Towards quality-of-service driven consistency for Big Data management

Álvaro García-Recuero

INRIA, Rennes-Bretagne Atlantique Research Center,
Campus de Beaulieu,
35042 Rennes Cedex, France
E-mail: alvaro.garcia-recuero@inria.fr

Sérgio Esteves and Luís Veiga*

INESC-ID Lisboa-Distributed Systems Group,
Instituto Superior Técnico,
Universidade de Lisboa,
Rua Alves Redol, 9,
1000-029 Lisbon, Portugal
E-mail: sesteves@gsd.inesc-id.pt
E-mail: luis.veiga@inesc-id.pt
*Corresponding author

Abstract: With the advent of Cloud Computing, Big Data management has become a fundamental challenge during the deployment and operation of distributed highly available and fault-tolerant storage systems such as the HBase extensible record-store. These systems can provide support for geo-replication, which comes with the issue of data consistency among distributed sites. In order to offer a best-in-class service to applications, one wants to maximise performance while minimising latency. In terms of data replication, that means incurring in as low latency as possible when moving data between distant data centres. Traditional consistency models introduce a significant problem for systems architects, which is specially important to note in cases where large amounts of data need to be replicated across wide-area networks. In such scenarios it might be suitable to use eventual consistency, and even though not always convenient, latency can be partly reduced and traded for consistency guarantees so that data-transfers do not impact performance. In contrast, this work proposes a broader range of data semantics for consistency while prioritising data at the cost of putting a minimum latency overhead on the rest of non-critical updates. Finally, we show how these semantics can help in finding an optimal data replication strategy for achieving just the required level of data consistency under low latency and a more efficient network bandwidth utilisation.

Keywords: Cloud storage; consistency; replication; geo-replication; data stores; NoSQL; quality-of-service; big data intelligence.

Reference to this paper should be made as follows: García-Recuero, Á., Esteves, S. and Veiga, L. (2014) 'Towards quality-of-service driven consistency for Big Data management', *Int. J. Big Data Intelligence*, Vol. 1, Nos. 1/2, pp.74–88.

Biographical notes: Álvaro García-Recuero received a joint double European Master Degree in Distributed Computing at Instituto Superior Técnico and Royal Institute of Technology respectively as of 2013. His interests include but are not limited to, distributed systems, data mining and machine learning. He is the author of a conference paper at IEEE CloudCom 2013, which this article considerably extends for inclusion of more results relevant to its context. Currently he is pursuing a PhD with special focus on Big Data at Inria, a major institution for research in Europe.

Sérgio Esteves received his BS and MS in Computer Science and Engineering in 2009, from Instituto Superior Técnico (IST)-ULisboa, Portugal. He is a PhD student at IST and a researcher in the Distributed Systems Group at INESC-ID since 2009. His research interests include Cloud Computing; massive scale software systems for Big Data, workflow data management; and machine learning.

Luís Veiga is an Assistant Professor at Instituto Superior Técnico (IST)-ULisboa, and Senior Researcher in the Distributed Systems Group at INESC-ID Lisboa, Portugal. He received the Best Paper Award at ACM/IFIP/Usenix Middleware 2007, the Best Paper Award Runner Up at IEEE CloudCom 2013 and Best Young Researcher Award for 2012 at INESC-ID Lisboa.

He serves as Chair of the IEEE Computer Society Chapter (IEEE Section Portugal), steering committee member of the ACM/IFIP/Usenix Middleware Conference, and Executive Group Manager of the Distributed Systems Group of INESC-ID Lisboa. He coordinates the Cloud For Europe European project at INESC-ID.

This paper is a revised and expanded version of a paper entitled 'Quality-of-data for consistency levels in geo-replicated cloud data stores' presented at IEEE CloudCom 2013, Bristol, UK, December 2013.

1 Introduction

Distributed systems in general, rely on geo-located infrastructures around the globe (Nygren et al., 2010). This is due to their replicated built-in nature, which provides highly available and consistent data at good performance levels. In particular, the Cloud Computing model has widely embraced that idea, and proven to be very convenient for users and providers willing to deploy and manage large clusters of machines with such on-demand services that promise the best possible scalability and performance.

On the other hand, with large amounts of information requiring to be consistent while highly available, there is a continuous need to find suitable workarounds to the early problem stated in the CAP theorem (Gilbert and Lynch, 2002). Brewer stated that it was not possible to ensure the three of these properties in a distributed system all at once (consistency, availability and partition tolerance). In this sense applications were traditionally compromising and choosing between two out of those three. More often this trade-off was a matter of a binary choice among data consistency or availability, assuming partition tolerance was a must due to failures in the network. Although possible, partitions are actually rare enough to give up entirely either consistency or availability first of all. Therefore, this trade-off later evolved into a discussion about consistency versus latency, more formally defined in PACELC (Abadi, 2012). To understand the meaning of that trade-off, one can think about how strong and eventual implementations of consistency differ in the way they replicate data, synchronous or asynchronously respectively. To tackle that, there is an ongoing wave of research efforts around that area for ensuring fully strong consistent geo-replicated data storage without giving up on availability. In that regard, for instance Google recently released a yet more robust geo-replicated storage system for fulfillment of such extreme needs (Corbett et al., 2012). All that comes at the extra cost of implementing several synchronisation techniques on top of each other such as atomic schema changes and extended clocks accounting for uncertainty.

Therefore, the key is to understand how to keep such distributed systems scalable while still delivering good performance to applications. These architectural requirements may change from application to application, and so will their end goals. For instance, today each non-relational database optimises its architecture with a particular goal in mind and therein considers also different trade-offs. For instance Cassandra (Lakshman and Malik, 2010) quorum approach can provide more consistency at the expense of more latency as well, as it will need to update a majority of replicas prior to starting reading new data from any of them. While this can reduce data inconsistencies and staleness, it is also important to reduce latency while providing a sufficiently strong level of consistency that caters for all types of applications and data semantics. In practice, that means replicating data across geo-located data centers without incurring into long network delays and optimising bandwidth utilisation (García-Recuero et al., 2013).

There are a number of existing systems where data semantics are analysed in order to provide operations with faster (eventual) or slower (stronger) consistency without compromising performance (Li et al., 2012). In some, causal serialisation and therefore commutative updates are provided also based on data semantics, but require redesigning application data types (Shapiro et al., 2011) or intercepting and reflecting APIs via middleware (Esteves et al., 2012). Unlike linearisability, eventual consistency does work well for systems with shared distributed data that is often queried and/or updated. That is because updates can be performed on any replicas at any given time (Burckhardt et al., 2012). Most systems implement eventual consistency, in order to avoid expensive synchronous operations across wide area networks, while still maintaining data consistent through low latency operations in large geo-located deployments.

HBase is a well-known and widely deployed open source extensible record data-store (Cattell, 2011), written and inspired on the idea of BigTable (Chang et al., 2008), which targets the management of large amounts of information. HBase provides eventual consistency through replication between sites (inter-cluster). Eventuality is therefore the promise and for that purpose a write-ahead log is maintained.

In this paper, we outline the advantages of applying a quality-of-service (QoS) to data replication, namely quality-of-data (QoD), as described in García-Recuero et al. (2013). Therefore, we will refer to it as QoD from this point onward. The concept is implemented for HBase, and targeting applications which require stonger levels of data consistency than just eventual, while being highly-replicated and available. Previous work in the area, such as Snapshot Isolation techniques (Sovran et al., 2011), work within but not across data centers. In several other models, bounded consistency is modelled based on generality but not practicality (Yu and Vahdat, 2001). We find the latter

to be more useful to applications, as stated in Alvaro et al. (2013).

1.1 Contributions

The main contributions here focus on what other consistency properties can No-SQL data stores such as HBase provide when geographically distributed during replication. Using a data-centric based approach, such as *QoD*, enables application developers to consider semantics of updates in a storage system and tag them for replication in a self-contained manner. This work therefore applies that concept into HBase as a first use case.

We offer custom levels of consistency guarantees to applications, geared with data semantics and driven towards bandwidth resource optimisation. Therefore, the behaviour of the system can be tuned and the data semantics become the key decision-maker of a more efficient consistency paradigm for geo-located data stores. This is very relevant for Big Data stores such as HBase, where some eventually replicated updates might be necessary earlier than others. Unlike a uniform processing of updates during replication, with a QoD in place, one can aim at satisfying a more fine-grained data consistency and delivery model. Interestingly, service level objective (SLOs) management have also been proposed for HBase, but in order to handle application multi-tenancy performance (Wang et al., 2012).

On the other hand, we use QoD so we can perform more functional and reliable decisions from the data storage layer upwards in order to fulfil the needs of consolidated application workloads. This is a step forward from the strictly eventual or strong consistency model in most cross-site replicated storage deployments. And consequently, shall evolve into more flexible consistency models for Big Data management. Our implementation provides several levels of data consistency, namely QoD-consistency fulfillment. This is used to ensure consistency among a group of updates as they become available to applications. For that, the value of one, several or a combination of the three dimensions of the vector-field consistency model in (Veiga et al., 2010) can be used.

The vector-field model defines the following data-semantics: K (θ, σ, ν), representing Time, Sequence and Value respectively. To realise our consistency guarantees, we extend HBase client libraries in order to provide grouping of operations during replication, where each grouping can support the required level of consistency: ANY, IMMEDIATE, or even with a specific-custom bound on data-consistency.

1.2 Roadmap

In the next sections of the article, we offer a brief overview of fundamental consistency models and background work in this well-studied area of distributed systems, having special focus on the concept of eventual versus strong consistency, and what possible variations of the two exist in the middle of the spectrum. As an intermediate approach, we position *QoD*, applied to HBase leveraging the aforementioned three-dimensional vector-field model.

The rest of the article is organised as follows: related work in Section 2, our HBase extension architecture in Section 3, the implementation details in Section 4, and evaluation in Section 5. The evaluation results show that from the architectural point of view our solution integrates well in HBase and provides the intended data guarantees. Finally, with Section 6 we conclude. And, in Section 7, we point to some future hints for Cloud Computing regarding Big Data management.

2 Background and related work

The architecture of HBase is inspired in previous work at Google, BigTable (Chang et al., 2008), a distributed, persistent and multi-dimensional sorted map. HBase is being used for instance at Facebook data centres for holding and managing the storage of messages and user data, as in partial replacement of Cassandra (Lakshman and Malik, 2010). Cassandra offers replica-set consistency tuning, but not divergence bounded consistency regarding data semantics. In geo-replicated scenarios, HBase provides eventual guarantees to data consistency through remote procedure call (RPC) mechanisms and only inside of a same cluster location strong consistency is supported (Aiyer et al., 2012).

Eventual consistency might be sufficient in most cases. Although, complex applications require stronger consistency guarantees and can be difficult to manage. Due to that, there have been recent research efforts to address these shortcomings in geo-replicated data centers, with Google developing earlier in 2012 an evolution of BigTable that provides external consistency through atomic clocks for instance, Spanner (Corbett et al., 2012). This makes applications highly-available while ensuring as much synchronicity among distant replicas as possible and more importantly, atomic schema changes. Data locality is also an important feature for partitioning of data across multiple sites. Spanner does use Paxos for strong guarantees on replicas.

Strong consistency does not work well for systems where we need to achieve low latency. So the reason for most systems to use eventual consistency is mostly to avoid expensive synchronous operations across wide area networks. In other cases such as COPS (Lloyd et al., 2011) causality is guaranteed, although it does not guarantee the quality of data by bounding divergence, which can still lead to outdated values being read. Previous inspiring work from Yu and Vahdat (2000) also shows divergence bounding approaches to be feasible in that regard. On that topic, Kraska et al. (2009) proposed rationing the level of consistency on data, rather than transactions. The missing gap in all cases, and what mainly differentiates them from this work, is taking into account semantics that are based in data itself.

Systems such as PNUTS from Yahoo (Cooper et al., 2008), introduced a novel approach for consistency on a per-record basis, but not explicitly stating QoD levels. Even though, it became clear that it is possible to provide low latency during heavy replication operations for large web scale applications. As in our work, they provide finer-grained guarantees on certain data, so in other words, new updates are not always seen right away by the clients (which is also the case with *QoD*), but only if strictly necessary. Keeping that in mind, it is not mandatory for applications to be highly-available and consistent both at once. That is applicable to our use case. Yahoo made the point for eventual consistency not being enough, and as in the case of social networks, stale replicas can introduce privacy issues if not handled adequately. We propose using grouping of operations within QoD in order to resolve the consistency issues among blocks of updates, and their more efficient and straightforward management.

3 Consistency architecture

QoD stands for quality-of-data consistency, which applied to a cloud data store such as HBase, allows entries to be evaluated and sorted in a priority queue prior to replication. That is achieved by scheduling data-transfers according to their priority, in the context of the application requirements. The unit of replication, includes HBase column family (used for instance to organise data from different business users of the application). For experimental purposes we also define a data-container that extends that notion. Data-containers are simply a concatenation of existing row-fields into the data store schema where to apply data semantics (e.g., *identified as tableName:columnFamily*).

In this context, solutions like FIFO (Data Structures I and Program Design, 1984) queues can be effective when treating all incoming items in an equal manner (e.g., priority in starting to propagate and take items out of the queue). Unlike the previous approach, our implementation uses a priority queue that resembles deadlines as defined in the earliest deadline first (EDF) algorithm by Liu and Layland (1973). EDF sets priorities based on the task absolute deadline. Basically, the ealier the deadline, the higher the priority. There is actually, recent research regarding HBase with EDF (Zhang et al., 2013) but unlike that approach, our intuition mostly relies on the use of data semantics applied to the replication mechanisms into HBase. To the best of our knowledge, ours is the first work in the area which takes into consideration system-round knowledge about data semantics in order to decide which items are most relevant to applications, so replicated first. This is a very useful consideration if we take into account that we are talking about data, Big Data to be more specific, and the implications of replication over wide-area networks (a.k.a geo-replication). Facebook users suffer about 66% stale reads on average (Wada et al., 2011), which suggests to treat data according to its meaning during replication so serving a just enough up to date version when reading it next.

In order to achieve the aforementioned goals, our solution focuses on data semantics and leverage them through a vector-field consistency model, which results into a sorted priority queue engine, namely *QoD* for consistency. That allows for a combination of one or several of the fields of the vector to act as actual deadlines (always enforcing first the most restrictive deadline among the deadlines concerned with a particular column family or data-container). This is particularly helpful in geo-replicated scenarios where data stores can hardly provide strong consistency guarantees without losig the benefits of eventuality. Solutions like Windows Azure, claim to provide such strong consistency (Calder et al., 2011) but at first, really seemed more to be a mixed of both, eventual and strong (Calder and Atkinson, 2011), which proves the burdens of dealing with consistency.

The parameters in the vector, defined as K(θ, σ, ν), correspond to θ (data refreshing time interval), σ (maximum number of fresh-data misses), and ν (maximum value divergence in data inconsistency allowed) in each case. They are the building blocks that define which consistency priority is applied to each column family or data container used. Vector-fields act as consistency bounds. Therefore, client insertion of newer updates at one site involve a new iteration at the QoD engine for first, evaluating the actual values of the vector for the update in the given column family or data container. And secondly, comparing them with their existing maximum vector-bounds for determining if replication must yet occur. If the maximum vector-bound threshold since the last replicated version of the update is reached, its priority is increased and therefore immediately scheduled for propagation as well as the priority reset back to its initial status.

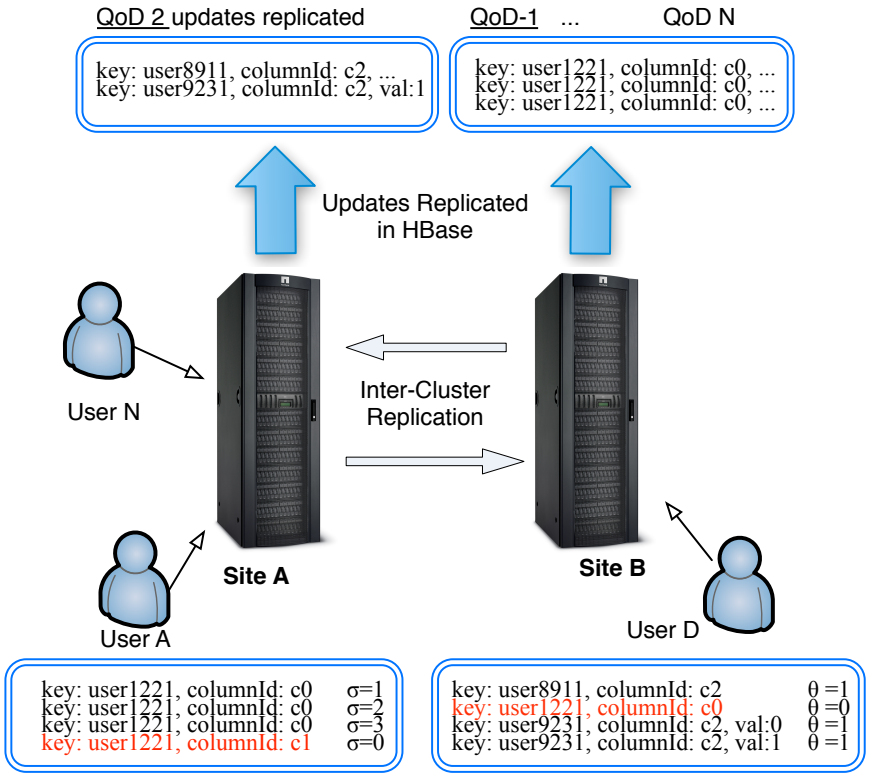
The time constraint can be always validated after every θ number of seconds, and the other two through Algorithm 1 as updates arrive. For the work presented here we use sequence (σ) as the main vector-field constraint to showcase the model in practice. For this, we define a set of customised data structures, which hold the values of database row items due to be checked for replication on a particular data container.

For instance, given actualK(θ, σ, ν) and containerMaxK (0,1,0) belonging to column family c0, we can determine updates assigned to that column family will only be replicated once the following condition is satisfied:

$$actualK(\sigma) >= containerMaxK(\sigma)$$

In order to track and compare QoD fields (which act as constraints during replication) against stored updates, data *containers* are defined for the purpose, controlling both current and maximum (then reset) bound values. Therefore, a QoD percentage is relative to the updates due to be propagated for a given vector-field combination (e.g., using σ).

Figure 1 HBase QoD high-level (see online version for colours)



QoD-1. **σ=3 for column family c0.** We increase value of σ up to 3 and replicate all items from c0 at that point.

QoD-2. **θ=1 for column family c2.** In this case θ is a time constant, so each second it replicates all items from c2 .

3.1 Typical distributed and replicated deployment

The underlying HDFS layer, is part of the HBase architectural design, which has built-in properties derived from it. In distributed clusters today Facebook (Harter et al., 2014) is currently using HDFS together with HBase in order to manage their messaging information across data centres. That is because of the layered architecture of HBase and its ability to handle both a short set of volatile data and ever-growing data, that rarely gets accessed more than once. In HBase, the WALEdit data structure can store data temporarily before being replicated, or be helpful when copying data between several HBase clusters. Originally the eventual consistency model provided, allows updates and insertions to be propagated asynchronously between clusters and Zookeeper (Hunt et al., 2010) is used for storing their positions in log files that hold the next log entry to be shipped in HBase. Therefore, if we can control the edits to be shipped, we can also decide what is replicated, when or in other words, how often.

3.2 Extensions to the HBase internal mechanisms

The QoD algorithm (shown in Algorithm 1) uses but needs to extend the WALEdit data structure for its purposes. We do so in order to retain meaningful information that supports the management of outgoing updates marked for replication within the *QoD* framework. We extend HBase, handling updates due to be replicated in a priority queue according to the QoD specified for each of their data containers. Thereafter, once the specified QoD threshold is reached, the working thread in HBase, in the form of Remote Procedure Call, collects and ships them as usual.

Algorithm 1 Simplified QoD algorithm using σ criteria (with time and value would be the same or similar) returns true means replicate

Require: $containerId$
Ensure: $maxBound \neq 0$ and $controlBound \neq 0$
1: **while** $enforceQoD(containerId)$ **do**
2: **if** $getMaxK(containerId) = 0$ **then**
3: **return** $true$
4: **else** $\{getactualK(containerId)\}$
5: $actualK(\sigma) \leftarrow actualK(\sigma) + 1$
6: **if** $actualK(\sigma) \geq containerMaxK(\sigma)$ **then**
7: $actualK(\sigma) \leftarrow 0$
8: **return** $true$
9: **else**
10: **return** $false$
11: **end if**
12: **end if**
13: **end while**

A high-level view of the mechanisms introduced with *QoD* are outlined in Figure 1, and it is based in a specific QoD bound in each case. This is applied to a defined data container in a per-user basis in this case. Replicating when the QoD is reached means here, every three updates for using σ in the case of QoD-1 for User A. Each second

for the User D with QoD-2 of θ field instead; in this case also showing the *last-writer wins* behaviour on the remote side, user N, for a same propagating remote data container value during replication. The architectural layout showcases a scenario accounting for propagation of updates during geo-replication.

3.3 Operation grouping

At the application level, it may be useful for HBase clients to enforce the same consistency level on groups of operations, despite the affected data containers possibly having different QoD bounds associated. In other words, there may be specific situations where write operations need to be grouped so that they can be all handled with the same consistency level and propagated atomically to slave clusters (*e.g., set of tightly-related log activities involved with a new status update on Facebook*).
For instance, publication of user status in social networks is usually handled at eventual consistency, but if they refer to new friends being added (*e.g., an update to the data container holding the friends of a user*), they should most likely be handled at a stronger consistency level to ensure they become visible atomically, along with the list of friends of the user, in respect to the semantics we describe here.

In order not to violate QoD bounds and maintain consistency guarantees, all data containers of operations being grouped must be propagated either immediately after the block execution, or when any of the QoD bounds associated to the operations has been reached. When a block is triggered for replication, all respective QoD bounds are naturally reset.

To enable this behaviour we devised extending the HBase client libraries to provide atomically consistent blocks. Namely, adding two new methods to HTable class in order to delimit the consistency blocks: *startConsistentBlock* and *endConsistentBlock*. Each block, through the method *startConsistentBlock*, can be parameterised with one of the two options:

1 *IMMEDIATE*, which enforces stronger consistency for the whole block of operations within itself.

2 *ANY*, which replicates groups of updates as a whole and as soon as the most stringent (smaller) QoD vector-field bound, associated with an operation inside the block, is reached.

4 Implementation details

To achieve the goals earlier described, we describe how to modify HBase libraries (HTable in particular). Regarding grouping of operations, that will be addressed from the source location into HBase before replication actually occurs, so apart from the multi-row atomically defined model in HBase, a more versatile system can also provide atomically replicated updates beyond row-level (e.g., column families or combinations of the fields in a row in HBase). This work is also an informal contribution which authors will aim to translate into a more formal addition to the ongoing efforts in HBase community for changing the Consistency model, 'pluggable replication framework' and branch hbase-10070 at *https://github.com/enis/hbase/tree/hbase-10070*.

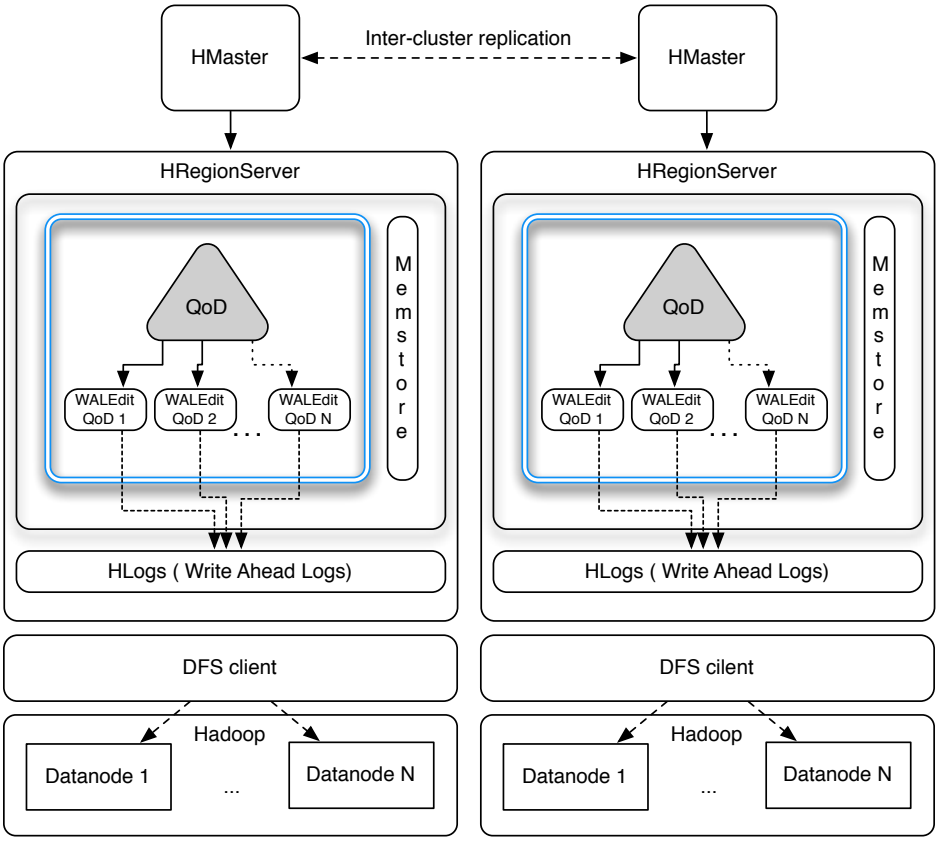
HBase replication mechanisms are based in a write ahead log (WAL), which is enabled in order to replicate information between distant data centers, and Zookeeper to keep track of the process and its state. The process of replication is currently carried out asynchronously, so there is not latency penalty for clients reading data and other processing can be in the meantime done at servers. Although, and since this process is not strongly consistent, in write heavy applications, a slave could still have stale data in the order of more than just seconds, and just until the last updates commit to local disk.

In our implementation, we overcome these pitfalls with a QoD approach, where we use a vector-model in order to handle updates selectively, depending of their classification and threshold in the online vector field (e.g., a maxBound for field σ in the three dimensional vector K). Therefore, we enforce on-demand delivery of updates at the remote cluster in all cases. For write intensive applications, that can be both beneficial in terms of reducing the maximum bandwidth peak-usage, while still delivering data according to application needs, and with improved semantics that take into account application logic processing.

The module in Figure 2 shows the implementation details introduced with QoD for consistency. We observe the module integrates into the core of the chosen cloud data store (HBase), intercepting incoming updates, and processing them into a priority queue, which is defined to store those for later replication to a given slave cluster.

Changing the logic of the shipping of edits in the write-ahead log, this process is therefore performed now according to the data semantics we define. Several data structures are required, some of them existing in HBase, as the WALEdit. That is in order to access different data containers, that we later query, to determine where and how to apply a given QoD at the column level (e.g., tablename:columnFamily). The data is replicated once we check the conditions shown in Algorithm 1 are met, and replication is triggered if there is a match with the threshold value in one, several or a combination of the existing and defined vector-constraints in each case (e.g., σ). As a remark then, the use of a QoD is also applicable for a selection of updates based into a combination of any of the fields from the three-dimensional vector, not just σ.

Figure 2 QoD for consistency (see online version for colours)



5 Simulation and evaluation

5.1 Overview

There has been extensive research with Consistency management on Cloud data stores for achieving the desired level of data synchronisations among distant data centers and similarly data replicas. Although, there is yet a fundamental aspect to cover in that spectrum regarding consistency and response times for SLA-based applications relying on distributed cloud storage systems. Big Data processing aims at establishing how to handle high volumes of information at large scale while still providing applications with a consistent and on-time data delivery. For that, several algorithms have been proposed in the area of Consistency in distributed systems, for instance Shapiro et al. (2011), and Lloyd et al. (2011). Also, in much earlier works such as Tarr and Clarke (1998), it is already proposed how an approximation to the idea of providing a wide range of data semantics is therefore desirable and necessary in several ranges of applications. In this sense, a QoD model is able to support these specific needs.

It has been already verified and presented in other reports and projects in the area of Hadoop, that a statically defined replication level is in itself insufficient, which therefore must be addressed and more efficiently adjusted in order to keep up with the scheduling of tasks (Lie et al., 2013). That is also related to the work here covered within HBase, as HDFS is its storage layer. A workaround on static replication constraints in HDFS and HBase is offering and enforcing on-demand replication with QoD for consistency. During evaluation of the model, a test-bed of several HBase clusters has been deployed, having some of them using the QoD engine enabled for consistency between remote replicas, and others running a regular implementation of HBase 0.94.8. All tests were conducted using six machines with an Intel Core i7-2600K CPU at 3.40 GHz, 11,926 MB of available RAM memory, and HDD 7200RPM SATA 6 Gb/s 32 MB cache, connected by 1 Gigabit LAN.

5.2 CPU micro-benchmarking of performance

We care about performance in terms of throughput and for that, we use a micro-benchmark based on built-in UNIX tools that shows how a given QoD does not highly impact performance. As observed in Figure 7, the throughput of the system itself is maintained during benchmarking of HBase with QoD enabled. A minimal difference in the expected throughput is concluded to be irrelevant mostly due to the obtention of similar results during several rounds of running the same input workload on the data store.

Next we conducted as shown in Figure 8, and *dstat* (Wieers, 2013) presents, an experiment to monitor the CPU usage using QoD for consistency. CPU consumption and performance remains roughly the same and therefore stable in the cluster machines as can be appreciated.

5.3 Analysis of workload QoD performance with YCSB

In order to realise the impact of the architecture modifications within HBase using a QoD, we define a set of workloads that are in relation to the percentage of reads and writes to an application as simulated in Yahoo Cloud Service Benchmark (YCSB).

We test that then with built-in workloads from YCSB plus a custom workload to intensively stress the data store with a high number of writes. This is the way updates in social networks target cloud data stores and as previously mentioned, most of them are mainly suffering from synchronisation issues regarding consistency of data during changes and new incoming updates. Measurements in regards to the following workloads and obtained results are outlined in Figures 6 to 12.

Figure 9 shows three different sets of QoD for the same workload (A).

YCSB workload A (R/W – 50/50)

- No QoD-consistency enforced.
- QoD-consistency fulfillment of σ = 0.5% of total updates to be replicated.
- QoD-consistency fulfillment of σ = 2% of total updates to be replicated.

During the execution of workload A, in Figure 9, the highest peaks in replication traffic are observed without any type of QoD consistency level, i.e. just using a regular HBase cluster. This is due to the nature of eventual consistency itself and the existing buffering mechanisms of HBase for replication.

With a QoD enabled as shown in the other two graphs, we rather control traffic of updates from being unbounded to a limited size, and accordingly saving resources' utilisation, while suiting applications that require smaller amounts of updates as they are only propagated as a group, when they are just needed.

We observe that a higher QoD implies replication traffic less often, although interactions reach high values on Bytes as they need to send more data. Smaller QoD optimises the usage of resources while sending priority updates more frequently (this could be the case of wall posts in a social network).

YCSB workload A modified (R/W – 0/100)

- No QoD-consistency enforced.
- QoD-consistency fulfillment of σ = 0.5% of total updates to be replicated.
- QoD-consistency fulfillment of σ = 2% of total updates to be replicated.

In Figure 10 we can see how a write intensive workload performs enabling QoD. As expected, results correlate the intuition in Figure 9 (please note the scale of the Y axis is modified in order to show the relevant difference in Bytes more accurately).

For smaller QoD (0.5%), overall we see lower peaks in bandwidth usage than with plain HBase, as well as in the following measurement used for QoD 2.0% (having that one higher maximum peak values than the previous QoD). Finally, HBase with no modifications shows a much larger number of Bytes when coming to maximum bandwidth consumption. Note we are not measuring, or find relevant, in any of these scenarios, to realise savings on average or total bandwidth usage (that would imply avoiding the propagation of earlier updates to containers, overwritten by subsequent modifications, as in state-transfer systems (Veiga et al., 2010)). The main goal of the system implementation is to have a way of controlling the usage of several resources in a data center, storage and bandwidth allocation. Also, to be able to leverage, and make more agile, the trading of strong for eventual consistency with a more robust atomic grouping of operations using vector bounded data semantics.

YCSB – Workload B

- Read/update ratio: 95/5.
- Default data size: 1 KB records (10 fields, 100 bytes each, plus key).
- Request distribution: zipfian.
- No QoD-consistency enforced.
- QoD-consistency fulfillment: σ = 10% of total updates to be replicated.
- QoD-consistency fulfillment: σ = 20% of total updates to be replicated.

Figure 11 shows the overall replication overhead with and without QoD enabled, for a Read intensive workload. The graph is significantly different from previous workloads here presented in terms of updates being replicated. That is due to the small fraction of writes in the workload, when compared to the percentage of QoD bounds for replication applied. If the QoD σ value was too high, then the activity on the network would decrease for longer periods, replicating updates rather later but in larger and higher bandwidth consuming batches than with a lower QoD. Therefore, increasing the amount of updates will result in more network utilisation, but for this particular workload, only a very limited amount of writes are going through the QoD for consistency module as required.

YCSB – Workload F

- Read/update ratio: 50/50.
- Default data size: 1 KB records (10 fields, 100 bytes each, plus key).
- Request distribution: zipfian.
- No QoD-consistency enforced.
- QoD-consistency fulfillment: σ = 20% of total updates to be replicated.
- QoD-consistency fulfillment: σ = 40% of total updates to be replicated.
- QoD-consistency fulfillment: σ = 60% of total updates to be replicated.

In the case of Figure 12, lower QoD values for σ slightly affect the height of the peaks of network communication during replication. This is due to the same reason as noted before: a bound on data staleness also puts a limit on the number of updates sent at the same time over the wire. In the case of σ = 60%, the replication overhead is kept acceptable and constant in respect to the previous with σ = 40%. This is as well due to the number of updates being issued running the workload, meaning that there is an upper ceiling with such access pattern. There is no need or advantage in batching more updates per second, unless the number of operations for this particular workload increased. Later on, we see how the QoD of σ = 0% (no-QoD in other words) has higher bandwidth consumption per second as expected.

5.4 Assessing data freshness

In order to assess data freshness, as observed by clients, a set of clients is set up so that a client is writing to a master cluster, and another reading from the slave. The writing client inserts 10 blocks of 100 updates interleaved between critical and non-critical into two different data containers with different QoD bounds. Therefore, it can be observed when, and which type of update, arrives at the destination by checking their delivery timestamp. That is also based on the data semantics offered by QoD for consistency.

5.4.1 Data arrival measured on sets of updates received

In Figure 3, we show how the latency varies by referring to the update timestamps. Higher QoDs approximate critical updates (in red) more to the beginning of the *Timeline*, while non-critical (green) keep being moved towards the right (later reading). We have therefore a better data freshness metric, in terms of critical updates, by prioritising them through our QoD for consistency engine. Critical updates move closer to the beginning of the time-line as σ bound in K vector increases, so that they actually receive higher priority during the replication process.

The results in Figure 3 are followed by Figure 4, the latter explains and validates the goal of grouping of critical versus non-critical operations. What we observe is, on a per-update basis, how the critical updates are favoured and tagged for replication first. This is inline with both, the intended implementation details, and the goal of providing certain data depending to which applications earlier at all given times.

5.4.2 Data arrival measured in a per update basis

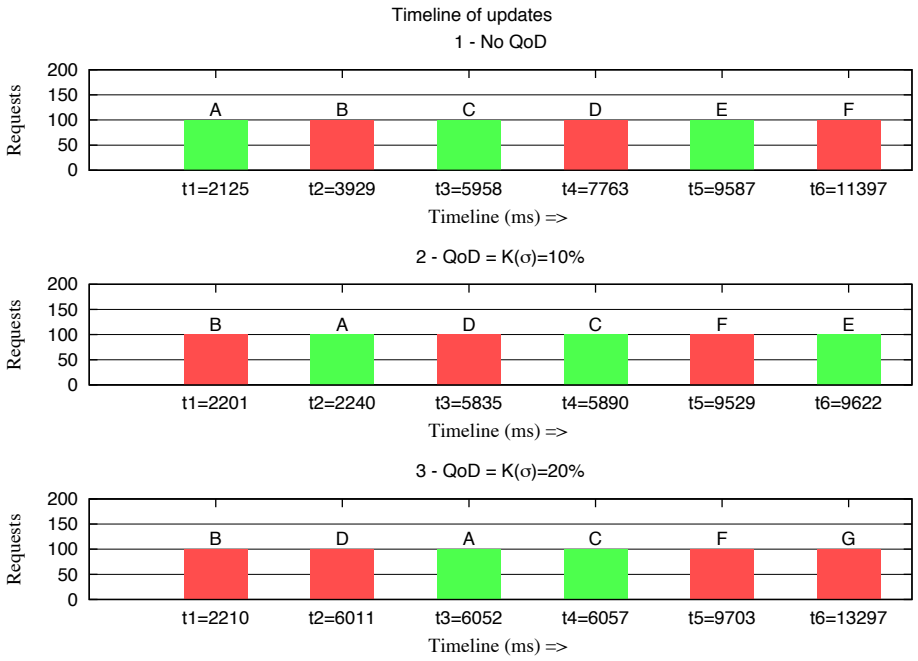
In this setup there is a client writing to a master HBase server using QoD for consistency, which writes 1,000 updates randomly mixed between critical and non-critical. We are introducing a delay of 40 ms in the network in order to emulate wide are network operation: the delay is set between master and slave cluster communication. For best readability, we are just showing a subset of the updates sent over time, from client 1 to the master cluster, and later read by client 2 from the replica at the slave cluster.

In Figure 5 it is represented the arrival of non-critical updates with different QoD applied for two different data containers. It is important to note there are two types of updates here, so for each QoD, one needs to take into account that not every remaining update not represented in this graph will exhibit the same behaviour; but approximately, all non-critical should arrive later on, or near the baseline of No-QoD while critical ones are prioritised. The horizontal lines denote the logic of shipment of updates once the QoD deadline expires, together.

Regarding maximum delay given the type of update, non-critical updates have higher time-stamps than the others and therefore arrive later as verified in Figure 5. In the case of being critical actually they do get read earlier in comparison to the baseline No-QoD (Figure 6) in most of the cases.

Figure 6 highlights how more critical updates arrive earlier in a per update basis over time. The larger is the QoD 'of the non-critical data container', the earlier critical updates are received by a slave cluster/data centre. The more latency or network overhead there is, the higher this difference appears to be. One could therefore trade-off on how much sooner critical updates arrive when the QoD of non-critical updates is relaxed.

Figure 3 Freshness of updates with several QoD-consistency bounds (see online version for colours)



Notes: Critical updates (in red), non-critical (in green)

Figure 4 Resulting flow of updates in a multi-site environment with HBase (see online version for colours)

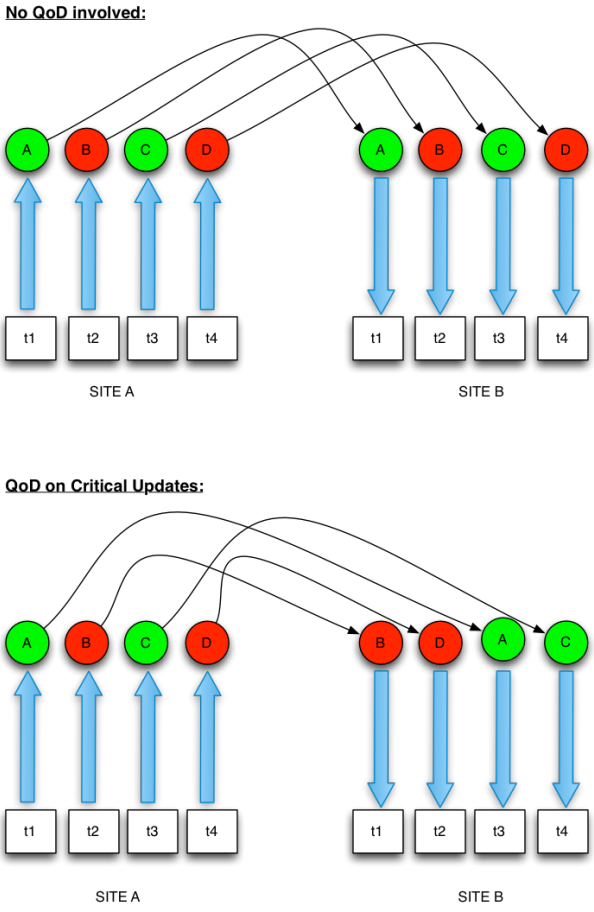


Figure 5 Impact of QoD in arrival time of non-critical updates (see online version for colours)

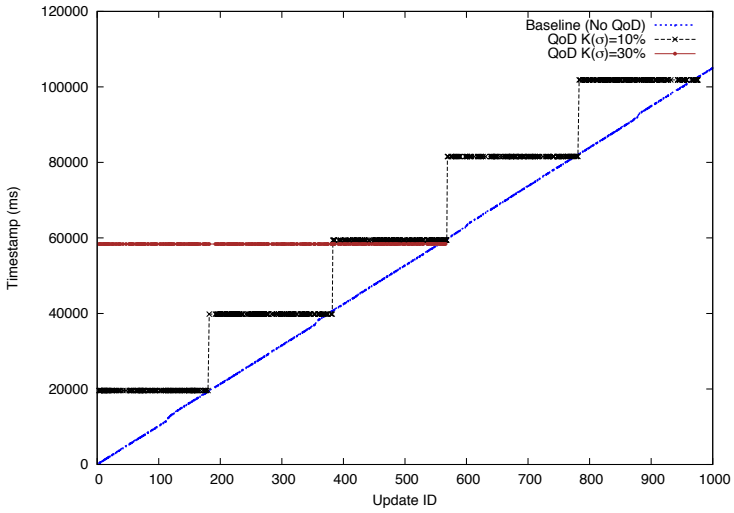


Figure 6 Impact of QoD in arrival time of critical updates (see online version for colours)

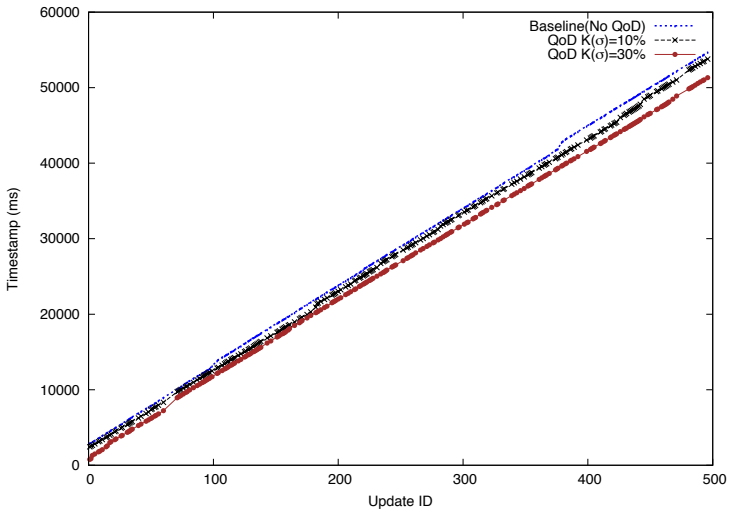


Figure 7 Throughput for several QoD configurations (see online version for colours)

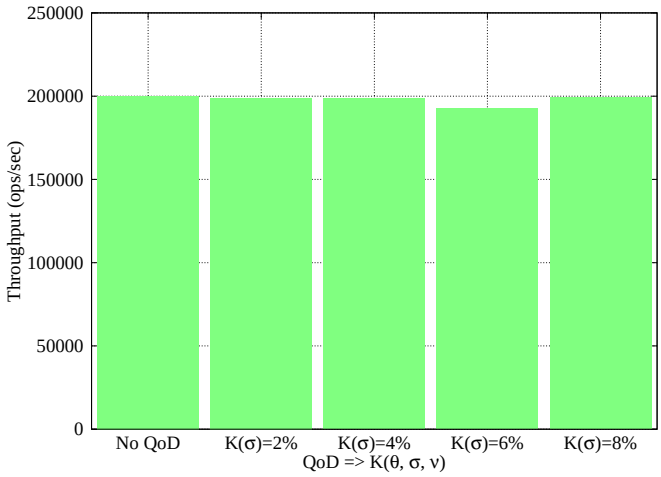


Figure 8 CPU usage over time with QoD enabled (see online version for colours)

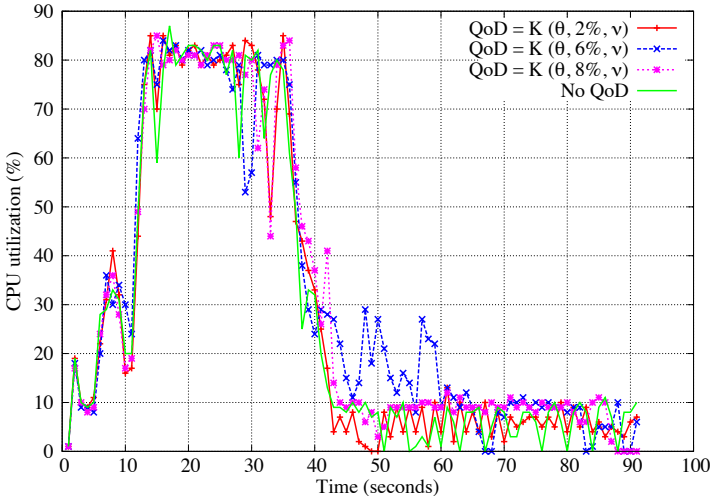


Figure 9 Bandwidth usage for Workload A with zipfian distribution, using 5M records using QoD bounds of 0.5 and 2% in the σ of K (see online version for colours)

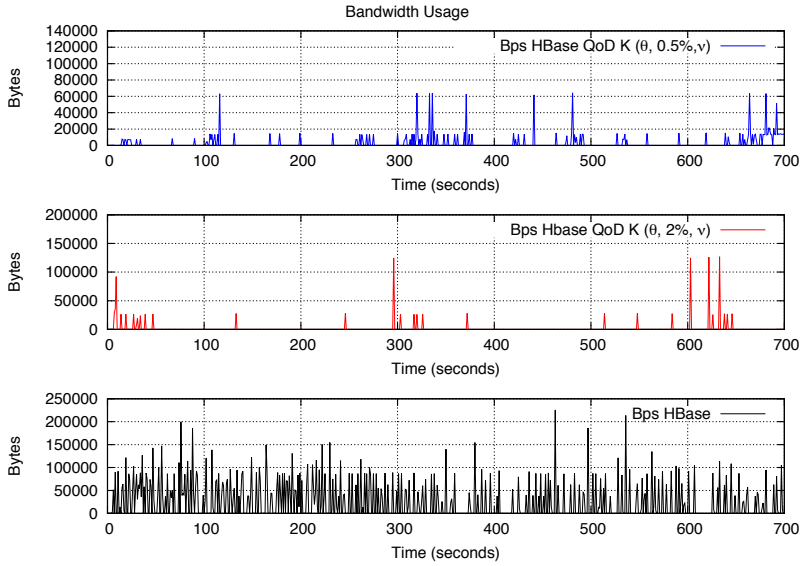


Figure 10 Bandwidth usage for custom Workload with uniform distribution, using 5M records and QoD bounds of 0.5 and 2% in the σ of K (see online version for colours)

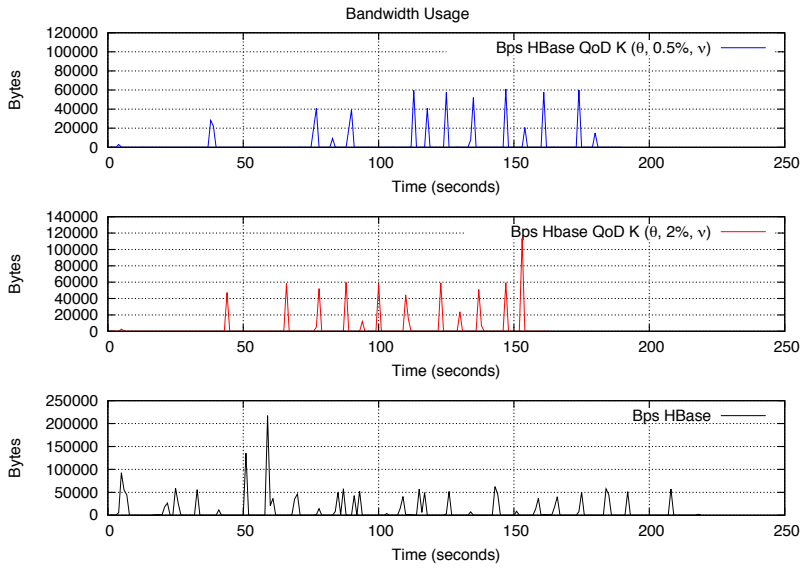


Figure 11 Bandwidth usage for Workload B using 500K operations in a total of 500K records using different QoD bounds for σ in K (see online version for colours)

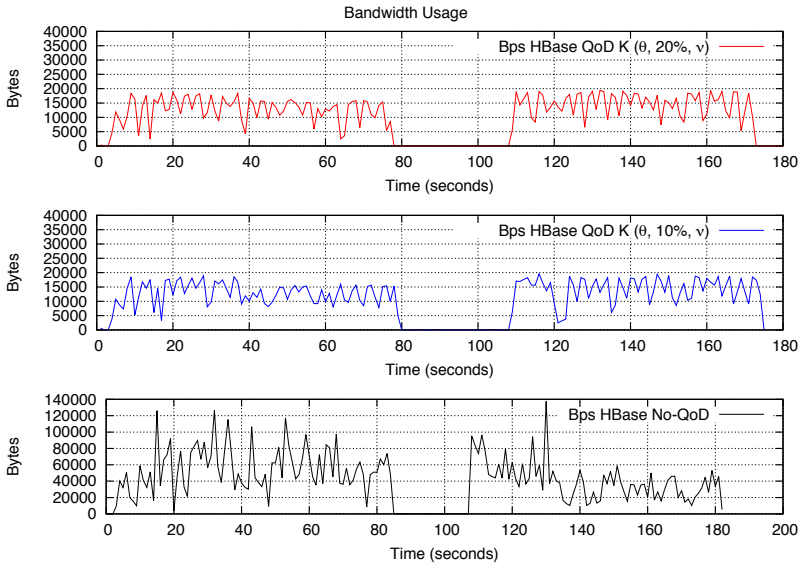
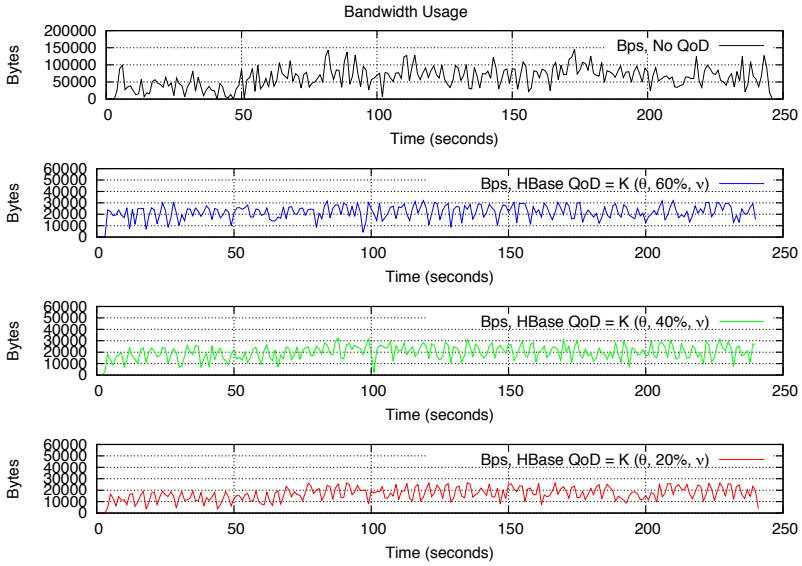


Figure 12 Bandwidth usage for Workload F using 500K operations in a total of 500K records using different QoD bounds for σ in K (see online version for colours)



6 Conclusions

Throughput performance with HBase improves as the number of resources increases. This has been realised in Carstoiu et al. (2010). Despite of the continuous drop in market prices regarding commodity hardware, it might not always be possible for existing data center facilities to dynamically adapt to the need of a growing population of machines. Following that trend, we continue seeing as of 2013, how large Internet companies such as Facebook keep investing in modern infrastructures that can collocate a high amount of resources across not just one, but several sites.

In this article we have introduced a consistency and replication mode for HBase that uses QoD to envision a very well-defined goal: adaptive consistency than one can tune in order to provide applications with just the required and on time information. All that, while saving on replication traffic and pursuing further improvements in regards to geo-replication techniques in Distributed Systems. That is, by defining boundaries through data semantics in a data store unit of storage, namely container (e.g. TableId:ColumnId, table name plus column family name), priority to each update can be assessed, established accordingly and used for replication under timely and on-demand application requirements.

With the consistency semantics presented, one can choose to trade-off short-timed consistency for wide area network bandwidth, which will translate into better bandwidth utilisation during peak-loads. This will significantly help to reduce replication overhead between distant data centers, in conditions where bandwidth

becomes an issue due to workload multi-tenancy and/or excessive latency in the network infrastructure. We evaluated our implementation on top of two HBase separate clusters.

The initial implementation is for HBase but can be extended for other platforms such as MongoDB (currently using Eventual consistency). This article brings insights about how data stores of this new Big Data era will handle data according to its semantics, whereas ultimate decision making for Consistency might be best integrated at the core of the system itself, avoiding middleware performance overheads (Das et al., 2010). In our case, experimental results indicate that we are able to maintain acceptable and enough client throughput, while reducing latency high percentiles, as well as optimise bandwidth usage. In the future, we aim to conduct more experiments using several other infrastructures from partner-institutions and public Cloud providers such as Amazon EC2 once resources become available.

7 Future quality-of-service in Cloud computing

7.1 Timeline consistency

From the point of view of Consistency, we believe that data-semantics based solutions, as here introduced, can be further developed for the provision of enhanced and adaptive replication paradigms. This will provide applications with just the required and consistent data at each point in time, despite of any faults, partitions or congestion in the underlying network fabric. Administrators and developers can easily tune any of the vector fields in the framework to be used as bounds, in order to perform selective replication in a more controlled and timely-fashion than typical eventually consistent approaches (e.g., that being the case of asynchronous replication in data stores such as HBase). Further enhancements can include innovative machine learning techniques that are able to classify data on-the-fly according to data usage properties and/or in accordance to defined application consistency needs.

7.2 QoS for Big Data management

When talking about QoS for data consistency, one should take into account the characteristics of the data being served; therefore, sites containing information with stricter SLO requirements on data delivery should provide not only the illusion of, but actually an accurate, accountable service to end-users. This is the case for many cloud-based storage providers, which promise to shape the future of Big Data. Therefore, with storage just becoming a commodity and large amounts of information available, there will be further efforts in the industry to provide efficient yet different quality of services to different customers, whereas the quality of the data being served in terms of semantics, will matter more than ever before.

Acknowledgements

This work was partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, projects PTDC/EIA-EIA/113613/2009, PTDC/EIA-EIA/108963/2008, PEst-OE/EEI/LA0021/2013.

The code implementation can be found in our repository online at: https://github.com/algarecu/hbase-0.94.8-qod.

References

Abadi, D. (2012) 'Consistency tradeoffs in modern distributed database system design: cap is only part of the story', *Computer Journal, IEEE Computer Society*, Vol. 45, No. 2, pp.37–42.

Aiyer, A.S., Bautin, M., Chen, G.J., Damania, P., Khemani, P., Muthukkaruppan, K., Ranganathan, K., Spiegelberg, N., Tang, L. and Vaidya, M. (2012) 'Storage infrastructure behind facebook messages: using hbase at scale', *IEEE Data Eng. Bull.*, Vol. 35, No. 2, pp.4–13.

Alvaro, P., Bailis, P., Conway, N. and Hellerstein, J.M. (2013) 'Consistency without borders', in *Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC '13*, pp.23:1–23:10, ACM, New York, NY, USA.

Burckhardt, S., Leijen, D., Fahndrich, M. and Sagiv, M. (2012) 'Eventually consistent transactions', in *Proceedings of the 21st European conference on Programming Languages and Systems, ESOP '12*, pp.67–86, Berlin, Heidelberg.

Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., Haridas, J., Uddaraju, C., Khatri, H., Edwards, A., Bedekar, V., Mainali, S., Abbasi, R., Agarwal, A., ul Haq, M.F., ul Haq, M.I., Bhardwaj, D., Dayanand, S., Adusumilli, A., McNett, M., Sankaran, S., Manivannan, K. and Rigas, L. (2011) 'Windows azure storage: a highly available cloud storage service with strong consistency', in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11*, pp.143–157, ACM, New York, NY, USA.

Calder, B. and Atkinson, M. (2011) 'Introducing geo-replication for windows azure storage, http://blogs.msdn.com/b/windowsazurestorage/archive/2011/09/15/introducing-geo-replication-for-windows-azure-storage.aspx (accessed December 2013).

Carstoiu, D., Cernian, A. and Olteanu, A. (2010) 'Hadoop hbase-0.20.2 performance evaluation', in *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*, May, pp.84 –87.

Cattell, R. (2011) 'Scalable sql and nosql data stores', *SIGMOD Rec.*, May, Vol. 39, No. 4, pp.12–27.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E. (2008) 'Bigtable: a distributed storage system for structured data', in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, OSDI '06*, Vol. 7, pp.15–15, USENIX Association, Berkeley, CA, USA.

Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H-A., Puz, N., Weaver, D. and Yerneni, R. (2008) 'Pnuts: Yahoo!'s hosted data serving platform', *Proc. VLDB Endow.*, August, Vol. 1, No. 2, pp.1277–1288.

Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R. and Woodford, D. (2012) 'Spanner: Google's globally-distributed database', in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI '12*, pp.251–264, USENIX Association, Berkeley, CA, USA.

Das, S., Agrawal, D. and El Abbadi, A. (2010) 'G-store: a scalable data store for transactional multi key access in the cloud', in *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10*, pp.163–174, ACM, New York, NY, USA.

Data Structures I and Program Design (1984) Prentice-Hall, Inc. Div. of Simon, I& Schuster, Englewood Cliffs, New Jersey, p.150.

Esteves, S., Silva, J. and Veiga, L. (2012) 'Quality-of-service for consistency of data geo-replication in cloud computing', in *Proceedings of the 18th International Conference on Parallel Processing, Euro-Par 12*, pp.285–297, Berlin, Heidelberg.

García-Recuero, Á., Esteves, S. and Veiga, L. (2013) 'Quality-of-data for consistency levels in geo-replicated cloud data stores', in *IEEE CloudCom*, IEEE, September.

Gilbert, S. and Lynch, N. (2002) 'Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services', *SIGACT News*, June, Vol. 33, No. 2, pp.51–59.

Harter, T., Borthakur, D., Dong, S., Aiyer, S., Tang, L., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H. (2014) 'Analysis of hdfs under hbase: a facebook messages case study', in *12th USENIX Conference on File and Storage Technologies*, USENIX, February.

Hunt, P., Konar, M., Junqueira, F.P. and Reed, B. (2010) 'Zookeeper: wait-free coordination for internet-scale systems', in *Proceedings of the USENIX Conference on USENIX Annual Technical Conference, USENIXATC '10*, pp.11–11, USENIX Association, Berkeley, CA, USA.

Kraska, T., Hentschel, M., Alonso, G. and Kossmann, D. (2009) 'Consistency rationing in the cloud: pay only when it matters', *Proc. VLDB Endow.*, August, Vol. 2, No. 1, pp.253–264.

Lakshman, A. and Malik, P. (2010) 'Cassandra: a decentralized structured storage system', *SIGOPS Oper. Syst. Rev.*, April, Vol. 44, No. 2, pp.35–40.

Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N. and Rodrigues, R. (2012) 'Making geo-replicated systems fast as possible, consistent when necessary', in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, OSDI '12*, pp.265–278, USENIX Association, Berkeley, CA, USA.

Lie, Y., Antoniu, G., Ibrahim, S., Chihoub, H. and Bouge, L. (2013) *Optimizing Energy Consumption in Cloud Storage Systems using Multiple Consistency Protocols*, Technical report, INRIA Rennes-Bretagne Atlantique.

Liu, C.L. and Layland, J.W. (1973) 'Scheduling algorithms for multiprogramming in a hard-real-time environment', *Journal of the ACM (JACM)*, Vol. 20, No. 1, pp.46–61.

Lloyd, W., Freedman, M.J., Kaminsky, M. and Andersen, D.G. (2011) 'Don't settle for eventual: scalable causal consistency for wide-area storage with cops', in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11*, pp.401–416, ACM, New York, NY, USA.

Nygren, E., Sitaraman, R.K. and Sun, J. (2010) 'The akamai network: a platform for high-performance internet applications', *SIGOPS Oper. Syst. Rev.*, August, Vol. 44, No. 3, pp.2–19.

Shapiro, M., Preguiça, N., Baquero, C. and Zawirski, M. (2011) 'Conflict-free replicated data types', in *Stabilization, Safety, and Security of Distributed Systems*, pp.386–400, Springer.

Sovran, Y., Power, R., Aguilera, M.K. and Li, J. (2011) 'Transactional storage for geo-replicated systems', in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11*, pp.385–400, ACM, New York, NY, USA.

Tarr, P. and Clarke, L.A. (1998) 'Consistency management for complex applications', in *Proceedings of the 20th International Conference on Software Engineering, ICSE '98*, pp.230–239, IEEE Computer Society, Washington, DC, USA.

Veiga, L., Negr ao, A.P., Santos, N. and Ferreira, P. (2010) 'Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency', *J. Internet Services and Applications*, Vol. 1, No. 2, pp.95–115.

Wada, H., Fekete, A., Zhao, L., Lee, K. and Liu, A. (2011) 'Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective', in *CIDR*, Vol. 11, pp.134–143.

Wang, A., Venkataraman, S., Alspaugh, S., Katz, R. and Stoica, I. (2012) 'Cake: enabling high-level slos on shared storage systems', in *Proceedings of the Third ACM Symposium on Cloud Computing, SoCC '12*, pp.14:1–14:14, ACM, New York, NY, USA.

Wieers, D. (2013) 'Dstat: versatile resource statistics tool' [online] http://dag.wiee.rs/home-made/dstat/ (accessed December 2013).

Yu, H. and Vahdat, A. (2001) 'Combining generality and practicality in a conit-based continuous consistency model for wide-area replication', in *Distributed Computing Systems, 21st International Conference on.*, pp.429–438.

Yu, H. and Vahdat, A. (2000) 'Design and evaluation of a continuous consistency model for replicated services', in *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation, OSDI '00*, Vol. 4, pp.21–21, USENIX Association, Berkeley, CA, USA.

Zhang, C., Wang, K. and Mu, H. (2013) 'A priority queue algorithm for the replication task in hbase', *Journal of Software*, (1796217X), Vol. 8, No. 7.