HBase++: Extending HBase With Client-Centric Consistency Guarantees for Geo-Replication

Alexandre Filipe Campos, Sergio Esteves, Luís Veiga
alexandre.t.campos@tecnico.ulisboa.pt, sesteves@gsd.inesc-id.pt,
luis.veiga@inesc-id.pt

Instituto Superior Técnico
INESC-ID

Abstract. Motivated by requirements of the growing social web applications, the necessity to store data across several data centres has become even more important in order to bring data as close to the users as possible. A plethora of non-relational databases raised in recent years in order to address this issue due to their ability to scale out. But in the presence of wide distances, to deliver high-availability and data consistency is not possible in the presence of partitions. Several works have been developed that try to reach the panacea of consistency and availability, at times sacrificing consistency in order to provide a better service or sacrificing latency to provide strongly consistent data, without taking into account that not all data needs the same consistency and not all data needs to be readily available. To tackle this problem, we propose a consistency model that will reason about data through user defined divergence bounding values, causal relationships and transactional enforcement within the data centre, in order to provide urgency to data in a geo-replication scenario.

1 Introduction

With the growth of the Internet and social web applications, and as larger amounts of information need to be processed and managed, more data centres are being deployed each year. As users are spread across the globe it becomes relevant, to deploy those data centres as close as possible to the largest amount of users, in order to avoid the large latency imposed by round trips across the globe. Amazon has shown that web users are sensitive to latency [13]: even a 100ms increase in latency causes measurable revenue losses. The focus therefore shifts, from simply having a well built distributed system that can serve from within a data centre, replicating data through several machines, but to be able to propagate and manage data through machines in various data centres. This, as said, brings data closer to the users and enables the system to be available even in the presence of full data centre outages due to various reasons (e.g. natural catastrophes).

Cloud storage systems, such as the currently popular class of NoSQL data stores, have been designed in accordance to this scenario, as they, in contrast to relational database management systems (RDMS), can scale out horizontally to thousands of machines. Due to data not being tightly coupled with several relations, it becomes easier to partition. Many of the NoSQL systems provide a relaxed form of consistency in order to provide high availability, while others strive for strong consistency and sacrifice latency. Other systems go even further and focus on the added feature of geo-replication, providing a system to manage data on a worldwide scale. This comes with inherent trade-off's as the already present problem of performance versus consistency, is highly accentuated due to the high communication latencies between data centres.

Choosing between either an eventual consistent store and a strongly consistent one becomes an arduous task, unless your data falls into a very specific category. As in most problems that need a data store solution, not all data are the same, and it could benefit from different consistency degrees. To address this problem, various works have been developed with the intent of providing support for developers, when deploying a geo-replicated data store, by offering consistency models in-between those opposite extremes. Some go even further as to offer dynamically different levels of consistency, to be defined at development time and others try to change the degrees of consistency at runtime.

HBase[1] is the open source version of Bigtable [4], and supports geo-replication in a eventually consistent manner between different clusters. Thus, one can not predict accurately enough how and when replication takes place, or ensure a given level of quality of service for delivering data to remote master replicas. Also, HBase alone offers only ACID guarantees on single row operations, making it impossible to group up certain related operations for geo-replication. Currently, there are some projects that try to provide transactional support for HBase, such as Omid.[2]

Given the current context, we propose a consistency model based on Vector Field Consistency (VFC) [12, 16] and further advancing on the work developed in the Distributed Systems Group [5, 11]. This model intends to enhance the geo-replication mechanism present in HBase, making it aware of the urgency of certain pieces of data updates, and giving them priority over less urgent data updates, taking into account operations incorporated into transactions and their causal relations.

2 Objectives

As stated, the overall aim of this work is to enhance the geo-replication mechanism present in HBase, making it aware of the urgency of certain pieces of data. In other words, we intend to improve the currently deployed eventual consistent model to be eventual consistency with notion of quality-of-data. With that said we can specify our objectives accordingly: (i) Make use of Omid to provide the basic underlying transactional support for HBase.(ii) Adapt the VFC

[1] http://hbase.apache.org

[2] http://yahoo.github.io/omid/

model to allow users to define consistency bounds on data in the storage. These bounds, will be used to define what is the urgency for each data item with relation to others, and will be placed as metadata information along or inside the HBase tables depending on the granularity of the enforcement.(ii) Incorporate the tracking of causal relationships between operations inside the HBase cluster either explicitly or implicitly. (iv) Develop a scheduling algorithm to define the ordering of operations for geo-replication. This algorithm will take into account the user defined bounds, as to reason about which operations are more urgent, as well as define the order of geo-replication, in a way that does not break causal relations. (v) Define metrics to infer cost of geo-replication in terms of queue disruption to assist the scheduling algorithm to make choices with the least cost possible. This will also help developers to reason about data that might be too strictly bounded. (vi) Develop a geo-replication protocol that enforces the order provided by the scheduling algorithm. (vii) Evaluate the proposed solution in qualitative, quantitative and comparative terms.

3 Related Work

Quality-of-Service for Consistency of Data Geo-replication in Could Computing [5, 11] is a system implemented on top HBase that provides support for geo-replication by providing a unified cache view across different clusters. It relies on a consistency model based on [12, 16], and enables the definition and dynamic enforcement of multiple consistency degrees over different groups of data, within the same application, across very large scale networks of cloud data centres.

The basis, as stated, of this consistency model relies on defining three dimensional vectors to be associated with data objects, where each dimension holds a numerical scalar that defines the maximum divergence of that value. The constraints are defined as time (θ), sequence (σ), value (ν). These values are not fixed, as developers can define intervals to allow the system as it reasons about access patterns, to relax or restrict the bound. As bounds get close to being broken the priority of the associated values for geo-replication increases, resulting in high priority data having stricter bounds and low priority data have relaxed bounds.

The unified cache view will hold frequently used database items and items within the same locality group (i.e., pre-fetch columns of an hot row). Also it will keep track of items that need to be replicated and handle the whole replication process across clusters. The system constantly checks bandwidth and can trigger data replication and synchronization on low bandwidth utilization periods, even if consistency constrains do not impose it. Concurrent updates in different data centres rely on the deterministic rule of last-writer wins to enforce convergence.

In 2011, Junqueira et al. [7], proposed a lock-free transaction management system, built initially for HBase, that relies on a "transaction status oracle" (TSO) for managing timestamp and read and write sets. It provides snapshot isolation and still requires some metadata (commit timestamps) to be stored

in HBase. It later evolved into replicating the commit timestamps to the client since they do not need to be persistently stored [2], further avoiding storage overhead inside the data store.

Currently, the open-souce implementation of the above system is Omid.[3] It relies on BookKeeper [8] for durability and relies on ZooKeeper [6], to enforce a singleton TSOServer instance operating over time. A transaction's life-cycle will start with the request for a start timestamp from the TSO. It will then access the data store and perform all the operations, reading only values whose timestamp is lower than the assigned start timestamp, and writing values with the assigned start timestamp as its version. At the end of the transaction the client submits the rows to the TSO to check for conflicts. If no conflicts arise, the transaction will commit and the commit timestamp for the affected rows will be updated on the TSO. This system can handle large amounts of transactions but it is limited by memory, which will eventually restrict the scalability of the system. A solution is currently in development to further scale Omid called MegaOmid. It will partition transactions among multiple status oracles, designated for certain areas, and will have a global status oracle for global transactions (spawning more than one area).

Here we analyse work that focus on providing different levels of consistency for differentiated data in the same storage. Thus, exploring the fundamental trade-offs between consistency, performance and availability it is argued that not all data needs the same consistency, and that multiple (sometimes dynamic) levels should be offered [14].

Consistency rationing in the cloud: Pay only when it matters [9]. In this system data is divided into three consistency categories, and present them in the context of a web shop: the first (A) covers data which a consistency violation would result in large penalty costs (e.g. credit card info); the second (B) focuses on data where the consistency requirements vary over time depending on actual availability (e.g. a products stock: a product that has a very low stock needs higher consistency than one who's stock is very high) of an item; finally the third (C) covers data where temporary inconsistencies are acceptable or negligible (e.g. logging info, users preferences). For data that falls in category A serializability is provided through pessimistic concurrency control: two-phase locking. For category C data, read-your-writes and monotonic session consistency is provided with last-writer wins conflict resolution for non-commutative updates. Finally, category B offers a runtime adaptive approach mixing both of the above. The adaptivity in category B is based on three policies:

i. **General Policy:** based on the frequency of accesses the probability of conflicts is calculated, if it breaks a certain threshold the consistency changes;
ii. **Time Policy:** when a certain temporal metric is reached the consistency changes (e.g. 5 minutes left for the auction to end);

[3] http://yahoo.github.io/omid/

iii. **Numeric Type Policies:** by defining or inferring through access frequency probabilistic analysis a value threshold, where if the policy is broken then strict consistency is enforced.

Transactions are also supported and since consistency guarantees are defined on a data level, operations within a transaction can cover various categories and each operation is handled with the consistency defined for that record, but the results of joins, unions, and any other operations between A and C data provide only C guarantees for that operation.

Pileus [15], is a key-value store with a range of consistency choices between eventual and strong consistency and offers the developers the choice to define service level agreements (SLA) that incorporate both consistency and latency. Their target applications are those that tolerate relaxed consistency but, nevertheless, benefit from improved consistency.

All puts and gets are within the context of a session, and all puts are strictly ordered at a primary site to avoid inter-site conflicts, making it the authority for strong consistency responses. With an SLA the developer can define and variable amount of subSLA's, ranking them in preference. Each subSLA defines a consistency-latency pair where the consistency value can have one of the following values: strong, eventual, read-my-writes, monotonic, bounded(t), causal. The utility of a subSLA is a number that allows applications to indicate its relative importance and along with network monitoring and storage node information gathered by the Client, will help making a decision on which SLA to attempt to satisfy and which storage nodes to contact.

4 Proposed Architecture

In this section we will present the overall architecture for HBase++, each of its main components and how they cooperate. The first subsystem/component addresses how to enforce serializable transactions and track causality within the cluster (Section 4.1). Afterwards we will detail the scheduling algorithm for geo-replication (Section 4.2), and finally we present the geo-replication protocol and how it enforces the scheduling order provided by the algorithm across clusters (Section 4.3).

4.1 Architecture Overview

Our goal is to provide the user the ability to associate bounding values, inspired by and enhancing the VFC^3 consistency model [5, 11], on data stored in HBase and therefore defining maximum divergence of the constraints:

- **Time (θ).** Specifies the maximum time a replica can be without being updated with is latest value. Considering $\theta(o)$ provides the time (e.g., seconds) passed since the last replica update of object o, constraint κ_θ, enforces that $\theta(o) < \kappa_\theta$ at any given time.

- **Sequence (σ).** Specifies the maximum number of updates that can be applied to an object without refreshing its replicas. Considering $\sigma(o)$ indicates the number of applied updates, this sequence constraint κ_σ enforces that $\sigma(o) < \kappa_\sigma$ at any given time.
- **Value (ν).** Specifies the maximum relative difference between replica contents or against a constant (e.g., top value). Considering $\nu(o)$ provides that difference (e.g., in percentage), this value constraint κ_ν enforces that $\nu(o)$ $<\kappa_\nu$ at any given time. It captures the impact or importance of updates on the objects internal state.

These constraints, coupled together with the transactional enforcement within the HBase cluster and causal relation tracking, will determine the creation of a schedule for the geo-replication of operations towards slave clusters.

This schedule has the challenge to improve upon the default eventual consistent replication provided by HBase by offering causal consistency not only between operations but groups of operations (Transactions) and at the same time to uphold the constraints defined by the VFC^3.

These constraints provide the ability to give urgency of replication to more important data and relax other data which might not be so urgent. In essence it allows awarding different quality of service to different data without breaking application semantics and the typical consistency assumptions developers make over accessed data. With this model, it will also be possible to infer a specific cost of replication for each operation, while enforcing the desired semantics. In Figure 1 we present a high-level view of the architecture of this project.

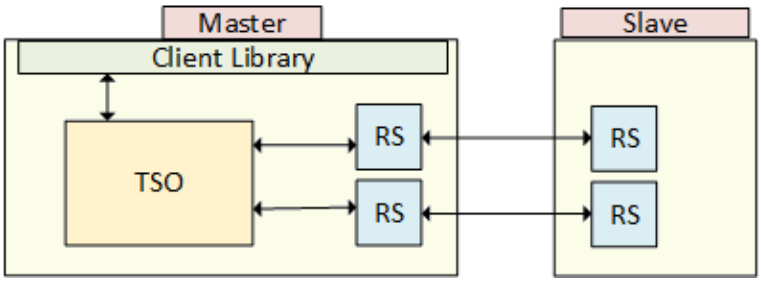

Fig. 1: The Transaction Status Oracle (TSO) state will be partially replicated in the client. It will push the epoch that results from the scheduling algorithm to the Region Servers (RS) for geo-replication.

The client library will communicate with the transaction Status Oracle (TSO) to coordinate their transactions, as transactions commit clients also submit the dependencies and progress on the bounding constraints. The TSO will reason about the serial order of operations and rearrange it to oblige to the constraints but without breaking causal relations. As the order is defined, the TSO pushes the queue to Region Servers (RS) in order for them to follow up with replicating

the operations to the other cluster. Each Region Server will look to the group of operations set to replicate, check which of them are they responsible for, and propagate them.

Transactional Enforcement and Causality Tracking. As described in section 3.3, Omid[4] is a system that provides transactional support for HBase with either snapshot isolation [3] or write-snapshot isolation [17] which guarantees serializability; it also relies on BookKeeper [8] for durability. We will leverage this as an underlying system for our transactions and extend it to track causal relations between client operations.

To add causal tracking to this system we simply need to add the transactions read and write set to the list of committed transactions already saved in the TSO. With this method we can track every potential causal relation between groups of operations. Another option with less storage and memory requirements is to track only explicit application defined causal relations where clients will, for each transaction, submit the nearest one-hop distance dependency [10] in the context of its session.

So in this second mode the client library, must maintain a causality graph where for each new transaction, submitted in a session, they will also submit the nearest one-hop dependency. In addition, they must also submit the constraint values, which are placed next to data they have accessed in the HBase tables as metadata, as is explained in the following section.

Bounding Values. Inside HBase we will store the user defined bounding values and associate them with the data whose constraints they are meant to represent. Four levels of granularity are considered here, ordered from the one with least storage and metadata overhead requirements to the most costly:

The first level, although very coarse it can fit some developer needs and requires the least storage requirements, is to associate a single bounding vector to an entire table. In Figure 2 is shown the second level, where different values for each row in a table are defined, making all columns in that row have the same bounds. Represented in Figure 3 is the next level of granularity that defines the values on a per column-family basis, covering all columns inside a column-family. In Figure 4, we show the highest demanding level but the most fine-grained where the bounding values can be defined differently for each cell leading to a column of metadata for each column.

	cf1:col1	cf1:col2	cf2:col1	cf2:col2	cf_vfc:col_vfc
row1	value	value	value	value	$(\theta, \sigma, v)_1$
row2	value	value	value	value	$(\theta, \sigma, v)_2$

Fig. 2: Different bounds for each row across all the column families (cf).

[4] http://yahoo.github.io/omid/

	cf1:col1	cf1:col2	cf1:col_vfc	cf2:col1	cf2:col2	cf2:col_vfc
row1	value	value	$(\theta, \sigma, v)_1$	value	value	$(\theta, \sigma, v)_3$
row2	value	value	$(\theta, \sigma, v)_2$	value	value	$(\theta, \sigma, v)_4$

Fig. 3: Different bounds for each row inside each column family (cf).

	cf1:col1	cf1:col1_vfc	cf1:col2	cf1:col2_vfc	cf2:col1	cf2:col1_vfc	cf2:col2	cf2:col2_vfc
row1	value	$(\theta, \sigma, v)_1$	value	$(\theta, \sigma, v)_3$	value	$(\theta, \sigma, v)_5$	value	$(\theta, \sigma, v)_7$
row2	value	$(\theta, \sigma, v)_2$	value	$(\theta, \sigma, v)_4$	value	$(\theta, \sigma, v)_6$	value	$(\theta, \sigma, v)_8$

Fig. 4: Different bounds for each cell.

4.2 Scheduling & Cost Inference

First we must define how the scheduling algorithm will reason about each operation's individual constraint values, and how it will define the constraint values of a transaction based on all of its operations. Considering all bound values are defined for individual operations, the bound closest to being broken will define that operation's level of urgency. The same logic will be applied to a transaction: when looking at all of its operations, the one that implies the most urgency will define the urgency of the entire transaction and enclosed operations, regardless of the presence of very relaxed urgency operations in the transaction.

The scheduling algorithm will reason broadly as follows: first, for all transactions it will identify the operations with the highest constraints and will define that constraint as the transaction constraint. It will then disregard arrival order and re-order transactions based on the constraint values but respecting causal order. For two independent causal dependencies of one transaction, the highest constraint one will be placed in front of the other in queue. This can be seen as a breath-first search where child selection is based on the highest constraint. For independent groups of transactions ordering will be solely based on constraint values and both groups will intertwine as it is shown in Figure 5.

As transactions are ordered with relation to other independent transactions, the idea is to satisfy all the constraints with the least cost possible. We define cost as the following:

- **Operation move cost.** The number of operations that had to be delayed in order for this operation to be propagated earlier (e.g. if an operation is moved from the 10th position to the 2nd, it has an associated cost of 8, as it causes the delay of 8 operations).
- **Total move cost.** The total cumulative number of queue positions of the moves made.

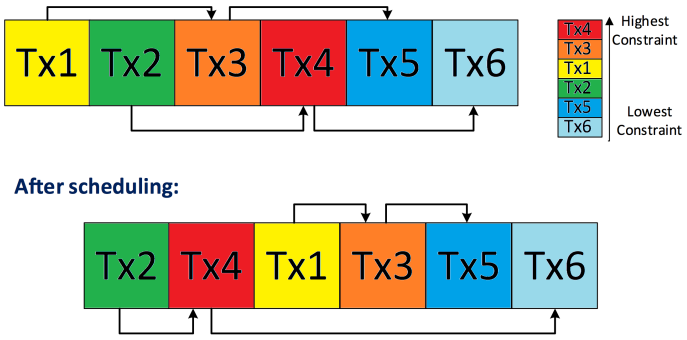

Fig. 5: The arrows represent the causal relations. Although Tx2 is less urgent than Tx1 it is placed first in the queue due to its causal relation with Tx4 and the need to satisfy Tx4 geo-replication.

This means we intend to minimize the amount of "moves" in the queue, done to operations with no causal relation, by stepping away the least possible from the serial order, but yet maintaining the constraints satisfied. Transactions can somewhat be viewed as a single operation due to the strictest constraint operation being the one that represents the bounding values of the transaction, but the cost of a move will be different as we move a transaction we are moving a certain amount of operations together.

4.3 Geo-Replication

As stated, the ordered queue is pushed to the Region Servers, so they can have the necessary info on the order of the updates to propagate. For this to work correctly, the TSO will have to make a decision about the head of the queue and define a final ordering that will not be changed, so it can be removed from the queue and safely pushed to Region Servers. This group of operations comprising the head will be called an *epoch*. The amount of updates to include in an epoch will be decided according to the size of the operations and their urgency to manage the lag and batching efficiently. For example if bounds are very close to being broken it is accepted that a smaller epoch is conceived so constraints are not broken. It is also accepted to delay somewhat the geo-replication (without breaking constraints) to create a larger epoch and augment the amount of updates in one commit protocol enforcement.

In the commit protocol (Figure 1) the TSO will act as a coordinator. It pushes the new epoch boundary to the Region Servers, and each one of them will analyse the epoch, pick out the updates they are in charge of, and propagate them to the slave cluster (phase 1). The RS on the slave cluster will acknowledge the reception and respond to the RS on the master (phase 2), similarly to the voting phase in two-phase commit but negative responses are not possible. On the master cluster the TSO will gather all responses and send commit command

along the new epoch (phase 3/phase 1 of the next round). This is very similar to the two-phase commit protocol but with a few improvements inspired by Sinfonia's [1] mini-transactions, and where replies are always positive and serve only to ensure consensus and global knowledge of epoch boundaries.

5 Evaluation

To evaluate our solution, we will focus on cost of the new added features in terms of how they will affect the performance and scalability of the system, versus the benefits these new features bring. We will make use of the PyTPCC HBase adaptation for transactional workloads and the YCSB benchmark.

- The correctness of the system must be evaluated, as to make sure causal relations are respected as the ordering algorithm acts. Another factor is the correctness of the commit protocol as to making sure the no later epoch is exposed before an earlier one, and also the behaviour in the presence of failures, when this is not achieved, quantify the extent and/or magnitude of the set of affected operations (with the metrics of Section 4.2 or with error measurements using geometric norms.

- The effectiveness of the scheduling algorithm to uphold the user specified constraints with the least possible cost, but it must be taken into account as above that certain constraint combinations will be simply impossible to serve. For example, in a worst case scenario where every operations has very strict constraints and no relaxed operations to leverage, the systems best option would be to simply default to serial order. We can thus assess the cost for each level of correctness and quality demands.

- In terms of causality tracking and constraint metadata we must evaluate the memory, storage and network overhead. Also the delay induced by the geo-replication commit protocol, for the propagation of epochs.

- The full behaviour of the system will be analysed in terms of performance and scalability. By emulating several clients, we plan to evaluate how the system will in terms of memory requirements in the TSO, identifying the point where queues growth speed will overcome the ability to propagate updates to slave data centres. We will also compare the performance of our geo-replication to the default HBase mode and balance the latency our enforcements will bring against the added features.

6 Conclusion

We have presented our proposal to improve HBase's eventual consistent geo-replicated approach with data urgency that abides to causal relations, by proposing a new scheduling algorithm. It will incorporate data bounding values and

causal relations into its reasoning as it disrupts serial order, as well as define a geo-replication protocol to enforce such ordering. We believe this algorithm will provide developers with the power to define and control the levels of priority on their data as well as reason about the cost of strict urgency data to the point where this cost can be extrapolated in a way to be related to monetary cost.

Acknowledgments: This work was supported by national funds through FCT Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2013.

References

[1] Marcos K Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch, and Christos Karamanolis. Sinfonia: a new paradigm for building scalable distributed systems. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 159–174. ACM, 2007.

[2] DGFFJ Benjamin and RM Yabandeh. Lock-free Transactional Support for Distributed Data Stores. *sigops.org*, pages 3–4.

[3] Hal Berenson, Elizabetb O Neil, Patrick O Neil, and Sybase Corp. A Critique of ANSI SQL Isolation Levels. pages 1–10, 1995.

[4] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[5] S Esteves, J Silva, and L Veiga. Quality-of-service for consistency of data geo-replication in cloud computing. *Euro-Par 2012 Parallel Processing*, pages 285–297, 2012.

[6] Patrick Hunt, Mahadev Konar, Flavio P Junqueira, and Benjamin Reed. Zookeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, volume 8, pages 11–11, 2010.

[7] Flavio Junqueira, Benjamin Reed, and Maysam Yabandeh. Lock-free transactional support for large-scale storage systems. *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 176–181, June 2011.

[8] Flavio P Junqueira, Ivan Kelly, and Benjamin Reed. Durability with bookkeeper. *ACM SIGOPS Operating Systems Review*, 47(1):9–15, 2013.

[9] Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency rationing in the cloud: Pay only when it matters. *Proc. VLDB Endow.*, 2(1):253–264, August 2009.

[10] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. Stronger semantics for low-latency geo-replicated storage. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI?13), Lombard, IL*, 2013.

[11] ÁG Recuero, S Esteves, IID Lisboa, and L Veiga. Quality-of-Data for Consistency Levels in Geo-replicated Cloud Data Stores. *gsd.inesc-id.pt*.

[12] Nuno Santos, Luís Veiga, and Paulo Ferreira. Vector-field consistency for ad-hoc gaming. *Middleware 2007*, 4834:80–100, 2007.

[13] Eric Schurman and Jake Brutlag. The user and business impact of server delays, additional bytes, and http chunking in web search. In *Presentation at the O?Reilly Velocity Web Performance and Operations Conference*, 2009.

[14] Doug Terry. Replicated data consistency explained through baseball. Technical report, Technical Report MSR-TR-2011-137, Microsoft Research, 2011.

[15] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 309–324, New York, NY, USA, 2013. ACM.

[16] Luís Veiga, André Negrão, Nuno Santos, and Paulo Ferreira. Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. *Journal of Internet Services and Applications*, 1(2):95–115, August 2010.

[17] Maysam Yabandeh and Daniel Gómez Ferro. A critique of snapshot isolation. *Proceedings of the 7th ACM european conference on Computer Systems - EuroSys '12*, page 155, 2012.