Lourenço *et al. Journal of Big Data* (2015) 2:18
DOI 10.1186/s40537-015-0025-0

Journal of Big Data
a SpringerOpen Journal

RESEARCH Open Access

Choosing the right NoSQL database for the job: a quality attribute evaluation

João Ricardo Lourenço[1*], Bruno Cabral[1], Paulo Carreiro[2], Marco Vieira[1] and Jorge Bernardino[1,3]

*Correspondence:
joaoml@student.dei.uc.pt
[1] CISUC, Department of Informatics Engineering, University of Coimbra, Pólo II – Pinhal de Marrocos, 3030-290 Coimbra, Portugal
Full list of author information is available at the end of the article

Abstract

For over forty years, relational databases have been the leading model for data storage, retrieval and management. However, due to increasing needs for scalability and performance, alternative systems have emerged, namely NoSQL technology. The rising interest in NoSQL technology, as well as the growth in the number of use case scenarios, over the last few years resulted in an increasing number of evaluations and comparisons among competing NoSQL technologies. While most research work mostly focuses on performance evaluation using standard benchmarks, it is important to notice that the architecture of real world systems is not only driven by performance requirements, but has to comprehensively include many other quality attribute requirements. Software quality attributes form the basis from which software engineers and architects develop software and make design decisions. Yet, there has been no quality attribute focused survey or classification of NoSQL databases where databases are compared with regards to their suitability for quality attributes common on the design of enterprise systems. To fill this gap, and aid software engineers and architects, in this article, we survey and create a concise and up-to-date comparison of NoSQL engines, identifying their most beneficial use case scenarios from the software engineer point of view and the quality attributes that each of them is most suited to.

Keywords: NoSQL databases; Key-value; Document store; Columnar; Graph; Software engineering; Quality attributes; Software architecture

Introduction

Relational databases have been the stronghold of modern computing applications for decades. ACID properties (Atomicity, Consistency, Isolation, Durability) made relational databases the solution for almost all data management systems. However, the need to handle data in web-scale systems [1–3], in particular Big Data systems [4], have led to the creation of numerous NoSQL databases.

The term NoSQL was first coined in 1988 to name a relational database that did not have a SQL (Structured Query Language) interface [5]. It was then brought back in 2009 for naming an event which highlighted new non-relational databases, such as BigTable [3] and Dynamo [6], and has since been used without an "official" definition. Generally speaking, a NoSQL database is one that uses a different approach to data storage and access when compared with relational database management systems [7, 8]. NoSQL databases lose the support for ACID transactions as a trade-off for increased availability and scalability [1, 7]. Brewer created the term BASE for these systems - they are Basically Available,

have a Soft state (during which they are not yet consistent), and are Eventually consistent, as opposed to ACID systems [9]. This BASE model forfeits the essential ACID properties of consistency and isolation in order to favor "availability, graceful degradation, and performance" [9]. While originally the term stood for "No SQL", it has recently been restated as "Not Only SQL" [1, 7, 10] to highlight that these systems rarely fully drop the relational model. Thus, in spite of being a recurrent theme in literature, NoSQL is a very broad term, encompassing very distinct database systems.

There are hundreds of readily available NoSQL databases, and each have different use case scenarios [11]. They are usually divided in four categories [2, 7, 12], according to their data model and storage: Key-Value Stores, Document Stores, Column Stores and Graph databases. This classification is due to the fact that each kind of database offers different solutions for specific contexts. The "one size fits all" approach of relational databases no longer applies.

There has been extensive research in the comparison of relational and non-relational databases in terms of their performance for different applications. However, when developing enterprise systems, performance is only one of many quality attributes to be considered. Unfortunately, there has not yet been a comprehensive assessment of NoSQL technology in what concerns software quality attributes. The goal of this article is to fill this gap, by clearly identifying which NoSQL databases better promote the several quality attributes, thus becoming a reference for software engineers and architects.

This article is a revised and extended version of our WorldCIST 2015 paper [13]. It improves and complements the former in the following aspects:

- Three more quality attributes (Consistency, Robustness and Maintainability) were evaluated.
- A new section describing the evaluated NoSQL databases was introduced.
- The state of the art was extended to provide more up to date and thorough information.
- All of the previously evaluated quality attributes were reevaluated in light of new studies and new developments in the NoSQL ecosystem.
- New conclusions and insights derived from the quality attribute based analysis of several NoSQL databases.

Henceforth, the main contributions of this article can be summarized as follows:

- The development of a quality-attribute oriented evaluation of NoSQL databases (Table 2). Software architects may use this information to assess which NoSQL database best fits their quality attribute requirements.
- A survey of the literature on the evaluation of NoSQL databases from a historic perspective.
- The identification of several future research directions towards the full coverage of software quality attributes in the evaluation of NoSQL databases.

The remainder of this article is structured as follows. In Section 'Background and literature review', we perform a review of the literature and evaluation surrounding NoSQL systems. In Section 'Research design and methodology', we introduce the methodology used to select the quality attributes and NoSQL databases that we evaluated, as well as the methodology used in that evaluation. In Section 'Evaluated NoSQL

databases', we present and describe the evaluated NoSQL databases. In Section 'Software quality attributes', we analyze the different quality attributes and identify the best NoSQL solutions for each of these quality attributes according to the literature. In Section 'Results and discussion', a summary table and analysis of the results of this evaluation is provided. Finally, Section 'Conclusions' presents future work and draws the conclusions.

Background and literature review

The word NoSQL was re-introduced in 2009 during an event about distributed databases [5]. The event intended to discuss the new technologies being presented by Google (Google BigTable [3]) and Amazon (Dynamo [14]) to handle high amounts of data. Interest in the research of NoSQL technologies bloomed since then, and lead to the publication of works, such as those by Stonebraker and Cattell [12, 15, 16]. Sonebraker began his research by describing different types of NoSQL technology and differences among those when compared to relational technology. The author argues that the main reasons to move to NoSQL databases are performance and flexibility. Performance is mainly focused on sharing and management of distributed data (i. e. dealing with "Big Data"), while flexibility relates to the semi-structured or unstructured data that may arise on the web.

By 2011, the NoSQL ecosystem was thriving, with several databases being the center of multiple studies [17–20]. These included Cassandra, Amazon SimpleDB, SciDB, CouchDB, MongoDB, Riak, Redis, and many others. Researchers categorized existing databases, and identified what kinds of NoSQL databases existed according to different architectures and goals. Ultimately, the majority agreed on four categories of databases [11]: Document Store, Column Store, Key-value Store and Graph-oriented databases.

Hecht and Jablonski [11] described the main characteristics offered by different NoSQL solutions, such as availability and horizontal scailability. Konstantinou et al. [19] performed a study based on the elasticity of non-relational solutions and compared HBase, Cassandra and Riak during execution of read and update operations. The authors concluded that HBase provided high elasticity and fast reads while Cassandra was capable of delivering fast inserts (writes). On the other hand, according to the authors, Riak did not show good scaling and high performance increase, regardless of the type of access. Many studies focused on evaluating performance [4, 11, 21].

Performance evaluations were made easier by the popularization of the Yahoo! Cloud Serving Benchmark (YCSB), proposed and implemented by Cooper et al. [21]. This benchmark, still widely used today, allows testing the read/write, latency and elasticity capabilities of any database, in particular NoSQL databases. The first studies using YCSB evaluated Cassandra, HBase, PNUTS and MySQL to conclude that each database offers its own set of trade-offs. The authors warn that each database performs at its best in different circumstances and, thus, a careful choice of the one to use must be made according to the nature of each project.

Since 2012, NoSQL databases have been most often evaluated and compared to RDBMSs (Relational Database Management Systems). Performance evaluation carried by [22] compared Cassandra, MongoDB and PostgreSQL, concluding that MongoDB is capable of providing high throughput, but mainly when it is used as a single server

instance. On the other hand, the best choice for supporting a large distributed sensor system was considered Cassandra due to its horizontal scalability. Floratou et al. [4] used YCSB and TPC-H to compare the performance of MongoDB and MS SQL Server, as well as Hive. The authors state that NoSQL technology has room for improvement and should be further updated. Ashram and Anderson [7] studied the data model of Twitter and found that using non-relational technology creates additional difficulties on the programmers' side. Parker et al. [23] also chose MongoDB and compared its performance with MS SQL Server using only one server instance. According to their results, when performing inserts, updates and selects, MongoDB is faster, but MS SQL Server outperforms MongoDB when running complex queries instead of simpler key-value access. In [24], Kashyap et al. compare the performance, scalability and availability of HBase, MongoDB, Cassandra and CouchDB by using YCSB. Their results show that Cassandra and HBase shared similar behaviour, but the former scaled better, and that MongoDB performed better than HBase by factors in the hundreds for their particular workload. The authors are prudent, and note that NoSQL is constantly evolving and that evaluations can quickly become obsolete. Rabl et al. [25] studied Cassandra, Voldemort, HBase, Redis, VoltDB and MySQL Cluster with regards to throughput, latency and scalability. Cassandra's throughput is consistently better than that of the other databases, but it exhibits high latency. Voldemort, HBase and Cassandra all show linear scalability, and Voldemort has the most stable, lowest latency. Of the tested NoSQL databases, VoltDB has the worst results and HBase also lagged behind the other databases in terms of throughput.

Already in 2013, with the research focus on performance, Thumbtack Technologies produced two white papers comparing Aerospike, Cassandra, Couchbase and MongoDB [26, 27]. In [26], the authors compare the durability and performance trade-offs of several state of the art NoSQL systems. Their results firstly show that Couchbase and Aerospike have good in-memory performance, and that MongoDB and Cassandra lagged behind in bulk loading capabilities. Regarding durability, Aerospike beats the competition in large balanced and read-heavy datasets. For in-memory datasets, Couchbase performed similarly to Aerospike as well. In their second paper [27], the authors study failover characteristics. Their results allow for many conclusions, but overall tend to indicate that Aerospike, Cassandra and Couchbase give strong availability guarantees.

In [28], MongoDB and Cassandra are compared in terms of their features and their capabilities by using YCSB. MongoDB is shown to be impacted by high workloads, whereas Cassandra seemed to experience performance boosts with increasing amounts of data. Additionally, Cassandra was found to be superior for update operations. In [29], the authors studied the applicability of NoSQL to RDF (Resource Description Framework data) processing, and make several key observations: 1) distributed NoSQL systems can be competitive with RDF stores with regards to query times; 2) NoSQL systems scale more gracefully than RDF stores when loading data in parallel; 3) complex SPARQL (SPARQL Protocol and RDF Query Language) queries, particularly with joins, perform poorly on NoSQL systems; classical query optimization techniques work well on NoSQL RDF systems; 5) MapReduce-like operations introduce higher latency. As their final conclusion, the authors state that NoSQL represents a compelling alternative to native RDF stores for simple workloads. Several other studies were performed in the same year regarding the applicability of NoSQL to diverse scenarios, such as [30–32].

More recently, as of 2014, experiments have stopped being so focused on performance, and having additional focus on applicability. NoSQL has seen validation and widespread usage, and so, in [10], a survey of some of the most popular NoSQL solutions is described. The authors state some of the advantages and main uses according to the NoSQL database type. In another evaluation, [33] performed their tests using real medical scenarios using MongoDB and CouchDB. They concluded that MongoDB and CouchDB have similar performance and drawbacks and note that, while applicable to medical imaging archiving, NoSQL still has to improve. In [34], the Yahoo! Cloud Serving Benchmark is used with a middleware layer that allows translating SQL queries into NoSQL commands. They tested Cassandra and MongoDB with and without the middleware layer, noting that it was possible to build middleware to ease the move from relational data stores to NoSQL databases. In [35], a write-intensive enterprise application is used as the basis for comparing Cassandra, MongoDB and Couchbase with MS SQL server. The results show that Cassandra outperforms the other NoSQL databases in a four-node setup, and that a MS SQL Server running on a single node vastly outperforms all NoSQL contenders for this particular setup and scenario.

The latest trends in NoSQL research, although still related to applicability and performance, have also concerned the validity of the benchmarking processes and tools used throughout the years. The authors of [36] propose an improvement of YCSB, called YCSB++, to deal with several shortcomings of the benchmark. In [37], the author proposes a method to validate previously proposed benchmarks of NoSQL databases, claiming that rigorous algorithms should be used for benchmarking methodology before any practical use. Chen et al., in [38], perform a survey of benchmarking tools, such as YCSB and TPC-C and list shortcomings and difficulties in implementing MapReduce and Big Data related benchmark systems, proposing methods for overcoming these difficulties. Similar work had already been done by [39], where benchmarks are reviewed and suggestions are given on building better benchmarks.

As we have seen, to the best of our knowledge, there are no studies focused on quality attributes and how each NoSQL system fits each of these attributes. Our work attempts to fill in this gap, by reviewing the literature, in Section 'Software quality attributes', with regards to the different quality attributes, finally presenting our findings in a summary table in Section 'Results and discussion'.

It is important to notice that the analysis of NoSQL systems is inherently bound to the CAP theorem [40]. The CAP theorem, proposed by Brewer, states that no distributed system can simultaneously guarantee Consistency, Availability and Partition-Tolerance. In the context of the CAP theorem [40, 41], consistency is often viewed as the premise that all nodes see the same data at the same time [42]. Indeed, of Brewer's CAP theorem, most databases choose to be "AP", meaning they provide Availability and Partition-Tolerance. Since Partition-Tolerance is a property that often cannot be traded off, Availability and Consistency are juggled, with most databases sacrificing more consistency than availability [43]. In Fig. 1, an illustration of CAP is shown.

Some authors (Brewer being one of them) have come to criticize the way the CAP theorem is interpreted and some have claimed that much has been written in literature under false assumptions [41, 44–46]. The idea of CA (systems which ensure Consistency and Availability) is now most often looked at as a trade-off on a micro-scale [41], where

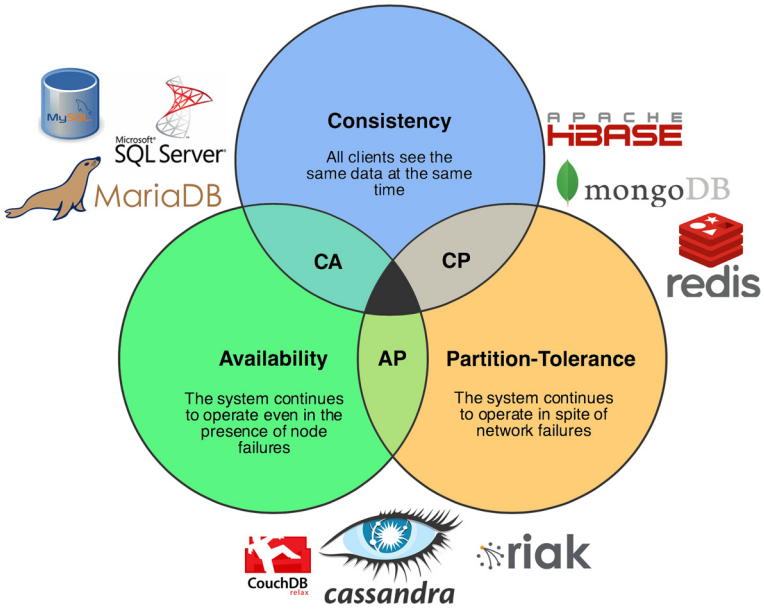

Fig. 1 CAP theorem with databases that "choose" CA, CP and AP

individual operations can have their consistency guarantees explicitly defined. This means that some operations can be tied to full consistency (in the ACID semantics sense), or to one of a vast range of possible consistency options. Modern NoSQL systems allow for this consistency tuning and should therefore not be looked at under such a simplistic view which narrows the whole system to "CA", "CP" or "AP".

Research design and methodology

This work was developed to answer the following research question: "Is there currently enough knowledge on quality attributes in NoSQL systems to aid a software engineer's decision process"? In our literature survey, we did not find any similar work attempting to provide a quality attribute guided evaluation of NoSQL databases. Thus, we devised a methodology to develop this work and answer our original research question.

We began by identifying several desirable quality attributes to evaluate in NoSQL databases. There are hundreds of quality attributes, yet some are nearly ubiquitous to every software project [47], and others are intimately tied to the topic of database systems, storage models and web applications (where the database backend often requires certain quality attributes) [48]. This lead us to identify the following quality attributes to evaluate: Availability, Consistency, Durability, Maintainability, Read and Write performance, Recovery Time, Reliability, Robustness, Scalability and Stabilization Time. These attributes are interdependent and have impact on most software projects. Most of these attributes have also been the target (even if indirectly) of some studies [18, 27, 27, 49, 50], rendering them ideal picks for this work.

Once these quality attributes had been identified, we identified which NoSQL systems were more popular and used, so as to narrow our research to a fixed set of NoSQL databases. This search lead us to selecting Aerospike, Cassandra, Couchbase, CouchDB,

HBase, MongoDB and Voldemort as the systems to evaluate. These are often found in literature [6, 10, 11, 26, 51, 52] and other sources [53] as the most popular and used systems, as well as the most versatile or appropriate to certain scenarios. For instance, while Couchbase and CouchDB share source-code and several similar original design goals, they have evolved into different systems, both with very high success and different characteristics. In much the same way, MongoDB and Cassandra, which are probably the most used NoSQL databases in the market, have fundamentally different approaches to storage model. Thus, our selection of databases attempted to find not only the most popular and mature databases in general, but also those that find high applicability in specific areas.

We surveyed the literature to evaluate the selected quality attributes on the aforementioned databases. This survey took into account already available evaluations regarding certain quality attributes, such as performance [51, 54], consistency [43] or durability [26]. Each of the surveyed papers was taken into account according to the versions of the database tested (e.g. papers with outdated versions were given less relevance), generality of results and overall relevance to this evaluation. The summary table presented in Section 'Results and discussion' is the result of this careful evaluation of the NoSQL literature, technical knowledge found on the NoSQL ecosystem and expert opinions and positions. We also took into account the overall architectures of each NoSQL system (e.g. systems built with durability limitations are intrinsically limited in terms of this quality attribute). The result of this methodology is the aforementioned summary table, which we hope will aid software engineers and architects in their decision process when selecting a given NoSQL database according to a certain quality attribute.

In the following sections, we present the databases that we evaluated from the literature, as well as that evaluation.

Evaluated NoSQL databases

There are several popular NoSQL databases which have gained recognition and are usually considered before other NoSQL alternatives. We studied several of these databases (Aerospike, Cassandra, Couchbase, CouchDB, HBase, MongoDB and Voldemort) by performing a literature review and introduce the first quality attribute based evaluation of NoSQL databases. In this section, these selected databases are presented, with a summary table at the end (Table 1) detailing their characteristics.

Aerospike

Aerospike (formerly known as Citrusleaf [10] and recently open-sourced) is a NoSQL shared-nothing key-value database which offers mainly AP (Availability and Partition-Tolerance) characteristics. Additionally, the developers claim that it provides high consistency [55] by trading off availability and consistency at low granularity in specific subsystems, restricting communication latencies, minimizing cluster size, maximizing consistency and availability during failover situations and automatic conflict resolution. Consistency is guaranteed by using synchronous writes to replicas, guaranteeing immediate consistency. This immediate consistency can be relaxed if the software architects view that as a necessity. Durability is ensured by guaranteeing the use of flash/SSD on every node and performing direct reads from flash, as well as replication on several different layers.

Table 1 Summary table with characteristics of the selected NoSQL databases

	Aerospike	Cassandra	Couchbase	CouchDB	HBase	MongoDB	Voldemort
Category	Key-Value	Column-Store	Document-Store	Document-Store	Column-Store	Document-Store	Key-Value
CAP	AP	AP/CP	CP	AP	CP	CP	AP
Consistency	Configurable (several options)	Configurable (several options)	Eventual Consistency	Eventual Consistency	Configurable (strong and eventual consistency)	Configurable (several options)	Read-Repair (client handles conflicts)
Durability	Notified written to replica nodes	Configurable (several options)	Configurable (several options)	Configurable (notified written to at least one disk)	Configurable (several options)	Configurable (several options)	Notified written to desired nodes
Querying	Internal API	Internal API, SQL like (CQL)	Internal API (MapReduce)	Internal API (MapReduce)	Internal API	Internal API, MapReduce, complex query support	Internal API (get, put delete)
Concurrency Control	Read-commited isolation level (support for optimistic concurrency control)	MVCC	MVCC (application can select Optimistic or Pessmistic locking)	MVCC (application can select Optimistic or Pessmistic locking)	Optimistic locking with MVCC	Master-slave with multi-granularity locking	Optimistic locking with MVCC
Partitioning Scheme	Proprietary (Paxos based)	Consistent Hashing	Consistent Hashing	Consistent Hashing (third party)	Range Based	Consistent Hashing	Consistent Hashing
Native Partitioning	Yes	Yes	Yes	No	Yes	Yes	Yes

Failover can be handled in two different ways [55]: focusing on High consistency on AP mode, or on Availability in CP (Consistency and Partition-Tolerance) mode. The former uses techniques to "virtually eliminate network based partitioning", including fast heartbeats and consistent Paxos based cluster formation. These techniques favor Consistency over Availability to ensure that the system does not enter a state of network partition. If, however, partitioning occurs, Aerospike offers two conflict handling policies: one relies on the database's auto-merging capabilities, and the other offloads the conflict to the application layer so that application developers can resolve the conflicts by themselves and re-write the right data back to the database. The second way that Aerospike manages failover is to provide Availability while in CP mode. In this mode, availability needs to be sacrificed by, for instance, forcing the minority quorum(s) to halt, thus avoiding data inconsistency if a network split occurs.

Aerospike is, henceforth, an in-memory database with disk persistence, automatic data partitioning and synchronous replication, offering cross data center replication and configurability in the failover handling mechanism, preferring full consistency or high consistency [10, 52, 55].

Cassandra

Cassandra is an open-source shared-nothing NoSQL column-store database developed and used in Facebook [10, 52, 56]. It is based on the ideas behind Google BigTable [3] and Amazon Dynamo [14].

Cassandra is similar to BigTable in what concerns the data model. The minimal unit of storage is a column, with rows consisting of columns or super columns (nested columns). Columns themselves consist of the name, value and timestamp, all of which are provided by the client. Since it is column-based, rows need not have the same number of columns [10].

Cassandra supports a SQL-like language called CQL, together with other protocols [10]. Indexes and secondary indexes are supported, and atomicity is guaranteed at the level of one table row. Persistence is ensured by logging. Consistency is highly tunable according to the desired operation – the application developer can specify the desired level of consistency, trading off latency and consistency. Conflicts are resolved based on timestamps (the newest record is kept). The database operates in master-master mode [52], where no node is different from another, and combines disk-persistence with in-memory caching of results, resulting in high write throughput operations [52, 56]. The master-master architecture makes it easy for horizontal scalability to happen [56]. There are several different partitioning techniques and replication can be automatically managed by the database [56].

CouchDB

Apache CouchDB is another open-source project, written in Erlang, and following a document-oriented approach [10]. Documents are written in JSON and are meant to be accessed with CouchDB's specific implementation of MapReduce views written in Javascript.

This database uses a B-tree index [10], updated during data modifications. These modifications have ACID properties on the document level and the use of MVCC (Multi-Version Concurrency Control) enables readers to never block [10]. CouchDB's

document manipulation uses optimistic locks by updating an append-only B-tree for data storage, meaning that data must be periodically compressed. This compression, in spite of maintaining availability, may hinder performance [10].

Regarding fault-tolerant replication mechanisms [57], CouchDB supports both master-slave and master-master replication that can be used between different instances of CouchDB or on a single instance. Scaling in CouchDB is achieved by replicating data, a process which is performed asynchronously. It does not natively support sharding/partitioning [10]. Consistency is guaranteed in the form of strengthened eventual consistency [10], and conflict resolution is performed by selecting the most up to date version (the application layer can later try to merge conflicting changes, if possible, back into the document). CouchDB's programming interface is REST-based [10, 57]. Ideally, CouchDB should be able to fit the whole dataset into the RAM of the cluster, as it is primarily a RAM-based database.

Couchbase

Couchbase is a combination of Membase (a key-value system with memcached compatibility) and CouchDB. It can be used in key-value fashion, but is considered a document store working with JSON documents (similarly to CouchDB) [10].

Documents, in Couchbase, have an intrinsic unique id and are stored in what are called data buckets. Like CouchDB, queries are built using MapReduce views in Javascript. The optimistic locking associated with an append-only B-tree is also implemented like in CouchDB. The default consistency level is eventual consistency (due to MapReduce views being constructed asynchronously). There is also the option of specifying that data should be indexed immediately [10].

A major difference when comparing Couchbase with CouchDB regards sharding [10]. Whereas CouchDB does not natively support sharding (there are projects, such as CouchDB Lounge [10] which enable this), Couchbase comes with transparent sharding off-the-shelf, with application transparency. Replication is also a major point of difference between the two databases, as couchbase supports intercluster and intracluster replication. The latter is performed within a cluster, guaranteeing immediate consistency. The former kind of replication ensures eventual consistency and is made asynchronously between geographically distributed clusters (conflict resolution is performed in the same way CouchDB does it). This database is mostly intended to run in-memory, so as to hold the whole dataset in RAM [10, 29].

HBase

HBase is an open-source database written in Java and developed by the Apache Software Foundation. It is intended to be the open-source implementation of the Google BigTable principles, and relies on the Apache Hadoop Framework and the Apache ZooKeeper projects. It is, therefore, a column-store database [10].

HBase's architecture is highly inspired by Google's BigTable [3, 10], and, thus, their capabilities are similar. There are, however, certain differences. The Hadoop Distributed File System (HDFS) is used for distributed storage, although other backends can be used (e.g. Hadoop MapReduce), in place of the Google File System. HBase also supports several master servers to improve system reliability, but does not support the concept of locality. Similarly to Google BigTable, it does not support full ACID semantics, although

several properties are guaranteed [58]. Atomicity is guaranteed within a row and consistency ensures that no rows result of interleaving operations (i.e. the row must have effectively existed at some point in time). Still on the topic of consistency, rows are guaranteed to only move forward in time, never backward, and scans do not exhibit snapshot isolation, but, rather, the "read commited" isolation level. Durability is established in the sense that all data which is read has already been made durable (i.e. persisted to disk), and that all operations returning success have ensured this durability property. This can be configured, so that data is only periodically flushed to disk [58]. HBase does not support secondary indexes, meaning that data can only be queried by the primary key or by a table scan. It is worth noting that data, in HBase is also absent of data types (everything is a byte array) [52]. Regarding the programming interface, HBase can be interfaced using a Java API, a REST interface, and the Avro and Thrift protocols [10].

MongoDB

MongoDB is an open-source document-oriented database written in C++ and developed by the 10gen company. It uses JSON (data is stored and transferred in a binary, more compact form named BSON), allowing for a schemaless data model where the only requirement is that an id is always present [10, 56].

MongoDB's horizontal scalability is mainly provided through the use of automatic sharding [56]. Replication is also supported using locks and the asynchronous master-slave model, meaning that writes are only processed by the master node and reads can be made from both the master node and from one of the slave nodes. Writes are propagated to the slave nodes by reading from the master's *oplog* (operation log) [56]. Database clients can choose the kind of consistency models they wish, by defining whether reads from secondary nodes are allowed and from how many nodes the confirmation must be obtained.

Document manipulation is a strong focus of MongoDB, as the database provides different frameworks (e.g. MapReduce and Aggregation Framework) and ways of interacting with documents [10]. These can be queried, sorted, projected, iterated with cursors, aggregated, among other operations. The changes to a document are guaranteed to be atomic. Indexing can be used on one or several fields (implemented using B-trees), with the possibility of using two-dimensional spatial indexes for geometry-based data [10]. There are many different programming interfaces supported by MongoDB, with most popular programming languages having native bindings. A REST interface is also supported [10].

Voldemort

Project Voldemort is an open-source key-value store implemented in Java which presents itself as an open-source implementation of the Amazon Dynamo database [10, 14, 59, 60]. It supports scalar values, lists and records with named fields associated with a single key. Arbitrary fields can be used if they are serializable [10].

Operations on the data are simple and limited: there are *put*, *get* and *delete* commands [10, 60]. In this sense, Voldemort can be considered (as the developers themselves put it), "basically just a big, distributed, persistent, fault-tolerant hash table" [59]. For data modification, the MVCC mechanism is used [10].

Replication is supported using the consistent hashing method (proposed in [61]) [10, 60]. Sharding is implemented transparently with support for adding and removing nodes in real-time (although this feature was not always easily available [62]). Data is meant to stay in RAM as much as possible, with persistent data storage using several mechanisms, such as Berkley DB [60]. Voldemort uses a Java API [52].

Summary

Table 1 summarizes the characteristics of the studied NoSQL databases, similar to the work seen in [1, 11, 17, 49, 63], but providing a broader and more up to date view of these characteristics. Its information is derived from the previous sections and additional relevant sources ([12, 64–71]). Each NoSQL database is described according to key characteristics: category (e.g. Key-Value database), positioning in the context of the CAP theorem, consistency guarantees and configurability, durability guarantees and configurability, querying possibilities and mechanisms (i.e. how are queries made and how complex can queries be?), concurrency control mechanisms, partitioning schemes and the existence of native partitioning. It should be noted that, as we have previously discussed, modern NoSQL databases often allow for fine-tuning of consistency and availability properties on a per-query basis, making the CAP-based classification ("AP", "CP", etc) overly simplistic [41, 44–46].

Software quality attributes

In the previous section we identified and described several NoSQL databases. In this section, we survey the literature on NoSQL databases to find how each of these satisfy the software quality attributes that we selected. Each subsection explores the NoSQL literature on a given quality attribute, drawing conclusions regarding all of the evaluated NoSQL databases. This information is then summarized in the following section (Section 'Results and discussion'), where a table is provided to aid software architects and engineers in their decision process.

Availability

Availability concerns what percentage of time a system is operating correctly [1]. NoSQL technology is inherently bound to provide availability more easily than relational systems. In fact, given the existence of Brewer's CAP theorem [40], and the presence of failures in real-world systems (whether they are related to the network or to an application crash), NoSQL databases oppose most relational databases by favoring availability instead of consistency. Thus, one can assert that the higher the availability of a NoSQL system, the less likely it is that it provides high consistency guarantees. Several NoSQL databases provide ways to tune the trade-off between consistency and availability, including Dynamo [14], Cassandra, CouchDB and MongoDB [9].

Apache CouchDB uses a shared-nothing clustering approach, allowing all replica nodes to continue working even if they are disconnected, thus being a good candidate for systems where high availability is needed [9]. It is worth noting, however, that this database periodically requires a compaction step which may hinder system performance, but which does not affect the availability of its nodes under normal operation [3].

In 2013, [27] tested several NoSQL Databases (Aerospike, Cassandra, Couchbase and MongoDB) concerning their failover characteristics. Their results showed that Aerospike

had the lowest downtime, followed by Cassandra, with MongoDB having the least favorable downtime. One should note that the results shown in the paper are limited to RAM-only datasets and hence might not be the best source for real-world scenarios. MongoDB's results are also not surprising, as even though it allows for fine-tuning (to adjust the consistency-availability trade-offs), several tests have shown that it is not the best choice for a highly available system, in particular due to overhead when nodes are rejoining the system (see, for instance, [1, 9] and our section on reliability). Lastly, [5] tested several NoSQL databases on the Cloud and noted that Riak could not provide high-availability under very high loads.

Thus, there is no obvious candidate for a highly available system, but there are several competing solutions, in particular when coupled with systems such as Memcached [2]. The specific architecture (number of replicas, consistency options, etc.) employed will play a major role, as pointed by several authors [27, 72]. Furthermore, the popular MongoDB and Riak databases seem less likely to be good picks for this use case scenario.

Consistency

Consistency is related to transactions and, although not universally defined, can be seen as the guarantee that transactions started in the future see the effects of transactions committed in the past, coupled with the insurance of database constraints [73–75]. It is useful to recall that, in the context of the CAP theorem [40, 41], consistency is often seen as the premise that all nodes see the same data at the same time [42] (i.e., the CAP version of consistency is merely a subset of the ACID version of the same property [41]). We have previously seen that consistency and availability are highly related properties of NoSQL systems.

Cassandra has several different consistency guarantees [76]. The database allows for tunable consistency at both the read and write level, even with near-ACID semantics if consistency level "ALL" is picked. MongoDB, in spite of being generally regarded as a CP system, offers similar consistency options [76, 77]. Couchbase offers strong consistency guarantees for document access, whereas query access is eventually consistent [67]. HBase provides eventual consistency without fine-tuning being possible [58] (there is only the choice of opting for strong or eventual consistency), and CouchDB, being an AP system, fully relies on eventual consistency [78]. The Voldemort project puts more stress on application logic to deal with inconsistencies in data, by using read repair [60].

Regarding concrete experiments, not much has been done to study consistency as a property in itself. Recent work by Bermbach et al. [76] has seen the proposal of a general architecture for consistency benchmarking. The authors test their proposal on Cassandra and MongoDB, concluding that MongoDB performed better, but also noting that they are merely proposing an architecture and that their tests might have been impacted negatively due to their testing environment. The authors of [54] study Cassandra and Couchbase in a real world microblogging scenario, concluding that Couchbase provided consistent results faster (i.e. the same value took less time to reach all node replicas). In [79], the authors study Amazon S3's consistency behavior and conclude that it frequently violates the monotonic read consistency property (other related work is presented by Bermbach et al. [76]). It seems that a general framework for testing consistency might provide with more in depth answers to the effectiveness of consistency trade-offs and techniques provided by each NoSQL database.

In summary, as the NoSQL ecosystem matures, there is a tendency towards micro-management of consistency and availability [41], with some solutions opting to provide consistency (withholding availability), others providing availability (withholding consistency) and another set, such as Cassandra and MongoDB, allowing for fine-tuning based on a query basis.

Durability

Durability refers to the requirement that data be valid and committed to disk after a successful transaction [1]. As we have previously covered, NoSQL databases act on the premise that consistency doesn't need to be fully enforced in the real world, preferring to sacrifice it in adjustable ways for achieving higher availability and partition tolerance. This impacts durability, as if a system suffers from consistency problems, its durability will also be at risk, leading to potential data loss [26].

In [26], the authors test Aerospike, Couchbase, Cassandra and MongoDB in a series of tests regarding durability and performance trade-offs. Their results featured Aerospike as the fastest performing database by a factor of 5-10 when the databases were set to synchronous replication. However, most scenarios do not rely on synchronous replication, but rather asynchronous (meaning that changes aren't instantly propagated among nodes). In that sense, the same authors, which in [27] studied the same databases in the context of failover characteristics, show that MongoDB loses less data upon node failure when asynchronous replication is used. Cassandra comes as forerunner to MongoDB by about a factor of 100, and Aerospike and Couchbase both lose very large amounts of data. In [1], MongoDB is found to have issues with data loss when compared to CouchDB, in particular during recovery after a crash. In the same paper, the authors highlight that CouchDB's immutable append only B+ Tree ensures that files are always in a valid state. CouchDB's durability is also noticed and justified by the authors of [2]. It should be noted that document-based systems such as MongoDB usually use a single-versioning system, which is designed specifically to target durability [49]. HBase's reliance on Hadoop means that it is inherently durable in the way requests are processed, as several authors have noted [80–82]. Voldemort's continuing operation as the backend to Linkedin's service is backed by strong durability [83], although there is a lack of studies focusing specifically on Voldemort's durability.

In conclusion, as with other properties, the durability of NoSQL systems can be fine-tuned according to specific needs. However, databases based on immutability, such as CouchDB, are good picks for a system with good durability due to their inherent properties [1]. Furthermore, single-version databases, such as MongoDB, should also be the focus of those interested in durability advantages.

Maintainability

Maintainability is a quality attribute that regards the ease with which a product can be maintained, i.e., upgraded, repaired, debugged and met with new requirements [84]. From an intuitive point of view, systems with many components (e.g. several nodes) should add complexity and difficult maintainability, and this is a view that several authors agree with [7, 85]. On the other hand, as some have hypothesized, the benefits of thoughtful modularity and task division make the case for a more maintainable system [86]. Assessing maintainability is a difficult problem which has seen vast amounts of research throughout

the years, but it has seldom been focused explicitly on the database itself (in particular due to the widespread usage of the relational model with similar database interfaces).

In spite of the perceived difficulty in assessing the maintainability of NoSQL systems, there has been some research on the subject. Dzhakishev [50] studied the usability and maintainability of several NoSQL solutions in a real enterprise scenario. The author notes how MongoDB and Neo4j have "great shell applications", easing maintainability, and that Neo4j even has a web interface (other NoSQL databases have such software, e.g. Couchbase Server). The authors of [87] study social network system implementation processes and rely on their own application-specific code to ensure maintainability of their application. They claim that versioning the schema using subversion is good for their goals. Throughout their work, maintainability seems to be moved more into the application layer and less into the database layer, possibly suggesting that NoSQL maintainability must be achieved with help of the developer. In [29], another real world experiment, the authors note the added maintainability difficulties in using HBase, Couchbase, Cassandra and CouchDB to replace their RDF data system. Similarly, Han [88] also faced maintainability problems with MongoDB when comparing it with the maintainability of relational alternatives. Although no particular study in literature has focused on the maintainability of Voldemort, from the point of view of ease of use, this database seems harder to configure (in particular in terms of node topology changes) than others [62].

It seems that most NoSQL systems offer limited maintainability when compared with traditional RDBMSs, but the literature has little to say with regards as to which is the more maintainable. Some authors [50, 87] point in the direction of the ease of use of web interfaces and the readiness of tools. In that sense, Couchbase and Neo4j are prominent examples of easy to use and setup databases. On the other hand, MongoDB and HBase are known to be hard to install [89, 90] or to confuse first users, thus limiting ease of use. Further research can and should be developed in this area so as to be able to truly compare the maintainability of NoSQL solutions.

Performance

When it comes to the performance and execution of different types of operations, NoSQL databases are divided mostly into two categories: read and write optimized [21, 91]. That means that, in part, regardless of the system type and records, the database has an optimization that is granted by its mechanisms used for storage, organization and retrieval of data. For example, Cassandra is known for being very highly optimized during execution of writes (inserts) and is not able to show the same performance during reads [21, 91]. The database achieves this by using a cache for write operations (updates are immediately written to a logfile, then cached in memory and only later written to disk, making the insertion process itself faster). In general, Column Store and Key-Value databases use more memory to store their data, some of those being completely in-memory (and, hence, completely unsuited for other attributes such as durability).

Document Stores, on the other hand, are considered as being more read optimized. This behavior resembles that of relational databases, where data loading and organization is slower, with the advantage of better preparing the system for future reads. Examples of this are MongoDB and Couchbase. If one compares most Column Store databases, such as Cassandra, to the document-based NoSQL landscape, with regards

to read-performance, then the latter wins. This has been seen in numerous works such as [51, 54] and [26]. We should also consider that databases such as MongoDB and Couchbase are considered more enterprise solutions with a set of mechanisms and functionality besides traditional key-value retrieval, which is mostly used not only by Key-Value stores but also by Column Store databases. This impacts performance significantly, as the need for additional functionality is usually associated with high performance costs.

Much work has been done with regards to performance testing of databases. Since NoSQL is constantly changing, past evaluations quickly become obsolete, but recent evaluations have been performed, some of which we now enumerate. In [51], a performance overview is given for Cassandra, HBase, MongoDB, OrientDB and Redis. The conclusions are that Redis is particularly well suited for all kinds of workloads (although this result should be taken lightly, since the database has many trade-offs with other quality attributes), that Cassandra performs very well for write/update scenarios, that overall OrientDB performs poorly when tested in this scenario and that HBase deals poorly with update queries. In [33], MongoDB and CouchDB are tested in a medical archiving scenario, with MongoDB showing better performance. In [92], MongoDB is shown to perform poorly for CRUD (create, read, update and delete) bulk operations when compared with PostgreSQL. Regarding write-heavy scenarios, a real-world enterprise scenario is presented in [35], where Cassandra, Couchbase and MongoDB are compared with MS SQL Server. In a four-node environment, Cassandra outperforms the NoSQL competition greatly (which is expected, since it is a write-optimized database), but is outperformed by a single-node MS SQL Server instance. Less recent, but also relevant, is the work presented in [26, 27], where Cassandra, Couchbase, MongoDB and Aerospike are tested. Aerospike is shown to have the better performance, with Cassandra coming in as second in terms of read-throughput, and Couchbase in terms of write-throughput. Rabl et al. [25] compared Voldemort, Redis, HBase, Cassandra, MySQL Cluster and VoltDB with regards to throughput, scalability and disk usage, and noted that while Cassandra's throughput dominated most tests, Voldemort exhibits a good balance between read and write performance, competing with the other databases. The authors also note that VoltDB had the worst performance and HBase's throughput is low (although it scales better than most other databases).

In conclusion, performance highly depends on the database architecture. Column Store databases, such as Cassandra, are usually more oriented towards writing operations, whereas document based databases are more read-oriented. This last group of databases is also generally more feature-rich, bearing more resemblance to the traditional relational model, thus tending to have a bigger performance penalty. Experiments have been validating this theory and we can conclude that, in contrast with some of the other quality attributes studied in this article, performance is definitely not lacking in terms of research and evaluation.

Reliability

Reliability concerns the system's probability of operating without failures for a given period of time [49]. The higher the reliability, the less likely it is that the system fails. Recently, Domaschka et al., in [49], have proposed a taxonomy for describing distributed databases with regards to their reliability and availability. Since reliability is significantly

harder to define than availability (as it depends on the context of the application requirements), the authors suggest that software architects consider the following two questions: "(1) How are concurrent writes to the same item resolved?; (2) What is the consistency experienced by clients?". With these in mind, and by using their taxonomy, we can see that systems which use single-version techniques, such as Redis, Couchbase, MongoDB and Neo4j, all perform online write conflict resolution detection, being good picks for a reliable system in the sense that they answer question (1) with reliable options. Regarding question (2), MongoDB, CouchDB, Neo4J, Cassandra and HBase all provide strong consistency guarantees. Thus, in order to achieve strong consistency guarantees and good concurrent write conflict resolution, as proposed by the authors, one should look at systems which have both these characteristics - MongoDB and Neo4j.

In conclusion, in spite of reliability being an important quality attribute, we have found that there is little focus in current literature about this topic, and, therefore, are limited in our answers to this research question.

Robustness

Robustness is concerned with the ability of the database to cope with errors during execution [93]. Relational technology is known for its robustness, but many questions still arise when such a topic is discussed in the context of NoSQL [4]. If, from one point of view, one might consider that NoSQL databases are more robust due to their replication (i.e. crashes are "faded out" by appropriate replication and consensus algorithms [94]), from another, lack of code maturity and extensive testing might make NoSQL less robust in general [4, 12]. Few [12, 95] has been written on this subject, although there have been some real world studies where the impact of NoSQL on a system's robustness was considered (even if only indirectly). In [88], Han experiments with MongoDB as a possible replacement for a traditional RDBMS in an air quality monitoring scenario. With regards to robustness, the author notes that as cluster scale and workloads increase, robustness becomes a more pressing issue (i.e. problems become more evident). Ranjan, in [95], studies Big Data platforms and notes that lack of robustness is a question in Big Data scheduling platforms and, in particular, in the NoSQL (Hadoop) case. In 2011, the authors of [12] postulated that robustness would be an issue for NoSQL, as the technology was new and needed testing. Neo4j is seen by some as a robust graph-based database [96, 97]. Lior et al. [98] reviewed security issues in NoSQL databases and found that Cassandra and MongoDB were subject to Denial of Service attacks, which can be seen as a system with a lack of robustness. Similarly, Manoj [77] presents a comparative table of features for Cassandra, MongoDB and HBase, where HBase is identified as having an intrinsic single point of failure that needs to be overcome by explicitly using failover clustering. Lastly, in [87], the authors claim Cassandra is robust due to Facebook's contribution to its development and the fact that it is used as one of the backends of the social network.

Overall, not much can be concluded for each individual database in terms of robustness. A benchmark for robustness is currently lacking in the NoSQL ecosystem, and software engineers looking for the most robust database would benefit from research into this area. The most up to date information and research indicates that more popular and used databases are more robust, although in general these systems are seen as less robust than their relational counterparts when tested in practice.

Scalability

Scalability concerns a system's ability to deal with increasing workloads [1]. In the context of databases, it may be defined as the change in performance when new nodes are added, or hardware is improved [99]. NoSQL databases have been developed specifically to target scenarios where scalability is very important. These systems rely on horizontal and "elastic" scalability, by adding more nodes to a system instead of upgrading hardware [3, 4, 9]. The term "elastic" refers to elasticity, which is a characterization of the way a cluster reacts to the addition or removal of nodes [99].

In [18], the authors compared Cassandra and HBase, improving upon previous work. They concluded that both databases scale linearly with different read and write performances. They also provided a more in-depth analysis at Cassandra's scalability, noticing how performing horizontal scalability with this platform leads to less performance hassles than performing vertical scalability.

In [99], the authors measure the elasticity and scalability of Cassandra, HBase and MongoDB. They showed surprise by identifying "superlinear speedups for clusters of size 24" when using Cassandra, stating that "it is almost as if Cassandra uses better algorithms for large cluster sizes". For clusters of sizes 6 and 12, their results show HBase the fastest competitor with stable performance. Regarding elasticity, they found that HBase gives the best results, stabilizing the database significantly faster than Cassandra and MongoDB.

Rabl et al. [25] studied the scalability (and other attributes) of Cassandra, Voldemort, HBase, VoltDB, Redis and MySQL cluster. They noted the linear scalability of Cassandra, HBase and Voldemort, noting, however, that Cassandra's latency was "peculiarly high" and that Voldemort's was stable. HBase, while the worst of these databases in terms of throughput, scaled better than the rest. Regarding the different scalability capabilities of the databases themselves, Cassandra, HBase and Riak all support the addition of machines during live operation Key-value databases, such as Aerospike and Voldemort, are also easier to scale, as the data model allows for better distribution of data across several nodes [12]. In particular, Voldemort was designed to be highly scalable, being the major backend behind LinkedIn.

Further studies regarding scalability are needed in literature. It is clear that NoSQL databases are scalable, but the question of which scale the most, or with the best performance, is still left unanswered. Nevertheless, we can conclude that popular choices for highly scalable systems are Cassandra and HBase. One must also take notice that scalability will be influenced by the particular choice of configuration parameters.

Stabilization Time and Recovery Time

Besides availability, there are other failover characteristics which determine the behavior of a system and might impact system stability. In the study made in [27], which we have already covered, the authors measure the time it takes for several NoSQL systems to recover from a node failure - the recovery time -, as well as the time it takes for the system to stabilize when that node rejoins the cluster -the stabilization time. They find that MongoDB has the best recovery time, followed by Aerospike (when in synchronous change propagation mode), with Couchbase having values an order of magnitude slower and Cassandra two orders of magnitude slower that MongoDB. Regarding the time to stabilize on node up, all systems perform well ($<$ 1ms) with the exception of MongoDB and Aerospike. The former takes a long 31 seconds to recover to stabilize on node reentry,

and Aerospike, in synchronous mode takes 3 seconds. These results tend to indicate that MongoDB and Aerospike are good picks if one is looking for good recovery times, but that these choices should be taken with care, such that when a node reenters the system, it does not affect its stability.

Overall, the topic of failover is highly dependent of configuration and desired properties and should be studied more thoroughly (we note this as part of our future work). The current literature is limited and does not allow for very general and broad conclusions.

Results and discussion

By gathering all the information we presented in the previous sections, we tried to establish a comprehensive summary table that indicates which database best suits each quality attribute. Each column in Table 2 represents a NoSQL database, and each row one of the studied software engineering quality attributes. A 5-point scale ranging from "Great for this quality attribute" (⊞) to "Bad for this quality attribute" (⊖) is presented. This allows for a direct comparison among databases. For instance, Cassandra is write-performance oriented, even more than Couchbase (hence their difference in label in the table). We assigned worse grades when a database is not an ideal pick, according to our literature revision. This doesn't mean that the database renders a given quality attribute unattainable, but that, according to the current literature, it is not the best when compared to the others. In cases where we were unsure what was the correct answer, we used the question mark symbol (?).

We used the criteria described in each of the previous sections to quantify the databases. Regarding availability, the downtime was used as a primary measure, together with relevant studies [5, 27]. Consistency was graded according to two essential criteria: 1) how much the database can provide ACID-semantics consistency and 2) how much

Table 2 Summary table of different quality attributes studied for popular databases

	Aerospike	Cassandra	Couchbase	CouchDB	HBase	MongoDB	Voldemort
Availability	⊞	⊞	⊞	⊞	−	−	⊞
Consistency	⊞	⊞	+	+	□	⊞	+
Durability	−	+	+	−	+	+	+
Maintainability	+	□	+	+	−	□	−
Read-Performance	+	−	⊞	□	−	⊞	+
Recovery Time	⊞	⊖	+	?	?	⊞	?
Reliability	−	+	−	+	+	⊞	?
Robustness	+	+	□	□	⊖	□	?
Scalability	⊞	⊞	⊞	−	⊞	−	+
Stabilization Time	⊖	+	+	?	?	⊖	?
Write-Performance	+	⊞	+	−	+	−	⊞

Legend:
⊞ Great
\+ Good
□ Average
− Mediocre
⊖ Bad
? Unknown/N.A.

can consistency be fine-tuned. Durability was measured according to the use of single or multi version concurrency control schemes, the way that data are persisted to disk (e.g. if data is always asynchronously persisted, this hinders durability), and studies that specifically targeted durability [26]. Regarding maintainability, the criteria were the currently available literature studies of real world experiments, the ease of setup and use, as well as the accessibility of tools to interact with the database. For read and write performance, we considered recent studies [27] and the fine-tuning of each database, as noted in the previous sections. Reliability is graded according to the taxonomy presented in [49] and by looking at synchronous propagation modes (databases which do not support them tend to be less reliable, as Domaschka et. al note). Database robustness was assessed with the real world experiments carried by researchers, as well as the available documentation on possible tendency of databases to have problems dealing with crashes or attacks (e.g. being subject to DoS attacks). With respect to scalability, we looked at each database's elasticity, its increase in performance due to horizontal scaling, and the ease of on-line scalability (i.e. is the live addition of nodes supported?). For recovery time and stabilization time, highly related to availability, we based our classification on the results shown in [27] (implying that our grading of these attributes is mostly limited to their particular study and should be taken with apprpriate care). We looked at the databases described in Section 'Evaluated NoSQL databases'.

By analyzing Table 2, we can see that Aerospike suffers from data loss issues, affecting its durability, and it also has issues with stabilization time (in particular in synchronous mode). Cassandra is a multi-purpose database (in particular due to its configurable consistency properties) which mostly lacks read performance (since it is tuned for write-heavy workloads). CouchDB provides similar technology to MongoDB, but is better suited for situations where availability is needed. Couchbase provides good availability capabilities (coupled with good recovery and stabilization times), making it a good candidate for situations where failover is bound to happen. HBase has similar capabilities to Cassandra, but is unable to cope with high loads, limiting its availability in these scenarios, and is also the worst database in terms of robustness (this is mostly due to research seen in [77, 95]). MongoDB is the database that mostly resembles the classical relational use case scenario - it is better suited for reliable, durable, consistent and read-oriented use cases. It is somewhat lacking in terms of availability, scalability and write-performance, and it is very hindered by its stabilization time (which is also one of the reasons for its low availability). Furthermore, it is not as efficient during write operations. Lastly, we lack some information on Voldemort, but find it to be a poor pick in terms of maintainability. It is, however, a good pick for durable, write-heavy scenarios, and provides a good balance between read and write performance (in line with [25]). We should highlight that there are more quality attributes that should be focused on, which we intend to do in future work, and, thus, that this table does not intend to show that "one database is better than another", but, rather, that some database is better for a particular use case scenario where these attributes are needed.

Many software quality attributes are highly interdependent. For example, availability and consistency, in the context of the CAP theorem, are often polarized. Similarly, availability, stabilization time and recovery time are highly related, since low stabilization and recovery time are bound to hinder availability. With this in mind, there are several interesting findings in the summary table we have presented.

Availability, stabilization time and recovery time, as mentioned, are highly related software quality attributes. In this sense, it is interesting to note that polarizing results are found for different databases. Software engineers looking at the availability software quality attribute should note that although Aerospike, Cassandra, Couchbase, CouchDB and Voldemort all provide high availability, some of these databases are not ideal picks for situations where a fast recovery time is needed. Indeed, of these databases, only Aerospike and MongoDB have a "Great" rating, with Cassandra having the worst possible grading. On the other hand, Aerospike and MongoDB have poor stabilization times, but Cassandra has a "Good" rating for this quality attribute. Couchbase, another highly available system, although not having any "Great" rating in these two quality attributes, has a "Good" rating. Thus, for systems which desire high availability with a balance of stabilization and recovery time, Couchbase is an ideal pick.

It is interesting to note that Aerospike and Cassandra achieve high availability and high consistency ratings. A naive application of the CAP theorem to distributed systems and NoSQL systems would tend to indicate that both of these quality attributes would have to ultimately be traded off. Nevertheless, as other authors have pointed out [41, 44–46], this is not the case, and our table reflects it. Systems such as Cassandra allow for these properties to be traded off on a query basis. This, ultimately combined with the other characteristics of each database, result in high ratings in both these quality attributes.

If inspecting only the availability and scalability quality attributes, it becomes clear that they are highly correlated. Nearly all systems with high availability also have high scalability. The only exception to this observation being CouchDB (this database does not support native partitioning, hindering its scalability). In cases where availability is limited, results are somewhat polarized: HBase achieves high scalability, whereas MongoDB is also hindered in terms of its scalability (this can easily be traced to MongoDB's locking mechanism; indeed, as we have mentioned, this database is the most similar one to the typical relational use case scenario).

There are other highly correlated quality attributes which can be surprising. For instance, there is a high correlation between scalability and write performance. When one of these quality attributes is tending towards being "Great", the other is too; similarly, when one tends to be "Bad", the other does too. This result provides insight into how scalability is achieved in many NoSQL systems: write optimizations (particularly found in column-store databases) help achieve scalability, and systems with poor write performance tend to be fairly limited in its scalability. Contrasting with this positive correlation, it seems that read performance is slightly negatively correlated with scalability: databases with high scalability tend to have higher write performance than read performance. This could be due to the fact that many of these databases rely on partitioning as an efficient way to scale, and the fact that partitioning improves write performance (through parallel writes) much more than it does read performance (nevertheless, this would be interesting to study as future work). Consistency and recovery time are also quality attributes that share a high degree of correlation. This result is also intuitive, since systems that react quickly to the loss of a node will tend to have fewer conflicts and, thus, fewer consistency problems. Still on the topic of consistency, it shares some similarity with robustness, in our table. Indeed, robust systems tend to also be consistent ones (notable exceptions are HBase and MongoDB). This relationship, however, is probably due to the nature of each database and not to any particular reason (i.e. there is no intrinsic relationship between consistency

and robustness). Finally, reliability and write performance are often in polarized positions (e.g., Aerospike has "Good" write performance but low reliability).

There are some quality attributes for which no "Great" rating has been attributed. Indeed, in terms of durability, maintainability, robustness and stabilization time, no NoSQL system was found to achieve optimal results. This indicates directions of future work for NoSQL databases. Some of these ratings can be explained due to the infancy of these systems – robustness and maintainability are properties that evolve with time, as systems mature, bugs are found and new functionality is added. These two quality attributes have the worst overall ratings and reveal weaknesses of NoSQL systems.

Another point of interest that becomes clear with the summary table is that while some quality attributes do not have a "Great" rating, there are others for which no "Bad" rating is given: availability, consistency, maintainability, reliability, scalability, read and write performance. Some of these quality attributes, such as consistency and scalability, are actually found to have generally high ratings. This implies that these quality attributes are among the key attributes offered by NoSQL databases. It is no surprise, then, that availability, consistency and scalability, some of the three major reasons for the initial development of NoSQL databases [63], are among these attributes.

Although performance is often considered an isolated quality attribute, read and write performance can be different. This difference is reflected in our table, and it is interesting to analyse the performance quality attribute as a whole with the data presented in the table. Most NoSQL databases polarize on performance characteristics, either having high ratings on read or write performance (Cassandra, Couchbase, HBase and MongoDB), but there are some exceptions. Aerospike provides a balance between write and read performance without reaching the "Great" rating in either of these quality attributes. On the other hand, Voldemort provides high write performance ("Great") and good read performance ("Good"), while Couchbase offers good write performance ("Good"), but high read performance ("Great"). This implies that software engineers can look to Voldemort, Couchbase and Aerospike as "balanced" systems in terms of performance, with Voldemort and Couchbase tending slightly towards more specific write or read scenarios, respectively.

The only quality attributes where there are neither "Great" or "Bad" ratings are durability and maintainability. Indeed, it would make little sense for a NoSQL system to have bad durability, since this is a key attribute of most database solutions. On the other hand, the trade-offs associated with NoSQL often mean that durability must be sacrificed, resulting in no system achieving the best durability yet (this is a clear are for future work in NoSQL systems).

Conclusions

In this article we described the main characteristics and types of NoSQL technology while approaching different aspects that highly contribute to the use of those systems. We also presented the state of the art of non-relational technology by describing some of the most relevant studies and performance tests and their conclusions, after surveying a vast number of publications since NoSQL's birth. This state of the art also intended to give a time-based perspective to the evolution of NoSQL research, highlighting four clearly distinct periods: 1) Database type characterization (where NoSQL was in its infancy and researchers tried to categorize databases into different sets); 2) Performance evaluations,

with the advent of YCSB and a surge in NoSQL popularity; 3) Real-world scenarios and criticism to some interpretations of the CAP theorem; and 4) An even bigger focus on applicability and a reinvigorated focus on the validation of benchmarking software.

We concluded that although there have been a variety of studies and evaluations of NoSQL technology, there is still not enough information to verify how suited each non-relational database is in a specific scenario or system. Moreover, each working system differs from another and all the necessary functionality and mechanisms highly affect the database choice. Sometimes there is no possibility of clearly stating the best database solution. Furthermore, we tried to find the best databases on a quality attribute perspective, an approach still not found in current literature – this is our main contribution. In the future, we expect that NoSQL databases will be more used in real enterprise systems, allowing for more information and user experience available to conclude the most appropriate use of NoSQL according to each quality attribute and further improve this initial approach.

As we have seen, NoSQL is still an in-development field, with many questions and a shortage of definite answers. Its technology is ever-increasing and ever-changing, rendering even recent benchmarks and performance evaluations obsolete. There is also a lack of studies which focus on use-case oriented scenarios or software engineering quality attributes (we believe ours is the first work on this subject). All of these reasons make it difficult to find the best pick for each of the quality attributes we chose in this work, as well as others. The summary table we presented makes it clear that there is a current need for a broad study of quality attributes in order to better understand the NoSQL ecosystem, and it would be interesting to conduct research in this domain. When more studies and with more consistent results have been performed, a more thorough survey of the literature can be done, and with clearer, more concise results.

Software architects and engineer can look to the summary table presented in this article if looking for help understanding the wide array of offerings in the NoSQL world from a quality attribute based overview. This table also brings to light some hidden or unexpected relationships between quality attributes in the NoSQL world. For instance, scalability is highly related to the write performance, but not necessarily the read performance. Additionally, broad sets of quality attributes that are highly related (e.g. availability, stabilization time and recovery time) can be individually studied, so that the appropriate trade-offs can be selected for a candidate system architecture.

Our literature review allows us to establish future directions on research regarding a quality-attribute based approach to NoSQL databases. It is our belief that the development of a framework for assessing most of these quality attributes would greatly benefit the lifes of software engineers and architects alike. In particular, research is currently lacking in terms of Reliability, Robustness, Durability and Maintainability, with most work in literature focusing on raw performance. Future work in this area, with the development of such a framework for quality attribute evaluation, would undoubtedly benefit the NoSQL research in the long term.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
JRL surveyed most of the literature. BC, JB and MV helped identifying and evaluating the quality attributes, as well as finding the appropriate NoSQL databases to study, guiding the research and iteratively reviewing and revising the work.

PC provided an initial case study from which this work originally sprouted, and helped identifying the evaluated quality attributes. All authors read and approved the final manuscript.

Acknowledgement
This research would not have been made possible without support and funding of the FEED - Free Energy Data and iCIS - Intelligent Computing in the Internet Services (CENTRO-07 - ST24 - FEDER - 002003) projects, to which we are extremely grateful.

Author details
[1]CISUC, Department of Informatics Engineering, University of Coimbra, Pólo II – Pinhal de Marrocos, 3030-290 Coimbra, Portugal. [2]Critical Software, Parque Industrial de Taveiro, lote 49, 3045-504 Coimbra, Portugal. [3]ISEC – Superior Institute of Engineering of Coimbra, Polytechnic Institute of Coimbra, 3030-190 Coimbra, Portugal.

Received: 2 June 2015 Accepted: 27 July 2015
Published online: 14 August 2015

References

1. Orend K (2010) Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer. Dissertation, Technische Universität München
2. Leavitt N (2010) Will nosql databases live up to their promise? Computer 43(2):12–14
3. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: A distributed storage system for structured data. ACM Trans Comput Syst (TOCS) 26(2):4
4. Floratou A, Teletia N, DeWitt DJ, Patel JM, Zhang D (2012) Can the elephants handle the nosql onslaught? Proc VLDB Endowment 5(12):1712–1723
5. Lith A, Mattson J (2013) Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data. Dissertation, Chalmers University of Technology
6. Sadalage PJ, Fowler M (2012) NoSQL Distilled: a Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, Upper Saddle River, NJ
7. Schram A, Anderson KM (2012) Mysql to nosql: data modeling challenges in supporting scalability. In: Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity. ACM, Tucson, Arizona, USA. pp 191–202
8. NoSQL. http://nosql-database.org/. Accessed June, 2015
9. Strauch C (2011) NoSQL Databases. Lecture: Selected Topics on Software-Technology Ultra-Large Scale Sites, Stuttgart Media University
10. Kuznetsov S, Poskonin A (2014) Nosql data management systems. Program Comput Softw 40(6):323–332
11. Hecht R, Jablonski S (2011) Nosql evaluation. In: International Conference on Cloud and Service Computing. IEEE, Hong Kong, China. pp 336–41
12. Cattell R (2011) Scalable sql and nosql data stores. ACM SIGMOD Record 39(4):12–27
13. Lourenço JR, Abramova V, Vieira M, Cabral B, Bernardino J (2015) Nosql databases: A software engineering perspective. In: New Contributions in Information Systems and Technologies. Springer, São Miguel, Azores, Portugal. pp 741–750
14. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: amazon's highly available key-value store. In: ACM SIGOPS Operating Systems Review. ACM, Stevenson, Washington, USA. Vol. 41. pp 205–220
15. Stonebraker M (2010) Sql databases v. nosql databases. Commun ACM. 53(4):10–11
16. Stonebraker M (2011) Stonebraker on nosql and enterprises. Commun ACM. 54(8):10–11
17. Tudorica BG, Bucur C (2011) A comparison between several nosql databases with comments and notes. In: Roedunet International Conference (RoEduNet), 2011 10th. IEEE, Iasi, Romania. pp 1–5
18. Dory T, Mejías B, Van Roy P, Tran NL (2011) Comparative elasticity and scalability measurements of cloud databases. In: Proc of the 2nd ACM Symposium on Cloud Computing (SoCC). IEEE, Iasi, Romania Vol. 11
19. Konstantinou I, Angelou E, Boumpouka C, Tsoumakos D, Koziris N (2011) On the elasticity of nosql databases over cloud management platforms. In: Proceedings of the 20th ACM international conference on Information and knowledge management, Glasgow. pp 24–28
20. Han J, Haihong E, Le G, Du J (2011) Survey on nosql database. In: Pervasive Computing and Applications (ICPCA), 2011 6th International Conference On. IEEE, Port Elizabeth, South Africa. pp 363–366
21. Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM Symposium on Cloud Computing. ACM, Indianapolis, Indiana, USA. pp 143–154
22. van der Veen JS, van der Waaij B, Meijer RJ (2012) Sensor data storage performance: Sql or nosql, physical or virtual. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference On. IEEE, Honolulu, HI, USA. pp 431–438
23. Parker Z, Poe S, Vrbsky SV (2013) Comparing nosql mongodb to an sql db. In: Proceedings of the 51st ACM Southeast Conference. ACM, Savannah, Georgia, USA. p 5
24. Kashyap S, Zamwar S, Bhavsar T, Singh S (2013) Benchmarking and analysis of nosql technologies. Int J Emerg Technol Adv Eng 3:422–426
25. Rabl T, Gómez-Villamor S, Sadoghi M, Muntés-Mulero V, Jacobsen HA, Mankovskii S (2012) Solving big data challenges for enterprise application performance management. Proc VLDB Endowment 5(12):1724–1735
26. Nelubin D, Engber B (2013) Ultra-High Performance NoSQL Benchmarking: Analyzing Durability and Performance Tradeoffs. Thumbtack Technology, Inc., White Paper
27. Nelubin D, Engber B (2013) Nosql failover characteristics: Aerospike, cassandra, couchbase, mongodb
28. Abramova V, Bernardino J (2013) Nosql databases: Mongodb vs cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering. ACM, New York, USA. pp 14–22

29. Cudré-Mauroux P, Enchev I, Fundatureanu S, Groth P, Haque A, Harth A, Keppmann FL, Miranker D, Sequeda JF, Wylot M (2013) Nosql databases for rdf: an empirical evaluation. In: The Semantic Web–ISWC 2013. Springer, Berlin. pp 310–325
30. Yang CT, Liu JC, Hsu WH, Lu HW, Chu WC-C (2013) Implementation of data transform method into nosql database for healthcare data. In: Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2013 International Conference On. IEEE, Taipei Taiwan. pp 198–205
31. Blanke T, Bryant M, Hedges M (2013) Back to our data-experiments with nosql technologies in the humanities. In: Big Data, 2013 IEEE International Conference On. IEEE, Silicon Valley, CA, USA. pp 17–20
32. Fan C, Bai C, Zou J, Zhang X, Rao L (2013) A dynamic password authentication system based on nosql and rdbms combination. In: LISS 2013. Springer, Berlin. pp 811–819
33. Silva LAB, Beroud L, Costa C, Oliveira JL (2014) Medical imaging archiving: A comparison between several nosql solutions. In: Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference On. IEEE, Valencia, Spain. pp 65–68
34. Rith J, Lehmayr PS, Meyer-Wegener K (2014) Speaking in tongues: Sql access to nosql systems. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, Gyeongju, Korea. pp 855–857
35. Lourenço JR, Abramova V, Cabral B, Bernardino J, Carreiro P, Vieira M (2015) Nosql in practice: a write-heavy enterprise application. In: IEEE BigData Congress 2015. New York, June 27-July 2, 2015
36. Wingerath W, Friedrich S, Gessert F, Ritter N (2015) Who Watches the Watchmen? On the Lack of Validation in NoSQL Benchmarking. In: Seidl T, Ritter N, Schöning H, Sattler K-U, Härder T, Friedrich S, Wingerath W (eds). Datenbanksysteme für Business, Technologie und Web (BTW 2015), Hamburg
37. George TB A proposed validation method for a benchmarking methodology. Int J Sustainable Econ Manag (IJSEM) 3(4):1–10
38. Chen Y, Raab F, Katz R (2014) From tpc-c to big data benchmarks: A functional workload model. In: Specifying Big Data Benchmarks. Springer, Berlin. pp 28–43
39. Qin X, Zhou X (2013) A survey on benchmarks for big data and some more considerations. In: Intelligent Data Engineering and Automated Learning–IDEAL 2013. Springer, Berlin. pp 619–627
40. Brewer EA (2000) Towards robust distributed systems. In: PODC. IEEE, Portland, Oregon, USA Vol. 7
41. Brewer E (2012) Cap twelve years later: How the "rules" have changed. Computer 45(2):23–29
42. Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News 33(2):51–59
43. Wada H, Fekete A, Zhao L, Lee K, Liu A (2011) Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In: CIDR. ACM, Asilomar, California, USA Vol. 11. pp 134–143
44. Stonebraker M (2010) In search of database consistency. Commun ACM 53(10):8–9
45. Abadi D (2010) Problems with CAP, and Yahoo's Little Known NoSQL System, DBMS Musings, blog, (2010); on-line resource. http://dbmsmusings.blogspot.pt/2010/04/problems-with-cap-and-yahoos-little.html. Accessed June 2015
46. Hale C (2010) You can't sacrifice partition tolerance. http://codahale.com/you-cant-sacrifice-partition-tolerance/. Accessed July 2015
47. Clements P, Kazman R, Klein M (2003) Evaluating Software Architectures. Tsinghua University Press, Beijing
48. Offutt J (2002) Quality attributes of web software applications. IEEE Softw 2:25–32
49. Domaschka J, Hauser CB, Erb B (2014) Reliability and availability properties of distributed database systems. In: Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International. IEEE, Ulm, Germany. pp 226–233
50. Dzhakishev D (2014) Nosql databases in the enterprise. An experience with tomra s receipt validation system
51. Abramova V, Bernardino J, Furtado P (2014) Which nosql database? a performance overview. Open J Databases (OJDB) 1(2):17–24
52. Gudivada VN, Rao D, Raghavan VV (2014) Nosql systems for big data management. In: Services (SERVICES), 2014 IEEE World Congress On. IEEE, Anchorage, AK, USA. pp 190–197
53. DB-Engines Ranking: Knowledge Base of Relational and NoSQL Database Management Systems. http://db-engines.com/en/ranking. Accessed July, 2015
54. Fonseca A, Vu A, Grman P (2013) Evaluation of NoSQL databases for large-scale decentralized microblogging, Universitat Politècnica de Catalunya
55. Aerospike (2014) ACID Support in Aerospike. Aerospike, Mountain View, California
56. Haughian G (2014) Benchmarking replication in nosql data stores. Dissertation, Imperial College London
57. Nocuń Ł, Nieć M, Pikuła P, Mamla A, Turek W (2013) Car-finding system with couchdb-based sensor management platform. Comput Sci 14(3):403–422
58. Apache Hbase ACID Semantics. http://hbase.apache.org/acid-semantics.html. Accessed July, 2015
59. Voldemort Project Github. https://github.com/voldemort/voldemort. Accessed July, 2015
60. Voldemort: Design – Voldemort. www.project-voldemort.com/voldemort/design.html. Accessed July, 2015
61. Karger D, Sherman A, Berkheimer A, Bogstad B, Dhanidina R, Iwamoto K, Kim B, Matkins L, Yerushalmi Y (1999) Web caching with consistent hashing. Comput Netw 31(11):1203–1213
62. Voldemort Rebalancing (as Seen in the Wayback Time Machine Archive in 2012). http://web.archive.org/web/20100923080327/http://github.com/voldemort/voldemort/wiki/Voldemort-Rebalancing. Accessed July, 2015
63. Pokorny J (2013) Nosql databases: a step to database scalability in web environment. Int J Web Inf Syst 9(1):69–82
64. Aerospike Clustering. https://www.aerospike.com/docs/architecture/clustering.html. Accessed July, 2015
65. MongoDB Concurrency FAQ. http://docs.mongodb.org/manual/faq/concurrency/. Accessed July, 2015
66. Couchbase Blog: Optimistic or Pessimistic Locking, Which One Should You Pick? http://blog.couchbase.com/optimistic-or-pessimistic-locking-which-one-should-you-pick. Accessed July, 2015

67. 10 Things Developers Should Know About Couchbase. http://blog.couchbase.com/10-things-developers-should-know-about-couchbase. Accessed July, 2015
68. Cassandra Concurrency Control. http://teddyma.gitbooks.io/learncassandra/content/concurrent/concurrency_control.html. Accessed July, 2015
69. Apache HBase Reference Guide. https://hbase.apache.org/book.html. Accessed July, 2015
70. Apache HBase Durability Javadoc. https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Durability.html. Accessed July, 2015
71. Sumbaly R, Kreps J, Gao L, Feinberg A, Soman C, Shah S (2012) Serving large-scale batch computed data with project voldemort. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies. USENIX Association, San Jose, CA, USA. pp 18–18
72. Beyer F, Koschel A, Schulz C, Schäfer M, Astrova I, Grivas SG, Schaaf M, Reich A (2011) Testing the suitability of cassandra for cloud computing environments. In: CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization. IARIA XPS, Venice/Mestre, Italy. pp 86–91
73. Ports DR, Clements AT, Zhang I, Madden S, Liskov B (2010) Transactional consistency and automatic management in an application data cache. In: OSDI. USENIX Association, Vancouver, BC, Canada Vol. 10. pp 1–15
74. Eswaran KP, Gray JN, Lorie RA, Traiger IL (1976) The notions of consistency and predicate locks in a database system. Commun ACM 19(11):624–633
75. Haerder T, Reuter A (1983) Principles of transaction-oriented database recovery. ACM Comput Surv (CSUR) 15(4):287–317
76. Bermbach D, Zhao L, Sakr S (2014) Towards comprehensive measurement of consistency guarantees for cloud-hosted data storage services. In: Performance Characterization and Benchmarking. Springer, Berlin. pp 32–47
77. Manoj V (2014) Comparative study of nosql document, column store databases and evaluation of cassandra. Int J Database Manag Syst (IJDMS) 6:11–26
78. CouchDB Consistency. http://guide.couchdb.org/draft/consistency.html. Accessed July, 2015
79. Bermbach D, Tai S (2011) Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In: Proceedings of the 6th Workshop on Middleware for Service Oriented Computing. ACM, Lisbon, Portugal. p 1
80. Konishetty VK, Kumar KA, Voruganti K, Rao G (2012) Implementation and evaluation of scalable data structure over hbase. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. USENIX Association, Chennai, India. pp 1010–1018
81. Harter T, Borthakur D, Dong S, Aiyer AS, Tang L, Arpaci-Dusseau AC, Arpaci-Dusseau RH (2014) Analysis of hdfs under hbase: a facebook messages case study. In: FAST. USENIX Association, Santa Clara, CA, USA. Vol. 14. p 12
82. Konishetty VK, Kumar KA, Voruganti K, Rao GVP (2012) Implementation and evaluation of scalable data structure over hbase. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. ICACCI '12. ACM, New York, NY, USA. pp 1010–1018. doi:10.1145/2345396.2345559. http://doi.acm.org/10.1145/2345396.2345559
83. Chandra DG, Prakash R, Lamdharia S (2012) A study on cloud database. In: Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference On. IEEE, Mathura, India. pp 513–519
84. Riaz M, Mendes E, Tempero E (2011) Towards predicting maintainability for relational database-driven software applications: Extended evidence from software practitioners. Int J Softw Eng Appl 5(2):107–121
85. Roijackers J, Fletcher G (2012) Bridging sql and nosql. Master's thesis, Eindhoven University of Technology
86. Fujimoto R, McLean T, Perumalla K, Tacic I (2000) Design of high performance rti software. In: Distributed Simulation and Real-Time Applications, 2000.(DS-RT 2000). Proceedings. Fourth IEEE International Workshop On. IEEE, San Francisco, CA, USA. pp 89–96
87. Škrabálek J, Kunc P, Nguyen F, Pitner T (2013) Towards effective social network system implementation. In: New Trends in Databases and Information Systems. Springer, Berlin. pp 327–336
88. Han M (2015) The application of nosql database in air quality monitoring. In: 2015 International Conference on Intelligent Systems Research and Mechatronics Engineering. Atlantis Press, Zhengzhou, China
89. Chodorow K (2013) MongoDB: the Definitive Guide. "O'Reilly Media, Inc.", 103a Morris Street, Sebastopol, CA 95472, USA
90. George L (2011) HBase: the Definitive Guide. "O'Reilly Media, Inc", 103a Morris Street, Sebastopol, CA 95472, USA
91. Gajendran SK (1998) A Survey on NoSQL Databases. Department of Computer Science, Donetsk
92. Hammes D, Medero H, Mitchell H (2014) Comparison of NoSQL and SQL Databases in the Cloud. Proceedings of the Southern Association for Information Systems (SAIS), Macon, GA, 21-22 March, 2014
93. Eager DL, Sevcik KC (1983) Achieving robustness in distributed database systems. ACM Trans Database Syst (TODS) 8(3):354–381
94. Feng H (2012) Benchmarking the suitability of key-value stores for distributed scientific data. Dissertation, The University of Edinburgh
95. Ranjan R (2014) Modeling and simulation in performance optimization of big data processing frameworks. Cloud Comput IEEE 1(4):14–19
96. Huang H, Dong Z (2013) Research on architecture and query performance based on distributed graph database neo4j. In: Consumer Electronics, Communications and Networks (CECNet), 2013 3rd International Conference On. IEEE, Xianning, China. pp 533–536
97. Schreiber A, Ney M, Wendel H (2012) The provenance store proost for the open provenance model. In: Provenance and Annotation of Data and Processes. IEEE, Changsha, China. pp 240–242
98. Okman L, Gal-Oz N, Gonen Y, Gudes E, Abramov J (2011) Security issues in nosql databases. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference On, IEEE, Changsha, China. pp 541–547
99. Kuhlenkamp J, Klems M, Röss O (2014) Benchmarking scalability and elasticity of distributed database systems. Proc VLDB Endowment 7(13):1219–1230