Quality-of-Data Consistency Levels in HBase for GeoReplication*

Álvaro García Recuero
Instituto Superior Técnico - UTL, Portugal
alvaro.recuero@ist.utl.pt

Luís Veiga
INESC-ID Lisboa, Distributed Systems Group
luis.veiga@inesc-id.pt

Abstract

This work first dives into the fundamentals of HBase to later explore a tailored architecture model for geo-replication in distributed systems, replacing the eventual consistency by an adaptive consistency model, extending some of the main components of HBase rather than building a middleware layer on top of it. Our aim is to give a better understanding of what other replication guarantees can such a system offer, its value to users, and how a novel consistency model can be applied to its core, in order to save wide area bandwidth costs and peak loads w.r.t. replication overhead between data centers, while catering for different users and applications needs (e.g., regarding timeliness, number of pending updates, divergence bounds). Therefore, this research is mainly targeting replication mechanisms HBase currently does not provide by assessing how one can extend those already in place and provided within its codebase.

1 Introduction and motivation

HBase is the open source version of BigTable [6], is developed in Java and targets the management of large amounts of information. HBase does not provide strong consistency outside of regional servers in replicated scenarios within a cluster. There has also been some recent research that addresses this shortcomings in geo-replicated data center scenarios like [1]. HBase does not use that approach, neither Paxos for synchronization of replicas. On the other hand, replication between remote sites is achieved with eventual consistency.

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 [7], 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 [5] via middleware.

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.

Optimizing replicated data: This work follows a previous middleware research effort (VFC^3) on adaptive replication for quality of service presented to users in geo-replicated scenarios [5]. The main goal is to incorporate a more flexible, fine-grained and adaptive consistency model at the HBase core architecture level. That feature can be part of HBase to have bandwidth savings on inter-site data center replication, helping to avoid peak transfer loads at times of high update rates, while still enforcing some *quality-of-data* to users regarding recency (or number of pending updates and value divergence between replicas).

In that regard, latency can be reduced by imposing some constrains (time bounds or others regarding number of pending updates and value divergence) on the replication mechanisms of HBase providing a two-fold advantage: i) ensure that a best-effort scenario does not overload a network with thousands of updates that might be too small (can be batched too if desired) and also and more importantly, ii) updates can be prioritized so that it is more able to achieve the quality of service agreed with the user or application class.

2 Architecting a QoS layer for HBase

This section outlines the main internal mechanisms proposed to include into HBase to rule updates selectively during replication. For that, we briefly introduce the core architecture of HBase and then dig into the proposed changes for defining a work in progress with quality of service for consistency of replicas.

*Álvaro García Recuero, Student, FAST'13 Work-in-Progress and Poster.

With eventual consistency, 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. 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 put into place with HBase. Therefore if we can control edits to be shipped, we can also decide what is replicated and how or when. In a similar research line, the VFC^3 framework defined the properties for a cloud tabular storage system like HBase so that can be flexible enough to accommodate all the needs of users when using a distributed cache in several data centers and locations during replication, providing an adjustable model of consistency. Keeping in mind that previous work, we plan to leverage the internal mechanisms of HBase regarding consistency.

To provide bounded consistency guarantees and a QoS for HBase we are tuning its inner workings with the same idea than for VFC^3. With the existing command line CopyTable in HBase one can manually define what is going to be replicated and is useful for cases where new replicas need to be put up to date. We are focusing our implementation in keeping an organized list of items (extending the structure reflecting updates to be shipped), where we can apply the QoS principles. We apply a well defined bounded replication model based on time constraints (t), sequence (number of pending updates) and value (percentage of changes) in `qosEngine.enforceQoS(tableName, put)`.

This way, every new update is checked for QoS and shipped for replication or delayed by insertion in a sorted queue ordered by allowed delay. Any new updates over previous ones (same data) are checked for number of pending updates or value difference from previously replicated update, and shipped or kept on list, accordingly.

To evaluate the gains we can simulate three distant locations, Barcelona, Lisbon and Stockholm. Therefore we set up a tool to create that illusion on delay of packets between sites, that is the unix built in tool netem (tc). We are using a workloads generation and benchmarking tool such as YCBS (Yahoo Cloud Benchmark Service), that we are already familiar with for testing and experimenting performance gains for our hypothesis about replication, but also standard tools (HBase shell) provided to enable and operate replication in a cluster. We expect to draw a solid evaluation and conclusions from the results obtained by having a specific data model to process replication in various ways flexibly.

3 Conclusion

We briefly outline the main features of HBase replication and those currently missing, noting the partial scope of the work as for now. Performance in HBase improves as the number of servers increases due to more memory available [3], but regardless of that fact, it is not trivial to scale always further following that approach. Therefore, having ways of providing different levels of service to users can save substantial traffic savings and therefore bandwidth costs. For that, the novel topic presented with selective replication based on QoS can help in deployments using HBase. Others have focused in providing an extra layer of middleware on top of HBase for caching reads and improve response times based on garbage collection avoidance [4] but the downside is the need for higher amounts of memory in place used as buffers. We might also explore that and several other options in later research efforts.

Acknowledgments: To my supervisor at IST and KTH coordinator, L. Veiga and J. Montelius, for their patience and support at all times, and specially to Sérgio Esteves for his advice at INESC-ID.

References

[1] J. C. Corbett, J. Dean, M. Epstein, et al. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[2] M. Shapiro, N. Preguiça, C. Baquero, M.Zawirski Conflict-free replicated data types. Technical Report RR-7687, July 2011.

[3] Carstoiu, D. and Cernian, A. and Olteanu, A. Hadoop Hbase-0.20.2 performance evaluation. In *2010 4th International Conference* on New Trends in Information Science and Service Science, May, 2010, pages 84–87.

[4] L. Pi. Slabcache, caching in hbase. *http://blog.cloudera.com/blog/2012/01/caching-in-hbase-slabcache/*

[5] S. Esteves, J.N. Silva and L. Veiga. Quality-of-service for consistency of Data Geo-replication in Cloud Computing. *Europar 2012*, August 2012.

[6] Chang, Fay and Dean, Jeffrey and Ghemawat, Sanjay and Hsieh, et al. Bigtable: a distributed storage system for structured data In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX

[7] Seth Gilbert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services. *SIGACT News*, 33(2):51–59, 2002.