Workload Management for Dynamic Partitioning Schemes in Replicated Databases

M. Louis-Rodríguez[1], J. Navarro[2], I. Arrieta-Salinas[1], A. Azqueta-Alzuaz[1], A. Sancho-Asensio[2] and J. E. Armendáriz-Iñigo[1]

[1]*Universidad Pública de Navarra, 31006 Pamplona, Spain*

[2]*La Salle - Ramon Llull University, 08022 Barcelona, Spain*

Keywords: Distributed Databases, Distributed Transactions, Fine-grained Partitioning, Lookup Tables, Cloud Computing.

Abstract: Recent advances on providing transactional support on the cloud rely on keeping databases properly partitioned in order to preserve their beloved high scalability features. However, the dynamic nature of cloud environments often leads to either inefficient partitioning schemes or unbalanced partitions, which prevents the resources from being utilized on an elastic fashion. This paper presents a load balancer that uses offline artificial intelligence techniques to come out with the optimal partitioning design and replication protocol for a cloud database providing transactional support. Performed experiments proof the feasibility of our approach and encourage practitioners to progress on this direction by exploring online and unsupervised machine learning techniques applied to this domain.

1 INTRODUCTION

Cloud-based storage was initially aimed to overcome the scalability limitations of transactional databases (Gray et al., 1996) and meet the ever-growing storage demands of daily software applications. To achieve such commitment, the properties of traditional databases were relaxed until achieving what was coined as the NoSQL paradigm (Stonebraker, 2010).

Thoroughly, novel NoSQL systems resign from the ACID (i.e., atomicity, consistency, isolation, and durability) features provided by classic databases and implement what is known as BASE (i.e., basically available, soft-state, and eventually consistent) properties (Brewer, 2012), which allow them to—ideally—scale up to infinity (Corbett et al., 2012; Chang et al., 2006; DeCandia et al., 2007). In order to meet the BASE principles, these systems often rely on (1) weak consistency models (Vogels, 2009), (2) in-memory key-value structures (Chang et al., 2006), and (3) super light concurrency control and replication protocols, which permit to reduce interdependences between data stored in the repository and thus boost its scalability.

However, the overwhelming amount of critical applications that cannot resign from their transactional nature (e.g., electronic transferences, billing systems) has driven practitioners to provide transactional support built on top of existing cloud repositories (Curino et al., 2011; Levandoski et al., 2011). Actually, these systems pursue the idea of deploying classic database replication protocols (Wiesmann and Schiper, 2005) over a cloud storage infrastructure (Das et al., 2010) and hence offer transactional facilities while inheriting the properties of the cloud. Typically, this is addressed by building a load balancer (Curino et al., 2010; Curino et al., 2011; Levandoski et al., 2011) that broadcasts transactions over a fixed set of partitions that are statically running a given replication protocol. Nevertheless, despite its broad adoption, this approach is not aligned with the cloud philosophy in the sense that it is not able to adapt itself to its intrinsic elastic nature, which paradoxically leads to underused or poorly scalable services.

The purpose of this paper is to propose a load balancer for transactional cloud databases that releases them from the aforementioned static configuration barriers. More specifically, this paper extends the proposal presented in (Arrieta-Salinas et al., 2012) (which describes a cloud database with transactional support) with a load balancing system targeted at (1) monitoring the key performance parameters (e.g., throughput, transactions per second, ac-

Louis-Rodríguez M., Navarro J., Arrieta-Salinas I., Azqueta-Alzuaz A., Sancho-Asensio A. and E. Armendáriz-Iñigo J..
Workload Management for Dynamic Partitioning Schemes in Replicated Databases.
DOI: 10.5220/0004375902730278
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 273-278
ISBN: 978-989-8565-52-5

cess pattern) of the database, (2) inferring the optimal partitioning scheme for the database through machine learning techniques, and (3) proposing the most suitable replication protocol to be run on each partition according to the current user demands.

The remainder of this paper is organized as follows. Section 2 stresses the need for applying a proper partitioning schema and using the appropriate replication protocol on a transactional cloud database. Section 3 presents the proposed system architecture. Section 4 details the experiments performed. Finally, Section 5 concludes the paper.

2 PARTITIONING THE CLOUD

As clarified in (Brewer, 2012), a distributed system may implement a weak form of consistency (Vogels, 2009), a relaxed degree of availability (White, 2012), and reasonable network partition tolerance (DeCandia et al., 2007) while still keeping itself scalable. Actually, most of the existing cloud data repositories behave in this direction (DeCandia et al., 2007; White, 2012). However, those applications that demand strict transactional support are not able to straightforwardly fit in this idea, since they generally require strong consistency to provide correct executions (Birman, 2012) while claiming for the appealing characteristics offered by the cloud, therefore refusing to relax availability constraints. In this context, tolerance to network partitions must be smartly addressed to truly benefit from the cloud features, which suggests practitioners to be very cautious when defining a partitioning scheme. This section (1) stresses the criticalness of managing data partitions, (2) describes the proposed graph-based partitioning technique that has been applied in the presented load balancer and (3) suggests supervised machine learning techniques as an effective way to address this matter.

2.1 Motivation and Related Work

Partitioning is a very effective way to achieve high scalability while leveraging data consistency in a distributed database. Transactions that are executed within one single data partition require no interaction with the rest of partitions, hence reducing the communication overhead (Aguilera et al., 2009; Curino et al., 2010; Das et al., 2010). However, configuring the partitioning scheme to minimize multi-partition transactions while avoiding extra costs derived from resources misusage requires a judicious criterion that must carefully address the workload pattern to determine the optimal data partitions.

However, if we miss the perspective here and do not replicate these partitions, the system will face the single point of failure issue or suffer from performance limitations due to the bottleneck effect. Consequently, besides deciding the most suitable partitioning strategy, it is important to study the workload nature that generated each partition to determine the most appropriate replication protocol. Roughly speaking, in an update-intensive data partition, the ideal candidate will be an update-everywhere replication protocol (Wiesmann and Schiper, 2005); otherwise, the candidate will probably be a primary copy replication protocol (Daudjee and Salem, 2006).

Overall, there is a strong connection between the amount of partitions, the database workload, and the replication protocol running on each partition. The following subsection discusses a graph-based approach used to infer these three features.

2.2 Graph-based Partitioning

We propose a structure based on undirected graphs, which can be used for determining the best partitioning scheme. This proposal will be explained by means of the example depicted in Figure 1, which defines a database consisting of two tables `PERSON` and `DEGREE` and a sample workload of four transactions.

This workload will drive the construction of the undirected graph structure shown in Figure 1. More specifically, each tuple of `PERSON` and `DEGREE` is represented by a node, whereas each edge between two nodes reflects that they are accessed within the same transaction. The weights of edges are increased according to the number of transactions accessing the connecting nodes. In addition, there is a counter associated to each node representing the number of transactions that access it.

The first statement of transaction 1 accesses the second tuple of `PERSON` and the first tuple of `DEGREE`. Hence, in Figure 1 we draw two nodes identified by the `ID` field of each tuple (i.e., node 2 and node 4) and plot an edge between them with an initial weight of one. In addition, we set to one the counters of nodes 2 and 4 to reflect the number of times they have been accessed. The next operation of transaction 1 modifies node 1, thus we connect it with the previous nodes (2 and 4) using edges of weight one and set the weight of node 1 to one. Finally, the last operation of transaction 1 is a `SELECT` that accesses node 3. We have to connect node 3 with nodes 1, 2 and 4 with edges of weight one and increase the counter of node 3 in one unit. Likewise, we proceed in the same way with the rest of transactions. For the sake of clarity, in Figure 1 we have surrounded the nodes accessed by each

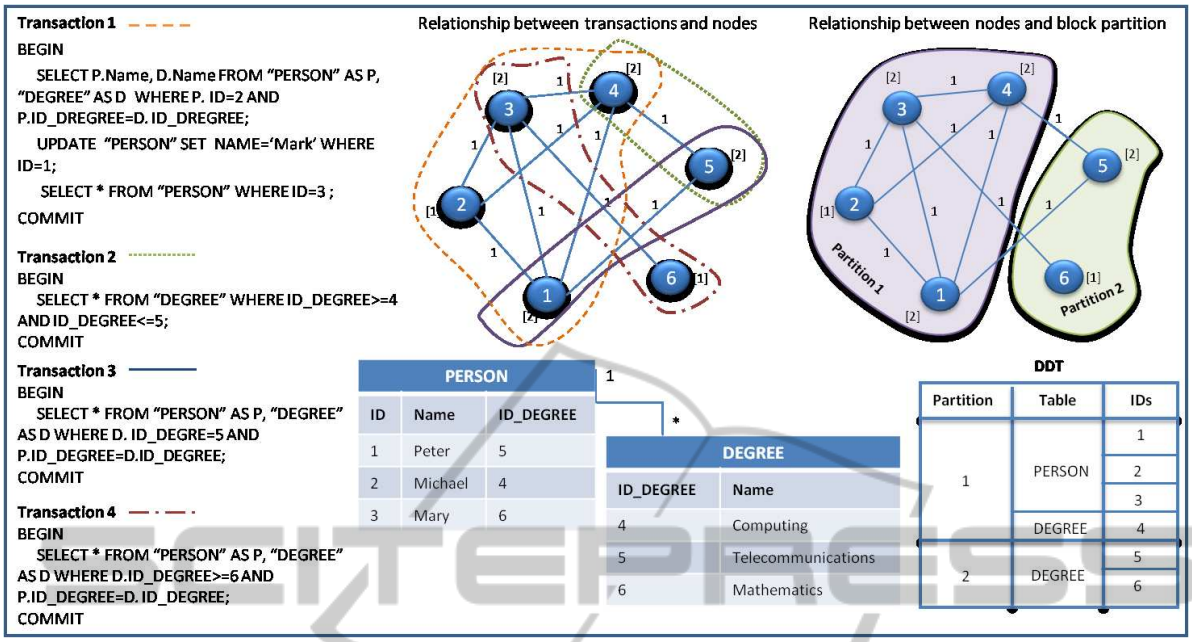


Figure 1: Example of a graph representation and its Distribution Data Table.

transaction with different dotted line formats.

Once all transactions have been executed and the graph has been built, data must be partitioned so that the nodes are distributed according to their relationships within transactions. Thus, we are pursuing a twofold goal: minimizing the number of multi-partition transactions (which implies cutting the edge with the minimum weight) while minimizing the size of each group of nodes, which leads to a NP-Complete problem. However, there exist algorithms that permit to circumvent this in an efficient way like MeTis and its variant hMeTis (Karypis and Kumar, 1998; Karypis, 2011).

In the previous example, the best cut will consist in two partitions: one partition would include nodes 1 to 4, and the other one nodes 5 and 6. This solution represents a cut of three edges, all of them with a weight of one. If we take a look at the executed transactions we can see that most of them access `PERSON`; thus, it makes sense to place these nodes together to minimize multi-partition transactions.

Once the partitioning scheme has been determined, the system will use the information provided by the aforementioned graph to properly forward client requests to their corresponding partition. With the aim of speeding up this task, we have developed a Distribution Data Table (DDT) which behaves like the structure presented in (Tatarowicz et al., 2012). A sample of the DDT is included in Figure 1. This structure, organized as a lookup table, contains a series of record intervals of each database table and the partition number where each interval is stored. The DDT, which is fully stored in main memory, is scanned when a client request arrives. Moreover, it can be partially cached in the client side to avoid continuously requesting the same information.

The approach herein presented analyzes the whole workload prior performing any kind of partition relying on the fact that transactions will tend to have the same access pattern. In order to cope with those situations where this assumption is too ambitious, the following subsection presents a prospective view on this area and discusses some strategies based on machine learning techniques.

2.3 Smart Partitioning on the Cloud

Data mining techniques can be divided into two kinds of families based on the desired outcome of the algorithm: (1) those that assume an *a priori* underlying structure and thus require the use of existing information to obtain its knowledge (referred to as supervised learning) and (2) those that do not assume any underlying structure (referred to as unsupervised learning).

Moreover, the two aforementioned families can be further classified into (1) offline and (2) online methods. Offline algorithms require all data to be analyzed in order to build a comprehensive system model. For instance, in (Curino et al., 2010) a C4.5 decision tree (Quinlan, 1993) is used to build an understandable model of the tuple dependencies existing on a database partition. On the other hand, online ap-

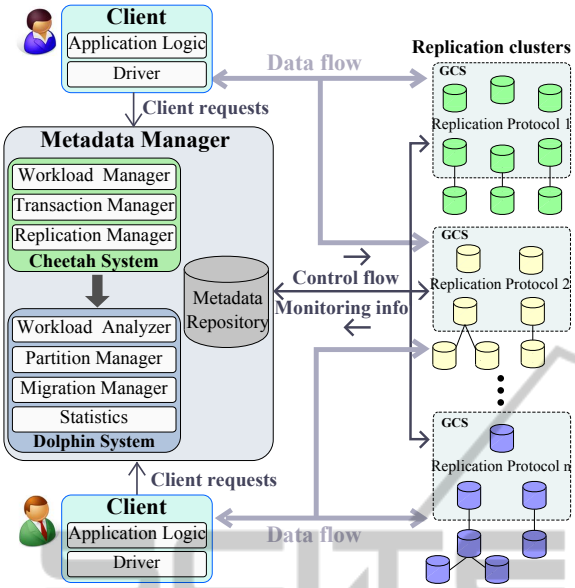


Figure 2: System model.

proaches build a dynamic system model that attempts to adapt itself to the environment specificities. For example, in (Hulten et al., 2001), an online decision tree has been designed to predict the class distribution on a time-changing environment.

Formally speaking, the technique presented in the previous subsection uses an offline approach to establish the best partitioning and replication schema. In fact, the whole workload schema is previously analyzed in order to build the aforesaid graphs and partitions. Although this ensures reaching an optimal solution, its application on some real environments is doubtful in the sense that transactions are rarely known in advance. This issue becomes even more relevant in cloud environments, where loads drastically change according to the elastic user demands.

Offline techniques can be exported to online environments if the variations of the system behavior are unusual. In such situations, a windowing technique can be used to continuously train the system and thus get nearly-online results. However, cloud databases cannot fully benefit from this approach since the characteristics of the workload can vary sharply, Hence, we propose to explore online machine learning techniques in order to come out with the best possible partitioning layout and replication protocol at any time.

3 SYSTEM MODEL

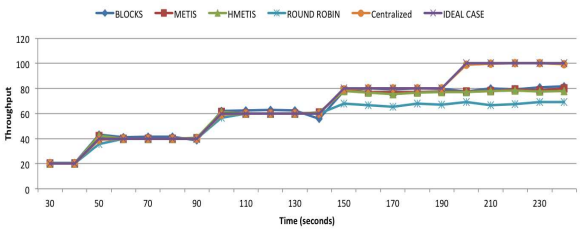
The experiments presented in this paper have been conducted over an extended version of the architecture proposed in (Arrieta-Salinas et al., 2012), which is an alternative approach to (Das et al., 2010; Levandoski et al., 2011; Curino et al., 2011) to provide transactional support on the cloud. As in other cloud systems, the core of our architecture (as shown in Figure 2) is a metadata manager that forwards all transactions executed by client applications and manages the replicas accordingly (Arrieta-Salinas et al., 2012). Taking into account (1) the importance of choosing a proper partitioning schema, (2) the significance of selecting a suitable replication protocol in each partition, and (3) the unavoidable existence of multi-partition transactions in cloud environments, already stressed in Section 2, this work extends the metadata manager (Cheetah) presented in (Arrieta-Salinas et al., 2012) and adds a new module coined as Dolphin with the following entities.

The *Workload Analyzer* examines each transaction to identify the accessed tuples by parsing each statement of a transaction through the `WHERE` clause, and generates a log file. The *Partition Manager* uses this log to build the graph that determines the relationships of tuples vs. transactions and creates the respective partitions as explained in Section 2.2. The Partition Manager is also in charge of periodically evaluating the system load information provided by the Statistics module to define the number of partitions and the amount of replicas per partition. Apart from this, the Partition Manager chooses the replication protocol for each data partition depending on the predominant type of operations (reads or updates) as pointed out in Section 2.1 and selects the number of hierarchy levels by inspecting the timeliness characteristics of queries and the total number of replicas. The *Migration Manager* distributes partitions across replicas by pointing out their position in the hierarchy level and, if a replica is at the core, which replication protocol must be run. Finally, the *Statistics Module* collects all kind of system information to study the performance in terms of scalability, handled TPS, number of replicas per partition, monetary costs, etc.

4 EXPERIMENTAL EVALUATION

We have built a prototype (using Java 1.6) that covers the basic functionality of all system components. We have also developed an implementation of a JDBC driver that has allowed us to run a popular set of benchmarks named *OLTPBenchmark* (Curino et al., 2012), in order to assess the performance of the developed prototype. In particular, we have used the OLTPBenchmark implementation of the Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al., 2010), which defines a single table of records com-

(a) Workload A.



(b) Workload B.

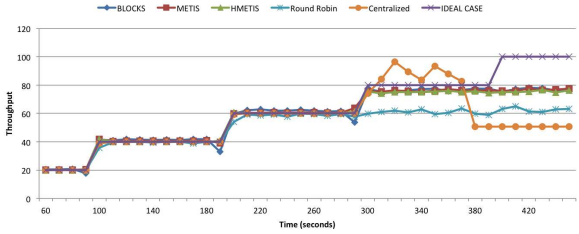


Figure 3: Maximum throughput depending on the scan workload.

posed by a primary key and ten text fields. In the experiments performed, this table has been filled with 500000 records for a total size of 500MB.

Our testing configuration consists of six computers in a 100 Mbps switched LAN, where each machine is equipped with an Intel Core 2 Duo processor at 2.13 GHz, 2 GB of RAM and a 250 GB hard disk. All machines run the Linux distribution OpenSuse v11.2 (kernel version 2.6.22.31.8-01), with a Java Virtual Machine 1.6.0 executing the application code. An additional computer with the same configuration as that of the replicas is used for running both clients and the metadata manager. For the sake of simplicity, instead of developing a distributed implementation of the metadata manager to provide fault tolerance and scalability, we have developed a centralized component. All the data stored at the metadata manager is kept in main memory. Moreover, each machine used as a replica holds a local PostgreSQL 8.4.7 database, whose configuration options have been tuned so that it behaves as an in-memory only database (i.e., it acts as a cache, without storing the database on disk). Spread 4.0.0 (Stanton, 2005) has been used as Group Communication System, whereas point-to-point communications have been implemented using TCP.

In the experiments, we have tested two graph-based heuristic algorithms to determine the partitioning schema, MeTis and hMeTis (Karypis, 2011; Karypis and Kumar, 1998). Both techniques try to find the optimal solution focusing on finding partitions by cutting as few edges as possible. The difference between them is that hMeTis is an optimization of MeTis that uses hypergraphs and performs more iterations in less time. Moreover, we have compared these two heuristic algorithms with two other approaches: Blocks and Round Robin. The former tries to split the database into uniform blocks according to the available capacity of the nodes, whereas the latter splits each consecutive record in a different partition across all nodes, so that there are no two consecutive rows in the same partition. For all the partitioning algorithms, we have performed an offline execution to determine the partitioning schema and the replication protocols to be run on the core layer of each partition. Besides, the algorithm reports the number of replicas per partition and their location along the replication hierarchy tree.

Due to the early stage of the development, we have been able to run only read-only transactions. From the set of transactions defined in the YCSB, we have chosen *scan* transactions, which read the set of records whose keys belong to a given interval. As we are only dealing with read-only transactions, all partitioning schemes define a primary copy replication protocol. The experiments have been performed using two workload types: i) *workload A*, which performs a scan over at most 10 records following a Zipfian distribution and ii) *workload B*, which is the same as workload A but scans at most 100 records.

The first parameter we have measured is the number of multi-partition transactions, which should be kept as low as possible in order to optimize system performance. We have obtained the following amount of multi-partition transactions for *workload A*: 1 with Blocks, 112 with MeTis, 1451 with hMeTis, and 7271 with Round Robin. For *workload B* we have obtained the following amount of multi-partition transactions: 1 with Blocks, 3360 with MeTis, 2182 with hMeTis, and 20339 with Round Robin. It is straightforward that the round robin algorithm shows the worst behavior due to the nature of the transactions performed, as we must travel across several partition to perform a scan. There is not such a big difference between MeTis and hMeTis; however, we can see that with workload B the hMeTis technique gives a better performance. Nonetheless, the Blocks approach outperforms them all, since with this configuration the probability of a scan accessing two partitions is rare.

Furthermore, we have analyzed the system throughput and added an ideal case and a centralized solution. Figure 3 shows that the performance is more or less the same for all the algorithms but the Round Robin. We have also noticed the fast degradation of system throughput (it does not reach 100 TPS even in the centralized case), due to the bottleneck caused by having a centralized metadata manager. A possible

solution would be to implement a distributed metadata manager using a a Paxos-like protocol (Lamport, 1998). Apart from this, holding the complete structure that represents the object interdependencies that determine the best partitioning policy is very expensive in terms of memory usage; thus, the main memory at the metadata manager becomes easily saturated. This problem could be alleviated by applying aging or windowing strategies to prune such structure.

5 CONCLUSIONS

This paper presents a load balancer that uses artificial intelligence techniques to obtain the optimal partitioning design for a cloud database with transactional support. The experiments performed have empirically verified that data partitioning can be seen as a multi-objective optimization problem, since it is targeted to come out with the maximum number of partitions with the minimum number of multi-partition transactions. Furthermore, the proposed architecture allows to heuristically define the most suitable replication protocol running on each partition according to the system workload.

In addition, we have explored the feasibility of further improving the obtained results by means of online data mining techniques, which would allow the system to automatically discover new workload patterns and therefore optimize the partitioning schema according to dynamic user demands.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the Spanish National Science Foundation (MINECO) (grant TIN2012-37719-C03-03) and from *Generalitat de Catalunya* for its support under grant 2012FI_B 01058 for Andreu Sancho-Asensio.

REFERENCES

Aguilera, M. K., et al. (2009). Sinfonia: A new paradigm for building scalable distributed systems. *ACM Trans. Comput. Syst.*, 27(3).

Arrieta-Salinas, I., et al. (2012). Classic replication techniques on the cloud. In *ARES 2012*, pages 268–273.

Birman, K. P. (2012). *Guide to Reliable Distributed Systems–Building High-Assurance Applications and Cloud-Hosted Services*. Texts in computer science. Springer.

Brewer, E. A. (2012). Pushing the CAP: Strategies for consistency and availability. *IEEE Comput.*, 45(2):23–29.

Chang, F., et al. (2006). Bigtable: A distributed storage system for structured data. In *OSDI 2006*, pages 205–218.

Cooper, B.F., et al. (2010). Benchmarking cloud serving systems with YCSB. In *SoCC 2010*, pages 143–154.

Corbett, J.C., et al. (2012). Spanner: Google's globally-distributed database. In *OSDI 2012*, pages 251–264.

Curino, C., et al. (2010). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57.

Curino, C., et al. (2011). Relational cloud: A database service for the cloud. In *CIDR 2011*, pages 235–240.

Curino, C., et al. (2012). OLTPBenchmark. Accessible in URL: http://oltpbenchmark.com.

Das, S., et a.l (2010). ElasTraS: An elastic transactional data store in the cloud. *CoRR*, abs/1008.3751.

Daudjee, K. and Salem, K. (2006). Lazy database replication with snapshot isolation. In *VLDB 2006*, pages 715–726.

DeCandia, G., et al. (2007). Dynamo: Amazon's highly available key-value store. In *SOSP 2007*, pages 205–220.

Gray, J., et al. (1996). The dangers of replication and a solution. In *SIGMOD 1996*, pages 173–182.

Hulten, G., et al. (2001). Mining time-changing data streams. In *SIGKDD 2001*, pages 97–106.

Karypis, G. (2011). METIS: A software package for partitioning meshes, and computing fill-reducing orderings of sparse matrices (v.5.0). In: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

Karypis, G. and Kumar, V. (1998). hMeTiS a hypergraph partitioning package (v.1.5.3). In: http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview.

Lamport, L. (1998). The part-time parliament. *ACM Trans. Comput. Syst.*, 16:133-169.

Levandoski, J.J., et al. (2011). Deuteronomy: Transaction support for cloud data. In *CIDR 2011*, pages 123–133.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Stanton, J. R. (2005). *The Spread Toolkit*. Accessible in URL: http://www.spread.org.

Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Commun. ACM*, 53(4):10–11.

Tatarowicz, A.L., et al. (2012). Lookup tables: fine-grained partitioning for distributed databases. In *ICDE 2012*, pages 102 –113.

Vogels, W. (2009). Eventually consistent. *Communications of the ACM*, 52(1):40–44.

White, T. (2012). *Hadoop–The Definitive Guide: Storage and Analysis at Internet Scale (3. ed.)*. O'Reilly.

Wiesmann, M. and Schiper, A. (2005). Comparison of database replication techniques based on total order broadcast. *IIEEE Trans. Knowl. Data Eng.*, 17(4):551–566.