NoSQL for Dependable Systems

Richard Metzler, Jan Schütze.

Hasso-Plattner-Institut
Universität Potsdam
http://www.hpi.uni-potsdam.de/

Abstract. The fault model for very large e-commerce websites like Amazon is fundamentally different from standard websites. These websites loose money when the aren't available (or just slow) for potential customers but can't risk to loose any data. The data has to be replicated between databases but traditional RDBMSs may not fit. This paper discusses some of the better known NoSQL software products available today

1 Fault Model

On very large e-commerce websites like Amazon people order every minute *TODO: WRITE SOME FACTS*. Amazon has statistics showing a causal connection between response time of the amazon.com website and the time potential customers spend on the website. *TODO: SOURCE?* The customer's shopping cart has to be always accessible for writes and the slightest outage has direct significant financial consequences.

But on the other side failures are the normal case, not an exception. Disks fail, the network experiences partitioning and whole data centers could become potentially unavailable because of natural disasters like hurricanes.

What our big e-commerce websites need is a datastore that is always read and write enabled, even in presence of network partitions. Data must be replicated across multiple datacenters and these datacenters may be located hundreds of kilometers away from each other and even on different continents.

2 Replication

Replication is one of the fundamental ideas for fault tolerant systems. But replicating data across datacenters located several hundreds of kilometers away from each other takes time. Using a traditional RDBMS with ACID style transactions to replicate data in a distributed transaction may be slow and not very scalable. Synchronous atomic updates would not be tolerant towards network partitions.

Asynchronous updates can't be atomic, but they are potentially more resistant in case of network partitioning as these are usually transient faults.

2.1 Split Brain

In distributed systems the interconnect between nodes is a weak spot. If it is broken, nodes are split into partitions unable to communicate and thus unable to share state. This scenario is called *split brain*. Nodes in split brain scenarios must be prevented from producing inconsistant state and one method to prevent inconsistency is the quorum consensus.

2.2 Quorum

As the system replica managers in different partitions cannot communicate with each other, the subgroup of replica managers within each partition must be able to decide independently whether they are allowed to carry out operations. A quorum is a subgroup of replica managers whose size gives it the right to carry out operations. **CITE: COULORIS** One possible criteria for a quorum may be having a majority. Any other partition would be smaller than the majority partition and as a consequence only the majority partition would be the quorum. Another possible quorum criteria could be the availability of a *quorum device*. The partition that is able to access the quorum device is allowed to carry out operations.

3 Brewer's CAP Theorem

In 2000 Eric Brewer at this time chief scientist of Inktomi hold a keynote at the "Principles of Distributed Computing" conference. He presented his assumption that was later proved in "Brewers Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services" stating that atomic data consistency, high availability (i.e. performance) and network partition tolerance can't be achieved all together at any given time and you may get only two of these properties for every distributed operation. This is called the CAP Theorem after the acronym for **C**onsistency, **A**vailability and **P**artition tolerance.

Because it is impossible to prevent network partitions in large networks the decision has to be between high availability and data consistency. As stated, large e-commerce websites usually go for high availability and trade consistency for that.

4 Eventual Consistent

Werner Vogels, CTO at Amazon, presented in his article "Eventually Consistent" **CITE: eventually-consistent**_ his idea of data being not consistent through atomic transactions but only eventually consistent. By trading ACID's atomicy and consistency for performance and partition tolerance it is possible to increase the response time and fault tolerance of websites. The database replications may not be fully consistent but a customer wouldn't usually experience any inconsistencies.

He defined the **inconsistency window** as "The period between the update and the moment when it is guaranteed that any observer will always see the updated value."

5 N / W / R Replica Configuration

Vogels introduces the reader to a short notation for replication configuration for *quorum* like systems:

- **N** is the number of nodes, that store replicas of the data
- **W** is the number of replicas that acknowledge a write operation
- **R** is the number of replicas contacted in a read operation

To avoid ties in failover scenarios usually an odd number is picked for N.

With these numbers you are guaranteed *strong consistency* if following condition holds:

- $N < W + R$

This is because the set of replicas for writing and reading the data overlap.

If your replica configuration only holds the condition

- $N >= W + R$

it only guaranties weak or eventual consistency.

A RDBMS is typically configured with $\{N = 2, W = 2, R = 1\}$ while $\{N = 3, W = 2, R = 2\}$ is a common configuration for fault tolerant systems.

It is possible to deduce different attributes from these configuration properties. Consistency over all nodes is reached if $W = N$.

Read optimized systems will use $R = 1$, while write optimized systems use $W = 1$.

Cassandra is able to run in application specific N / W / R configuration. This helps Cassandra to recover from transient and permanent failures.

6 Ring topology

Cassandra and Riak are both heavily inspired by Amazon's Dynamo and both organize nodes in a ring topology. Also Cloudant's BigCouch **CITE: BIGCOUCH** of CouchDB is closely modeled after Dynamo and features a ring to replicate CouchDB instances. N consecutive nodes on the ring form one replica set and replica sets overlap. By distributing the nodes in the physical space a better fault tolerance should be obtained.

7 Products

There are several NoSQL systems available. We focused on 4 of the major ones:

- Riak (document oriented)
- Cassandra (column oriented)
- CouchDB (document oriented)
- MongoDB (document oriented)

In terms of the CAP theorem: Riak, Cassandra and CouchDB provide availability and partition tolerance. MongoDB on the other hand provides consistency and partition tolerance.

We installed those on multiple virtual machines, connected them with each other and ran some very simple tests to figure out how they behave in case of a fault.

7.1 Riak 0.11.0

Riak is created and maintained by Basho (a company founded by Ex-Akamai employees) and the de facto open source reference implementation of the dynamo paper. It is released under the terms of Apache License 2.0 .

Riak is a document oriented key value store and also supports links between them. Documents are stored in so called buckets. Riak is written entirely in Erlang and has an HTTP interface to read and write data.

More information about Riak may be found at `https://wiki.basho.com/display/RIAK/Riak`.

Replication Config One of Riaks features is the variable configuration of N W R:

- N can vary for each bucket
- R & W can vary for each operation (read/write/delete)

There are also additional quorums for:

- durable writes to disk
- deletes

7.2 Cassandra 0.6.3

Cassandra is a column oriented distributed database system written in Java. It was created by Facebook and donated to the Apache Foundation. Cassandra is released under the Apache License 2.0.

Cassandra's architecture is inspired by Amazon's Dynamo and the Google BigTable.

When developing applications with Cassandra the developer need to configure the collumns before Cassandra can be started. Thus Cassandra is not schema

less like one may expect from the term NoSQL. Once configured you can search and order the documents by these columns.

More information about Cassandra may be found at the official website at `http://cassandra.apache.org/` or at the wiki at `http://wiki.apache.org/cassandra/FrontPage`.

7.3 MongoDB 1.4.3

MongoDB is a document oriented distributed database system by 10gen. It is available under the terms of GNU Affero General Public License.

It uses a custom TCP protocal (BSON) and is written in C++.

More information about MongoDB is available at the official website at `http://www.mongodb.org/` or at the wiki at `http://www.mongodb.org/display/DOCS/Home`.

7.4 Replication

There is Master/Slave replication in MongoDB available. In this case one can read from the slaves and the master, but write only into the master.

Additionally there are Replica Pairs available. When using Replica Pairs only one of two nodes is the master at any time. Read and write is only possible on the master of the Replica Pair.

In case one node of the Replica Pair fails the other one is made the new master. Deciding who is the master can be done by an external arbiter (Quorum Device).

The MongoDB Team is currently working on Replica Sets, which are meant to allow more then 2 machines to be part of the replication configuration.

7.5 Crash

In case of a MongoDB crash, the entire database must be reindexed again. According to David Mytton's blogpost `http://blog.boxedice.com/2010/02/28/notes-from-a-production-mongodb-deployment/` this takes up to 72 hours for 664.000.000 database entries.

That's why MongoDB has an increased MTTR (Mean Time To Repair).

7.6 CouchDB 0.11.0

CouchDB is a document oriented database written in Erlang with support for JavaScript views. CouchDB is an Apache Project.

Access to the data is made available via an HTTP interface by exchanging JSON objects.

More information about CouchDB is available at the official website at `http://couchdb.apache.org` or at the wiki at `http://wiki.apache.org/couchdb/FrontPage`.

8 Experiments

To test the fault tolerance features of the distributed database systems, we focused on the behavior in case of network splits and synchronization after adding new nodes.

For this purpose we set up the virtual machines “Alice” and “Bob”. Both run a vanilla Debian Squeeze release in Virtual Box. For the Cassandra experiments we added a third identical machine called “Charly”.

8.1 Experiment 1

The first experiment is meant to show what happens if a new node joins the distributed database.

For this purpose we set up the node Alice and pushed 1000 data records into Alice. Then Bob joined the network. To check if Bob already had all data we frequently tried to read the 1000th entry from Bob. If this was possible we assumed that Bob was in sync or at least capable to answer in a consistent way.

Results (Replicating to a new node):

- Riak: 1 second
- Cassandra (3 nodes): 20 seconds
- CouchDB: 1 second
- MongoDb: 2 seconds

8.2 Experiment 2

To test how the distributed database system is able to manage a network split, we set up the second experiment.

The two nodes Alice and Bob were synchronized and connected. Then we deactivated Bob’s network and wrote 1000 data records into Alice. Because of the network partition it is impossible for Bob to have the fresh data. We turned on the network and timed how long it takes for Bob to receive all data entries (by querying for the 1000th entry).

Results (replication after network split):

- Riak: 6 seconds
- Cassandra (3 nodes): 20 seconds
- CouchDB: 8 seconds
- MongoDb: Failed, because reading from Slave returns an error

When configured as replica pair it is not possible to read from the MongoDB Slave.

8.3 Experiment 2b

Since we had MongoDB as Replica Pair in the experiment 2, it was impossible to read from the slave. That's why we made an experiment 2b with a slightly different configuration.

We set up Alice and Bob as Replica Pair. Another MongoDB instance "Charly" was used as arbiter (Quorum Device). MongoDB chose the first one (Alice) to be the master and Bob the slave. Then we pushed 1000 data records into Alice.

We stopped Alice in three different ways:

- by removing the network connection
- by stopping it gracefully
- by using "kill –9"

After that we timed how long Bob needs to recognize that Alice has disappeared and Bob become master.

Result: It took 1 second for Bob to become Master and thus allowing the client to read from Bob.

We noticed that stopping the node and kill –9 worked great. But Bob did not notice the network split if we just removed the network connection. We assume that this is because the virtual machine does not send any interupt like a physical network interface would waiting for the TCP connection to time out.

9 Sources

- Eric Brewer: "Towards Robust Distributed Systems" `http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf`
- Gilbert, Lynch: "Brewers Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services"
- Werner Vogels: "Eventual Consistent"
- W. Vogels et all: "Dynamo: Amazon's highly Available Key-Value Store
- Lakshman, Malik: "Cassandra - A Decentralized Structured Storage System"
- David Mytton: "Notes from a production MongoDB deployment" `http://blog.boxedice.com/2010/02/28/notes-from-a-production-mongodb-deployment/`
- Coulouris et al: "Distributed Systems. Concepts and Design"

References

1. Google research publication: Mapreduce.
2. Joshua Bloch. *Effective Java (2nd Edition)*. The Java Series. Prentice Hall PTR, Upper Saddle River, NJ, USA, second edition, 2008.
3. Jason Venner. *Pro Hadoop*. Apress, Berkely, CA, USA, 2009.