Evaluation of ACE Properties of Traditional SQL and NoSQL Big Data Systems

Maria Teresa Gonzalez-Aparicio
Department of Computing
University of Oviedo
Gijón, Spain
maytega@uniovi.es

Muhammad Younas
School of ECM
Oxford Brookes University
Oxford, United Kingdom
m.younas@brookes.ac.uk

Javier Tuya
Department of Computing
University of Oviedo
Gijón, Spain
tuya@uniovi.es

Rubén Casado
Accenture Digital
Spain
ruben.casado.tejedor@accenture.com

ABSTRACT
Traditional SQL and NoSQL big data systems are the backbone for managing data in cloud, fog and edge computing. This paper develops a new system and adopts the TPC-DS industry standard benchmark in order to evaluate three key properties, availability, consistency and efficiency (ACE) of SQL and NoSQL systems. The contributions of this work are manifold. It evaluates and analyses the tradeoff between the ACE properties. It provides insight into the NoSQL systems and how they can be improved to be sustainable for a more wide range of applications. The evaluation shows that SQL provides stronger consistency, but at the expense of low efficiency and availability. NoSQL provides better efficiency and availability but lacks support for stronger consistency. In order for NoSQL systems to be more sustainable they need to implement transactional schemes that enforce stronger consistency as well as better efficiency and availability.

CCS Concepts
• Information systems→Data management systems →Database management system engines→Database transaction processing→Data locking

Keywords
Big data; SQL; NoSQL; Riak; TPC-DS, Data consistency.

1. INTRODUCTION
Cloud computing delivers on-demand IT services, such as storage, compute power and servers, over the Internet in order to offer flexibility, scalability and elasticity in service provisioning. Cloud service consumers only pay for the services they use. This reduces their operational and maintenance cost of IT services. The common model of cloud service provisioning is built around data centers where cloud services are centrally stored and managed. In order to alleviate issues of centralized cloud new models of edge and fog computing have been emerged. Edge computing offers users and developers cloud services and resources at the edge of a network or Internet. It delivers compute, storage and data services much closer to end devices and/or end users [1]. Fog computing model can be defined as an additional layer that provides a bridge between edge computing (resources) and the (centralized) cloud. For example, fog computing can help in cloud resource virtualization in order to dynamically distribute workload across different (edge) computing nodes.

Despite the differences between cloud, fog and edge computing models, they all share the need of storing, processing and analysing data for different types of applications. The work presented in this paper focuses on the traditional SQL and NoSQL big data systems which are used by all the three models, cloud, edge and fog computing. It evaluates the three key properties, availability, consistency and efficiency (ACE). Availability means that data is available. For instance, if one node (of a system) is failed or overloaded (with many requests) then data can be accessed from another node. Consistency means that data must remain in consistent state whenever it is updated. Efficiency refers to the process that data is efficiently accessed and/or updated.

Traditional SQL databases have widely been used for a number of years by various organizations and companies. SQL databases (such as MySQL, Oracle) are built using rigorous theoretical and mathematical models such as relational algebra. They follow the principles of data normalization and integrity constraints in order to maintain strong data consistency. SQL database systems have been used for applications that need strong consistency and data integrity constraints, for example, banking applications, customers and products data, online shopping and so on.

NoSQL big data systems (such as Riak, MongoDB, Couch) are relatively new and they do not generally adopt strong theoretical/mathematical models. They give preference to efficiency and availability over data consistency. They do not follow data normalization principles (as in SQL). Instead they follow weaker or eventual consistency model.

NoSQL systems have been used for applications that need high efficiency and availability but weaker consistency. For instance, NoSQL systems are capable of processing hundreds of thousands of social media messages/per sec. Social media data may tolerate weaker consistency. But applications such as financial transactions or online shopping may not tolerate weaker consistency. In such applications problems caused by inconsistency could be more serious than having lower efficiency or availability.

We believe that the three ACE properties are crucial to different applications that use cloud, fog and/or edge computing. We propose and develop a new system in order to evaluate and analyse

these properties of SQL and NoSQL big data systems. We implement an online shopping cart as a case study and use TPC-DS industry standard benchmark [2] in order to evaluate the ACE properties.

The premise behind implementing online shopping cart is that it is more appropriate to evaluate and analyse the ACE properties of SQL and NoSQL big data systems (see Section 4). This work is based on the work [3], which developed a new transactional model and tested the effects of velocity on consistency in NoSQL big data systems.

The contributions of this work are manifold:

- It evaluates and analyses the ACE properties of SQL and NoSQL big data systems. This reveals the strengths and weaknesses of SQL and NoSQL systems.
- It uses TPC-DS as a benchmark which has been used for evaluating real life and commercial data systems. TPC (Transaction Processing Council) is a non-profit corporation which define benchmarks for transaction processing and database systems which are used by industry [2].
- The results show that SQL provides stronger consistency but at the expense of low efficiency and availability. NoSQL big data systems provide better efficiency and availability but they lack support for stronger consistency.
- It develops a transactional scheme in order to improve NoSQL big data systems so that they are more sustainable for a wide range of applications (e.g., online shopping, financial systems) that need stronger consistency.

The reminder of this paper is sructured as follows. Section 2 presents the rationale and background of this research work. Section 3 reviews and analyses related work and technologies. Section 4 illustrates the case study and the proposed approach. It also presents experimental results and analysis. Section 5 presents the conclusion.

2. RATIONALE AND BACKGROUND

The number of cloud, fog, and edge users and services are increasing at a very high speed. As a consequence, data storage systems have to manage enormous volume of data from a wide range of sources such as online social media, online shopping, commerce and business, web searching/browsing, client's reviews, and so on. This phenomenon is fostering a giant leap in the 21st century economy growth [4], as strategic decision making information can be inferred from the data sets, using different techniques. Data therefore play a crucial role in different fields such as commerce, business, healthcare, and research among others [5].

The large-scale data, generated from different sources, has led to the creation of the concept of "big data" [6], which is characterized by 3Vs, 4Vs or 5Vs model [7], i.e. Volume, Variety, Velocity, Veracity and Value. Volume refers to the massive amount of data which is gathered from different sources and is processed by scalable cloud systems. Variety is the different types of data gathered, including, structured, semi-structured and unstructured format (images, text, data logs, etc.). Velocity is the speed at which data is created, stored, analysed and visualized. Veracity is related to quality of data such as data accuracy, trust, and reliability. Value is the process of obtaining (monetary, social, business) value from the data.

Traditional relational SQL databases are not well-equipped to manage the big data models. Thus a new generation of databases named as NoSQL ("Not only SQL" or "NoSQL") has emerged. In general, NoSQL big data systems support schema-free, replication of data and do not require normalization of data (as in traditional SQL systems). They use CRUD (Create, Read, Update, Durability) operations for data manipulation. They generally implement "weak consistency", meaning that there is no guarantee that an application (or user) will access the latest version of a specific data. They are based on the concept of eventual consistency which states that eventually all the updates will reach all the replicas and data will eventually become consistent.

In order to illustrate the issue of eventual consistency in NoSQL databases, let's consider the commonly used architecture of a NoSQL database, as shown in Fig. 1. In NoSQL databases, data is generally replicated across three nodes so as to provide better efficiency and availability of data. When one replica is updated then, ideally, the update should be instantly reflected across all the three nodes. This is to ensure that all the replicas (at different nodes) are consistent — which is the case of strong consistency.

However, according to eventual consistency it is permissible that some of the replicas (at some nodes) may not be updated instantly due to network or some other latency. It means that for a certain period of time some replicas remain inconsistent and different applications (or users) could read different values after an update operation.

In some applications, such as social media, it is tolerable if some of the replicas are not updated instantly. For example, it could be acceptable if social media messages (e.g., 'Our lunch break is at 12:30pm.' or 'We plan to have grilled fish for lunch.') are not instantly updated across all the nodes. Fig. 1 shows the situation where some social media data is updated at Node 1 (Replica 1) and Node 2 (Replica 2) but not at Node 3 (Replica 3). In this situation, data available from Node 3 will not be consistent but it could be tolerable.

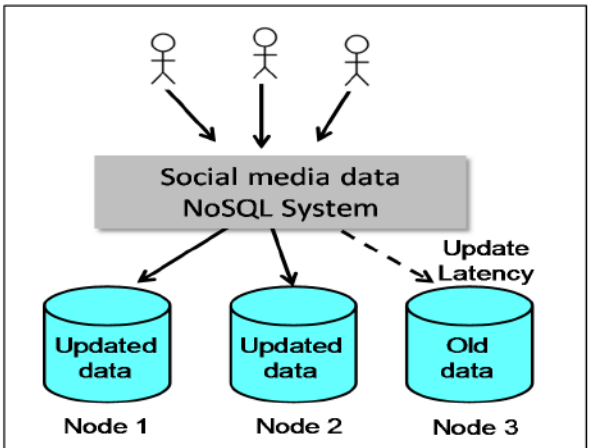

Figure 1. NoSQL and social media data.

Now consider an example of a bank data which is updated by an online (shopping) application or transaction, as shown in Fig. 2. If a customer has 50 EUR in the account and s/he buys something for 20 EUR, then all the replicas (at three nodes) should store the updated value of 30 EUR. This is to maintain the consistency of data. In addition, any other transaction over 20 EUR should be rejected by the system. But the technique of eventual consistency may not prevent such transaction as data could not be updated instantly. In Fig. 2, data read from Node 3 (50 EUR) is not consistent, as this node does not reflect the updated version of data, which is 30 EUR.

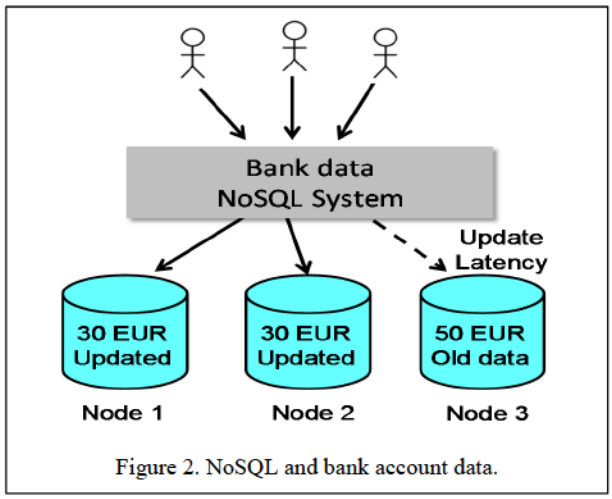

Figure 2. NoSQL and bank account data.

Similarly, in other e-commerce systems such as Amazon's Auctions and eBay it is critical to ensure an e-commerce quality [8], such as reliability, efficiency and product availability among others. But at the same time, data about products, transactions, and shopping cart, must be stored and processed consistently.

It is observed from the above discussion that high efficiency and availability are important for big data systems, but consistency must be dealt with properly. If not, there are serious consequences of inconsistent data as seen in the example of bank data.

3. RELATED WORK AND TECHNOLOGIES

This section illustrates the SQL and NoSQL big data systems. It also explains the key-value NoSQL big data system, Riak, which is used in this study. Further it reviews existing work on transactional services in NoSQL big data systems.

3.1 NoSQL databases

Relational databases enforce relationship between data tables (relations) and support ACID (Atomicity, Consistency, Isolation, Durability) properties that guarantee strong consistency of data and concurrency of transactions. SQL has been widely used in relational databases. But with the emergence of new technologies such as service-oriented computing, cloud, fog or edge computing, there has been significance increase in the amount of data which is generated through applications or services such as online social media, web searching/browsing, customers reviews, road traffic and weather data. In such applications it is difficult to maintain strong consistency (as in relational databases) as well as availability and scalability. According to the CAP theorem [9], consistency, availability and partition tolerance cannot be guaranteed simultaneously. This has led to a new trend in databases, named as NoSQL databases (as described above).

NoSQL databases process large volume of data and generate results in real time such as analysis of millions of tweets or processing of live road traffic data. Therefore, such applications demand high efficiency, response time, scalability and availability. Different NoSQL databases follow different data models and provide different levels of consistency, availability and efficiency. The most common NoSQL databases models are Document databases, Key-value databases, Column store and Graph databases [10]. These models are implemented in different NoSQL databases such as ongoDB, Cassandra, MemcacheDB, Neo4J, Riak and so on.

Comparing the different NoSQL databases is beyond the scope of this paper. In this paper, we use Riak which is one of the most widely used NoSQL big data systems. Riak works as a cluster which is composed of multiple physical nodes. Each node is logically divided into virtual nodes. Efficiency and availability are achieved through data partitioning and replication. Each data pair is replicated at 'N' virtual nodes, which are located in distinct physical nodes. Moreover, key/value pairs are grouped into a namespace called "bucket". This is to allow storing different pairs with the same key but in different buckets. Buckets are grouped in another namespace named "bucket type", where a set of system behavioural properties could be established. For instance, properties like the number of replicas (N) and the level of consistency/availability could be initialized at the bucket type, in order to determine when a read ("r") or a write ("w") operation will be considered successful or not. In general, consistency is maintained by a quorum technique and a decentralized replica synchronization protocol. The system will provide stronger consistency if $(r + w > N)$ than if $(r + w \leq N)$.

3.2 Transactional services for NoSQL databases

NoSQL systems provide availability, scalability and efficiency in a completely different way in comparison with the SQL databases. The key-value data models do not adopt strict relationships between data entities as relational databases do. Moreover, the NoSQL query language is simplified to Get/Put operations. ACID transactions are not guaranteed as NoSQL databases prioritize efficiency and availability over consistency. Examples of such NoSQL databases are BigTable [9], Facebook Cassandra [11], and Windows Azure [12].

Different approaches have been proposed to implement transactions in NoSQL systems, in order to provide different levels of consistency — which can be implemented at three different layers such as data store, middleware and client side. Systems such as Spanner [13] or COPS [14] have been developed to support transactions at the data store level. But this method may compromise on scalability and availability. Middleware approaches include Google Megastore [15], CloudTPS [16], or CumuloNimbo [17]. Such approaches implement transactional services at the middleware level which is an interface between clients and a database. It means that concurrency control and ACID properties are managed at the middleware level. Finally, in the client layer approach, API's are developed that send and receive metadata from client's applications. Examples include, Percolator [18] and ReTSO [19].

Existing research proposes techniques to address the issue of consistency in NoSQL systems but they do not provide evaluation of consistency with respect to availability and efficiency.

4. THE PROPOSED APPROACH

This section presents the proposed approach for evaluating the SQL and NoSQL big data systems which are commonly used in cloud, fog and edge computing.

4.1 Online shopping cart – case study

We first explain the problem of ACE properties of SQL and NoSQL big data systems using a case study of an online purchase order of a shopping cart. We then use this case study to evaluate the effects of consistency (and inconsistency) on the efficiency and availability of data.

As described above, we use the TPC benchmark, TPC-DS [2] in the evaluation. It provides a business model of a retail company

— i.e., recording client purchases, modifying prices according to promotions, maintaining customer profiles are some of the examples of business processes. In this paper, our study is focused on part of the recording client purchases process.

Specifically, we explore the problem of how concurrently issued purchase orders (or transactions) can affect availability, efficiency and consistency. If multiple (purchase) orders or transactions are concurrently updating the data (e.g., buying same product or unit) then the chances of data inconsistency are also increased. During an update operation data may remain inconsistent for a certain period of time. The inter-arrival time since the update operation has started until it has finished and recorded updated data in the database is referred to as "inconsistency window" [20]. This can happen due to several reasons such as network communication delay, number of replicas or a system load.

The issue of inconsistency arises when two or more clients concurrently order the same product/unit and their orders overlap during a purchase process. The issue is illustrated through an example in the Fig. 3. At time t1 and t2 the number of available units (#units) in the database for a specific product ("id_product") is the same for both client 1 and client 2. Clients place orders independently of each other. The number of available items (in the database) at the time of the clients's orders, should be: #units_client 1 ≤ #units or #units_client 2 ≤ #units. That is, the number of units should be equal to or greater than the units specified in client's order 1 or client order 2.

If client 1 order is completed first, then the number of available items will decrease. Thus it is possible that client 2 may not get sufficient units if (#units_client 2 > #units) after the purchase of client 1.

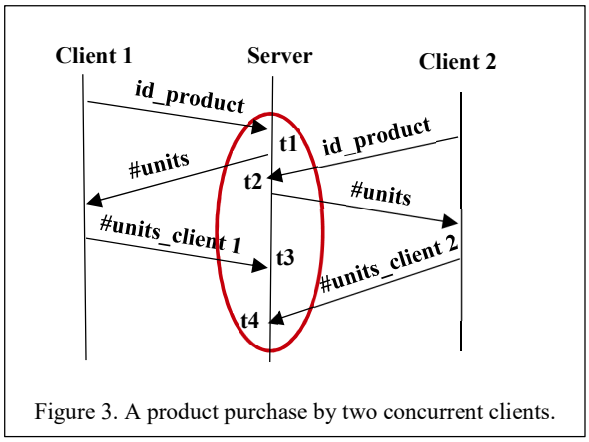

Figure 3. A product purchase by two concurrent clients.

Therefore, it is necessary to process client requests so that data is consistent while maintaining appropriate level of efficiency and availability.

4.2 The process model of an online shopping cart

This section explains the process model of an online shopping cart. We use the online shopping cart case study as it is more appropriate to evaluate the ACE properties of the SQL and NoSQL big data systems. For instance, maintaining consistency is crucial in online shopping cart as clients' orders and product records must be consistently processed and stored in databases. Efficiency is also important as clients requests need to be processed quickly. Availability is important too as sellers want their products to be available to many clients.

The aim of describing the process model is to study the behaviour of an online purchase system which simultaneously manages purchase requests from different clients. When a client makes a purchase request for a specific product the system will display the number of units (or products) available in the database. If there are no units available, the client's request will be declined, otherwise the system proceeds with the purchase request. The system must manages a client's request and keeps consistency of the database.

The purchase process used in our study is illustrated in the Fig. 4. Note that there exist various models of implementing the purchase process in online shopping carts. However, comparing different online shopping carts is beyond the scope of this research.

In Fig. 4, the flow of different steps is numbered in order to provide a better understanding of how the system carries out the purchase process.

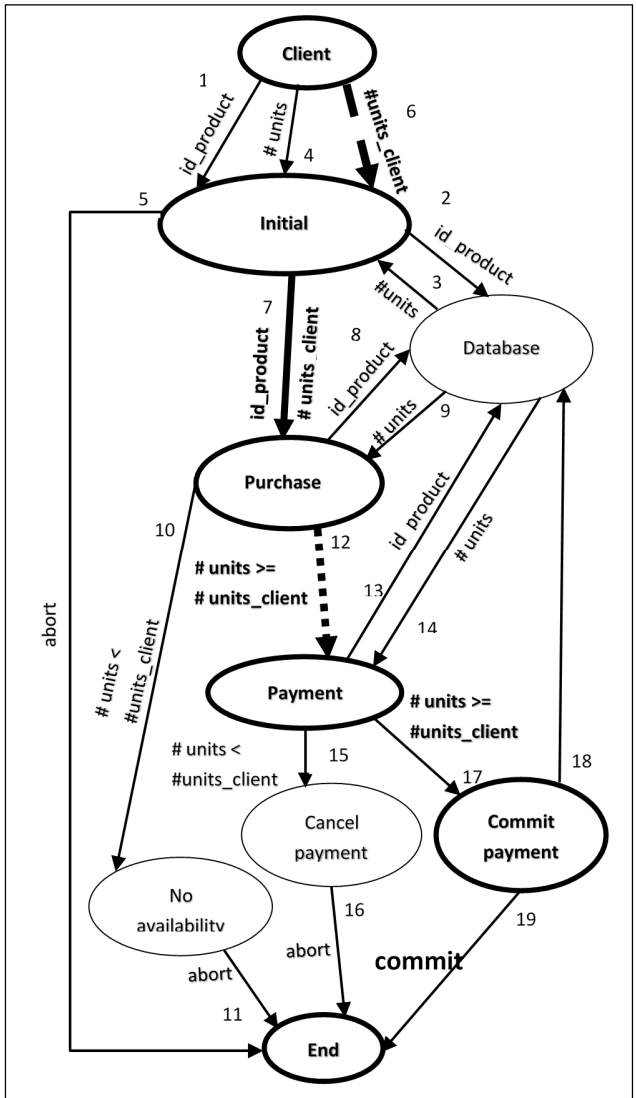

Figure 4. An online shopping cart purchase process model.

The client's purchase process comprises the following three main phases:

1. **Initial phase**: the system receives a client's request to purchase a product (1). It checks the number of available units (2, 3), and sends the information to the client (4). If there are

no units, then the system discards client's request (5), otherwise the client should choose number of units which are to be purchased (6), and the purchase will start.

2. **Purchase phase**: the system receives the number of units chosen by the client (7). However, it has to be taken into account that other clients coexist due to the concurrent nature of the purchase system. Therefore, any client is highly likely to make decisions based on a stale version of a product in their initial phase. For this reason, it is necessary to check if there are still enough number of units available in the database that fulfil client's needs (8, 9). If there are not enough units available, then the client's request is declined (10, 11), otherwise the system proceeds with the purchase order.

3. **Payment phase**: this is the most critical point during the purchase procedure as during this phase client's decision will be finalized (and recorded) in the database (commit or abort). This implies that a certain concurrency control technique [21] should be implemented in order to consistently manage data updates in the database. At this stage, the system definitely confirms the availability of units (13, 14). Then, if the check-out is successful the purchase will commit (17, 18, 19), otherwise it will abort (15, 16). The goal is to avoid or reduce the number of database inconsistencies and payment conflicts, especially in a database system that lacks appropriate concurrency control technique.

According to the purchase process model, every client's request starts at the "Initial" phase. But it is possible that all clients may not be able to finish their purchases successfully due to the level of competitiveness between them – that is, system allows multiple clients at the same time to enable high availability of data (or products). It is possible that some purchase orders are discarded at the beginning of the process as they cannot get sufficient units (i..e, #units_client > #units). Other purchase requests (where #units_client ≤ #units) will pass to the "Purchase" phase. At the end, only when a client's request has enough available units along the three phases ("Initial", "Purchase", "Payment") then his/her purchase will complete successfully. Otherwise the client's request will be cancelled at the specific phase of the purchase process.

4.3 Evaluation and testing

The purchase process model is implemented using Java, SQL and NoSQL big data system. The goal is to analyse how a purchase system works with two different data systems, a traditional SQL database and a NoSQL big data system. We use MySQL and Riak for traditional SQL and NoSQL big data system respectively. Specifically, we analyse how concurrent read-write operations affect the ACE properties. With MySQL we use locks and no locks in the experiments. Riak does not use locks and it lacks appropriate concurrency control mechanisms.

Several experiments have been carried out in order to simulate concurrent client's requests (or transactions) with different degrees of concurrency. The number of requests (transactions) per experiment varies exponentially from 1 to 2048 (2, 4, 8, 16, …, 2048), so the order of magnitude in the input size changes. Therefore, an experiment with an input size 'x' ($1 \leq x \leq 2048$) means that the system is trying to perform 'x' transactions concurrently. Note that from 4096 transactions and onward, the system could not cope with high number of requests due to constraints on hardware resources used in the experiments.

Given a specific number of transactions, each experiment has been run 30 times; which comply with recommended sample in statistical analysis. In our study, the percentage of transactions which reach the "Payment" phase is represented in Fig. 5. It is to be noted that there is a big difference in the results between using locks and no locks. In MySQL with locks, most of the transactions have the possibility of reaching the "Payment" phase. But in Riak and MySQL with no locks, the percentage plummet from 32 transactions and onward, i.e. most of the transactions are discarded in the first "Initial" and second "Purchase" phases of the purchase process.

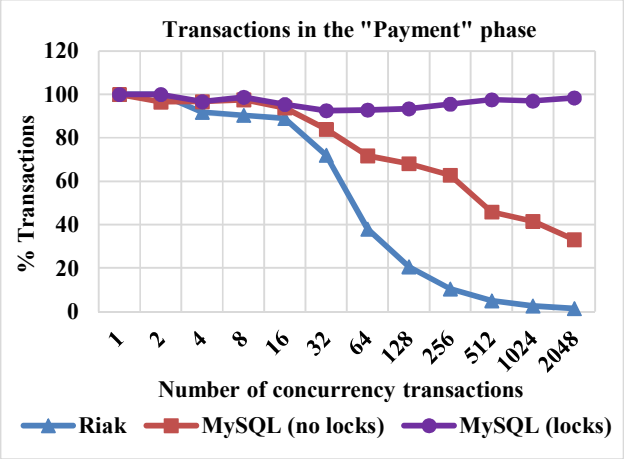

Figure 5. Percentage of transactions in the "Payment" phase.

Moreover, the lack of any lock-based concurrency control mechanism leads to the absence of controlling the conflict between transactions. Thus the speed of the database system plays an important role, i.e. how fast the transactions arrive at the system. In summary, the faster the system is, the higher the level of competitiveness (or conflict).

The goal is to establish a global mean time and a mean percentage of transactions that are able to complete successfully per group of transactions. However, any transaction can be discarded at any of the three phases, so the mean execution time and the percentage of successful transactions are worked out for each specific phase, i.e. not all transactions will reach the "Purchase" or the "Payment" phase. In addition, the relation between the number of transactions that arrive at a specific phase divided by the mean execution time of that phase provides a measure in relation to the number of transactions per millisecond ('tpm') that the purchase system can cope with. For each specific phase, the combination of the measure 'tpm' with the percentage of successful transactions is a plausible way of comparing MySQL and Riak.

The experiments were carried out using the following hardware/software features: a CPU core with 2.4 GHz Intel(R) Core(TM) i7-5500, an operating system Ubuntu 14.04 LST with 64 bits, Eclipse Luna 4.42 as IDE (Integrated Development Environment), Oracle Java 7 as the programming language, a client API supported by MySQL and NoSQL key/value Riak (by Basho) 2.1.1. The simulation run over the SQL store MySQL and the NoSQL key-value Riak with a cluster of five nodes over one CPU.

4.3.1 "Initial" phase

At the beginning of the purchase process, a group of transactions start their execution at the "Initial" phase, but not all will accomplish the next phase for several reasons such as the level of competitiveness (or concurrency) between them and the features of the database management system. The results for this "Initial" phase are shown in the Fig. 6.

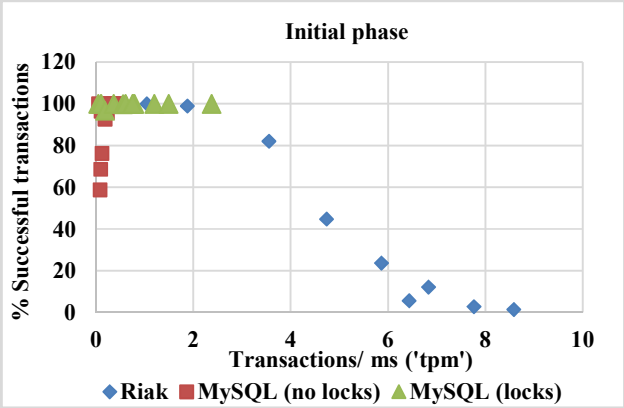

Figure 6. MySQL and Riak performance in the "Initial" phase.

In MySQL (with no locks) the percentage of successful transactions is always above 58% where the 'tpm' varies from 0.05 to 0.5. At the beginning, the 'tpm' increases from 0.05 to 0.5 as the number of concurrent transactions increases from 1 to 16, but with 32 and onward the system starts decreasing the 'tpm' rate, and it falls from 0.3 to 0.08, and so does the percentage of successful transactions. In MySQL with locks, the 'tpm' rate increases from 0.4 to 2.4 as the number of concurrent transactions increases and with a 100% of successful transactions. On the contrary, Riak is able to manage higher 'tpm' rates than MySQL in most concurrent transactions. It varies from 0.2 to 8.6.

MySQL with no locks reaches its maximum 'tpm' value (0.5) for 16 concurrent transactions. MySQL with locks handles 2048 with 2.4 'tpm'. However, the 'tpm' in Riak is higher with a value between 1.8 and 3.5 for 16 and 32, and above 4.7 'tpm' for more than 32 concurrent transactions.

According to the results, the level of competition for a specific number of units of the same product is higher in Riak than in MySQL. As a consequence, it has a direct impact on the percentage of successful transactions, especially when the number of concurrent transactions is above 64, then the percentage of successful transactions plummets from 45% to 1%.

4.3.2 "Purchase" phase

Some transactions have already finished (discarded) in the "Initial" phase. Indeed, depending on the database management system, the number of transactions which achieve the "Purchase" phase is different as explained in the previous section. The results of second phase are shown in Fig. 7.

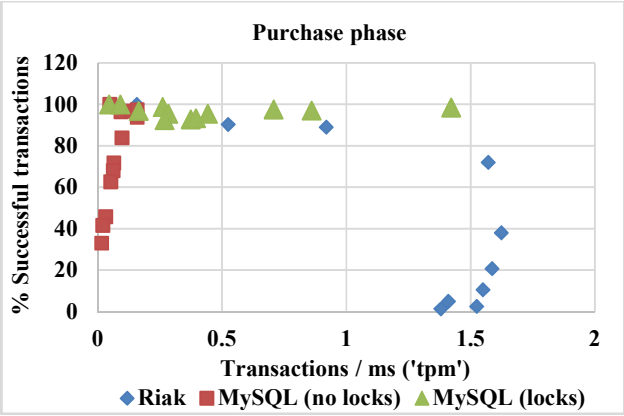

Figure 7. MySQL and Riak performance in the "Purchase" phase.

In Riak, the percentage of transactions which achieve the "Payment" phase falls drastically from 82% to 1% (from 32 concurrent transactions and onwards). This fact implies a decrease in the 'tpm' in the "Purchase" phase. For example, the 'tpm' in the "Initial" phase for 32 concurrent transactions is 3.5 and in the "Purchase" phase it is 1.6. This is due to the fact that the number of concurrent transactions which reaches the second phase is lower, i.e. 26 out of 32 (82%). This results in an enormous decrease in the number of 'tpm' which moves down from 1.5 to 1.3 in comparison with the "Initial" phase which is around 8 'tpm'. In MySQL with no locks, as the 'tpm' increases from 0.01 to 0.16 the percentage of transactions which pass to the next phase moves from 30% to 100%. Finally, in MySQL with locks the percentage of successful transactions is mainly always above 95% with a 'tpm' from 0.04 to 1.4.

In summary, MySQL with locks enable a high percentage of transactions which reach the final stage (Payment), than MySQL with no locks and Riak. Though Riak is faster than both versions of MySQL (with and with no locks), it results in a drastic fall in the number of successful transactions — above 64 concurrent transactions, where the 'tpm' decreases from 1.6 to 1.4 due to the high level of competition.

4.3.3 "Payment" phase

Finally, the "Payment" phase will determine the number of transactions which are able to commit or abort at the end of the purchase process, as shown in Fig. 8.

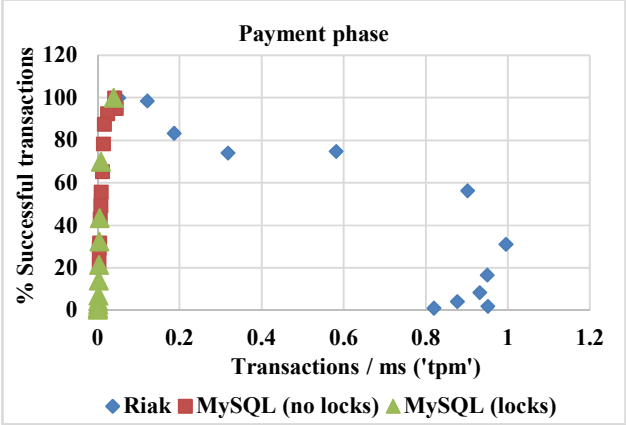

Figure 8. MySQL and Riak performance in the "Purchase" phase.

In MySQL the 'tpm' is below 0.04 in both cases (with and without locks), and the percentage of transactions which achieve this phase is neither high. In MySQL, the number of successful transactions is lower than 50% with a 'tpm' below 0.005 (with locks) and 0.007 (with no locks). On the contrary, Riak is faster than MySQL, where the percentage of successful transactions is below 50% with a 'tpm' between 0.8 and 1. Moreover, Riak provides better results than MySQL with locks when 'tpm' is lower than 0.6 — with more than 70% of successful transactions. This is because not many transactions are competing, otherwise the percentage will drop when the 'tpm' increases from 0.8 to 1 'tpm', i.e. more clients in the system. MySQL with no locks achieves a higher percentage of successful transactions than both MySQL with locks and Riak, with a 'tpm' below 0.04 and more than 2 concurrent transactions.

4.3.4 *Consistency versus availability*

Since product data is not locked in Riak and MySQL with no locks, it cannot be guaranteed that a purchase request ends up in a consistent state at the "Payment" phase. This fact is highly likely to cause an inconsistent database state. Our approach therefore makes a distinction between transactions which conflict and those which do not. The detection of conflicts is carried out with the read-write conflict rule [22]. Two transactions t1 with (ts1, tc1) and t2 with (ts2, tc2) may lead to consistency breach in the database when their execution time overlap, i.e. (ts1 < tc2) or (ts2 < tc1), and when both are allowed to commit. In our approach, if some transactions are conflicting but there are enough available units (or products), then they are allowed to commit, otherwise they will abort. We therefore differentiate between the following four cases during the "Payment" phase:

a) Availability (A) and Consistency (C): the number of units in the database is sufficient to fulfil one transaction's needs (client's request), and it does not conflict with others.
b) Availability (A) and No Consistency (NC): the number of units in the database is sufficient to fulfil one transaction's needs, but it conflicts with others.
c) No Availability (NA) and Consistency (C): the number of units in the database is not sufficient to fulfil one transaction's needs, and it does not conflict with others.
d) No Availability (NA) and No Consistency (NC): the number of units in the database is not sufficient to fulfil one transaction's needs, and it conflicts with others.

Based on the above cases, different experiments were conducted. The results are explained as follow. In the first case "Availability and Consistency" (AC), Riak is faster than MySQL and all transactions conflict with each other, i.e. there is a complete lack of consistency as it can be observed in the Fig. 9. In MySQL (with locks) there is a complete avoidance of conflicts, so the first transactions to lock the system will be the one to purchase the number of units it requires. Although, in MySQL (with no locks) more transactions can make purchases in comparison to MySQL (with locks), and in Riak, the data in the database is highly likely to end up in an inconsistent state, i.e. negative values. This is due to the lack of an appropriate concurrency control at the beginning and at the end of the "Payment" phase.

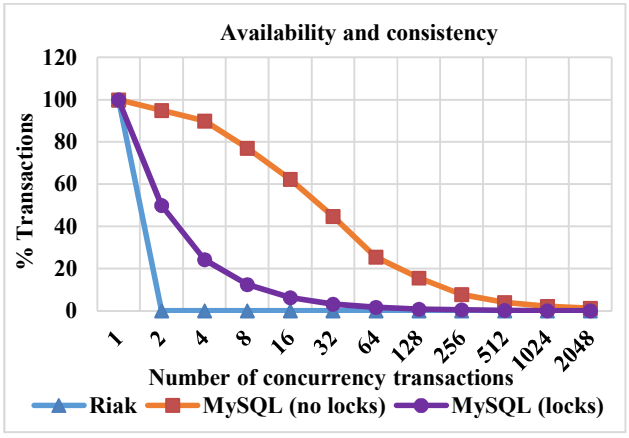

Figure 9. AC in MySQL and Riak in the "Payment" phase.

Figure 10 shows the results of the "Availability and No Consistency" (ANC) case. In it, when the number of concurrent transactions is very low, there is a high likelihood that their arrival happens at once. They arrive extremely close to each other, i.e. there is no difference at all, so the system might set the same start time to some of them. Moreover, if there is a lack of concurrency control their commit time can be close to each other too. Therefore, the percentage of transactions which clashes increases. Indeed, in MySQL with no locks and Riak the percentage is bigger than in MySQL with locks. On the contrary, in MySQL with locks the percentage is lower because the implementation of locks imposes an order of execution on the transactions. That is, a transaction has to wait until another one is completed. Thus the commit time between the transactions, which are allowed to commit, are not so close to each other.

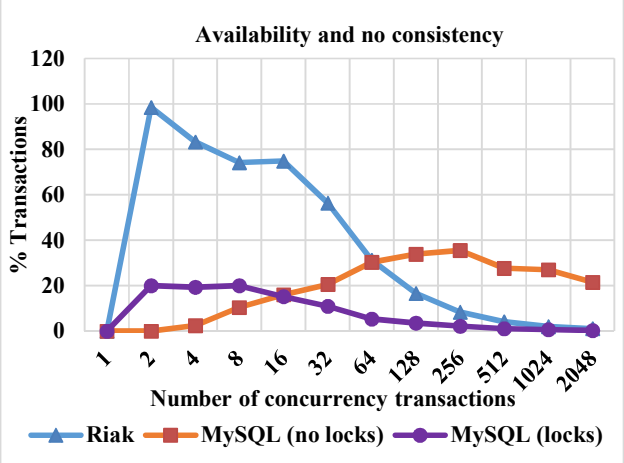

Figure 10. ANC in MySQL and Riak in the "Payment" phase.

The last two cases "No Availability and Consistency" (NAC) and "No Availability and No Consistency" (NANC) are analyzed together. The results are illustrated in the Fig. 11. It can be observed that in MySQL with locks a high percentage of transactions do not commit because there were not enough available units in the database. Indeed, only those transactions that get the lock first, are able to satisfy their needs and get the required number of units. In MySQL with no locks and in Riak the percentage of transactions that fail due to the shortage of units in the database is very low, because the number of transactions which reach this phase is also low, as it is explained in the Section 4.3.

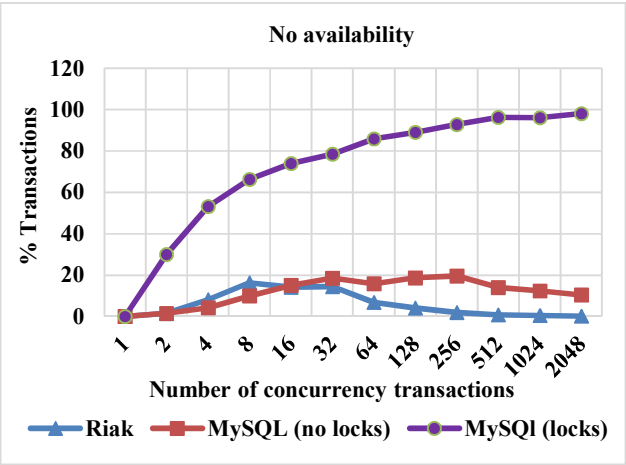

Figure 11. No availability in MySQL and Riak in the "Payment" phase.

5. CONCLUSIONS

This paper studied the three main ACE properties, availability, consistency and efficiency (or performance) of the traditional SQL and NoSQL data systems which are used in cloud, fog and edge computing for storing and processing data. In it, we developed a new system using real case study of an online shopping cart and the industry standard benchmark of the TPC-DS in our experiments. We also used the widely used MySQL (traditional database system) and Riak (NoSQL big data system) in the design and implementation of the proposed system and experimentation.

Our work is first that studied the ACE properties of SQL and NoSQL big data systems. It provided greater insights into the strengths and weaknesses of both SQL and NoSQL big data systems. Our extensive experimentation produced various interesting results which show that MySQL with locks provide better consistency. Thus, it is more appropriate for applications that need strong consistency such as online shopping or banking. However, in terms of efficiency and availability Riak outweighs MySQL. But Riak does not ensure strong consistency. Thus, in its current form Riak is not sustainable to be used for applications (such as online shopping or banking). Our recommendation is that, in order for Riak, to be used in such applications it needs to support appropriate concurrency control and transaction management mechanisms.

6. REFERENCES

[1] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*, 2015, pp. 37-42.

[2] TPC. (2001-2018). Available: http://www.tpc.org/tpcds/

[3] M.T. González-Aparicio, M. Younas, J. Tuya, and R. Casado, "Testing of transactional services in NoSQL key-value databases," *Future Generation Computer Systems,* vol. 80, pp. 384-399, 2018.

[4] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications,* vol. 19, pp. 171-209, 2014.

[5] M. Cecowski, S. Becker, and S. Lehrig, "Cloud computing applications," in *Engineering Scalable, Elastic, and Cost-Efficient Cloud* Computing *Applications*, ed: Springer, 2017, pp. 47-60.

[6] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[7] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information Systems,* vol. 47, pp. 98-115, 2015.

[8] L. Jiang, M. Jun, and Z. Yang, "Customer-perceived value and loyalty: how do key service quality dimensions matter in the context of B2C e-commerce?," *Service Business,* vol. 10, pp. 301-317, 2016.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.,* vol. 26, pp. 1-26, 2008.

[10] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191,* 2013.

[11] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review,* vol. 44, pp. 35-40 2010.

[12] D. G. Campbell, G. Kakivaya, and N. Ellis, "Extreme scale with full SQL language support in microsoft SQL Azure," presented at the Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, Indianapolis, Indiana, USA, 2010.

[13] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's Globally Distributed Database," *ACM Trans. Comput. Syst.,* vol. 31, pp. 1-22, 2013.

[14] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don't settle for eventual: scalable causal consistency for wide-area storage with COPS," presented at the Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, Cascais, Portugal, 2011.

[15] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, J. M. Leon, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services," in *CIDR*, 2011, pp. 223-234.

[16] K. Chitra and B. Jeevarani, "CLOUD TPS: SCALABLE TRANSACTION IN THE CLOUD COMPUTING," *International Journal of Engineering and Computer Science,* vol. 2, pp. 2280-2285, July, 2013.

[17] R. Jimenez-Peris, M. Patino-Martinez, B. Kemme, I. Brondino, J. O. Pereira, R. Vilaça, F. Cruz, R. Oliveira, and M. Y. Ahmad, "CumuloNimbo: A Cloud Scalable Multi-tier SQL Database," *IEEE Data Eng. Bull.,* vol. 38, pp. 73-83, 2015.

[18] D. Peng and F. Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications," in *OSDI*, 2010, pp. 1-15.

[19] F. Junqueira, B. Reed, and M. Yabandeh, "Lock-free transactional support for large-scale storage systems," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, 2011, pp. 176-181.

[20] W. Vogels. (2008). *All things distributed*. Available: https://www.allthingsdistributed.com/2008/12/eventually_consistent.html

[21] C. G. D. J. K. Tim, *Distributed Systems. Concepts and design.*: Addison-Wesley 2001.

[22] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," *SIGMOD Rec.,* vol. 24, pp. 1-10, 1995.