technology (IAJIT)	Q4	0.52	292	78	Q3	0.72	502	93
IEEE Software	Q2	0.82	1638	55	Q1	2.19	2547	69
IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)	Q1	2.48	6465	245	Q1	3.44	9370	239
IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS)	Q1	2.66	5080	282	Q1	4.18	8313	271
IEEE Transactions on Software Engineering (IEEE TSE)	Q1	1.51	4221	62	Q1	3.27	6712	59
International Journal of Data Warehousing and Mining (IJDWM)	Q4	0.63	146	17	Q4	0.73	219	15
International Journal of Information Management (IJIM)	Q1	2.69	1937	73	Q1	3.87	3087	115
International Journal of Information Processing and Management (IJIPM)	Q1	1.40	2296	63	Q1	2.39	3067	72
International Journal of Information Technology and Decision Making (IJITDM)	Q3	1.18	627	45	Q3	1.66	742	56
International Journal of Information Technology and Management (IJITM)	Q4	0.60	226	23	Q4	1.07	281	29
International Journal of Software Engineering and Knowledge Engineering (IJSEKE)	Q4	0.24	216	55	Q4	0.30	345	52
Information and Software Technology (IST)	Q1	1.57	2145	153	Q1	2.69	3448	122
Journal of Database Management (JDM)	Q4	0.12	131	7	Q4	0.27	182	9
Journal of Information Technology (JIT)	Q1	4.78	1695	24	Q1	6.95	2515	19

Journal of Management Information Systems (JMIS)	Q1	3.03	3818	41	Q1	2.36	4456	30
Journal of Software: Evolution and Process	Q4	0.73	140	46	Q4	1.03	319	50
Journal of Parallel and Distributed Computing (JPDC)	Q1	1.32	1983	94	Q1	1.93	2740	84
The Journal of Strategic Information Systems (JSIS)	Q2	2.60	1159	17	Q2	3.49	1580	15
Journal of Systems and Software (JSS)	Q1	1.42	3243	181	Q1	2.44	5161	229
Knowledge and Information Systems (KAIS)	Q2	1.70	1559	110	Q2	2.00	2146	117
Software Quality Journal (SQJ)	Q4	0.79	280	24	Q3	1.86	486	33
Software Testing, Verification & Reliability (STVR)	Q3	1.08	363	25	Q3	1.59	612	20

TABLE A2 CORE conferences for the mapping study

CORE conferences	CORE 2014	CORE 2017
ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)	A*	A*
Computer Aided Verification (CAV)	A*	A*
IEEE International Conference on Data Mining (IEEE ICDM)	A*	A*
International Conference on Data Engineering (IEEE ICDE)	A*	A*
Special Interest Group on Management of Data Conference (SIGMOD)	A*	A*
Very Large Data Bases Conference (VLDB)	A*	A*
Automated Software Engineering (ASE)	A	A
Biennial Conference on Innovative Data Systems Research (CIDR)	A	A
Empirical Software Engineering and Measurement (ESEM)	A	A
European Conference on Parallel Processing (EURO-PAR)	A	A
European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)	A	A
International Conference on Database Theory (ICDT)	A	A
International Conference on Distributed Computing Systems (ICDCS)	A	A
International Conference on Extending Database Technology (EDBT)	A	A
International Conference on Information and Knowledge Management (CIKM)	A	A
International Conference on Software Engineering (ICSE)	A	A
International Conference on Statistical and Scientific Database Management (SSDBM)	A	A
International Symposium on Cluster Computing and the Grid (CCGRID)	A	A
International Symposium on Intelligent Data Analysis (IDA)	A	A
International Symposium on Software Testing and Analysis (ISSTA)	A	A
Joint International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)	A	A
Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)	A	A
Parallel Computing Technologies International Conferences Series (PaCT)	A	A
SIAM International Conference on Data Mining (SDM)	A	A
Symposium on Large Spatial Databases (SSTD)	A	A
ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)	A	B
Advances in Databases and Information Systems (ADBIS)	B	B
Australasian Data Mining Conference (AusDM)	B	B
Australasian Database Conference (ADC)	B	B
Databases and Programming Language (DBPL)	B	B
European Software Engineering Conference (ESEC)	A	B
IEEE International Conference on Cloud Computing (IEEE CLOUD)	B	B
IEEE International Enterprise Distributed Object Computing Conference (IEEE EDOC)	B	B

International Baltic Conference on Databases and Information Systems (DB&IS)	B	B
International Conference on Data Warehousing and Knowledge Discovery (DaWaK)	B	B
International Conference on Database and Expert Systems Applications (DEXA)	B	B
International Conference on Database Systems for Advanced Applications (DASFAA)	B	B
International Conference on Management of Data (COMAD)	B	B
International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)	B	B
International Conference on Quality Software (QSIC)	B	B
International Conference on Software and Data Technologies (ICSOFT)	B	B
International Conference on Tests and Proof (TAP)	B	B
International Database Engineering and Applications Symposium (IDEAS)	B	B
International Workshop on Data Warehousing and OLAP (DOLAP)	B	B
Software Engineering and Knowledge Engineering (SEKE)	B	B
Symposium on Applied Computing (SAC)	B	B
Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)	C	C
Evolution and Change in Data Management (ECDM)	C	C
IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom)	C	C
International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)	C	C
International Conference on Software Testing, Verification and Validation (ICST)	C	C
International Workshop on Formal Approaches to Testing of Software (FATES)	C	C
Symposium on Principles of Database Systems (PODS)	C	C

B OTHER JOURNALS AND CONFERENCES USED FOR THE MAPPING STUDY

TABLE B3 Other journals for the mapping study

ACM DATA BASE	International Journal of Intelligent Information and Database Systems (IJIIDS)
ACM SIGSOFT Software Engineering Notes (ACM SIGSOFT)	International Journal of Information Quality (IJIQ)
ACM Transactions on Management Information Systems (ACM TMIS)	International Journal of Information Systems and Change Management (IJISCM)
Big Data Research	International Journal of Information Technologies and Systems Approach (IJITSA)
Computing and Information Technology (CIT)	International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)
European Journal of Information Systems (EJIS)	Journal of Cases on Information Technology (JCIT)
Foundations and Trends in Databases (FTDB)	Journal of Data and Information Quality (JDIQ)
IEEE Cloud Computing	Journal of Digital Information Management (JDIM)
IEEE Computer	Journal of Enterprise Information Management (JEIM)
IEEE Distributed Systems Online (IEEE DS)	Journal of Information and Data Management (JIDM)
IEEE Transactions on Big Data	Journal of Information & Knowledge Management (JIKM)
IEEE Transactions on Cloud Computing (IEEE TCC)	Journal of Information Processing (JIP)
International Journal of Big Data Intelligence (IJBD)	The Journal of Information Processing Systems (JIPS)
International Journal of Cloud Applications and Computing (IJCAC)	Journal of Information Technology Research (JITR)
International Journal of Cloud Computing (IJCC)	Journal of Systems and Information Technology (JSIT)
International Journal of Distributed Systems and Technologies (IJDST)	Transactions on Large-Scale Data- and Knowledge-Centered Systems (Transactions LDKS)
International Journal of Enterprise Information Systems (IJEIS)	Journal of Enterprise Information Management (JEIM)

TABLE B4 Other conferences for the mapping study

Advances in Model-Based Testing (A-MOST)	Industrial Conference on Data Mining (ICDM)
Alberto Mendelzon Workshop on Foundations of Data Management (AMW)	International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)
International Conference on Big Data Analytics (BDA)	Internet and Distributed Computing Systems (IDCS)
International Conference Beyond Databases, Architectures, and Structures (BDAS)	IEEE/ACM International Symposium on Big Data Computing (BDC)
International Conference on Big Data and Smart Computing (BigComp)	IEEE International Conference on Big Data (IEEE BigData)
International Congress on Big Data (BigData Congress)	IEEE Symposium on Large-Scale Data Analysis and Visualization (IEEE LDAV)
Workshop on Scalability in Model Driven Engineering (BigMDE)	International Conference on Algorithms for Big Data (ICABD)
British National Conference on Databases (BNCOD)	International Conference on Big Data and Cloud Computing (BdCloud)
International Conference on Cloud and Autonomic Computing Conference (CAC)	International Conference on Big Data Cloud and Applications (BDCA)
International Conference on Cloud and Green Computing (CGC)	International Conference on Big Data Computing and Communications (BigCom)
International Conference on Cloud Computing and Services Science (CLOSER)	International Conference on Big Data Computing Service and Applications (BigDataService)
Cloud Computing (CloudComp)	International Multiconference on Computer Science and Information Technology (IMCSIT)
Conference on Data and Application Security and Privacy (CODASPY)	International Workshop on Machine Learning, Optimization, and Big Data (MOD)
International Computer Software and Applications Conference (COMPSAC)	Symposium on Network Cloud Computing and Applications (NCCA)
International Conference on Cloud and Service Computing (CSC)	Conference on Next Generation Information Technologies and Systems (NGITS)
European Joint Conference on Theory and Practice of Software (ETAPS)	ACM Symposium on Cloud Computing (SoCC)
International Conference on Future Data and Security Engineering (FDSE)	SPIN Workshop on Model Checking of Software (SPIN)
Federated Conference on Computer Science and Information Systems (FEDCSIS)	Symposium on Computational Intelligence in Big Data (CIBD)
International Conference on Future Internet of Things and Cloud (FICLOUD)	Symposium on Information Management and Big Data (SIMBig)
USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)	Testing: Academic & Industrial Conference - Practice And Research Techniques (TAIC PART)
International Conference on Advanced Cloud and Big Data (CBD)	International Conference on Testing Communicating Systems (TestCom)
International Conference on Cloud Engineering (IC2E)	Workshop on Big Data Benchmarking (WBDB)
International Conference on Algorithms for Big Data (ICABD)	Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware (BPOE)
International Conference on Innovative Computing and Cloud Computing (ICCC)	Workshop on Mobile Big Data (Mobidata)
International Conference on Data Engineering and Management (ICDEM)	

C PRIMARY STUDIES

TABLE C5 Primary studies

Ref.	Year	Contribution	Number of citations *	Summary
[116]	2013	Conference	20	A study and characterization of MapReduce-like failures
[85]	2013	Conference	30	A prediction model of individual MapReduce jobs based on important properties
[93]	2013	Conference	25	A performance prediction based on network properties and configuration of the cluster
[111]	2013	Conference	22	A performance prediction based on a representation of the architecture with some information of the MapReduce program
[128]	2015	Conference	7	Generator of representative data to testing Big Data programs based on input space partitioning
[117]	2010	Conference	311	A study and characterization of more than 170000 MapReduce executions
[87]	2011	Conference	46	A simple performance prediction model that considers the program and the system
[92]	2013	Journal	47	A model that obtains several metrics about the MapReduce programs performance and resource utilization
[82]	2013	Briefing	19	Classification of testing in Big Data and the underlying challenges
[119]	2013	Conference	8	Classification of MapReduce faults based on empirical changes in the programs
[121]	2015	Conference	13	Checking of the commutativity problem in the Reduce functions
[94]	2013	Conference	30	A performance prediction model based on information about the MapReduce program and the cluster
[109]	2012	Doctoral dissertation	14	A simulator of MapReduce program that obtains a prediction of the performance
[90]	2014	Journal	50	A performance prediction model of MapReduce program using Mean Field Analysis and information of the program, system and data
[100]	2015	Conference	7	A performance prediction model of MapReduce program in the cloud considering the program and the data
[125]	2012	Conference	34	A failure injector in the architecture using the cloud manager in order to test the MapReduce programs
[83]	2013	Conference	7	Challenges of software testing in Big Data
[115]	2013	Conference	97	A study and characterization of three Hadoop clusters
[102]	2016	Journal	31	Prediction of the performance and optimization of resource utilization based on deadline requirements
[112]	2011	Conference	54	Monitoring of the MapReduce program that generates detailed reports of the execution
[114]	2013	Journal	54	A study and characterization of several bugs in Big Data programs
[101]	2015	Conference	10	A performance prediction model of the MapReduce programs considering the deployment in virtualized cloud and the characteristics of the program
[91]	2012	Conference	11	A performance prediction model of the MapReduce programs considering several samplings of the input data
[124]	2015	Journal	11	A testing framework to run the MapReduce programs under architectural failures in order to test
[95]	2013	Conference	7	A performance prediction model of the MapReduce programs considering the resource contention and the task failures
[120]	2014	Conference	7	Classification of several MapReduce faults with a series of challenges in order to reveal the faults
[122]	2011	Conference	28	Functional Testing of the Reduce function based on symbolic execution
[118]	2014	Conference	22	Characterization of the MapReduce programs based on empirical study
[96]	2014	Conference	5	A performance prediction model of the MapReduce programs considering the memory shared and disk I/O
[76]	2014	Journal	9	Prediction of the MapReduce performance based on empirical executions and an adjustment based on micro benchmarks

[113]	2014	Journal	19	Performance analysis model for MapReduce applications based on ISO 25010 that establishes a relationship between the performance and reliability measures
[88]	2013	Conference	1	Obtains the performance of the MapReduce programs based on Stochastic Petri Nets
[106]	2015	Conference	2	Performance prediction of HIVE-QL queries through the underlying MapReduce applications based on multiple lineal regression
[78]	2013	Journal	13	Performance prediction of PIG queries through the underlying MapReduce applications
[86]	2014	Conference	2	Performance prediction for a MapReduce program and optimization based on the type of application and potential bottlenecks
[97]	2014	Conference	6	Mathematical model for performance prediction of the RDMA-Enhanced MapReduce programs
[77]	2013	Conference	32	A performance prediction model of the MapReduce programs considering information of the program and the performance for several parts of the program
[103]	2015	Conference	3	Model that predicts the performance of MapReduce applications in hybrid clouds
[98]	2015	Conference	33	Simulation of the Spark applications in order to obtain performance information
[99]	2014	Conference	1	A performance prediction model of the MapReduce programs considering the heterogeneity of the cluster
[89]	2011	Conference	29	A performance prediction model of the MapReduce programs based on the mean time between failures
[84]	2014	Conference	8	Overview and challenges of performance testing in Big Data
[129]	2013	Conference	8	Data generation for dataflow programs based on symbolic execution
[108]	2014	Conference	8	Simulating the MapReduce program under configurable hardware in order to obtain a performance prediction
[110]	2014	Conference	2	Simulating the scheduler of the MapReduce program in order to test the best configuration
[80]	2015	Conference	6	Test factory model for Big Data development
[131]	2011	Conference	71	Static analysis of the MapReduce configuration in order to detect misconfigurations and avoid failures
[130]	2011	Conference	12	Automatic checking of the java types inside MapReduce programs in order to detect incompatible types
[127]	2012	Dissertation	7	Data generator for MapReduce programs based on bacteriological algorithm in order to test the program
[126]	2015	Conference	6	Testing technique for MapReduce programs based on data flow and the MapReduce specifics
[123]	2013	Conference	5	Checking the correctness of the dataflow programs based on the operators properties
[107]	2013	Journal	0	Performance prediction of the join queries in Pig
[81]	2013	Journal	11	Overview and challenges of testing in Big Data
[132]	2011	Conference	17	Formal verification of the MapReduce program based on a model of the program/specification and invariants

* Number of citations obtained from Google Scholar[140] in 2018

References

1. *ISO/IEC JTC 1 - Big Data, preliminary report.* 2014.

2. Dean Jeffrey, Ghemawat Sanjay. MapReduce: Simplified Data Processing on Large Clusters. *Proc. of the OSDI - Symp. on Operating Systems Design and Implementation.* 2004;:137–149.

3. Apache Spark: a fast and general engine for large-scale data processing. https://spark.apache.org. Accessed: June 2018.

4. Apache Flink: Scalable batch and stream data processing. https://flink.apache.org. Accessed: June 2018.

5. Apache Hadoop: open-source software for reliable, scalable, distributed computing. https://hadoop.apache.org/. Accessed: June 2018.

6. Institutions that are using Apache Hadoop for educational or production uses. https://wiki.apache.org/hadoop/PoweredBy. Accessed: June 2018. .

7. Schatz M. C.. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics.* 2009;25(11):1363–1369.

8. Kocakulak Hakan, Temizel Tugba Taskaya. A Hadoop solution for ballistic image analysis and recognition. In: :836–842IEEE; 2011.

9. ISO/IEC/IEEE 29119-1:2013 - ISO/IEC/IEEE International Standard for Software and systems engineering - Software testing - Part 1: Concepts and definitions. 2013:1–64.

10. Bertolino Antonia. Software Testing Research: Achievements , Challenges , Dreams. In: :85–103; 2007.

11. Bertolino Antonia, Inverardi Paola, Muccini Henry. *Software architecture-based analysis and testing: A look into achievements and future challenges.* 2013.

12. NewVantage Partners LLC . *Big Data Executive Survey 2016 An Update on the Adoption of Big Data in the Fortune 1000.* 2016.

13. Gartner. *How to Take a First Step to Advanced Analytics.* 2015.

14. Xerox. *Big Data in Western Europe Today.* 2015.

15. Cai Li, Zhu Yangyong. The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal.* 2015;14:2.

16. Marr Bernard. *Where Big Data Projects Fail. http://www.forbes.com/sites/bernardmarr/2015/03/17/where-big-data-projects-fail/.* Accessed: June 2018.

17. Pure Storage. *BIG DATA'S BIG FAILURE: The struggles businesses face in accessing the information they need.* 2015.

18. Capgemini Consulting. *Big Data survey.* 2014.

19. Marx Vivien. Biology: The big challenges of big data. *Nature.* 2013;498(7453):255–260.

20. Bachlechner Daniel, Leimbach Timo. Big data challenges: Impact, potential responses and research needs. In IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech): :257–264; 2016.

21. Kitchenham Barbara. Procedures for performing systematic reviews. *Keele, UK, Keele University.* 2004;33(TR/SE-0401):28.

22. Kitchenham Barbara, Charters S. Guidelines for performing Systematic Literature Reviews in Software Engineering. *Engineering.* 2007;2:1051.

23. Petersen Kai, Feldt Robert, Mujtaba Shahid, Mattsson Michael. Systematic mapping studies in software engineering. *EASE'08 Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering.* 2008;68–77.

24. Kitchenham Barbara, Brereton Pearl, Budgen David. The educational value of mapping studies of software engineering literature. *In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*: :589–598; 2010.

25. Zhang He, Babar Muhammad Ali, Ali Babar Muhammad. Systematic reviews in software engineering: An empirical investigation. *Information and Software Technology.* 2013;55(7):1341–1354.

26. Sharma Megha, Hasteer Nitasha, Tuli Anupriya, Bansal Abhay. Investigating the inclinations of research and practices in Hadoop: A systematic review. In Proceedings of the 5th International Conference on Confluence 2014: The Next Generation Information Technology Summit: 227–231; 2014.

27. Camargo Luiz Carlos, Vergilio Silvia Regina. Testing MapReduce Programs: A Mapping Study. In 32nd International Conference of the Chilean Computer Science Society (SCCC): 85–89; 2013.

28. Ferrera Pedro, De Prado Ivan, Palacios Eric, Fernandez-Marquez Jose Luis, Di MarzoÂăSerugendo Giovanna. Tuple MapReduce and Pangool: an associated implementation. *Knowledge and Information Systems.* 2014;41(2):531–557.

29. Lin Jimmy. Mapreduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail!. *Big Data.* 2013;1(1):28–37.

30. Babu Shivnath. Towards automatic optimization of MapReduce programs. *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing.* 2010; 137–142.

31. Vishwanath Kashi Venkatesh, Nagappan Nachiappan. Characterizing Cloud Computing Hardware Reliability. *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*. 2010; 193.

32. JUnit: a simple framework to write repeatable tests. http://junit.org. Accessed: June 2018.

33. Apache MRUnit: Java library that helps developers unit test Apache Hadoop map reduce job. http://mrunit.apache.org. Accessed: June 2018.

34. Minicluster: Apache hadoop cluster in memory for testing. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/CLIMiniCluster.html. Accessed: June 2018.

35. Herriot: Large-scale automated test framework. https://wiki.apache.org/hadoop/HowToUseSystemTestFramework. Accessed: June 2018.

36. Anarchy Ape: Fault injection tool for Hadoop cluster from Yahoo anarchyape. https://github.com/david78k/anarchyape. Accessed: June 2018.

37. Chaos Monkey: Fault injector. https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey. Accessed: Junes 2018.

38. Hadoop Injection Framework. https://hadoop.apache.org/. Accessed: June 2018.

39. Pan Zhongdang, Kosicki Gerald. Framing analysis: An approach to news discourse. *Political Communication*. 1993;10(1):55–75.

40. Hart Geoff. The Five W's: An Old Tool for the New Task of Audience Analysis. *Technical Communication*. 1996;43(2):139–145.

41. Kipling Rudyard. *Just so stories*. Macmillan and Co; 1902.

42. Jia Changjiang, Cai Yan, Yu Yuen Tak, Tse T.H.. 5W+1H pattern: A perspective of systematic mapping studies and a case study on cloud software testing. *Journal of Systems and Software*. 2016;116:206–219.

43. Kitchenham B. A., Budgen D. (David), Brereton Pearl. *Evidence-based software engineering and systematic reviews*. CRC Press; 2015.

44. ISI/IEC 25010:2011. *Software Process: Improvement and Practice*. 2011;2(Resolution 937):1–25.

45. *ISO/IEC 9126-1: Software Process: Improvement and Practice*. 2001.

46. DBLP - digital bibliography & library project. http://dblp.uni-trier.de. Accessed: June 2018.

47. Thomson reuters. http://thomsonreuters.com. Accessed: June 2018.

48. CORE - Computing research and education. http://www.core.edu.au. Accessed: June 2018.

49. IEEE Xplore Digital Library. http://ieeexplore.ieee.org. Accessed: June 2018.

50. ACM Digital Library. http://dl.acm.org. Accessed: June 2018.

51. Scopus. http://www.scopus.com. Accessed: June 2018.

52. EI compendex. https://www.engineeringvillage.com/. Accessed: June 2018.

53. ISI web of science. https://www.accesowok.fecyt.es. Accessed: June 2018.

54. Palacios Marcos, García-Fanjul José, Tuya Javier. Testing in Service Oriented Architectures with dynamic binding: A mapping study. *Information and Software Technology*. 2011;53(3):171–189.

55. Kitchenham Barbara, Brereton Pearl. A systematic review of systematic review process research in software engineering. *Information and Software Technology*. 2013;55(12):2049–2075.

56. Kitchenham Barbara, Brereton Pearl, Budgen David. Mapping study completeness and reliability - a case study. In IET Seminar Dig. :126–135; 2012.

57. Pan Xinghao, Tan Jiaqi, Kavulya Soila, Gandhi Rajeev, Narasimhan Priya. Ganesha: BlackBox Diagnosis of MapReduce Systems. *SIGMETRICS Perform. Eval. Rev.*. 2010;37(3):8–13.

58. Joshi Pallavi, Gunawi Haryadi S., Sen Koushik. PREFAIL: A Programmable Tool for Multiple-Failure Injection. *ACM SIGPLAN Notices*. 2011;46(10):171.

59. Tilley Scott, Parveen Tauhida. HadoopUnit: Test Execution in the Cloud. In Software Testing in the Cloud: 2012 (pp. 37–53).

60. Saleh Iman, Nagi Khaled. HadoopMutator: A Cloud-Based Mutation Testing Framework. *14th International Conference on Software Reuse, ICSR 2015*. 2014;172–187.

61. ISO/IEC/IEEE 29119-4:2015 - ISO/IEC/IEEE International Standard for Software and systems engineering - Software testing - Part 4: Test techniques. 2015;1–149.

62. *ISO/IEC 12207:2008 Systems and software engineering - Software life cycle processes.* 2008.

63. *Foundation Level Syllabus Reference, International Software Testing Qualifications Board (ISTQB) Std.* 2011.

64. Shaw Mary. Writing Good Software Engineering Research Papers. In Proceedings of the 25th International Conference on Software Engineering: 726–736; 2003.

65. Cruzes Daniela S., Dyb Tore. Research synthesis in software engineering: A tertiary study. In Information and Software Technology: 440–455; 2011.

66. Cruzes Daniela S., Dyba Tore. Recommended Steps for Thematic Synthesis in Software Engineering. *International Symposium on Empirical Software Engineering and Measurement*. 2011;(7491):275–284.

67. Noblit George W., Hare R. Dwight RD. *Meta-ethnography: Synthesizing qualitative studies*. Qualitative research methods series Newbury Park. 1988.

68. Strauss A, Corbin J. Basics of qualitative research: grounded theory procedure and techniques. *Qualitative Sociology*. 1990;13(1):3–21.

69. Da Silva Fabio Q B, Cruz Shirley S J O, Gouveia Tatiana B., Capretz Luiz Fernando. Using meta-ethnography to synthesize research: A worked example of the relations between personality and software team processes. In International Symposium on Empirical Software Engineering and Measurement: 153–162; 2013.

70. ACM SIGSOFT - SEWORLD Mailing List https://www.sigsoft.org/resources/seworld.html. Accessed: June 2018.

71. Brereton Pearl, Kitchenham Barbara A., Budgen David, Turner Mark, Khalil Mohamed. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*. 2007;80(4):571–583.

72. Petticrew Mark, Roberts Helen. *Systematic reviews in the social sciences : a practical guide*. John Wiley & Sons; 2008.

73. Petersen Kai, Vakkalanka Sairam, Kuzniarz Ludwik. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology: :1–18; 2015.

74. Cohen Jacob. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*. 1960;20(1):37–46.

75. McHugh Marry L.. Interrater reliability: the kappa statistic. *Biochemia Medica*. 2012;:276–282.

76. Zhang Zhuoyao, Cherkasova Ludmila, Loo Boon Thau. Parameterizable benchmarking framework for designing a mapreduce performance model. *Concurrency Computation Practice and Experience*. 2014;26(12):2005–2026.

77. Zhang Zhuoyao, Cherkasova L, Loo B T. Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments. *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. 2013;:839–846.

78. Zhang Zhuoyao, Cherkasova Ludmila, Verma Abhishek, Loo Boon Thau. Performance Modeling and Optimization of Deadline-Driven Pig Programs. *ACM Transactions on Autonomous and Adaptive Systems*. 2013;8(3):1–28.

79. Kitchenham Barbara, Brereton O Pearl, Budgen David, Turner Mark, Bailey John, Linkman Stephen. Systematic literature reviews in software engineering - A systematic literature review. *Information and Software Technology*. 2008;51:7–15.

80. Thangaraj Muthuraman, Anuradha Subramanian. State of art in testing for big data. In IEEE International Conference on Computational Intelligence and Computing Research; 2016.

81. Mittal Akhil. Trustworthiness of Big Data. *International Journal of Computer Applications*. 2013;80(9):35–40.

82. Gudipati Mahesh, Rao Shanthi, Mohan Naju D, Kumar Gajja Naveen. *Big Data: Testing Approach to Overcome Quality Challenges*. Big Data: Challenges and Opportunities. (11):65–72. 2013.

83. Nachiyappan S., Justus S. Getting ready for BigData testing: A practitioner's perception. In 4th International Conference on Computing, Communications and Networking Technologies; 2013.

84. Liu Z. Research of performance test technology for big data applications. *IEEE International Conference on Information and Automation, ICIA 2014*. 2014;53–58.

85. Song Ge, Meng Zide, Huet Fabrice, Magoules Frederic, Yu Lei, Lin Xuelian. A Hadoop MapReduce performance prediction method. In IEEE International Conference on High Performance Computing and Communications, and IEEE International Conference on Embedded and Ubiquitous Computing:820–825; 2014.

86. Yin Jinsong, Qiao Yuanyuan. Performance modeling and optimization of MapReduce programs. In IEEE 3rd International Conference on Cloud Computing and Intelligence Systems:180–186; 2014.

87. Yang Xiao, Sun Jianling. An analytical performance model of MapReduce. *IEEE International Conference on Cloud Computing and Intelligence Systems*. 2011; 306–310.

88. Cheng Sheng-Tzong, Wang Hsi-Chuan, Chen Yin-Jun, Chen Chen-Fei. Performance Analysis Using Petri Net Based MapReduce Model in Heterogeneous Clusters. In Advances in Web-Based Learning: 170–179 Springer, Berlin, Heidelberg; 2015.

89. Jin Hui, Qiao Kan, Sun Xian He, Li Ying. Performance under failures of mapReduce applications. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing:608–609; 2011.

90. Castiglione Aniello, Gribaudo Marco, Iacono Mauro, Palmieri Francesco. Exploiting mean field analysis to model performances of big data architectures. *Future Generation Computer Systems*. 2014;37:203–211.

91. Xu Lijie. MapReduce Framework Optimization via Performance Modeling. In 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum: 2506–2509; 2012.

92. Vianna Emanuel, Comarela Giovanni, Pontes Tatiana, et al. Analytical performance models for mapreduce workloads. *International Journal of Parallel Programming*. 2013;41(4):495–525.

93. Han Jungkyu, Ishii Masakuni, Makino Hiroyuki. A Hadoop performance model for multi-rack clusters. In 5th International Conference on Computer Science and Information Technology: 265–274; 2013.

94. Ishii Masakuni, Han Jungkyu, Makino Hiroyuki. Design and performance evaluation for Hadoop clusters on virtualized environment. In International Conference on Information Networking: 244–249; 2013.

95. Cui Xiaolong, Lin Xuelian, Hu Chunming, Zhang Richong, Wang Chengzhang. Modeling the Performance of MapReduce under Resource Contentions and Task Failures. *IEEE 5th International Conference on Cloud Computing Technology and Science*. 2013;(1):158–163.

96. Ahmed Sarker Tanzir, Loguinov Dmitri. On the performance of MapReduce: A stochastic approach. In IEEE International Conference on Big Data: 49–54; 2014.

97. Rahman Md., Lu Xiaoyi, Islam Nusrat Sharmin, Panda Dhabaleswar K.. Performance Modeling for RDMA-Enhanced Hadoop MapReduce. *43rd International Conference on Parallel Processing*; 2014:50–59.

98. Wang Kewen, Khan Mohammad Maifi Hasan. Performance prediction for apache spark platform. In IEEE 17th International Conference on High Performance Computing and Communications, IEEE 7th International Symposium on Cyberspace Safety and Security and IEEE 12th International Conference on Embedded Software and Systems: 166–173; 2015.

99. Fan Yuanquan, Wu Weiguo, Xu Yunlong, et al. Performance prediction model in heterogeneous MapReduce environments. In IEEE International Conference on Computer and Information Technology: 240–245; 2014.

100. Wu Xing, Liu Yan, Gorton Ian. Exploring Performance Models of Hadoop Applications on Cloud Architecture. In 11th International ACM SIGSOFT Conference on Quality of Software Architectures: 93–101; 2015.

101. Mytilinis Ioannis, Tsoumakos Dimitrios, Kantere Verena, Nanos Anastassios, Koziris Nectarios. I/O Performance Modeling for Big Data Applications over Cloud Infrastructures. In IEEE International Conference on Cloud Engineering: 201–206; 2015.

102. Khan Mukhtaj, Jin Yong, Li Maozhen, Xiang Yang, Jiang Changjun. Hadoop Performance Modeling for Job Estimation and Resource Provisioning. *IEEE Transactions on Parallel and Distributed Systems*. 2016;27(2):441–454.

103. Ohnaga Hayata, Aida Kento, Abdul-Rahman Omar. Performance of Hadoop Application on Hybrid Cloud. In International Conference on Cloud Computing Research and Innovation (ICCCRI): 130–138; 2015.

104. Apache Hive: data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. https://hive.apache.org/. Accessed: June 2018.

105. Apache Pig: platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. https://pig.apache.org/. Accessed: June 2018.

106. Sangroya Amit, Singhal Rekha. Performance Assurance Model for HiveQL on Large Data Volume. In 22nd IEEE International Conference on High Performance Computing Workshops: 26–33; 2016.

107. Mogrovejo Renato Javier M., Monteiro José Maria, Machado Javam C., Viana Carlos Juliano M., Lifschitz Sérgio. Towards a Statistical Evaluation of PigLatin Joins. *Journal of Information and Data Management*. 2013;4(3):483.

108. Bian Zhaojuan, Wang Kebing, Wang Zhihong, et al. Simulating Big Data Clusters for System Planning, Evaluation, and Optimization. *2014 43rd International Conference on Parallel Processing*. 2014;391–400.

109. Wang G. Evaluating Mapreduce system performance: A Simulation approach. PhD thesis Virginia Polytechnic Institute and State University. 2012.

110. Chauhan Jagmohan, Makaroff Dwight, Grassmann Winfried. Simulation and performance evaluation of the hadoop capacity scheduler. In 24th Annual International Conference on Computer Science and Software Engineering: 163–177; 2014.

111. Barbierato E, Gribaudo M, Iacono M. A performance modeling language for big data architectures. *27th European Conference on Modelling and Simulation*. 511-517. 2013.

112. Dai Jinquan, Huang Jie, Huang Shengsheng, Huang Bo, Liu Yan. HiTune: dataflow-based performance analysis for big data cloud. *Proceeding USENIXATC'11 Proceedings of the 2011 USENIX conference on USENIX annual technical conference*. 2011;7.

113. Bautista Villalpando Luis, April Alain, Abran Alain. Performance analysis model for big data applications in cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*. 2014;3(1):19–38.

114. Rabkin Ariel, Katz Randy Howard. How hadoop clusters break. *IEEE Software*. 2013;30(4):88–94.

115. Ren Kai, Kwon YongChul, Balazinska Magdalena, Howe Bill. Hadoop's adolescence. *Proceedings of the VLDB Endowment*. 2013;6(10):853–864.

116. Li Sihan, Zhou Hucheng, Lin Haoxiang, et al. A characteristic study on failures of production distributed data-parallel programs. In 35th International Conference on Software Engineering (ICSE): 963–972; 2013.

117. Kavulya Soila, Tan Jiaqi, Gandhi Rajeev, Narasimhan Priya. An Analysis of Traces from a Production MapReduce Cluster. In 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing: 94–103; 2010.

118. Xiao Tian, Zhang Jiaxing, Zhou Hucheng, et al. Nondeterminism in MapReduce considered harmful? an empirical study on non-commutative aggregators in MapReduce programs. In Companion Proceedings of the 36th International Conference on Software Engineering: 44–53; 2014; New York, New York, USA.

119. Camargo Luiz C, Vergilio Silvia R. Classificacao de Defeitos para Programas MapReduce: Resultados de um Estudo Empirico. 2013.

120. Morán Jesús, Riva Claudio, Tuya Javier. MRTree: Functional Testing Based on MapReduce's Execution Behaviour. In International Conference on Future Internet of Things and Cloud: 379–384; 2014.

121. Chen Yu-Fang, Hong Chih-Duo, Sinha Nishant, Wang Bow-Yaw. Commutativity of Reducers. In Springer Berlin Heidelberg 2015 (pp. 131–146).

122. Csallner Christoph, Fegaras Leonidas, Li Chengkai. New Ideas Track: Testing Mapreduce-style Programs. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering.* 2011;504–507.

123. Xu Zhihong, Hirzel Martin, Rothermel Gregg, Wu Kun Lung. Testing properties of dataflow program operators. In 28th IEEE/ACM International Conference on Automated Software Engineering: 103–113; 2013.

124. Marynowski João Eugenio, Santin Altair Olivo, Pimentel Andrey Ricardo. Method for testing the fault tolerance of MapReduce frameworks. *Computer Networks.* 2015;86:1–13.

125. Faghri Faraz, Bazarbayev Sobir, Overholt Mark, Farivar Reza, Campbell Roy H., Sanders William H.. Failure scenario as a service (FSaaS) for Hadoop clusters. *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management - SDMCMM '12.* 2012;:1–6.

126. Morán Jesús, Riva Claudio, Tuya Javier. Testing data transformations in MapReduce programs. In 6th International Workshop on Automating Test Case Design, Selection and Evaluation: 20–25ACM Press; 2015; New York, New York, USA.

127. Mattos Antonio Junior. Test data generation for testing mapreduce systems. PhD thesis Federal University of Paraná. 2011.

128. Li Nan, Escalona Anthony, Guo Yun, Offutt Jeff. A Scalable Big Data Test Framework. In IEEE 8th International Conference on Software Testing, Verification and Validation (ICST): 1–2; 2015.

129. Li Kaituo, Reichenbach Christoph, Smaragdakis Yannis, Diao Yanlei, Csallner Christoph. SEDGE: Symbolic example data generation for dataflow programs. In 28th IEEE/ACM International Conference on Automated Software Engineering: 235–245; 2013.

130. Dörre Jens, Apel Sven, Lengauer Christian. Static type checking of Hadoop MapReduce programs. *Proceedings of the second international workshop on MapReduce and its applications - MapReduce '11.* 2011;:17.

131. Rabkin Ariel, Katz Randy. Static extraction of program configuration options. *2011 33rd International Conference on Software Engineering (ICSE).* 2011;131–140.

132. Ono Kosuke, Hirai Yoichi, Tanabe Yoshinori, Noda Natsuko, Hagiya Masami. Using Coq in specification and program extraction of Hadoop MapReduce applications. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): 350–365; 2011.

133. Chen Yanpei, Ganapathi Archana, Griffith Rean, Katz Randy. The case for evaluating mapreduce performance using workload suites. In IEEE 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS): :390–399; 2011.

134. GridMix: benchmark for Hadoop clusters https://hadoop.apache.org/docs/stable1/gridmix.html. Accessed: June 2018.

135. Li Min, Tan Jian, Wang Yandong, Zhang Li, Salapura Valentina. SparkBench: a spark benchmarking suite characterizing large-scale in-memory data analytics. *Cluster Computing.* 2017;20(3):2575–2589.

136. Ghazal Ahmad, Rabl Tilmann, Hu Minqing, et al. Bigbench: Towards an industry standard benchmark for big data analytics. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data.* 2013;:1197–1208.

137. TPCx-BB: TPC Big Data Benchmark http://www.tpc.org/tpcx-bb/. Accessed: June 2018.

138. Wohlin Claes, Runeson Per, Da Mota Silveira Neto Paulo Anselmo, Engström Emelie, Do Carmo Machado Ivan, De Almeida Eduardo Santana. On the reliability of mapping studies in software engineering. *Journal of Systems and Software.* 2013;86(10):2594–2610.

139. Major Louis, Kyriacou Theocharis, Brereton O. Pearl. Systematic literature review: Teaching novices programming using robots. *IET Software.* 2012;6(6):502–513.

140. Google Scholar https://scholar.google.com/. Accessed: June 2018.