CAP in practice: HBase

Thomas Uyttendaele

Abstract

Keywords:

1. Introduction

In the end many choices can be made, but how does it reflect to the behaviour in practice? This paper will present result from experiments with HBase and MongoDB compared to the CAP theorem. The paper is organised as follows. Sect

2. The CAP Theorem[1][2]

The CAP Theorem was introduced by E. Brewer [1] in 2000 and discusses 3 properties of which each network shared-data system can guarantee at most 2: (definitions based on [2])

- (Strict)**C**onsistency: The system acts like there is only single storage
- High **a**vailability: The system is available (for updates)
- **P**artition tolerance: A split in the network let the different partitions still act as a single system.

Designers used this model to explain their design decisions, others used it to compare different system but sometimes it was misused. As E. Brewer explains 12 year after the launch, the "2 of 3" can be misleading.

One of the reasons is that there exist several types of consistency, partition tolerance and different availability guarantees. These choices need to be made several times in the different subsystems and the end solution is not black or white.

At first glance, it also looks like partition tolerance has to be implemented and therefore consistency and availability needs to be given up. In the practice, partition splits are only rare and therefore both consistency and availability can be allowed most of the time.

Each of the 3 choices will be discussed in more detail, how their implementation could work, what the influences are and some examples.

2.1. CA

When forfeiting partition tolerance, these systems provide all the time consistent data available to all nodes, except when there are one or more nodes unavailable. In that case, write requests will be not allowed.

These systems can be build around the 2 phase commit and have cache invalidation protocols. Examples of this types are the typical relational databases roll out in clusters.

2.2. CP

A consistent system with partition tolerance will provide all the time the last data, even in the case of network splits. This comes with the loss of all nodes available all the time.

The system will allow operations only on the majority partition. In case multiple splits are present and no partition has a majority of nodes, the whole system can be unavailable. These systems can be build around a master/slave principle where the operations will be directed to the master, the slaves are present to continue operation when the master fails.

In practice, systems like MongoDB, HBase and Redis chooses for CP.

2.3. AP

In a highly available systems with partition tolerance, is it possible to read inconsistent data. As read and write operations are still allowed when there are different partitions, it is possible that the database has other content depending on the used node. When the split is dissolved, a need for manual conflict resolution can be needed. In case a record is adapted in both partitions, the user will need to choose the correct version.

Example systems following AP are Cassandra, Riak and Voldemort.

3. Overview of HBase and MongoDB

In this article, 2 CP systems will be discussed more in detail regarding their choices to forfeit availability and the influence on their behaviour in practice. The systems are HBase and MongoDB, an architectural overview will be given in this section.

3.1. HBase

HBase[3] is an open-sourced, distributed, versioned database designed after Google's BigTable [4]. HBase relies on Zookeeper for the distributed task coordination and the persistent storage can be done on the local hard disc, Hadoop Distributed File System or Amazon S3. In this installation is chosen for Hadoop.

HBase nodes exists out of HMasters and HRegionServers, the coordination of the system is done by one HMaster, the handling of data is done by the HRegionServer. To store the data, on each HRegionServer a Hadoop datanode instance should be deployed for data storage, the data will be replicated to a configurable amount of other nodes. The data is stored in a table, which is split in one or more regions. A region is leased to a given HRegionServer for a defined time. During this time, only this server will provide the data of the region to the different users. This way the consistency of data can be guaranteed because there is for each record only a single system responsible. Consistency on a single record is provided by a readers/writer lock on a single record for the according queries, this way there is a guarantee to atomicity on a single record, the full procedure is explained by Lars Hofhansl[5].

To be partition tolerant, the partition with the majority of the Zookeeper servers and a HMaster will appoint regions to available HRegionServers for them, let's call this the data serving partition. With this approach, it is important to place the Zookeeper and HMaster servers in diverse location as otherwise a partition of only this servers will make the whole system unavailable.

In HBase, a node will be able to answer to requests if the node is present in the data serving partition. Only in rare cases that all data copies of Hadoop are stored in the other, inactive cluster, the data will be unreachable. The nodes not in the data serving partition, will be unable to complete any requests.

When a server goes down, he can release the lease in case there is a graceful shut down (the HBase server is notified) and another server can get the lease immediately. In other cases, a new lease can only be given after the decay of the old lease, if the server comes back on-line in meantime, he will still be responsible.

3.2. MongoDB

MongoDB[6] is an open-sourced, distributed database designed for document storage, this are data entries where the format of each record can be different. According to their website they provide high performance and high availability, but this is incorrect to the given definition in this article.

MongoDB provides data replication and data distribution, the first is done by grouping different MongoDB servers into a ReplicaSet, the second is done by grouping different of these ReplicaSets.

A ReplicaSet exists out of different MongoDB server which work as a master/slave configuration. A master is a primary and the slave is called a secondary. The primary is responsible for the write transactions, by default a query will succeed once it has a confirmation that the write has been executed on the primary. The read operation will go by default on the primary as well. Both query methods are configurable to give other guarantees, for a write operation there are multiple *write concerns*, it is possible to wait till it has written on hard disc or a number of secondary servers, however all need a primary. For read operation there are multiple *read preferences*, it is possible to read from a secondary or the closest server.
In the default configuration, there will be consistency guarantee.

In a ReplicaSet, for the primary election there is at least half of the ReplicaSet needed. As there is always a primary needed, the system has partition tolerance but no high availability, contrary to the statement on the documentation. However, it is possible to read from a secondary but writing is not possible.

The state of the different members of a ReplicaSet is maintained by a heartbeat system: a server is marked as off-line if no beat has been received for 10 seconds. In case the primary goes off-line, election will be started to re-elect a new one. In other words, a primary has a lease of 10 seconds. This value of 10 seconds is non-configurable.

De data distributions happens by merging replica sets in a cluster. Furthermore, there is the need for access server (as many as you want) and configuration servers (1 or 3). The availability and partition tolerance of the data is the same as in the ReplicaSets as it handled by the ReplicaSets. In case a access server can't reach a primary, another access server will be needed to write data, in case a majority of the configuration servers are not reachable, their will be no reconfiguration of the data over the different servers.

3.3. Differences between databases

Both systems provide consistency and partition tolerance and forfeit high availability, but some differences are in their implementation.

First of all in their partition tolerance, in HBase there are dedicated management servers (HMaster and Zookeeper) to distribute the responsibilities of regions and if the management servers are in a partition with a minority of the data servers, data will be not available. In MongoDB the management of a ReplicaSet is done internally and the write queries will be available in the partition with the majority of the data servers.

Both decision has their influence, in HBase it is possible to write every record, as a new region can be created if the old region is unavailable. In MongoDB it is possible that in sharding you can read all the data from multiple secondaries but only write given ranges of records.

In availability, there is a small difference: where in MongoDB it is possible to read from secondaries, this is impossible in HBase.

In section 5, a detailed analyse will be done to the consistency and the behaviour of the systems in the case of network or server failure.

4. Test method

To test the behaviour of database systems towards consistency and availability, the Yahou Cloud Serving Benchmarking (YCSB [7]) has been extended with event support for availability and reader-writers for consistency.

Each of this tests follows the same steps: calibrating the system, records are preloaded, the test is started and in the end all the records are removed again and they are executed for HBase and MongoDB. During the calibration, a workload is chosen so there is a medium load on the different databases.

4.1. Event support

The implementation of event support integrates a way to execute given UNIX commandos at specified moments, the execution time and result code is logged.

In these tests were 3 kinds of tests executed of which each one took 900 second. The first action takes place at 300 seconds, the second at 600 seconds.

- Graceful shut down of a data service and restart of the service
- Hard kill of the data service and a restart of the service
- Blocking of all network traffic from and to a server and the allowance of all network traffic

Each one tests the behaviour of the system under different circumstances, the first two checks what happens in case of a respectively planned and crash shut down, the latter check the behaviour in case the network fails.

These tests are executed on each of the data nodes of HBase and MongoDB.

5. Results

(50 threads, 600 queries/second) and MongoDB (15 threads, 200 queries/second). These values were based

6. Related work

7. Conclusion

[1] E. A. Brewer, Towards robust distributed systems, in: PODC, 2000, p. 7.
[2] E. Brewer, Cap twelve years later: How the" rules" have changed, Computer 45 (2) (2012) 23–29.
[3] Hbase, apache hbase.
URL `https://hbase.apache.org/`
[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, Bigtable: A distributed storage system for structured data, ACM Transactions on Computer Systems (TOCS) 26 (2) (2008) 4.
[5] L. Hofhansl, Hbase: Acid in hbase (3 2012).
URL `http://hadoop-hbase.blogspot.be/2012/03/acid-in-hbase.html`
[6] The mongodb 2.6 manuel.
URL `http://docs.mongodb.org/manual/`
[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with ycsb, in: Proceedings of the 1st ACM symposium on Cloud computing, ACM, 2010, pp. 143–154.