Comparison of Eager and Quorum-based Replication in a Cloud Environment

Alexander Stiemer Ilir Fetai Heiko Schuldt
Department of Mathematics and Computer Science
University of Basel, Switzerland
firstname.lastname@unibas.ch

***Abstract*—Most applications deployed in a Cloud require a high degree of availability. For the data layer, this means that data have to be replicated either within a data center or across Cloud data centers. While replication also allows to increase the performance of applications if data is read as the load can be distributed across replica sites, updates need special coordination among the sites and may have an adverse effect on the overall performance. The actual effects of data replication depend on the replication protocol used. While ROWAA (read-one-write-all-available) prefers read operations, quorum-based replication protocols tend to prefer write operations as not all replica sites need to be updated synchronously. In this paper, we provide a detailed evaluation of ROWAA and quorum-based replication protocols in an amazon AWS Cloud environment on the basis of the TPC-C benchmark and different transaction mixes. The evaluation results for single data center and multi data center environments show that in general the influence of transaction coordination significantly grows with the number of update sites and a growing number of update transactions. However, not all quorum-based protocols are well suited for high update loads as they may create a hot spot that again significantly impacts performance.**

I. Introduction

Data is an essential part of any application deployed in the Cloud [1]. Data replication is a means to increase data availability by hosting the data at different *replica sites*. For read-only transactions, replication can additionally increase the performance as the load can be distributed between the sites. However, there are fundamental trade-offs in the design of distributed, thus replicated databases, that are captured by the CAP-theorem [2], [3]. According to the CAP theorem, any distributed database faces the *C*onsistency, *A*vailability and *P*artition Tolerance tradeoff, and it is only possible to jointly provide two out of these three of these properties – even though the majority of applications would need all three at the same time. Usually, tolerance to network partitions must be considered [4]. For update-transactions in presence of network partitions, this means that availability must be sacrificed if strong consistency is a demand [4], [5]. As a consequence of the CAP trade-offs, new relaxed consistency models such as Causal Consistency [6] or Eventual Consistency [7] have been defined that are compatible with the high availability demands of applications. Relaxed consistency models are the default models in most NoSQL databases [8], [9], [10]. However, with weak consistency models, it is difficult to reason about the execution semantics and the execution guarantees of concurrent applications. This poses a challenge to application developers as they need to compensate for possible inconsistencies at application level [11], [12]. In reaction to that, many NoSQL databases have newly incorporated stronger consistency models and the possibility to specify the desired consistency guarantees at operation level. That allows clients to choose the optimal consistency model and design their applications accordingly [8], [13], [14].

The stronger the consistency model, the higher is the performance penalty for applications. The intuition is that for strong consistency models, such as Strong One-Copy Serializability (1SR) [15][1], 1SR [17], and Snapshot Isolation (SI) [18], [19], [20] the database system has to 'invest' more resources in coordination compared to weaker models such as eventual consistency [4], [6], [21]. In the latter case, it is the application developer who has (if at all) to guarantee consistency. This consistency vs. latency trade-off is prevalent in both centralized and distributed databases and has been well-known in databases such as Oracle, MySQL as they have ever since included consistency models that are weaker than serializability [5]. However, it is especially eminent in distributed databases with geo-replication as in those configurations the network communication becomes a critical factor for the overall latency [5], [22].

In the past decade(s), different protocols have been developed that implement a certain consistency model and that target the optimization of one or more parameters, such as latency and/or throughput [23], [24], [25]. However, while all of them are well suited for a certain application workload, they are at the same time less suited for others.

The goal of this work is to asses the performance of the 1SR protocol defined in [17] using different *replication protocols (RP)* on the basis of the TPC-C benchmark [26]. For this, we have considered the following RPs: read-one-write-all-available (ROWAA) and quorum protocols (e.g., majority-based (MQ) [27] and a tree-based quorum (TQ) [28]). The latter target the reduction of communication overhead by reducing the number of sites involved in the consensus required for providing 1SR consistency.

The contributions of this paper are twofold: First, we provide a thorough analysis of the performance implications

[1]equivalent to linearizability for distributed shared memory [16].

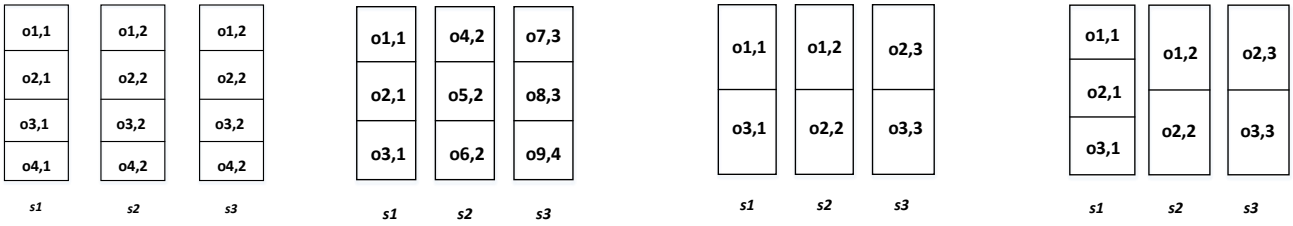


Figure 1: (a) fully replicated, (b) partitioned, (c) pure-partially replicated, and (d) hybrid-partially replicated configurations

of different RPs in a single data center environment for different transaction mixes on the basis of the TPC-C benchmark using the real Cloud infrastructure. Second, we asses the impact of geo-replication to the overall performance of the replication protocols by deploying the replica sites in different data centers of the AWS infrastructure. Both evaluations show how the number of sites needed for read and write operations influence the overall transaction throughput. In short, ROWAA outperforms quorum-based approaches for read-dominated workloads as reads can be executed on a single site. Quorum-based approaches, however, outperform ROWAA if the transaction mix is dominated by update transactions and the quorum construction strategy considers the avoidance of hotspot-sites.

This paper is structured as follows: Section II presents the system model. In Section III introduce replication protocols for 1SR consistency. Section V discusses implementation details of these protocols and provides in detail the results of the evaluation. In Section VI we summarize related work and Section VII concludes.

II. System Model

Typical Cloud environments host applications in a three-tier architecture consisting of a web tier, an application tier in the middle, and a data layer hosting the database system (DBS) which, in turn, consists of the actual database (DB) and the database management system (DBMS). It is important that all these layers guarantee the desired availability and scalability level. This is particularly true for the DBS, as otherwise the guarantees of the entire application may degrade [29], [1], [30], [31], [32]. As a consequence, DBSs are usually distributed and replicated across different data centers [29], denoted as *Distributed DBSs (DDBS)*.

In our system model, we assume flat transactions spawned by the applications and a DDBS consisting of a set of sites $S = \{s_1, s_2, s_3, \ldots\}$. We distinguish between logical objects LOs with $LO = \{o_1, o_2, \ldots\}$ and physical copies (PCs) denoting copies of the LOs hosted at the sites. $pc_{i,j}$ denotes a physical copy of logical object o_i located at the site s_j.

There are two types of operations, namely reads and writes: $OP = \{r, w\}$. An action ac denotes an operation that acts on a specific $lo \in LO$, i.e., $ac \in OP \times LO$. A read $r(o_i)$ returns the value of o_i without any side-effects, whereas a write $w(o_i)$ changes the value of o_i. Let A denote the set of all actions, then a transaction t is a tuple with:

$$\begin{aligned} t &= (A_t, <_t) \\ A_t &= \{ac_1, ac_2, \ldots, ac_k\} \cup term, \quad A_t \subseteq A \end{aligned} \tag{1}$$

where $term \in \{c, a\}$ is a termination action (*c*ommit or *a*bort) and $<_t \subseteq (A_t \times A_t)$ denotes the precedence relation defined on the transaction's actions. The termination action must be ordered after all other actions of a transaction t with regards to $<_t$, i.e., $ac_i <_t term$ for all $ac_i \in A_t$.

A *read-only* transaction (t_r) –in contrast to an *update* (t_u) transaction– does not include any write action.

As depicted in Figure 1, different mappings of LOs to sites lead to different types of DDBSs: i.) Fully replicated DB in which each LO is present at each site. ii.) Partially replicated DB. Each $o \in LO$ is available at a subset of sites. We distinguish between *pure partial replication* and *hybrid partial replication*. In a pure partial replication, no site contains all data objects. Hybrid partial replication means that some sites contain copies of all data objects, whereas others contain copies of only a subset of data objects. iii.) Partitioned DBs in which LOs are available at exactly on site, without replication. Compared to a partially replicated system, in a partitioned system the sets of objects at the sites are disjoint.

III. Replication Protocols

Transaction execution in a DDBS needs to address two challenges: First, the concurrent access of transactions to distributed data should be coordinated. This task is commonly referred to as *concurrency control (CC)* and implemented by a *concurrency control protocol (CCP)* [17], [33]. Second, the actions on LOs must be mapped to actions on a set of physical copies available at different sites (Figure 2). This task is commonly referred to as *replica control (RC)* and implemented by a *replication protocol (RP)* [33].

A. Classification of Replica Control Protocols

RPs can be classified according to *where* transactions are executed and *when* the results of updates are propagated to other replica sites in the system. The '*where*' defines

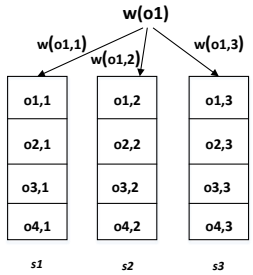


Figure 2: Action on LO mapped to physical copies

which site is allowed to execute a transaction [34]. In a primary-copy approach there is a dedicated site that executes transactions, whereas in the update-everywhere any site in the system can execute transactions.

The '*when*' defines the point in time in the transaction lifecycle at which updates are propagated to other replica sites in the system. RPs can be either *eager* or *lazy*. Eager RPs update all replica sites in the scope of the running transaction, i.e., before a response is returned to the client. Lazy RPs postpone the update propagation to other replica sites to dedicated refresh transactions and return results immediately to the client.

A read $r(o_i)$ is mapped to $R(o_i)$ with $R(o_i) \subseteq \{r(pc_{i,1}), r(pc_{i,2}), \ldots, r(pc_{i,n})\}$, i.e., depending on the replication protocol used, either one or several physical copies have to be read. For a write $w(o_i)$, eventually all physical objects of o_i have to be updated; however, depending on the replication protocol used, within the update transaction, the changes have to be applied either at a subset of the physical copies (lazy replication) or at all physical copies (eager replication).

B. Transaction Lifecycle

Depending on the desired consistency level, the system configuration (e.g., number of sites, site configuration), etc., transactions will undergo the following phases during their life-cycle:

1.) Processing: consists of the actions (reads, writes) needed for the execution of the transaction. One of the main tasks of this phase is the choice of the optimal site for executing the transactions. Optimality can be defined in terms of monetary cost, response time, etc.

2.) Concurrency control: the objective of this phase is to guarantee the correct concurrent execution of transactions; usually, correctness means equivalence to a serial (non-concurrent) execution. Example of CCPs include, for instance, Two-Phase Locking (2PL), timestamp-based protocols, etc. [35] that differ in the optimizations they incorporate.

3.) Commit: consists the activities needed for making the transaction results durable. Commit protocols differ in the choice of site: commit only a subset of all sites, commit locally at a site, or commit at all sites. If only a subset of sites is committed in the context of a transaction, then the commit strategy may incorporate a means of choosing the optimal sites for commit.

4.) Refresh: consists of activities for pulling the recent updates from one or more sites (in case not all sites are jointly committed in the previous phase). It may incorporate different strategies for choosing the optimal sites to pull updates from.

5.) Synchronization: consists of activities for pushing updates to other sites. The strategies usually incorporated are related to choosing the optimal time for propagation. Usually, updates of many transactions are batched instead of sending updates of each transaction separately.

Depending on the actual protocols used, phases 4.) and/or 5.) may be superfluous.

IV. Replication Protocols and 1SR Consistency

One-Copy Serializability (1SR) requires that the effects of an interleaved transaction execution on a replicated system is equivalent to a serial execution on an one-copy (single-copy) database. In [17], this is achieved by using Two-Phase Locking (2PL) as a CCP and Two-Phase Commit (2PC) for the eager propagation of updates. We have implemented the 1SR protocol with the goal of assessing the performance of different RPs. The RPs differ in the number of sites that are updated eagerly before a response is delivered to the client leading to different flavors of the 1SR protocol.

In what follows we will shortly summarize the RPs that are evaluated as part of this work.

A. ROWA and ROWAA

ROWA (read-one-write-all) follows the update everywhere approach (i.e., it can be executed at any site) and updates all available physical copies eagerly. Thus, a read can be executed at a single site. ROWAA (read-one-write-all-available) also provides update-everywhere eager replication semantics. Again, reads can be executed at a single site. However, in contrast to ROWA, it provides a higher degree of availability as updates are performed only at available sites. As a consequence, ROWAA requires costly and complex reconciliation algorithms once a failed site recovers.

B. Quorum Protocols

In quorum-based RPs, only a subset of replica sites is updated eagerly. However, the subsets must be chosen in such a way that any two writes or a write and read on the same data object overlap. This is known as the *intersection property*. For quorum-based RPs, this is crucial as it allows a consistent decision taken by a subset of sites on behalf of all sites [36], which, in turn, is necessary for guaranteeing strong consistency as demanded by 1SR. It is well known that quorum-based RPs have a lower overhead compared to ROWA(A) for writes at the cost of increased overhead for

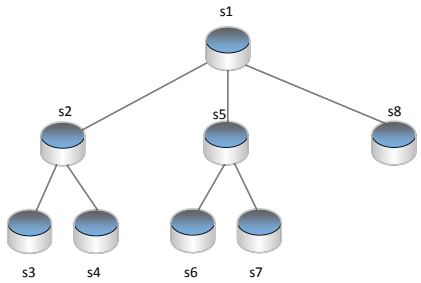


Figure 3: Site Structure for a Log-Write Tree Quorum

reads [36]. Reads must access a subset of sites that form a read quorum. Based on timestamps that are attached to the updates, it is possible to determine the most recent version in a read quorum, which is then also guaranteed to be globally the most recent one due to the intersection property.

Different quorum RPs have been developed, such as the *majority quorum (MQ)* [27], [37] or the *tree quorum (TQ)* [28]. All have the intersection property in common, but they differ on the costs generated for transaction execution. Additionally, they differ in the overhead generated for organizing the sites in a certain logical structure, which is a precondition for the intersection property in some protocols, and for maintaining that structure [36].

Quorum RPs can be *static* or *dynamic* w.r.t. quorum construction. In static protocols the quorums do not change except in cases of site failures, whereas dynamic protocols are able to adapt the quorums to application workload.

Majority Quorum (MQ): is a simple quorum RP, in which each site has a nonnegative number of votes. The quorums are then chosen in such a way so that they exceed half of total votes [27], [37], [36]:

$$\begin{aligned} wq &= \lfloor \frac{\#votes}{2} \rfloor + 1 \\ rq &= \lfloor \frac{\#votes + 1}{2} \rfloor \end{aligned} \tag{2}$$

In the simplest form, each site has the same amount of votes, all with the same weight. It follows that wq can be created from a majority of sites, whereas an rq by half of the sites if $|S|$ is even, or majority of sites if $|S|$ is odd.

Tree Quorum (TQ): it logically organizes the sites in a tree structure [28] where each tree is defined by its *degree* d (the maximum number of children of a node or site, resp.) and its *height* h (the longest path from the root to a leaf node). A tree quorum $q = \langle l, w \rangle$ for a tree is constructed as follows: First, the root of the tree is selected and w children are added to the root. Then, child nodes are recursively added to each node until the depth l is reached [38].

In order to guarantee the intersection property, quorums must overlap both in height and degree. For a read quorum $rq = \langle l_r, w_r \rangle$ and a write quorum $wq = \langle l_u, w_u \rangle$ to overlap, the following must hold: $l_r + l_u > h$ and $w_r + w_u > d$ and for two write quorums to overlap: $2 \cdot l_u > h$ and $2 \cdot w_u > d$.

Transaction mix	Description
Readonly	Consists of read-only transactions.
RW8020	Consists of 80% read-only and 20% update transactions.
RW5050	Consists of 50% read-only and 50% update transactions.
Writeonly	Consists of update transactions only.

Table I: Transaction Mixes

Log-Write Tree Quorum (LWTQ): it is a special instance of the generic TQ protocol with $rq = \langle 1, d \rangle$ and $wq = \langle h, 1 \rangle$ [38]. Thus, in a failure-free environment, reads access a single site, namely the root – and update operations are executed on a path down the tree. As depicted in Figure 3, a read quorum consists of the root node (s_1). In case s_1 fails, a rq consisting of the root's children must be found. In the example in Figure 3, the rq would then consist of the sites s_2, s_3 and s_8. In general, if a node is not available, then for a rq all its children must be accessed and this increases the quorum size. Write quorums must access one node at each level. In the tree depicted in Figure 3, the wq may consist of the sites s_1, s_2, and s_3.

V. Implementation and Evaluation

In [39], we have introduced PolarDBMS (*Po*licy-based and modu*lar* DBMS for the Cloud), our prototype system that is based on a modular architecture. Each module provides certain data management functionality, such as a RP, data consistency, atomic commitment, consensus protocols and others, and for each module, different implementations may exist (e.g., ROWA, ROWAA, quorums for RP). The main objective of PolarDBMS is to automatically select and dynamically adapt the modules and their implementations to best provide the guarantees requested by Cloud users. All RPs considered in this evaluation are implemented as PolarDBMS modules, with each module providing a web service interface.

A. Implementation of RPs

In the RPs evaluated as part of this work, transaction lifecycles consist of three phases, namely a locking phase, a processing phase, and a 2PC (commit coordination) phase. Once a transaction has been submitted for execution at a certain site, that site will become the coordinator. In the first step, the site will acquire the necessary locks at a centralized `LockManager` as part of Strict Two-Phase Locking (S2PL). Next, the processing is initiated which is dependent on the RP. In ROWAA, transactions can be executed at any site. In the quorum RPs, read-only transactions must access all sites as defined by the read quorum (rq) in order to determine the most recent values for the objects to be read. This can be achieved by considering the commit

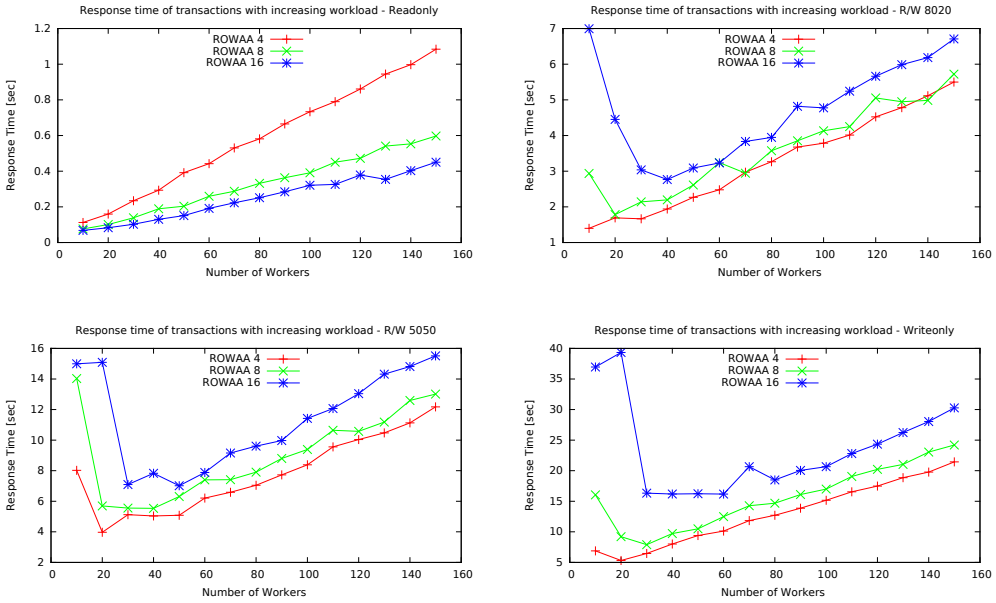


Figure 4: Response time of transactions in ROWAA

timestamp of the objects. During the 2PC phase, the updates will eagerly propagate their changes to other sites using a log-based approach [33]. ROWAA will eagerly update all available sites, whereas the quorum RPs will update those sites that consist the write quorum (wq).

The quorums for MQ are constructed using a random strategy, in which each site constructs its quorum by randomly selecting a set of sites so that conditions in Equation (2) are satisfied. Clearly, using such a strategy, certain sites may be included in too many quorums and thus become a bottleneck. In our implementation, each site has the same amount of votes with the same weight.

In LWTQ, the rq consists of the root site, whereas the wq will randomly choose a path from the root down a leaf and include the sites that consist that path.

B. Evaluation Set-Up

TPC-C is an On-Line Transaction Processing (OLTP) benchmark, that models transactions of a wholesale supplier [26] and consists of a mix of read-only and update transactions. In our implementation of TPC-C, we have defined different transaction mixes, which are depicted in Table I. Transactions according to a specific mix are generated by client terminals, which run separately from the system under test (SUT). The client will start a number of worker threads that will submit transactions to a specific site for execution. The distribution of transactions to sites is done in a round-robin manner. The workers will wait for the response of their submitted transactions, report the statistics to a `StatisticsModule` before submitting a new transaction for execution. The TPC-C data is generated with the following parameters: the number of warehouses is set to 1, the number of districts to 10, the number of customers to 3'000, and the number of stock entries to 100'000. Each object has the same probability of being accessed, i.e., there are no hot-spots. Thus, the conflict rate between transactions is mainly influenced by the r/w ratio of the transaction mix.

We have conducted our evaluations in the AWS Cloud Environment using `c1.medium`[2] as machine type. Each evaluation is executed 10 times for each different RP and configuration, and the results are averaged over the runs.

C. Single Data Center

The goal of this evaluation, which is similar to the size-up test defined in [40], is to analyze the response time of transactions with increasing throughput in order to depict the overhead of the different transaction phases. All sites are deployed in the *eu-west* region. We have measured a round-trip time (RTT) between sites of up to 3 ms, thus network overhead is negligible.

We have evaluated all RPs for different number of sites (4, 8 and 16) by varying the read/write ratio as described in Table I. The client will initially start with 10 workers, which will continuously submit new transactions for execution, after having received the response of previously submitted transactions. During a period of 30 seconds the number of workers is kept constant and afterwards increased by 10 until the maximum number of 150 workers is reached.

[2]http://aws.amazon.com/de/ec2/instance-types/

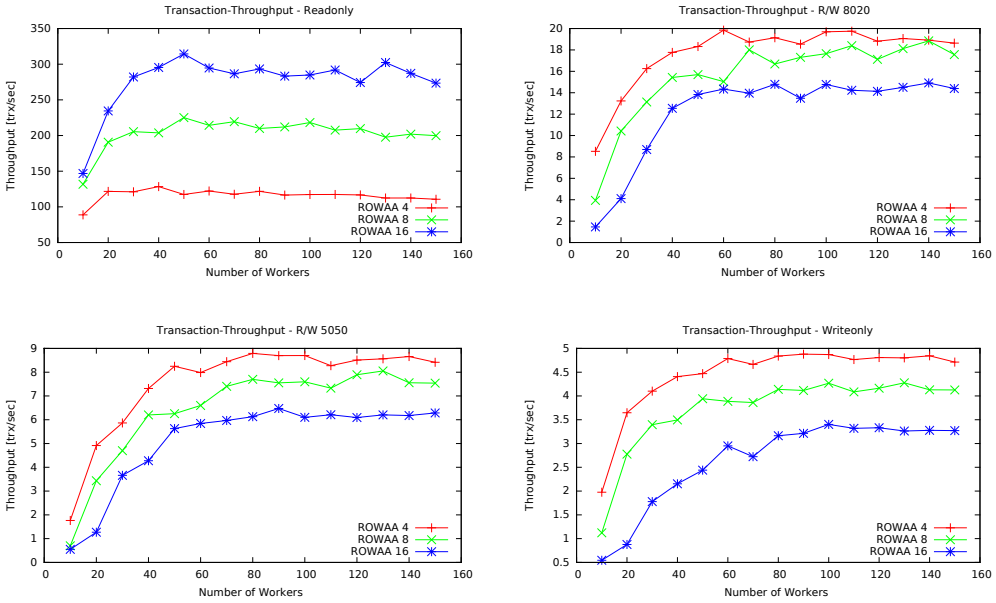


Figure 5: Throughput of transactions in ROWAA

	ROWAA			LWTQ		
R/W ratio	**4**	**8**	**16**	**4**	**8**	**16**
RW8020: total	3.60	3.94	4.81	2.33	2.06	1.80
RW8020: 2PL	3.33	3.74	4.61	1.24	0.93	0.73
RW5050: total	8.27	9.34	11.44	3.74	3.14	2.62
RW5050: 2PL	7.99	9.04	11.14	2.50	1.82	1.36
Writeonly: total	14.21	16.64	22.34	6.52	4.94	4.12
Writeonly: 2PL	13.93	16.24	21.79	4.82	3.14	2.31

This table summarizes the average results over all runs, rounded to two decimal places. As the proportion of update transaction increases the 2PL overhead also increases and considerably impacts the overall response time of transactions. Compared to ROWAA, the 2PL overhead in LWTQ is lower as each transaction must be forwarded to the root, and as the root gets overloaded the load at `LockManager` gets reduced.

Table II: Total response time and 2PL duration in [s] for ROWAA and LWTQ

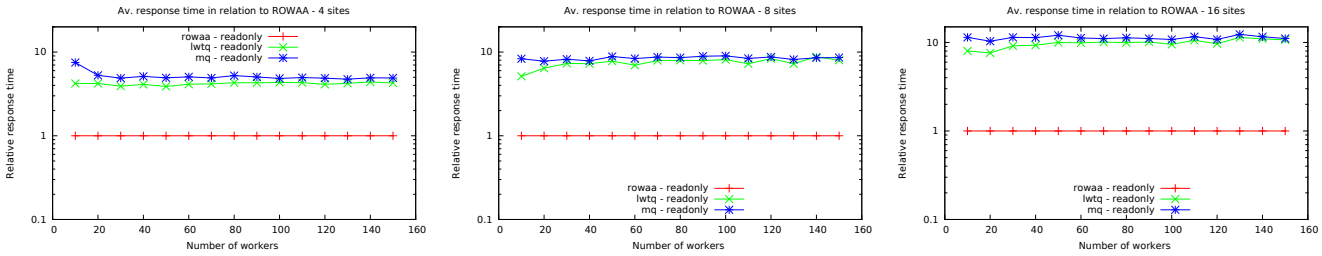
Figure 4 depicts the average response time of transactions in ROWAA for varying read/write ratio and site numbers as the load increases. Figure 5 shows the corresponding transaction throughput.

The response time of LWTQ and MQ relative to ROWAA is depicted in Figures 6 to 9 and the throughput in Figures 10, 11 and 13. Note that the y-axis is log-scaled and that the ROWAA response time serves as baseline.

Readonly mix: ROWAA performs best for the read-only mix as transactions can be executed at any site and it can thus fully exploit the capacity of all sites in the system. As depicted in Figures 4 and 5, the ROWAA performance improves as the number of sites increases.

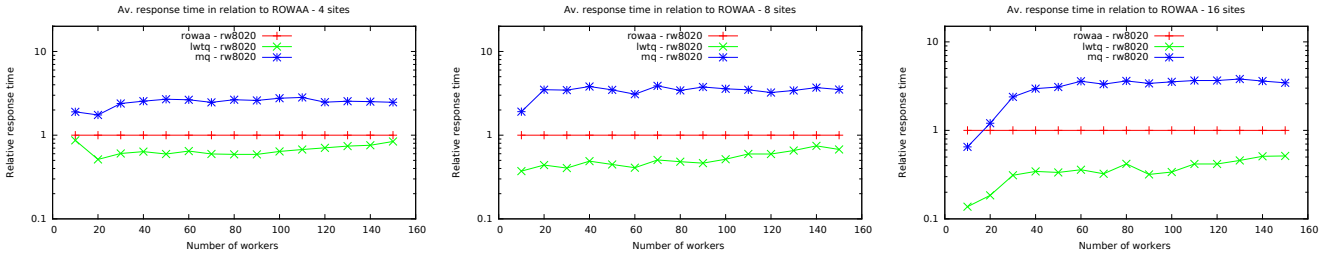
LWTQ and MQ have a higher response times and lower throughput compared to ROWAA (see Figure 6). In LWTQ, each read transaction must be executed by the root in order to guarantee 1SR consistency and this leads to the root becoming the bottleneck. In MQ, each read transaction must read from the rq in order to determine the most recent values of object and this leads to an increase of the response time and decrease of throughput compared to ROWAA. Note that the requests sent to the members of the rq in MQ is done using a multicast. Thus, the slowest site in the rq determines the additional read overhead.

Read/write mix: In ROWAA the higher the proportion of update transactions in the mix, the higher is the 2PL



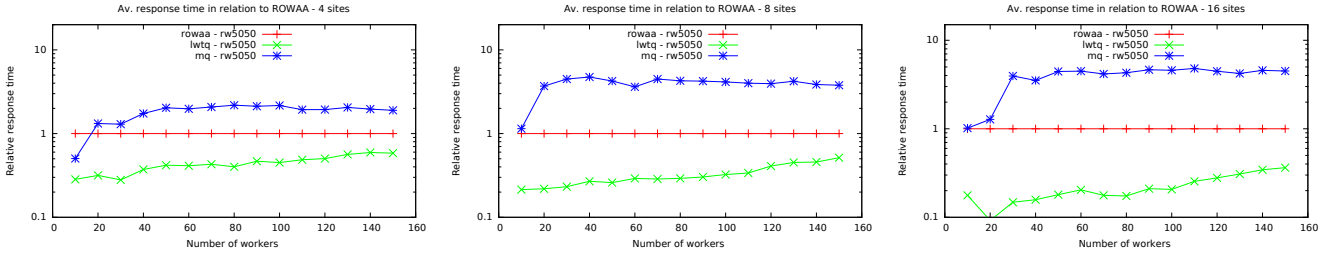
(a) 4 sites (b) 8 sites (c) 16 sites

Figure 6: Readonly: response time of transactions



(a) 4 sites (b) 8 sites (c) 16 sites

Figure 7: RW8020: response time of transactions



(a) 4 sites (b) 8 sites (c) 16 sites

Figure 8: RW5050: response time of transactions

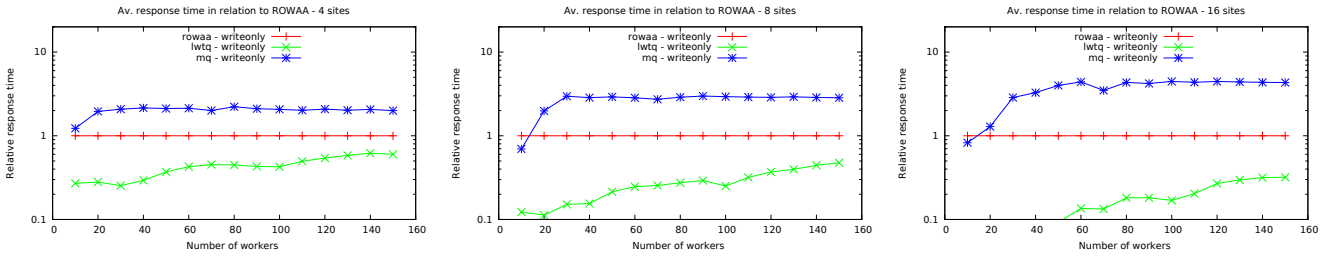
overhead as the conflict rate between transactions increases (Table II). The processing and 2PC overhead are rather low compared to the 2PL overhead. Thus, the conflict rate, which is influenced by the ratio of update transactions and the transaction throughput, is the limiting factor for the performance in ROWAA. Notice that, since we use round-robin distribution of transaction to sites, the higher the number of workers and sites, the higher is the number of concurrent transactions and thus the conflict rate.

LWTQ outperforms ROWAA for transaction mixes that contain update transactions (Figures 7 to 9). However, as Table II shows, this is related to the lower concurrency degree at the `LockManager`. As transactions need to be forwarded from the sites to the root, the requests at the `LockManager` arrive with a delay compared to ROWAA.

MQ performed worst of all RPs. In our evaluations, we have constructed the quorums using a random strategy, in which each site randomly chooses other sites for its quorum. This may lead to some sites being more frequently included in write quorums and thus become a bottleneck (see Figure 15 and Figure 14). 2PC duration increases the overall transaction duration which, in turn, increases the 2PL overhead.

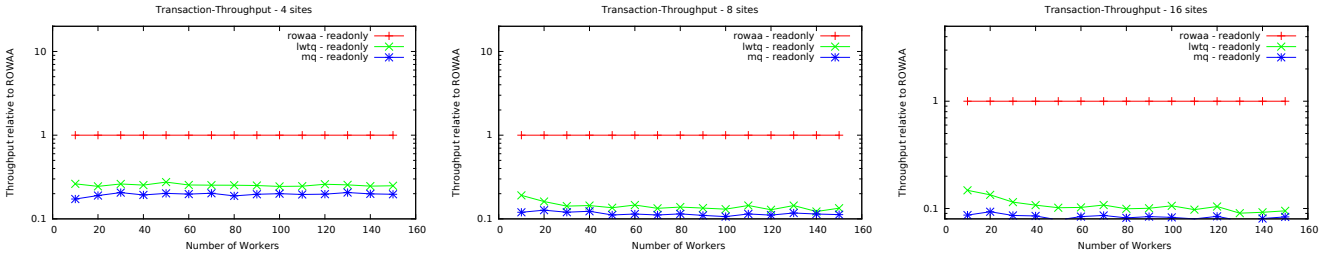
In summary, 2PL considerably impacts the overall performance of transactions in all evaluated RPs, and accounts in some cases for up to 98% of the overall overhead. The application workload, i.e., transaction mix and the corresponding access patterns of transactions, is a critical factor that influences the cost of 2PL. However, one cannot influence the application workload. So, it is important to invest in clever strategies with the goal of reducing 2PC and processing overhead.

As we have seen with MQ, it is important to avoid situations in which some sites become a hotspot by including



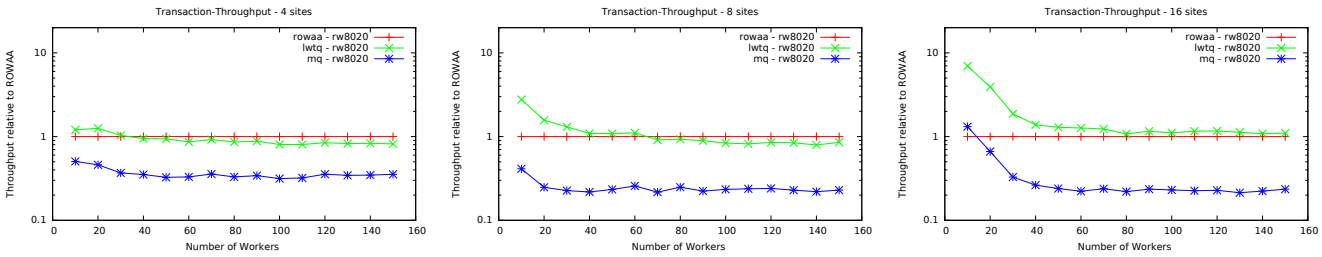
(a) 4 sites (b) 8 sites (c) 16 sites

Figure 9: Writeonly: response time of transactions



(a) 4 sites (b) 8 sites (c) 16 sites

Figure 10: Readonly: transaction throughput



(a) 4 sites (b) 8 sites (c) 16 sites

Figure 11: RW8020: transaction throughput

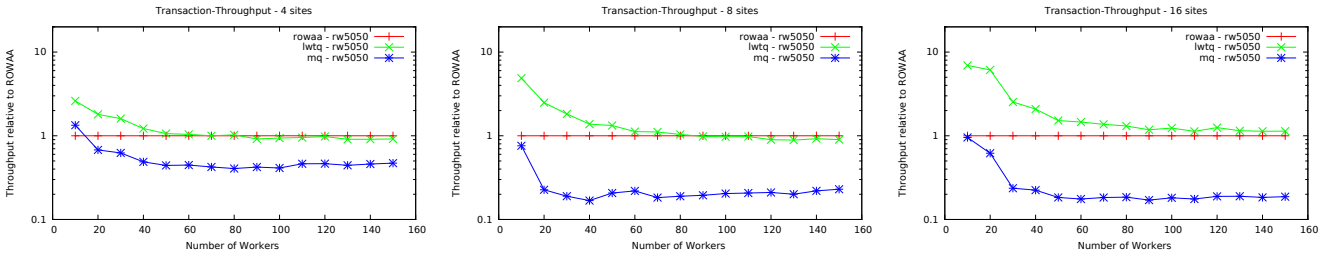
them more frequently in quorums compared to others. By doing that, one can impact both the 2PC and processing overhead and in turn the lock duration.

D. Multi Data Center

This experiment is similar to the Single Data Center with the difference that now the network delay at two sites is increased with the goal of assessing the impact of network RTT to the overall performance of the RPs. We used the `netem`[3] tool to increase the delay as follows. The delay of the first site was increased by 85ms and that of the second site to 175ms. The values correspond to the half of the distance from the `eu-west-1` datacenter to `us-west-1` (170ms) and `ap-southeast-2` (350ms) datacenters [22]. We halved the distance in order to cope with the additional overhead generated by the entire web service stack. However, it is still 20 - 40 times higher compared to the RTT inside the same datacenter. The client submits transactions only to the sites in the `eu-west-1` datacenter, i.e., to the closest sites[4], as the goal is to analyze the impact of the increased RTT to the 2PC overhead and then to the overall performance. In LWTQ, the root is located in the `eu-west-1` and the write quorums are chosen randomly on a per transaction basis, which means that for some transactions the quorums will consist of sites located in the `eu-west-1` datacenter. Other transactions however may pick a quorum that includes one of the distant sites located in the `us-west-1` or `ap-southeast-2` datacenter. Clearly, as only two sites are located in remote datacenters, the more sites available in the system, the lower the probability of accessing a distant site as part of a wq.

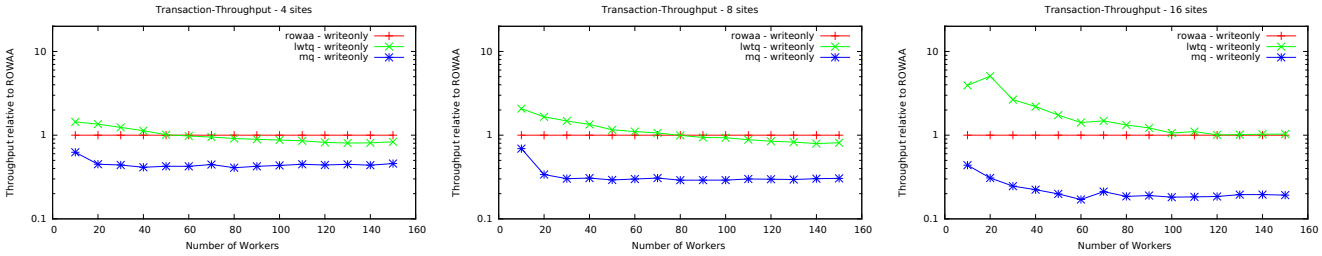
[3]www.linuxfoundation.org/collaborate/workgroups/networking/netem

[4]This corresponds to how usually load balancing is done in Multi Data Centers [41].



(a) 4 sites (b) 8 sites (c) 16 sites

Figure 12: RW5050: transaction throughput



(a) 4 sites (b) 8 sites (c) 16 sites

Figure 13: Writeonly: transaction throughput

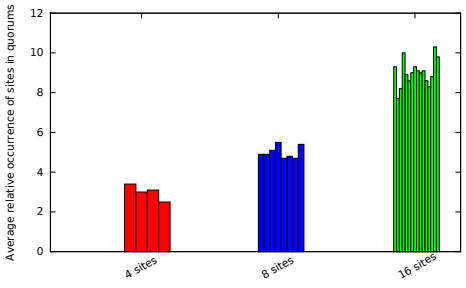


Figure 14: Average occurrence of sites in MQ quorums for 10 runs

RP	4	8	16
ROWAA	0.1331	0.1713	0.2971
LWTQ	0.1118	0.1148	0.1180
MQ	0.3975	0.7451	1.6340

Figure 15: 2PC overhead in [s] for different RPs (writeonly)

2PC overhead (rounded to four decimal places) for MQ is between 3-5 times higher compared to ROWAA, and 3-14 times compared to LWTQ.

As depicted in the evaluation results in Table III, ROWAA and MQ show a similar performance behavior. One can observe that the increase of number of sites in the `eu-west-1` data center does not impact the 2PC overhead in ROWAA and MQ as now the geographical distance (increased RTT) of the sites that reside outside the `eu-west-1` data center are the limiting factor to the performance. This additional 2PC overhead has a cascading effect to 2PL and subsequently to the overall performance. As locks need to be kept longer compared to the Single Data Center case, conflicts between the transactions are more eminent.

LWTQ achieves lower throughput compared to ROWAA and MQ mainly due to the necessity of forwarding transactions to the root node, which becomes a bottleneck. In Table III one can observe that with an increasing number of sites, the 2PC overhead decreases as the probability of including one of the distant sites in the wq decreases, which leads to a lower 2PL/total overhead. Notice however that with an increase of the update ratio in the transaction mix, it is more likely that a distant site is involved in a commit. This explains the higher 2PC overhead for the writeonly mix in LWTQ compared to the $50\% - 50\%$ mix (Table III).

E. Lessons Learned

In summary, the number of sites is not the determining factor for the overall performance of the RPs, but rather the properties of the sites, such as their load and the RTT between sites.

In Single Data Centers, in which the RTT between the sites is negligible, 2PL is the limiting factor to performance. For transaction mixes containing update transactions, 2PL overhead accounts for $40\% - 98\%$ of the total overhead and this is the best case as there are no hotspot-objects. Hotspots would make things only worse. If, however, 2PL is mandatory in a certain application, then one can invest in reducing the overhead for the processing phase by incorporating load

	ROWAA			MQ			LWTQ		
R/W ratio	**4**	**8**	**16**	**4**	**8**	**16**	**4**	**8**	**16**
RW5050: total	216.25	234.34	224.23	217.46	197.73	213.86	70.79	34.00	5.88
RW5050: 2PL	211.91	229.76	219.84	202.37	191.16	207.66	65.60	30.80	3.56
RW5050: 2PC	4.07	4.41	4.23	5.67	4.04	4.21	1.46	0.79	0.19
RW5050: #Trx	1'223 / 2'533	1'231 / 2'370	1'231 / 2'497	848 / 1'722	1'120 / 2'325	1'176 / 2'329	659 / 1'865	296 / 1'496	541 / 2'212
Writeonly: total	451.48	455.56	458.70	434.15	420.07	423.21	246.92	178.95	20.18
Writeonly: 2PL	442.90	446.86	449.85	425.90	412.08	414.87	235.47	165.14	15.10
Writeonly: 2PC	8.42	8.49	8.59	8.06	7.81	7.83	4.58	4.11	0.96
Writeonly: #Trx	1'244 / 1'244	1'244 / 1'244	1'244 / 1'244	1'257 / 1'257	1'325 / 1'325	1'247 / 1'247	698 / 698	304 / 304	496 / 496

Average results over all runs, rounded to two decimal places. Due to space limitations, we have only depicted the results for rw5050 and writeonly, which are also the most interesting results as they generate more 2PC overhead.

Table III: Total response time, 2PL & 2PC duration in [s], no. of executed write transactions for ROWAA, MQ & LWTQ

balancing strategies, as this would shorten the lifetime of transactions and thus reduce the conflict rate between them [42].

In Multi Data Centers, the quorum construction strategy is crucial in order to avoid bottlenecks. As we have seen in case of MQ (Figure 14), if certain sites are included more frequently in the quorums they will become overloaded and thus degrade the overall performance. The same applies for LWTQ, which needs to consider the site properties when constructing the tree and the resulting quorums. Compared to the Single Data Centers, it is necessary to jointly address both the load and the RTT when constructing the quorums. As depicted in Table III, 2PC costs are doubled by increasing the RTT of some sites by a factor of 20 to 40. The increased 2PC overhead leads to an increase of the transaction lifetime and this considerably impacts the conflict rate and by that the overall performance (Figure 16).

In the context of the Cloud, users are charged for each resource in a fine-grained way. Thus, actions (e.g., 2PC messages) consuming those resources generate a precisely defined monetary cost. In addition to performance, application providers should also consider the monetary cost when choosing the RP. For example, each 2PC message generates a cost (due to bandwidth consumption). The overall 2PC cost per transaction is dependent of the number of sites involved in the commit, which is dependent on the RP.

VI. Related Work

Different replication protocols have been developed with the goal of reducing the overhead for transactions when strong consistency is required [33], [36], [28], [27], [43]. However, currently there is a lack of work that assesses the performance of different replication protocols using a real implementation on the basis of a concrete benchmark.

In [44], [45], [46], the performance of replication protocols are characterized on the basis of an asymptotic analysis and considered as being representative for the overall system performance.

The work in [47] analyzes the performance of different replication protocols and considers the response time and the ratio of committed transactions to the total number of transactions.

[36] provides an analysis of the scalability and availability of different replication protocols by defining the upper bound of what can be achieved in the best case. However, as important aspect, the impact of 2PL on the overall performance, is neglected.

In contrast to existing work, we have assessed the performance of different replication protocols by considering all components (phases) of transactions and also their impact on the overall performance on the basis of the widely accepted TPC-C benchmark and by also considering geo-replicated databases in the Cloud.

VII. Conclusion and Outlook

In this paper, we have provided an in-depth analysis of the performance of different replication protocols on the basis of the TPC-C benchmark using the AWS Cloud infrastructure. Our evaluations are run in two configurations, a single and a multi-data centers environment, and analyze the overhead generated by the different transaction phases. Compared to existing work, our results are based on a concrete benchmark and provide a more detailed view on where exactly the

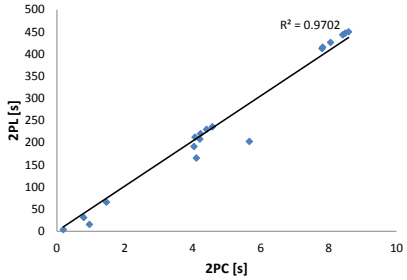


Figure 16: Correlation between 2PC and 2PL

overhead is generated. As part of a future work, we plan to complement our evaluations by considering additional RPs, and also by considering additional parameters, such as the capacity of sites and the monetary costs that incur in a Cloud due to the use of the resources needed by the RPs.

VIII. Acknowledgment

This work has been partly supported by the SNSF (project ClouDMan) and an AWS research grant.

References

[1] S. Das, D. Agrawal, and A. El Abbadi, "G-store: a scalable data store for transactional multi key access in the cloud," in *Proc. SoCC*, 2010.

[2] E. A. Brewer, "Towards robust distributed systems," in *Proc. PODC*, 2000.

[3] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, 2002.

[4] C. Hale, "You can't sacrifice partition tolerance," 2010.

[5] P. Bailis *et al.*, "HAT, Not CAP: Towards Highly Available Transactions," in *Proc. HotOS*, 2013.

[6] ——, "Bolt-on causal consistency," in *Proc. SIGMOD*, 2013.

[7] W. Vogels, "Eventually consistent," *Commun. ACM*, 2009.

[8] "Oracle NoSQL Consistency," online; accessed September-2015.

[9] A. Cassandra. Apache Cassandra. Online; accessed September-2015. [Online]. Available: http://incubator.apache.org/cassandra/

[10] MongoDB. MongoDB. Online; accessed September-2015. [Online]. Available: http://www.mongodb.org/

[11] J. Hamilton, "I love eventual consistency but..." 2010, [Online; accessed September-2015].

[12] D. Obasanjo, "When Databases Lie: Consistency vs. Availability in Distributed Systems." 2009, [Online; accessed September-2015].

[13] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," in *Proc. SIGOPS*, 2010.

[14] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," in *Proc. SIGOPS*, 2007.

[15] V. Zuikevičiūtė and F. Pedone, "Correctness criteria for database replication: Theoretical and practical aspects," in *Proc. OTM*. Springer, 2008.

[16] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, 1990.

[17] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1986.

[18] H. Berenson *et al.*, "A critique of ansi sql isolation levels," in *Proc. SIGMOD*, 1995.

[19] K. Daudjee and K. Salem, "Lazy database replication with snapshot isolation," in *Proc. VLDB*, 2006.

[20] S. Elnikety *et al.*, "Database replication using generalized snapshot isolation," in *Proc. SRDS*, 2005.

[21] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," *Computer*, 2012.

[22] P. Bailis *et al.*, "Highly available transactions: Virtues and limitations," in *Proc. VLDB*, 2013.

[23] B. Kemme and G. Alonso, "Don't be lazy, be consistent: Postgres-r, A new way to implement database replication," in *Proc. VLDB*, 2000.

[24] I. Fetai and H. Schuldt, "Cost-based data consistency in a data-as-a-service cloud environment," in *Proc. CLOUD*, 2012.

[25] T. Kraska *et al.*, "Consistency rationing in the cloud: Pay only when it matters," in *Proc. VLDB*, 2009.

[26] TPC-C Benchmark, http://www.tpc.org/tpcc/, online; accessed September-2015.

[27] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Trans. Database Syst.*, 1979.

[28] D. Agrawal and A. E. Abbadi, "The tree quorum protocol: An efficient approach for managing replicated data," in *Proc. VLDB*, 1990.

[29] D. J. Abadi, "Data management in the Cloud: Limitations and opportunities," *IEEE Data Eng. Bull.*, vol. 32, no. 1, 2009.

[30] D. Kossmann and T. Kraska, "Data management in the cloud: promises, state-of-the-art, and open questions," *Datenbank-Spektrum*, 2010.

[31] R. Rawson and J. Gray, "HBase at Hadoop World NYC." http://www.docstoc.com/docs/12426408/HBase-at-Hadoop-World-NYC/, 2009.

[32] F. Yang, J. Shanmugasundaram, and R. Yerneni, "A Scalable Data Platform for a Large Number of Small Applications," in *Proc. CIDR*, 2009.

[33] B. Kemme *et al.*, "Database replication," *Synthesis Lectures on Data Management*, 2010.

[34] J. Gray *et al.*, "The dangers of replication and a solution," in *Proc. SIGMOD*, 1996.

[35] G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2001.

[36] R. Jiménez-Peris *et al.*, "Are quorums an alternative for data replication?" *ACM Trans. Database Syst.*, 2003.

[37] D. K. Gifford, "Weighted voting for replicated data," in *Proc. SOSP*, 1979.

[38] D. Agrawal and A. El Abbadi, "The generalized tree quorum protocol: An efficient approach for managing replicated data," *ACM Trans. Database Syst.*, 1992.

[39] I. Fetai, F. M. Brinkmann, and H. Schuldt, "PolarDBMS: Towards a cost-effective and policy-based data management in the cloud," in *Proc. ICDEW*, 2014.

[40] B. F. Cooper *et al.*, "Benchmarking cloud serving systems with YCSB," in *Proc. SoCC*, 2010.

[41] L. A. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The google cluster architecture," *Micro, Ieee*, vol. 23, no. 2, pp. 22–28, 2003.

[42] P. A. Bernstein and E. Newcomer, *Principles of transaction processing*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2009.

[43] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The grid protocol: A high performance scheme for maintaining replicated data," in *Proc. KDE*, 1990.

[44] S. Rangarajan, S. Setia, and S. K. Tripathi, "A fault-tolerant algorithm for replicated data management," *IEEE Trans. Parallel Distrib. Syst.*, 1995.

[45] Y. Amir and A. Wool, "Optimal availability quorum systems: Theory and practice," 1998.

[46] M. Ahamad and M. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *Software Engineering, IEEE Transactions on*, 1989.

[47] C. S. Keum *et al.*, "Performance evaluation of replica control algorithms in a locally distributed database system," in *Proc. DASFAA*, 1995.