Trade-offs between consistency and latency in Apache Cassandra

Noura Rabardeau, Alexandre Tamborrino, Thomas Fattal
School of Computer Science and Communication (CSC)
Royal Institute of Technology KTH, Stockholm, Sweden

May 13, 2013

Abstract

Used by some large-scale companies like Facebook, Ebay and Spotify, Apache Cassandra is a major NoSQL distributed storage system. Data consistency and request latency are two important concerns for applications. Since strong consistency cannot be obtained at the same time as very low request latency and high availability (CAP theorem), trade-offs need to be made according to applications' needs. We focus here on the Apache Cassandra storage system, which provides tunable consistency. This study shows what are the possible trade-offs between consistency and latency that can be obtained with Apache Cassandra to suit best the needs of applications that have different requirements.

1 Introduction

Apache Cassandra [1] is a key-value pairs distributed storage system able to manage very large amounts of structured data over several servers. This system provides high availability and has no single point of failure. Apache Cassandra is built using a column-based data model and sparse storage such as Google BigTable and has a replication and distribution model that is close to the one used by Amazon Dynamo.

Cassandra [2] is designed as a peer-to-peer system, where the peers (nodes) communicate between each other via gossiping to discover and exchange cluster membership information. Cassandra's distribution and replication model is based on quorum-based Dynamo-like model [3].

One of the key features of Cassandra is providing tunable consistency. The consistency level is based on how many replicas will be contacted for an operation (read/write) and can be set for a cluster, for a data center or for one operation. Choosing the consistency level allows to obtain the trade-off between request latency and data accuracy that suits best the needs of the application.

Strong consistency ensures that the data will always be up-to-date. It corresponds to $R + W > N$ where R = number of replicas contacted for read operations, W = number of replicas contacted for write operations, and N = total number of replicas. Quorums are often used to achieve strong consistency: $R = W = floor(N/2) + 1$.

Weak consistency allows data to be out-dated for a while, while preserving eventual consistency (there is a time after which data will be up-to-date). Weak consistency corresponds to situations where R and W don't overlap: $R + W \leq N$. Weakening consistency allows to lower latency and thus to provide higher availability.

2 Study of trade-offs between consistency and latency

2.1 Benchmark environment

We used CCM (for Cassandra Cluster Manager) which is an open-source project [4] that allows to create, launch and remove an Apache Cassandra cluster on a local machine. The version of Cassandra used in our experiments is 1.2.0. All our experiments have been carried out on a dedicated Thinkpad computer. This system has bi-core Intel i7 CPU, 8 GB of memory, a HDD with a rotational speed of 7200 rpm and runs a Linux 3.7.1 kernel. Our Cassandra cluster has 8 nodes and a replication factor of 3 (3 is a standard in real Cassandra cluster deployments). To run the workload on our cluster, we used the tool cassandra-stress [5] which runs 10 000 write operations (test data's keys chosen at random) followed by 10 000 read operations.

2.2 Case 1: When strong consistency is a requirement

In this case study, we assume that strong consistency is a requirement for the application that will use Cassandra. We suppose that an application can be read-intensive (resp. write-intensive) if it performs a lot more reads (resp. writes) than writes (resp. reads) and/or read (resp. write) latency is greater than write (resp. read) latency. Consequently, the possible couples of values for R and W and (R=1, W=3), (R=2, W=2) and (R=3, W=1) so that $R + W > 3$. (R=1,W=3) corresponds to a read-intensive setting as only one node storing a replica has to be contacted for a read operation to complete. Same for the write-intensive configuration with (R=3, W=1). (R=2, W=2) is the middle case where read and write operations are equally important. Using our benchmark environment, Figure 1 shows the resulting latency of read and write operations in these different configurations.

We see that the (R=1, W=3) configuration increases a lot the write latency but has not a much better read latency than (R=3, W=1). Indeed, write latency in the read-intensive configuration is 9.4 times worse than in the write-intensive configuration, but read latency in the read-intensive configuration is only 2.4 times better than in the write-intensive configuration.
These results show that the read-intensive configuration should only be used when write requests are very rare or when write latency is not important at all (for instance in the case of read-only systems where write operations are performed only by administrators). In other cases, a write-intensive configuration is a better trade-off.

2.3 Case 2 : When weak consistency is acceptable

Now we study the case where the application that uses Cassandra can live with weak consistency ($R+W \leq N$). We show that weak consistency allows to decrease the latency. We define the following configurations: (R=1, W=2), (R=2, W=1) and (R=1, W=1). Logically (R=1, W=1) provides the weakest consistency level as R+W is equal to 2 instead of 3 in the two other configurations. Figure 1 shows the resulting latency of read and write operations in these different configurations using our benchmark environment.

We see that (R=1, W=2) has still a relatively high write latency (3.5 times worse than when W=1) and should be used only in systems where write latency is not important. R=2 has a read latency 1.7 times worse than R=1. As expected, (R=1, W=1) provides the best latency for both writes and reads, but at the price of the weakest consistency.

But what do we exactly mean by "weaker consistency"? We need metrics that better define the level of weak consistency than the values of R and W. In

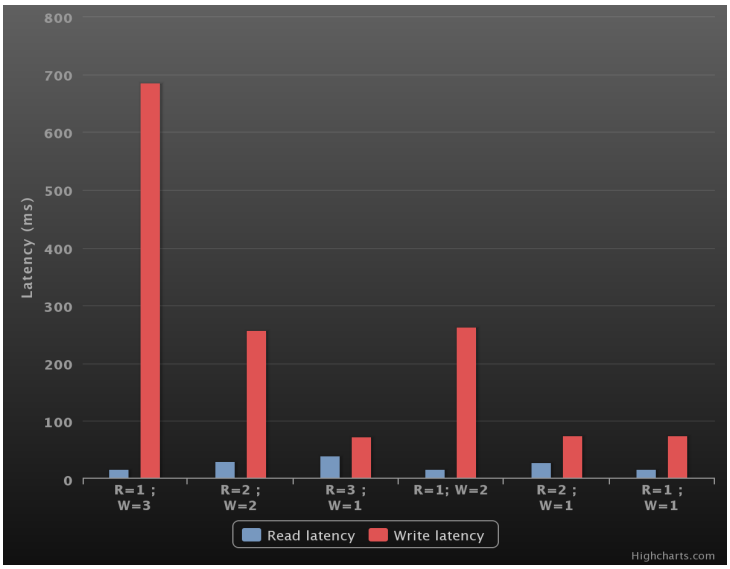

Figure 1: Read latency (ms) and write latency (ms) in strong and weak consistency cases

order to refine our study we use the Probabilistically Bounded Staleness algorithm (PBS) [6]. PBS can make predictions about the probability of consistent reads for Dynamo-style quorum replication[3] (which is the replication model of Cassandra). Two metrics can be tuned to refine the predictions[6]: **t-visibility** which provides the probability of reading a write t seconds after it returns, and **k-staleness** which provides the probability of reading one of the last k versions of a data item.

T-visibility answers the question "How eventual is eventual consistency?" and k-staleness answers to the question "How consistent is eventual consistency?". We use an implementation of a PBS predictor released in a module of Cassandra 1.2.0. It needs the Cassandra cluster to be manually loaded to make predictions, so we can use the same benchmark environment than previously to perform the load. First, we vary the t-visibility for a fixed value of k-staleness equals to 1. In other words, we get the probability of reading the last written version when we read a value t-visibility ms after it has been written. Figure 2 shows the resulting graph when using our benchmark environment. We can see that we have much more useful metrics to compare our different weak consistency configurations than without using PBS. As expected (R=1, W=1) provides the weakest consistency for small t-visibility values, but the probability of consistent read reaches 1 for a k-visibility of 1.5 seconds (not visible in the graph).

Now we vary the k-staleness for a fixed value of t-visibility equals to 100 ms. In other words, we get the probability of reading one of the k-staleness latest versions of a value we have last written 100 ms ago. Figure 3 shows the resulting graph when using our benchmark environment. As expected (R=1, W=1) gives the lowest probability of reading an up-to-date version of the data.

In the end, the use of the PBS algorithm on our benchmark environment allowed us to measure with much more accuracy how much we "lose" in terms of probability of consistent read when we lower the consistency level. These metrics can be very useful for an application developer that wants to decide the level of consistency that suits the most the nature of the application.

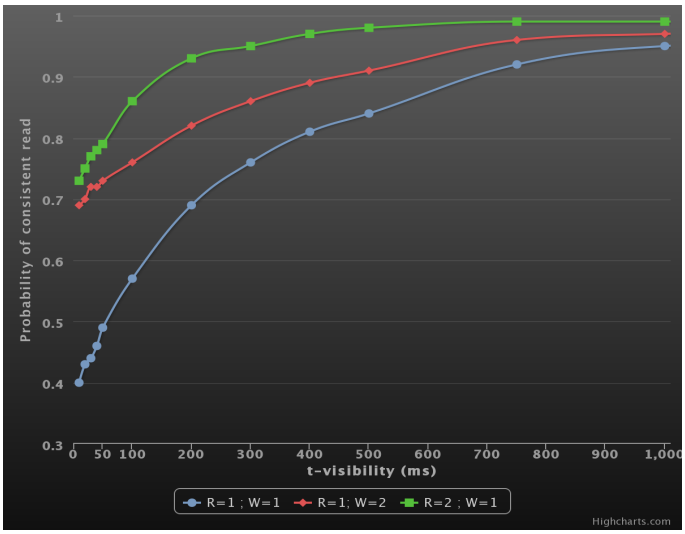

Figure 2: Probability of consistent read as a function of t-visibility for a fixed value of k-staleness (equal to 1)

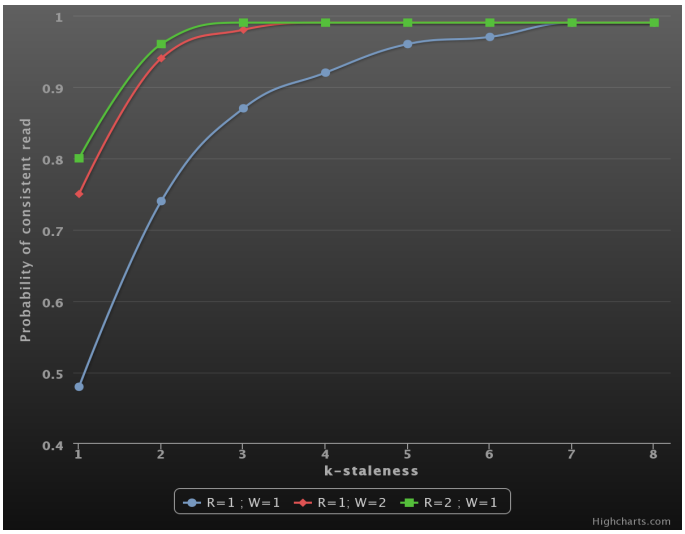

Figure 3: Probability of consistent read as a function of k-staleness for a fixed value of t-visibility (equal to 100 ms)

3 Some future challenges

All our experiments were carried out on a single machine where 8 nodes were virtualized. Results have been computed as if the cluster was really distributed (emulation of latency as if the nodes were running on different machines distant from each other). In order to prove the results obtained, it would be interesting to run the same experiments on a real cluster. Moreover, the fact that our benchmark environment was local prevented us to kill nodes during the stress-tests. It would be interesting to measure how the latency and/or consistency are affected in presence of churn, when nodes often fail and join.

References

[1] Prashant Malik (Facebook) Avinash Lakshman (Facebook). Cassandra - a decentralized structured storage system. *Lakshman*, 2009.

[2] Apache Cassandra 1.2 Documentation. `http://www.datastax.com/docs/1.2/index`. [Online].

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels (Amazon.com). Dynamo: Amazons highly available key-value store. 2007.

[4] Sylvain Lebresne. Github Repository for Cassandra Cluster Manager (CCM). `https://github.com/pcmanus/ccm/`, 2013. [Online].

[5] Apache Cassandra 1.1 - Tool cassandra-stress. `http://www.datastax.com/docs/1.1/references/stress_java`. [Online].

[6] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. 2012.