Transaction Consistency in the Cloud: Old Paradigms Revisited

J. E. Armendáriz-Iñigo[1], M. I. Ruiz-Fuertes[2]

[1]Departamento de Ingeniería Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía s/n, 31006 Pamplona, Spain
[2]Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

enrique.armendariz@unavarra.es, miruifue@iti.upv.es

Technical Report TR-ITI-SIDI-2010/004

Transaction Consistency in the Cloud: Old Paradigms Revisited

J. E. Armendáriz-Iñigo[1], M. I. Ruiz-Fuertes[2]

[1]Departamento de Ingeniería Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía s/n, 31006 Pamplona, Spain
[2]Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

Technical Report TR-ITI-SIDI-2010/004

e-mail: enrique.armendariz@unavarra.es, miruifue@iti.upv.es

September, 2010

Abstract

Initial distributed and replicated databases used some mechanisms and paradigms that were slowly replaced by new and more efficient techniques that make cluster-based databases successful. Recently, cloud computing environments appeared, demanding database solutions with higher availability and scalability than those offered by cluster-based systems. In this search for proper solutions, current management techniques are again set aside, and looks are focusing on those initial and previously discarded mechanisms in order to adapt them to the elastic nature of the cloud.

1 Introduction

Middleware-based database replication is a good approach to be developed with web application systems to provide consistent and persistent storage with transactional support. This is a very nice approach to increase system performance and tolerate site failures. Clustered-based solutions have been considered as the proper choice so far, either as primary copy [15, 35] or update everywhere [22, 28] systems using total order broadcast [10], despite their static and scalability issues [19], present even after relaxing to weak consistency [18, 41]. However, database systems are critical elements for business in many companies, having stringent availability requirements and often demanding a high degree of scalability. This is where cloud storage systems come in hand with their promise of high scalability at low cost. These systems, a network of multiple virtual servers, are hosted and maintained by third parties and offer online storage to companies. In terms of scalability, the cloud gives the perception of infinite resources that can be elastically removed or added with little cost in terms of latency. Its low cost stems from the "*pay-as-you-go*" model: you only pay for the resources you are really using and thus it saves investments in huge data centers that cannot be affordable for many companies.

Unfortunately, to the best of our knowledge, cloud storage systems do not provide full transactional support. Besides, it is well known that it is not possible to provide strong consistency *and* good scalability, as stated in the CAP theorem [8]. Therefore, traditional assumptions, such as full replication or interactive transaction support, need to be relaxed. However, this does not suppose a big restriction as these web-based systems mainly use stored procedures [39]. These stored procedures may lead to a proper data partitioning schema and, with the current development of technology, the resulting partitions can perfectly fit in main memory. In the latter, there is no need for concurrency at all: transactions can be executed one after the other as there is no gain for multi-threading transactions [21, 32].

Recently, the database research community has proposed several ways to provide transactional support in the cloud [2, 13, 14, 12, 21, 36, 40] bearing in mind the aforementioned limitations and/or chances. In this paper, we cover the current proposals presented so far and discover that some previously discarded distributed data management techniques, such as the two-phase commit protocol [5], are having a revival [2], along with several mixed combinations of serializable [5], multiversion [5] and snapshot [4] concurrency control approaches providing different isolation levels. This will allow us to say to some extent that no system can be taken as good or bad for granted, as it goodness will depend on the workload; furthermore, due to their nature, there is no standard benchmark for these systems rather than the TPC benchmark [39] and, very recently, the Yahoo! proposal [11]. We will see how cloud storage systems are very elastic and are thought to adapt themselves accordingly. The rest of the paper is organized as follows. Section 2 describes the system model generally assumed in cloud database systems. Section 3 explores different approaches for maintaining consistency in the presence of partitioning. Lastly, Section 4 offers a final discussion and concludes the paper.

2 System Model

In this paper we will assume an abstraction that encompasses most of the systems already proposed in the literature, and use it as a base to compare different approaches. A general system model is depicted in Figure 1, composed of four different layers. On top, the clients containing the application logic use a client library to interact with the system. At the bottom of the stack, a storage system persistently stores data. The real core of the system lies on the two intermediate layers: the Transaction Management layer, responsible for data consistency, and the Replication Management layer, which ensures data availability. These two layers may be closely interconnected, since the replication policy may potentially affect the transaction management [19]. On the other hand, the Replication Management and Storage System layers are merged in those systems [14] relying on a fault-tolerant distributed storage system, such as HDFS [20] or Amazon S3 [6], which internally exploit replication to ensure the availability of the stored data.

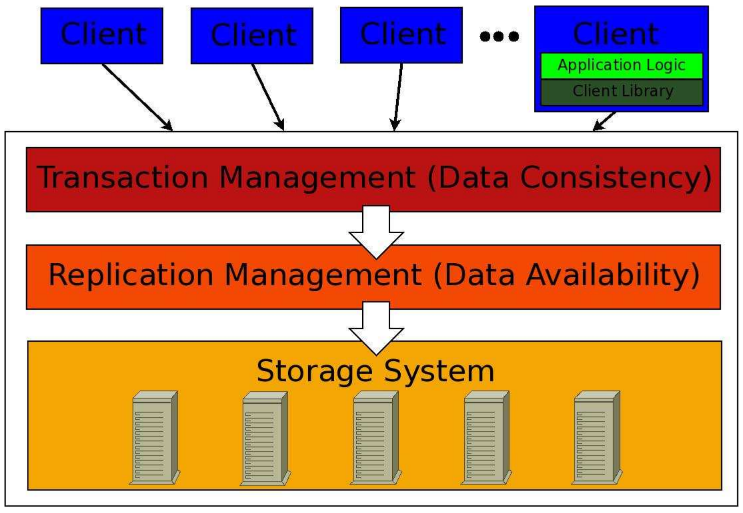
Client
Client
Client
•••
Client
Application Logic
Client Library
Transaction Management (Data Consistency)
Replication Management (Data Availability)
Storage System

Figure 1: An abstraction of a cloud storage system with transactional support

2.1 Transaction Management

Most of the architectures assume that the transaction workload is made of stored procedures. Hence we remove client stalls and reduce the need for concurrency, as shown in main memory databases [21, 32]. By using stored procedures and keeping the database in main memory, both user and disk stalls are avoided. The next step forward to achieve good scalability is database partitioning, where each partition executes transactions independently. The challenge is how to partition the data so that transactions can be entirely executed into a single partition –it is already well known that standard TPC benchmarks, like TPC-C [21, 14, 32] and TPC-E [21], can be split in such a way. It is very often, however, that some transactions need to access several partitions during their execution and then certain coordination is needed for managing them. This type of transactions are called multi-partition transactions. The issue here is twofold: on one hand, network stalls appear as transactions are fragmented [21] (e.g.: imagine that it is needed to write on item x in partition P_1 the value read from item y that belongs to partition P_2); and, on the other hand, some coordination has to be provided to commit a transaction and globally maintain consistency across partitions. With regard to the first issue, some solutions are to speculatively execute transactions (without notifying the clients to avoid cascading aborts) that are ordered after a fragment that is waiting for its commit, as done in [21], or to use transaction flow graphs to determine rendezvous points along the fragments of a multi-partition transaction [32, 12]. In order to commit a multi-partition transaction and thus give a consistent view of data through different partitions, several techniques have been proposed so far: a single coordinator [21, 36]; multiple coordinators [1, 30]; the Paxos algorithm and its variants [26, 27, 31]; the two-phase commit rule and its variants [5, 2, 13, 40, 12]; or through rendezvous protocols [32].

2.2 Replication Management

Data replication is a good way to tolerate node failures and increase the throughput of the system. However, it is well known that traditional replication does not scale well, specially the eager update everywhere technique or the primary copy approach when there are lots of updates [19]. One of the most valued features of the cloud computing environment is its elasticity to achieve scalability. Replicas are added and removed on the fly, so it is needed a well defined policy that decides which partitions have to be replicated (all or certain partitions) and where to replicate a given partition (in all or certain servers).

In general, as it has been roughly stated previously, none of the systems replicate all data in all replicas, they do it up to a given K level [40, 12, 14, 21, 30]; i.e., there exist up to K physical copies of a certain partition. The most common replication technique is the primary backup mechanism using either an optimistic approach (sending the reply back as soon as it is executed in the primary, and updating the secondaries in the background) [40, 2] or a pessimistic approach [21]. Replication can also be done through Paxos [2] or state machine replication [2, 30, 36]. Finally, ElasTraS relies on a fault-tolerant distributed storage to ensure availability of data [13, 14], delegating replication to the lower level of its architecture.

On the subject of which partitions should be replicated and to what level, there are different alternatives. Firstly, we should consider read-only transactions and replicate their associated partitions in all replicas to exploit the benefit of access locality [40]. Otherwise, the replica placement policy follows again different approaches that can be defined through graph analysis [12], histogram analysis of accesses to a given range [40], or autonomously through monitoring the workload until the K level is met.

3 Consistency in the Presence of Partitioning

Traditional cluster-based replication protocols started with serializable databases, being 1-Copy-Serializability (1CS) [5] their correctness criterion: the system behaves as a serializable database with a single logical copy of the set of data items, in spite of existing several physical copies of them. This replicated system was a source of blocks and deadlocks since each individual transaction operation was executed in all replicas, and a two-phase commit protocol was required to ensure that a transaction could be committed. The use of a particular quorum in the system [33], such as the Read-One-Write-All policy [5], along with the advent of Group Communication Systems [10], derived in much more efficient implementations [22, 34]. These replicated systems execute transactions concurrently in the same and different replicas; each read operation is executed in one replica while update operations must be propagated to all replicas at commit time, requiring only a single message per transaction. Parallel to this, the database community developed a new isolation level called Snapshot (SI) [4] that was implemented in commercial and open source databases. This kind of isolation level is slightly weaker than serializable since read operations never block and conflicting updates are serially executed. This meant a great advance in web information systems and was widely used in replicated databases, where the most outstanding solutions were the certification-based replication –an update everywhere protocol with a single message interaction per transaction [22, 28]– and the traditional primary copy approach [15, 35]. All these aspects brought out a new consistency criterion that was called 1SI[1] after its equivalent 1CS for serializable databases [17, 29].

It is well known that, even with the usage of partial replication, database clusters do not scale due to the strong consistency maintained among replicas, which leads to additional network and database stalls [37, 3]. This was already shown in the CAP theorem: if we want to obtain higher scalability, we have to relax the consistency guarantees [8]. The way to efficiently support transactions in the cloud is to reduce to the minimum the interaction among replicas. A common solution is to partition the data: transactions executed in a single partition would not need coordination with other sites.

However, developers go further, consider that the database itself is the main bottleneck for scalability and propose key-value data stores with poorer semantics than traditional SQL statements in order to achieve greater performance. Here we can consider basic systems such as Amazon's Dynamo [16], which only supports simple read and write operations to a data item that is uniquely identified by a key; more complex solutions like Google's Bigtable [9] and Cassandra [25], which enhance the basic model by adding structure to the data and offering range queries and single-row transactions; and ecStore [40], which further enhances previous models by providing transactional semantics across multiple rows.

In some cloud solutions, each replica maintains all its assigned partitions in main memory, therefore avoiding all writing to disk [38]. In these systems, the replication degree needs to be high enough to ensure durability, but disk stalls or the cost of a distributed storage [14] are saved. Other systems [25, 40] rely on in-memory data structures to reduce response time, but periodically dump this information to disk. Main memory can hold both key-value stores [25, 40], to further boost their performance, and SQL databases [38], to enhance them enough to be scalable. Although the theoretical functionality of these two systems is very different, in common OLTP applications there are no ad-hoc queries, and data is accessed through primary keys, equivalent to the keys of a key-value store. Indeed, several *NoSQL* systems [14, 21] have been used to implement TPC-C, and the Yahoo!'s benchmark YCSB is targeted at both key-value stores and main memory databases. As a result, both system types are practically identical solutions for providing transactional behavior over a replicated storage system.

The thread-to-data policy has been shown to be effective through exploiting the regular pattern of data accesses [21, 32]: there exists one thread per partition that sequentially executes operations belonging to different transactions. Together with this, the use of stored procedures avoids any interaction with the user during the execution of a transaction [21, 32, 12]. Single-partition transactions are trivially serialized [12, 21, 40, 13, 14, 2] while multi-partition transactions demand additional mechanisms to ensure serializability. As explained in Section 2.2, this is mostly achieved by the use of 2PC variations [2, 13, 21, 40], through state machine replication [36, 30], or via Paxos[2] [36].

The elastic nature of cloud resources makes the replicated data storage system to almost instantaneously scale up and down according to varying workloads. Ideally, the system will stabilize to execute most of its transactions as single-partition ones. However, there are also backup copies for fault tolerance that can be used as additional replicas to execute transactions. In the case of asynchronous propagation of updates [30, 40], we can have a hierarchy of replicas with different degrees of outdatedness. There is a primary copy that propagates changes to the secondaries and, respectively, the secondaries propagate changes to slave replicas. Other works propose to timestamp the transaction and try to find the appropriate consistency version according to this timestamp, either by quorums [40] or by a direct request to the primary that is subsequently temporarily cached [36]. The existence of different versions of a data item poses a challenging question: why not to take into account 1SI as in traditional database replication? Most of the transactions are read-only and single-partition and so can be serially executed anywhere. However, it is not that cheap to execute 1SI multi-partition transactions without incurring in extra messages [40, 36] that penalize performance. Therefore, it can be assumed that no notion of consistency across data is generally provided for multi-partition transactions, but that data versions are obtained from a valid committed snapshot in each partition. The resulting schedule does not satisfy any of the conditions stated in [29] so we

[1]For the sake of rigorousness we also assume here 1GSI as defined in [17].

[2]Other systems, like ElasTraS [13, 14], use Paxos for replicating metadata exclusively.

obtain a one-copy-multi-version schedule.

In general, a replicated system can give either strong or weak data consistency [18]. The preferred form of weak consistency is eventual consistency [41]: if no new updates are made to an item, eventually all accesses will return the last updated value. However, we have to take into account that augmenting the consistency pays its price on performance and scalability [24].

4 Discussion

Database replication researchers have developed ROWA protocols, either certification-based [22, 28] or primary copy ones [15, 35], since the 2PC protocol does not scale [19]. Furthermore, they initially worked with 2PL databases [5] to be exchanged afterwards for SI [4] databases. All these factors were established for multithreaded concurrent transactions that can potentially span the whole database. On the other hand, we have already mentioned that partitioned databases with relaxed consistency can perform better. Actually, this relaxation is not such, since transactions are known in advance (stored procedures), are mostly single-partitioned, and do not involve any interaction at all with the final user. Under these assumptions, it is easier to provide serializable with a modification of the 2PC rule from how it was originally stated.

The general system model used in cloud solutions is also very similar to that used in federated databases, where multiple and possibly heterogeneous database management systems are coordinated to create the illusion of a logical integration of data. In [7], authors present a federated database that allows the concurrency of both global, system-wide transactions that span over several sites, and local transactions that execute only at one node. This situation is similar to that of single- and multiple-partition transactions in a cloud storage system. Concurrency control mechanisms from [7] avoid deadlocks by using site graphs, where edges connect sites accessed by global transactions. As long as no cycles are introduced in the graph, no deadlocks may arise. This is used to select the site where a read operation will be executed, among the replicas of the required data item. Write operations add all replicas to the graph. If no cycle is formed, the operation is allowed; otherwise the transaction is aborted and retried later. Similar mechanisms can be used for concurrency control and replica placement in the cloud, e.g. authors of [12] follow this technique to heuristically determine the best partitioning and to find the appropriate replication degree of certain partitions, required to avoid scalability problems. Finally, 2PC is used in [7] to commit global transactions, a solution also possible for committing multi-partition transactions, as done in [13, 2, 40, 21].

We are then witnessing a complete revisiting of old paradigms adapted to the elastic nature of the cloud. This reasoning serves us to claim that, in general, there is no perfect system out there. Even though standards like those proposed by TPC [39] or Yahoo! [11] can give a good hint, cloud computing is aimed to be totally dynamic in the sense that it will adapt itself to the best replication technique to provide good performance without sacrificing consistency. Of course, the performance of these systems will depend too on the cloud system where the replicated data storage has been deployed, being this specially true for overload situations and high volumes of data [23].

Acknowledgments

This work has been supported by the Spanish Government under research grant TIN2009-14460-C03, by the Spanish MICINN under grant TIN2010-17193, and by the Spanish MEC under grant BES-2007-17362.

References

[1] A. Adya, R. Gruber, B. Liskov, U. Maheshwari. Efficient optimistic concurrency control using loosely synchronized clocks, in: SIGMOD Conf., ACM, San Jose, CA, USA, 1995, pp. 23–34.

[2] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, C. Karamanolis. Sinfonia: a new paradigm for building scalable distributed systems, ACM Trans. Comput. Syst. 27(3) (2009) 11:1–11:49.

[3] J.E. Armendáriz-Iñigo, J.R. Juárez-Rodríguez, J.R. González de Mendívil, H. Decker, F.D. Muñoz-Escoí. *K*-bound GSI: a flexible database replication protocol, in: Symp. on Applied Comput. (SAC), ACM, 2007, pp. 556–560.

[4] H. Berenson, P.A. Bernstein, J. Gray, J. Melton, E.J. O'Neil, P.E. O'Neil. A critique of ANSI SQL isolation levels, in: SIGMOD Conf., ACM Press, 1995, pp. 1–10.

[5] P.A. Bernstein, V. Hadzilacos, N. Goodman. Concurrency Control and Recovery in Database Systems, Addison Wesley, 1987.

[6] M. Brantner, D. Florescu, D. Graf, D. Kossmann, T. Kraska. Building a database on S3, in: SIGMOD Conf., ACM, Vancouver, BC, Canada, 2008, pp. 251–264.

[7] Y. Breitbart, A. Silberschatz, G.R. Thompson. An Update Mechanism for Multidatabase Systems, IEEE Data Eng. Bull. 10(3) (1987) 12–18.

[8] E.A. Brewer. Towards robust distributed systems (abstract), in: PODC Conf., ACM, Portland, OR, USA, 2000, pp. 7.

[9] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber. Bigtable: a distributed storage system for structured data, ACM Trans. Comput. Syst. 26(2) (2008).

[10] G. Chockler, I. Keidar, R. Vitenberg. Group communication specifications: a comprehensive study, ACM Comput. Surv. 33(4) (2001) 427–469.

[11] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears. Benchmarking cloud serving systems with YCSB, in: SoCC, ACM, Indianapolis, IN, USA, 2010, pp. 143–154.

[12] C. Curino, Y. Zhang, E. Jones, S. Madden. Schism: a workload-driven approach to database replication and partitioning, Proceedings of the VLDB Endowment (PVLDB) 3(1) (2010).

[13] S. Das, D. Agrawal, A. El Abbadi. ElasTraS: an elastic transactional data store in the cloud, in USENIX Workshop on Hot Topics in Cloud Computing, San Diego, CA, USA, 2009.

[14] S. Das, S. Agarwal, D. Agrawal, A. El Abbadi. ElasTraS: an elastic, scalable, and self managing transactional database for the cloud, UCSB Computer Science Technical Report 2010-04, Santa Barbara, CA, USA (2010).

[15] K. Daudjee, K. Salem. Lazy database replication with snapshot isolation, in: 32nd Intnl. Conf. on Very Large Data Bases (VLDB), Seoul, Korea, 2006, pp. 715–726.

[16] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. Dynamo: Amazon's highly available key-value store, in: SOSP, ACM (2007) pp. 205–220.

[17] S. Elnikety, F. Pedone, W. Zwaenepoel. Database replication using generalized snapshot isolation, in: Symp. on Reliable Distrib. Syst. (SRDS), Orlando, FL, USA, IEEE-CS, 2005, pp. 73–84.

[18] A.D. Fekete, K. Ramamritham. Consistency models for replicated data, in: B. Charron-Bost, F. Pedone, A. Schiper (Eds.) Replication. Theory and Practice, Monte Verita, Switzerland, Springer, 2009, LNCS 5959, pp. 1–18.

[19] J. Gray, P. Helland, P.E. O'Neil, D. Shasha. The dangers of replication and a solution, in: H.V. Jagadish, I.S. Mumick (Eds.), SIGMOD Conf., ACM Press, 1996, pp. 173–182.

[20] The Apache Software Foundation, Welcome to Hadoop distributed file system, in: `http://hadoop.apache.org/hdfs/`, 2010.

[21] E.P.C. Jones, D.J. Abadi, S. Madden. Low overhead concurrency control for partitioned main memory databases, in: SIGMOD Conf., ACM, Indianapolis, IN, USA, 2010, pp. 603–614.

[22] B. Kemme, G. Alonso. A new approach to developing and implementing eager database replication protocols, ACM Trans. Database Syst. 25(3) (2000) 333–379.

[23] D. Kossmann, T. Kraska, S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud, in: SIGMOD Conf., ACM, Indianapolis, IN, USA, 2010, pp. 579–590.

[24] T. Kraska, M. Hentschel, G. Alonso, D. Kossmann. Consistency rationing in the cloud: pay only when it matters, Proceedings of the VLDB Endowment (PVLDB) 2(1) (2009) 253–264.

[25] A. Lakshman, P. Malik. Cassandra - A decentralized structured storage system, in: 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, 2009.

[26] L. Lamport. The part-time parliament, ACM Trans. Comput. Syst. 16(2) (1998) 133–169.

[27] L. Lamport. Fast Paxos, Distributed Computing 19(2) (2006) 79–103.

[28] Y. Lin, B. Kemme, M. Patiño-Martínez, R. Jiménez-Peris. Middleware based data replication providing snapshot isolation, in: SIGMOD Conf., ACM, Baltimore, MD, USA, 2005, pp. 419–430.

[29] Y. Lin, B. Kemme, R. Jiménez-Peris, M. Patiño-Martínez, J.E. Armendáriz-Iñigo. Snapshot isolation and integrity constraints in replicated databases, ACM Trans. Database Syst. 34(2) (2009) 11:1–11:49.

[30] F. Maia, J.E. Armendáriz-Iñigo, M.I. Ruiz-Fuertes, R. Oliveira. Scalable transactions in the cloud: partitioning revisited, in: 12th International Symposium on Distributed Objects, Middleware, and Applications (DOA), Springer, Crete, Greece, 2010. *Accepted for publication.*

[31] P.J. Marandi, M. Primi, N. Schiper, F. Pedone. Ring Paxos: a high-throughput atomic broadcast protocol, in: DSN Conf., IEEE-CS Press, Chicago, IL, USA, 2010.

[32] I. Pandis, R. Johnson, N. Hardavellas, A. Ailamaki. Data-oriented transaction execution, Proceedings of the VLDB Endowment (PVLDB) 3(1) (2010).

[33] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, G. Alonso. Are quorums an alternative for data replication?, ACM Trans. Database Syst. 28(3) (2003) 257–294.

[34] F. Pedone, R. Guerraoui, A. Schiper. Transaction reordering in replicated databases, in: Symp. on Reliable Distrib. Syst. (SRDS), 1997, pp. 175–182.

[35] C. Plattner, G. Alonso, M.T. Özsu. Extending DBMSs with satellite databases, VLDB J. 17(4) (2008) 657–682.

[36] D. Sciascia, M. Primi, P.J. Marandi, N. Schiper, F. Pedone. Tapioca: flexible data storage for datacenter applications, University of Lugano, Technical Report 2010/04, Lugano, Switzerland (2010).

[37] D. Serrano, M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation, in: 13th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), IEEE-CS Press, Melbourne, Victoria, Australia, 2007.

[38] M. Stonebraker, S. Madden, D.J. Abadi, S. Harizopoulos, N. Hachem, P. Helland. The end of an architectural era (it's time for a complete rewrite), in: 33rd International Conference on Very Large Data Bases (VLDB), ACM, Vienna, Austria, 2007, pp. 1150–1160.

[39] The Transaction Processing Performance Council, TPC - Homepage, in: `http://www.tpc.org/`, 2010.

[40] H.T. Vo, C. Chen, B.C. Ooi. Towards elastic transactional cloud storage with range query support, Proceedings of the VLDB Endowment (PVLDB) 3(1) (2010).

[41] W. Vogels. Eventually consistent, Commun. ACM 52(1) (2009) 40–44.