HBase-QoD: Vector-Field Consistency for Replicated Cloud Storage

Álvaro García Recuero
Instituto Superior Técnico (IST)
Lisbon, Portugal
alvaro.recuero@ist.utl.pt

***Abstract*—There has been extensive recent research in distributed systems regarding consistency in geo-replicated systems. NoSQL databases have introduced a new paradigm in this field, where user data location and replication mechanisms work together to be able to offer best-in-class distributed applications. As such, that means running costly operations under lower system latency while still ensuring data read is still valid for application logic processing. Therefore, several consistency models (like the eventual) use a flexible data consistency approach, that consists in taking advantage of those where staleness of data can be tolerated temporarily.**

We materialize that the idea in the context of an extensible record store. We assign priority to data depending of data semantics and serve it to applications when it is just required, avoiding overloading the network during large periods of disconnection or partitions. It is a practical and flexible approach that currently is not implemented in HBase as such, but even though, it is a desired feature in some community forums of the Apache project itself. We will introduce the data semantics used for non-relational data stores and built around the core architecture of an existing well-known system as such, enabling it to trigger data replication selectively with the support of a three-dimensional field vector. In addition, grouping of operations is possible, ensuring atomic changes and therefore valid updates are propagated.

Using this paradigm we can realize bandwidth savings in simulated peak loads scenarios in regard to high overhead of replication between data centers. We still provide different levels of consistency to different users or applications needs (e.g., regarding timeliness, number of pending updates and divergence bounds). We verify the implementation of the Quality-of-Data based vector on a set of distributed clusters. Levels of data consistency are enforced while performance of the cluster is monitored to ensure it remains stable, and updates can be tagged and grouped atomically in logical batches, akin to transactions.

I. Introduction

In Cloud Computing, replication of data in distributed systems is becoming a major challenge with large amounts of information that require consistency and high availability as well as resilience to failures. Nowadays, there are several solutions to the problem, none of them applicable in all cases, as they are determined by the type of system built and its end goals. As the CAP theorem states [1], one can not ensure the three properties of a distributed system all at once, therefore applications have to compromise and choose two out of three between consistency, availability and tolerate or not partitions in the network. Several other relaxed consistency techniques have been also devised in the area for innovative consistency models but require redesigning application data types [2] or intercepting and reflecting APIs [3] via middleware. The main challenge nowadays in many data center environments is to understand how one makes such distributed systems scalable while still delivering good performance to user applications. Data availability is for instance always a desired property, while a sufficiently strict level of consistency can be used to handle data effectively across locations without long network delays (latency) and optimizing bandwidth usage.

The work described in this article has been accepted for publication at IEEE CloudCom 2013 (short paper), titled *Quality-of-Data for Consistency Levels in Geo-replicated Cloud Data Stores*

HBase is a well-known and deployed open source Java based solution inspired on the idea of BigTable [4] which targets the management of large amounts of information. HBase does not provide strong consistency outside of the local cluster itself. Eventuality is the promise and a write-ahead log is maintained for that. There have also been recent research efforts which address these shortcomings in geo-replicated data centers. For instance, Google with Spanner [5], where time clocks and GPS is used to achieve strong consistency with as much availability and consistency as possible.

HBase is the best example we have found of a large scale data store. We explore the idea of a system architecture with custom levels of consistency, providing finer-grain replication guarantees through application of data semantics (data consistency levels). We propose that having a strategy to best serve clients, while keeping control of geo-replicated and distributed databases can optimize usage of resources while still providing an acceptable experience to the end user through eventual consistency. Application behavior is more efficient and involves a slightly different shift into the consistency paradigm. This is carried out in this paper by modifying the extending the eventual consistency framework of HBase, with a more flexible and innovative approach, with which we use to tune its replication mechanisms, originally developed for treating updates in a self-contained manner.

Fully distributed HBase deployments have one or more master nodes (HMaster), which coordinate the entire cluster, and many slave nodes (RegionServer), which handle the actual data storage. Therefore a write-ahead log (WAL) is used for data retention in replication for high availability. Currently the architecture of Apache HBase is prepared to provide eventual consistency, as updates are replicated asynchronously between data centers. 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.

There are a number of existing systems where data semantics are analyzed to provide operations with faster (eventual) or slower (stronger) consistency without compromising performance [6]. In some, causal serialization and therefore commutative updates are provided based on the semantics of data [2]. Unlike linearizability, eventual consistency does work well for systems with shared distributed data to be queried and/or updated, because updates can be performed on any replicas at any given time [7]. It is also easier to achieve lower latency so most systems use eventual consistency in order to avoid expensive synchronous operations across wide area networks and still keeping data consistent through low latency operations in large geo-located deployments.

A. Roadmap

In the first part of the paper we review the fundamentals of several well-known existing consistency models in the area of distributed systems taking particular attention to the concept of eventual and strong consistency and possible variations of the two of them, extremes of the spectrum. Like an intermediate approach we put in practice a Quality-Of-Data model (QoD) which can be defined by up to three different vector-based constraints. That way, one can provide tailored consistency guarantees at replica nodes during high bandwidth peak-loads.

The rest of the paper is organized 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 corresponding vector-bounded replication guarantees. Finally, with Section 6 we conclude this work.

B. Contributions

The main contributions are focused on what other consistency properties Hbase can provide at the master server level, using a flexible and tunable framework, developed for treating group of updates tagged for replication in a self-contained manner. The work presented includes the design and development of a custom replication model and supporting framework to non-relational cloud-based data stores. The framework follows a three-dimensional vector model *K* (θ, σ, ν), which is based on data semantics.

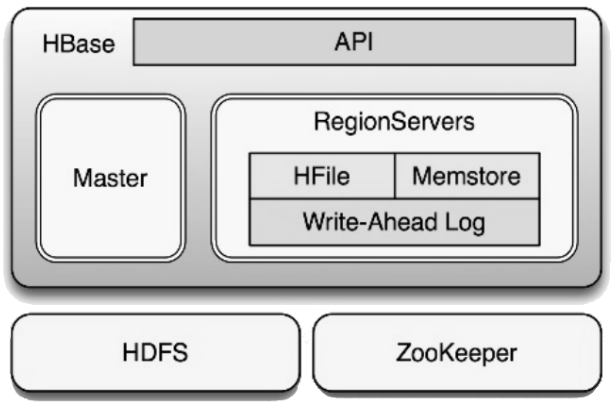
We take a step forward from the eventual consistency model at inter-site replication scenarios with HBase deployments to prove the validity of the model. It is possible to evaluate it by using an adaptive consistency model that can provide different levels of consistency depending of the desired Service Level Objective (SLO) for the QoD fulfillment. The idea of QoD fulfillment is based on the percentage of updates that need to be replicated within a given period using the three-dimensional vector model *K* previously mentioned.

We also propose to extend the HBase client libraries to provide blocks of transactions where each block can be customized using the desired level of consistency: causal, more relaxed, average, or bound specific. To achieve this we modify the HBase libraries (Htable). Basically, we provide grouping of operations from the source location before replication occurs, so apart from the multi-row atomically defined model in HBase, a more versatile system can also provide atomically replicated updates beyond the row-level (e.g column families or combinations of the fields in a row in HBase). The work is also an informal contribution that we aim to turn into a formal one to complete the efforts of a the "pluggable replication framework" proposed by the Apache HBase community [8].

II. Background and Related Work

BigTable [9] is example of a highly available and scalable key-value store, provides good performance for most workloads, it is lexicographically sorted, data families have the same type. It uses Chubby [10] as locking service, Google File Systems to store logs and data files (as SSTables).

Fig. 1 The main HBase architecture from [11]



HBase is open source, and it is based in the previous work BigTable [9], a distributed, persistent and multi-dimensional sorted map and it is being used for example at Facebook for structured storage of the messaging and user data in partial replacement of Cassandra [12]. Written in Java, it can handle large amounts of information, providing very good write latency at the expense of some durability concerns (data integrity is ensured with a minimum provided replica set of 3 in HDFS) and not so good results for reads as observed in [13]. In master to master scenarios, typically in geo-located and replicated clusters, the system provides eventual guarantees to data consistency through RPC (Remote Procedure Call) mechanisms.

That might be sufficient in most cases, more complex applications that require stronger consistency guarantees can be difficult to manage with BigTable and due to that, Google developed later in 2012 an evolution for BigTable that provides external consistency through atomic clocks for instance, Spanner [5]. This makes applications highly-available while ensuring synchronicity among distant replicas 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 of replicas.

Systems such as PNUTS from Yahoo [14] introduced a novel approach for consistency on a per-record basis. Therefore, became possible to provide low latency during heavy replication operations for large web scale applications. As in our work, they provide finer-grain guarantees for certain data, so in other words, new updates are not always seen right away by the clients (which is the case also with our HBase extension), but only if strictly necessary. Keeping that in mind, it is not always necessary for applications to be highly-available and consistent at once. That is applicable to our implementation too. Yahoo also realized that eventual consistency is not enough in the case of social and sharing networks, as stale replicas can result in undesired cases of users having the opportunity to see or use data they were not supposed to access. Ordering of events must be also taken into account. The grouping of operations feature of our model covers that idea.

A growing number of systems where data semantics are reviewed exist in order to provide operations with faster (eventual) or slower (stronger) performance without compromising consistency [6]. Also with those where causal serialization and therefore commutative updates are provided based on the semantics of data [2]. 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 [15] causality is guaranteed, although it does not guarantee the quality of data by bounding divergence, which can lead to outdated values being read.

In this work we introduce a replication module that integrates into HBase mechanisms and which targets applications which can cope and might require finer-grain levels of consistency at some point. Other systems use a variant of Snapshot Isolation techniques [16], which works within, but not across data centers. In a previous paper, a.k.a the conit consistency model from Duke University [17], the model is based on generality not practicality. We prefer the last one, because we find it more rewarding to users that need to integrate quality of data into a fully functional and reliable data store for cost savings on geo-replication resource usage. We cite the HBase community site for this, where it is reflected as an opened issue on their site [8]. Similar previous work and model from [18] also inspires diverge bounding approaches are we are addressing.

Others have focused in providing an extra layer on top of HBase for caching reads and improve response times based on avoiding garbage collection altogether [19] but the downside is the need for higher amounts of memory in place and used as buffers. Having the integration of a novel QoD at the core of HBase is more interesting to us, both in read dominated applications with added flexibility, as well as for more flexibility in update propagation in the case of write intensive applications.

III. HBase-QoD Architecture

The QoD paradigm here introduced allows for entries to be evaluated prior to replication based on one or several of the three parameters in a three-dimensional vector K (θ, σ, ν), corresponding to Time, Sequence, Value respectively in our case. Secondly, we take care of updates that collide with previous ones (same keys but different values). They can also be checked for number of pending updates or value difference from previously replicated updates, and then shipped or kept on the data structure accordingly. The time constraint can be always validated every X seconds, and the other two constraints are validated through Alg. 1, whenever updates arrive. For the work presented here we use Sequence (σ) as the main vector-field bound applied to a container.

The three-dimensional vector constraint model is based on [20] and implemented in the form of the corresponding QoD paradigm, shipping updates for replication, or retaining them for later shipment as mentioned. For that to be possible, we have used a set of customized data structures, which hold the values of the database rows we desire to check according to some specific field we might be interested in (e.g column family) for replication from a containerId. That container identified is composed as *tableName:columnFamily* in this case but can be a combination of any relevant row-fields that differentiate between user or application data semantics.

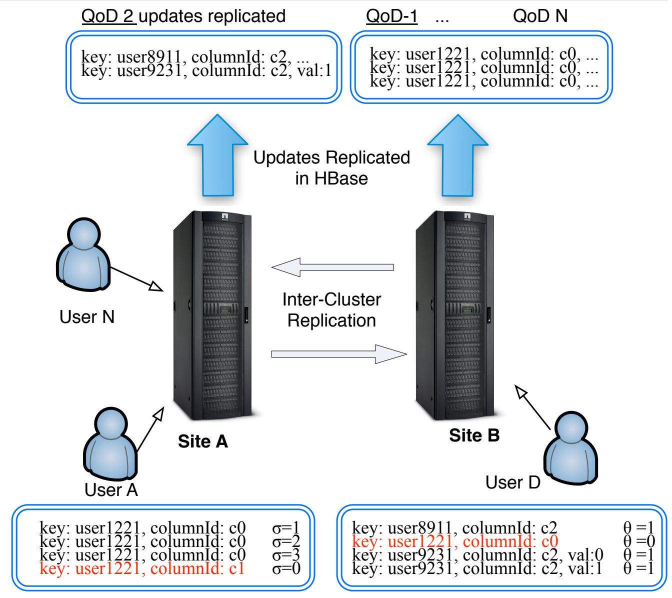
To compare and track the QoD fields, that act as constraints to replicate updates, against these stored entries, we defined data *containers* which are useful to keep track of the current value of the vector-field selected to bound replication to, and secondly the maximum value it will be allowed to reach before updates are flushed to the slave cluster and it is reset again. That is as what we call the QoD percentage of updates replicated (according to the selected vector-field bound, e.g σ). The process is partly automated, of by now, we just define it at run-time (or by the developer later) by adding a parameter into the HBase console to define a vector-field specific bound.

Extensions to the HBase internal mechanisms: We now outline the main internal mechanisms proposed to include into the HBase architecture, in order to rule updates selectively during replication. For that, we previously introduced the core architecture of HBase as shown in Figure 1. The overall architecture layout is presented in Figure 2.

The original HBase architecture has built-in properties derived from the underlying HDFS layer. As part of it, the WALEdit data structure is used to store data temporarily before being replicated, useful to copy data between several HBase locations. The QoD algorithm (shown in Algorithm. 1) uses that data structure, although we extend it to contain more meaningful information that help us in the management of the outgoing updates marked for replication. We extend HBase, adding updates due to be replicated in a priority queue according to their own QoD in each case. Thereafter once the specified QoD threshold is reached another thread from HBase in the form of Remote Procedure Call collects and ships all of them at once.

Typical distributed and replicated deployment: In distributed clusters Facebook is currently using HBase to manage the messaging information across data centers. That is because of the simplicity of consistency model, as well as the ability of HBase to handle both a short set of volatile data and ever-growing data, that rarely gets accessed more than once. More specifically, in their architecture reports, a Key for each element is the userID as RowKey, word as Colum and messageID as Version and finally the value like offset of word

Fig. 2 HBase QoD high-level


QoD 2 updates replicated
QoD-1 ... QoD N
key: user8911, columnId: c2, ...
key: user9231, columnId: c2, val:1
key: user1221, columnId: c0, ...
key: user1221, columnId: c0, ...
key: user1221, columnId: c0, ...
Updates Replicated in HBase
User N
Inter-Cluster Replication
Site A
Site B
User A
User D
key: user1221, columnId: c0 σ=1
key: user1221, columnId: c0 σ=2
key: user1221, columnId: c0 σ=3
key: user1221, columnId: c1 σ=0
key: user8911, columnId: c2 θ =1
key: user1221, columnId: c0 θ =0
key: user9231, columnId: c2, val:0 θ =1
key: user9231, columnId: c2, val:1 θ =1

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 .

in message (Data is sorted as: <userId, word, messageID>). That implicitly means that searching for the top messageIDs of an specific user and word is easily supported, and therefore queries run faster in the backend.

With the eventual consistency enforcement provided, updates and insertions are propagated asynchronously between clusters so Zookeeper is used for storing their positions in log files that hold the next log entry to be shipped in HBase. To ensure cyclic replication (master to master) and prevent from copying same data back to the source, a sink location with remote procedure calls invoked is already into place with HBase. Therefore if we can control the edits to be shipped, we can also decide what is replicated, when or in other words, how soon or often.

Algorithm 1 QoD algorithm for selecting updates 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
```

Operation Grouping: At the application level, it may be useful for HBase clients to enforce the same consistency level on groups of operations despite affected data containers 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 at the same consistency level and propagated atomically to slave clusters.

For example, publication of user statuses 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 they should be handled at a stronger consistency level to ensure they are atomically visible along with the list of friends of the user in respect to the semantics we describe here.

In order to not 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 behavior we propose 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 parameterized with one of the two options: i) *IMMEDIATE*, which enforces stronger consistency for the whole block of operations within it; and ii) *ANY*, which replicates a whole block as soon as any QoD vector field bound, associated with an operation inside the block is reached. We depict this scenario in Listing 1.

Listing 1: Operation grouping

```
htable.startConsistentBlock(
    ConsistencyType.IMMEDIATE)
Put put1 = new Put(Bytes.toBytes("row1"))
    ;
put1.add(Bytes.toBytes("SocialNetTable"),
    Bytes.toBytes("status"), Bytes.toBytes
    ("friend 12345 added"));

Put put2 = new Put(Bytes.toBytes("row2"))
    ;
put2.add(Bytes.toBytes("SocialNetTable"),
     Bytes.toBytes("friends"), Bytes.
    toBytes("12345"));

Put put3 = new Put(Bytes.toBytes("row3"))
    ;
put3.add(Bytes.toBytes("SocialNetTable"),
     Bytes.toBytes("wall"), Bytes.toBytes(
    "12345 is now a friend"));

htable.put(put1);
htable.put(put2);
htable.put(put3);

htable.endConsistentBlock();
```

The above listing 1 showcases an illustrative simple example of a social network where three containers with different consistency levels are modified. Note that we are not aiming at full transactional support, as it would be possible to change the same data containers modified by a set of grouped operations, at the same time, from other operations individually.

IV. Implementation Details

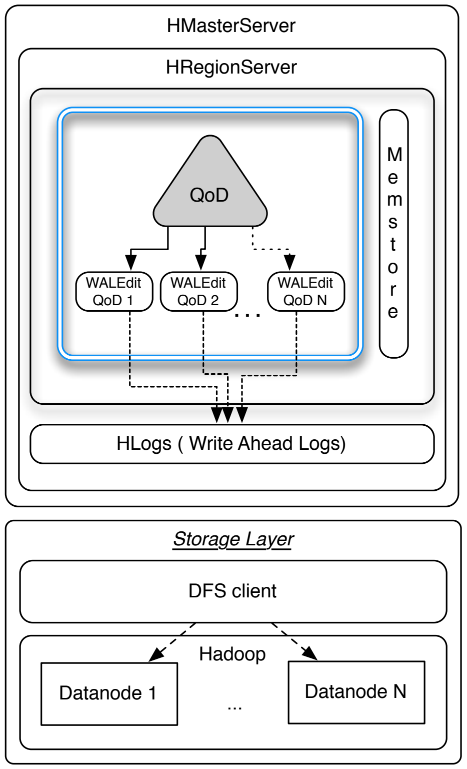
HBase replication mechanism is based in a Write Ahead Log (WAL), which must be enabled in order to be able to replicate between distant data centers. The process of replication is currently carried out asynchronously, so there is no introduced latency in the master server during that operation. Usually, it operates in the following manner: when the buffer is filled or the reader hits the end of the file, the buffer is sent to a random region server on the slave cluster. Although, since that process is not strongly consistent, in write heavy applications a slave could still have stale data for an order of more than just seconds, and just until the last updates commit to local disk.

In our implementation we overcome the pitfalls of such an approach, and also handle well the burden of latency during long communication periods between clusters in different data-centers. This is because updates are shipped earlier depending on the QoD, therefore lower values of it (e.g maxBound of σ in the three dimensional vector K). We take advantage and build a filtered and sorted list of edits we scope for shipping during replication in HBase. We are able to enforce QoD bounds on data for delivery at the other end cluster. For write intensive applications, that can be both beneficial in terms of reducing the magnitude of peaks in bandwidth usage, while also delivering data according to application needs and improved semantics.

The QoD module in Figure 3 shows the overall resulting inner-implementation details from the changes introduced in HBase. We observe the QoD module is plugged into the core of the systems, for intercepting the incoming updates, and processing them into a priority queue defined to store those for later replication to a slave cluster. For that purpose, several Java files such as *ReplicationSource.java* are modified in the HBase code-base.

We have changed the logic for shipping edits to the write-ahead log so that the process is carried out according to the data semantics we already presented. Several data structures are required, some of them are of existing types in HBase, as WALEdit, others for instance ConcurrentHashMap in order access different data containers that we later query to determine where and how to apply a given QoD at the row 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 for any of the vector-constraints (e.g σ). The use of the QoD is also applicable to the selection of those updates to be replicated according to a combination of any the three-dimensional vector constraints, not only σ.

Fig. 3 HBase QoD operation



V. Simulation and Evaluation

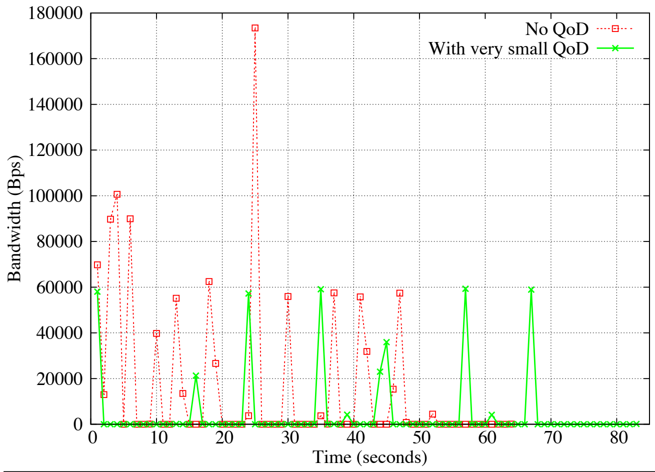
It has been already verified and presented in other reports and projects in the area of Hadoop, that the level of replication statically defined is a limitation in itself, which therefore must be more efficiently adjusted to match scheduling of tasks. That is also related to the work here covered within HBase, as HDFS is the storage layer of it. We avoid the statically defined replication constraints in HDFS and HBase by enforcing the shipment of updates during replication as frequently as we desire by using our HBase-Qod defined vector constraints.

During the evaluation of the QoD prototype a test-bed with several HBase cluster has been deployed at INESC-ID and IST in Lisbon, some of them with an HBaseQoD-enabled engine for quality of data between replicas, and others running a regular implementation of HBase 0.94.8. All tests were conducted using 6 machines with an Intel Core i7-2600K CPU at 3.40GHz, 11926MB of available RAM memory, and HDD 7200RPM SATA 6Gb/s 32MB cache, connected by 1 Gigabit LAN.

We expect to continue obtaining results using a network tool as [21] for delaying and simulating network bandwidth and latency between a distant set of locations. We are also currently trying to confirm that the appropriate QoD does have a positive impact when bounding staleness (monitoring time an pending updates, instead of number of updates) with varying delays, but not now due to space constraints in this text, we do not present it. Even though, it is obvious this will just spread replication activity in time, but expecting a reduced max-value on each network peak-load observed.

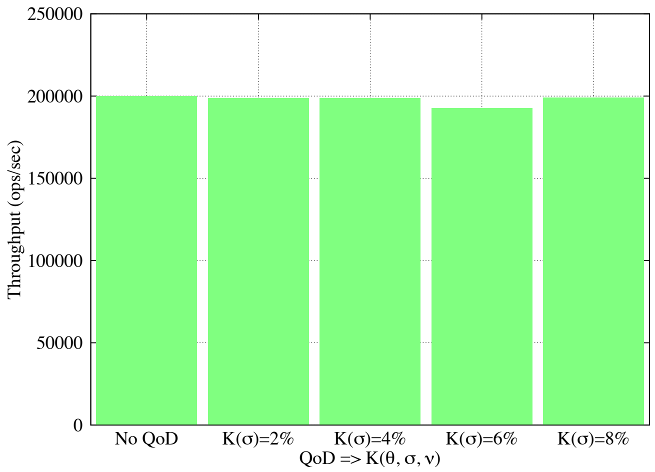
In Figure 4 for a very small QoD of K using σ we realize

Fig. 4 Bandwidth usage and replication frequency for a typical workload with and without HBase-QoD



that there is a lower limit where latency can not be reduced any further. We have measured that same value in the subsequent Figure 8 with a larger QoD of K using $\sigma = 0.5\%$. Recall this means the percentage of records allowed to be modified, or of updates applied in relation to total number of records, before replication is triggered (a very small value, where strict consistency would mean 0.00%).

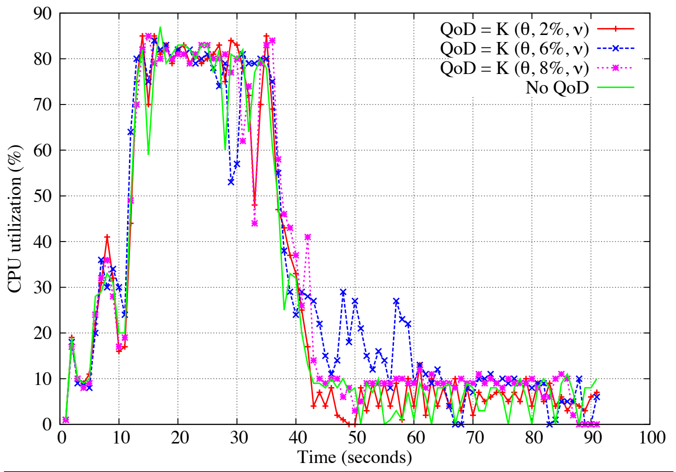
Fig. 5 Throughput for several QoD configurations



We also confirm the QoD does not hurt performance as we observe from the throughput achieved for the several levels of QoD chosen during the evaluation of the throughput achieve with the benchmark for our modified version with HBase=QoD enabled, Figure 5. The differences in throughput are irrelevant and mostly due to noise in the network, that is the conclusion after obtaining similar results to that one in several rounds of tests with the same input workload on the data store.

Next we conducted as shown in Figure 6, and *dstat* presents, an experiment to monitor the CPU usage using HBase-QoD. CPU consumption and performance remains roughly the same and therefore stable in the cluster machines as can be appreciated.

Fig. 6 CPU usage over time with QoD enabled



We have also taken measurements for the following workloads obtaining results as follows:

a) Workloads for YCSB: We have tested our implementation in HBase with several built-in workloads from YCSB plus one customer workload with 100% writes to stress the database intensively as the target updates in the social network previously described is about changes and new insertions.

Figure 7 shows three different sets of Qualities of Data for the same workload (A):

1) YCSB workload A (R/W - 50/50)
   - No QoD enforced.
   - QoD fulfillment of σ=0.5% of total updates to be replicated.
   - QoD fulfillment of σ=2% of total updates to be replicated.

   During the execution of the workload A, in Figure 7, the highest peaks in replication traffic are observed without any type of QoD, i.e. just using plain HBase. This is due to the nature of eventual consistency itself and the buffering mechanisms in HBase.
   With a QoD enabled as shown in the other two graphs, we rather control traffic of updates from being unbounded to a limited amount, accordingly to save resources' utilization, while suiting applications that require small amounts of information to be only propagated as a group, when they are just needed.
   We observe that higher QoD requires replication traffic less frequently, although interactions reach higher values on Bytes as they need to send more data. Small QoD optimizes the usage of resources while sending priority updates more frequently (this could be the case of wall posts in a social network).
2) YCSB workload A modified (R/W - 0/100)
   - No QoD enforced.
   - QoD fulfillment of σ=0.5% of total updates to be replicated.
   - QoD QoD fulfillment of σ=2% of total updates to be replicated.

In Figure 8 we can see how a write intensive workload performs using a QoD. Similar results are expected and later

Fig. 7 Bandwidth usage for Workload A using 5M records using QoD bounds of 0.5 and 2% in the σ of K.

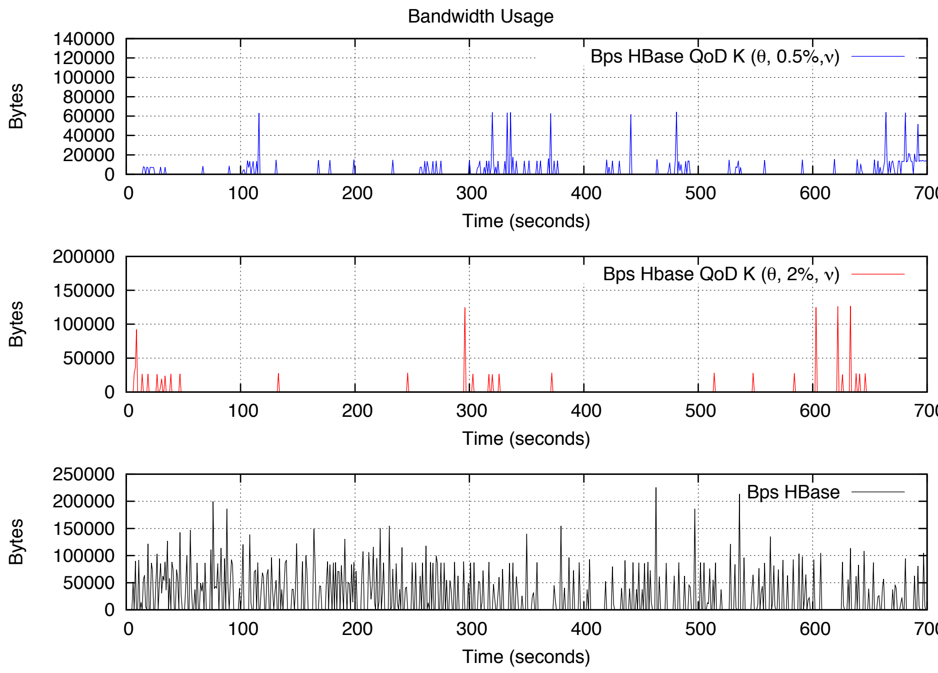
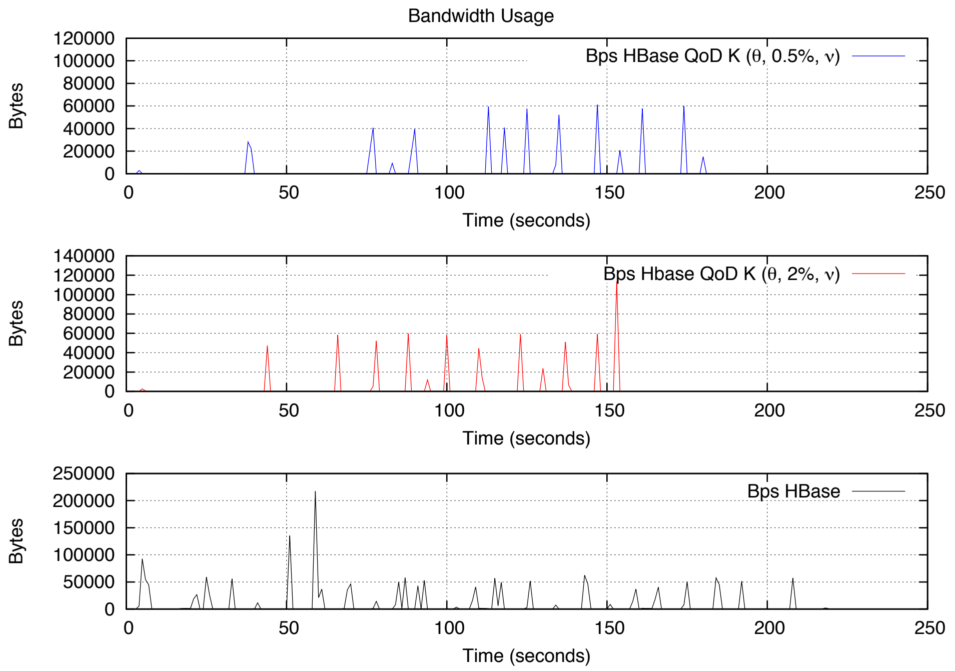


Fig. 8 Bandwidth usage for Workload A-Modified using 5M records using QoD bounds of 0.5 and 2% in the σ of K.



also confirmed in this graph (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%) we see lower peaks in bandwidth usage, as well as in the following measurement used (2.0%). 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 realize savings on average bandwidth usage. The principal source of motivation of the paper is to find a way of controlling the usage of the resources in a data center. Also, to be able to leverage the trading of strong for eventual consistency with a more robust atomic grouping of operations using vector bounded data semantics.

VI. Conclusion

Performance in HBase improves as the number of servers increases due to more memory available [4], but regardless of that fact, it is not trivial to scale always further by following the later approach. Therefore, having ways of providing different levels of consistency to users regarding data in cloud environments translates into substantial traffic savings and therefore associated costs to potential service providers or even customers, which is a very relevant matter as seen in [22] for consistency cost-efficiency. Therefore it is always good to be evaluate how selective replication can support that goal in distributed deployments with HBase.

In this paper we presented a mechanism QoD (Quality-of-Data), that applied to HBase provides a flexible consistency model for replication between distant locations. It can provide applications with just the required consistent data at each point in time. 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 usual eventually consistent approaches in these sort of data stores.

With these semantics we trade-off between short timed consistency and wide area bandwidth cost savings during peak loads. Achieving the last, can help to significantly reduce replication overhead between data centers when there are periods of disconnection or bottlenecks in the network. We evaluated our implementation on top of HBase clusters distributed across several locations.

In summary, we successfully apply a model based on the mechanisms of a previous work [20] to show that HBase consistency can be tuned at the core-system level, without requiring intrusion to the data schema and avoiding more middleware overhead such as in [23]. In our case, experimental results indicate that we are able to maintain an acceptable throughput, reduce latency peaks, as well as optimize bandwidth usage. In the future we would like to conduct more experiments using Amazon EC2 infrastructure and also several other cluster locations in partner-universities if there is any chance to do so.

References

[1] *Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services.*, 2002.

[2] N. P. Marc Shapiro and M. Z. Carlos Baquero, "Conflict-free replicated data types," INRIA, rocq, rr RR-7687, July 2011. [Online]. Available: http://lip6.fr/Marc.Shapiro/papers/RR-7687.pdf

[3] L. Veiga and S. Esteves, "Quality-of-service for consistency of datageo-replication in cloud computing," *Europar 2012*, August 2012.

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

[5] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally-distributed database," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 251–264. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387905

[6] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 265–278. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387906

[7] S. Burckhardt, D. Leijen, M. Fahndrich, and M. Sagiv, "Eventually consistent transactions," in *Proceedings of the 21st European conference on Programming Languages and Systems*, ser. ESOP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 67–86. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28869-2_4

[8] A. Purtell, "Priority queue sorted replication policy," August 2011.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323

[10] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 335–350. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298487

[11] L. George, "Advanced hbase," http://www.slideshare.net/jaxlondon2012/hbase-advanced-lars-george, 2012. [Online]. Available: http://www.slideshare.net/jaxlondon2012/hbase-advanced-lars-george

[12] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[14] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1277–1288, Aug. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1454159.1454167

[15] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "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*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 401–416. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043593

[16] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 385–400. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043592

[17] A. Haifeng Yu, Vahdat, "Combining generality and practicality in a conit-based continuous consistency model for wide-area replication," April 2001.

[18] H. Yu and A. Vahdat, "Design and evaluation of a continuous consistency model for replicated services," in *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4*, ser. OSDI'00. Berkeley, CA, USA: USENIX Association, 2000, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251229.1251250

[19] LiPi, "Slabcache, caching in hbase." [Online]. Available: http://blog.cloudera.com/blog/2012/01/caching-in-hbase-slabcache

[20] L. Veiga, A. Negrão, N. Santos, and P. Ferreira, "Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency," *Journal of Internet Services and Applications*, vol. 1, no. 2, pp. 95–115, 2010. [Online]. Available: http://dx.doi.org/10.1007/s13174-010-0011-x

[21] S. Hemminger, "Netem-emulating real networks in the lab," in *Proceedings of the 2005 Linux Conference Australia, Canberra, Australia*, 2005.

[22] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. Pérez, "Consistency in the Cloud:When Money Does Matter!" in *CCGRID 2013- 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Delft, Pays-Bas, May 2013. [Online]. Available: http://hal.inria.fr/hal-00789013

[23] S. Das, D. Agrawal, and A. El Abbadi, "G-store: a scalable data store for transactional multi key access in the cloud," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 163–174. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807157