NOSQL in Dependable Systems

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.

Keywords: Fault Tolerance, CAP Theorem, NoSQL, Dynamo, Riak, Cassandra, MongoDB, CouchDB

Authors: Richard Metzler @rmetzler, Jan SchÃ¼tze @dracoblue

TODO: EXPLAIN QUORUMS (Fault tolerant patterns II page 11) TODO: Explain Amnesia TODO: Explain Split Brain

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.

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.

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 "Brewerâ s 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 you can't do anything against network partitions in large networks you have to pick between high availability and data consistency. As stated, large e-commerce websites usually go for high availability and trade consistency for that.

Eventual Consistent

Werner Vogels, CTO at Amazon, presented in his famous article *TODO:SOURCE* his idea of data being "Eventual 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 â The period between the update and the moment when it is guaranteed that any observer will always see the updated value.â

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 as 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 speciï¬ c N / W / R conï¬ guration. This helps Cassandra to recover from transient and permanent failures.

Products

we installed and ran some simple tests on

- Riak
- Cassandra
- CouchDB
- MongoDB

Disclaimer: very, very simple tests...

Riak 0.11.0

- Key-Value Store (Document oriented)
- Links
- de facto reference implementation of the Dynamo paper
- created by Basho (ex Akamai), APL 2.0
- written in Erlang, HTTP Interface
- https://wiki.basho.com/display/RIAK/Riak

Replication Config

- N can vary per bucket
- R & W can vary per operation
- additional Quorums for
- durable writes to disk
- deletes

Cassandra 0.6.3

- column oriented
- created by Facebook, maintained by Apache
- inspired by Amazon Dynamo and Google BigTable
- written in Java
- http://cassandra.apache.org/
- http://wiki.apache.org/cassandra/FrontPage

MongoDB 1.4.3

- Key Value Store (Document oriented)
- uses custom TCP Protocol (BSON)
- written in C++
- created by 10gen, AGPL
- http://www.mongodb.org/
- http://www.mongodb.org/display/DOCS/Home

Replication

- Master/Slave Replication
- Replica Pairs
- Arbiter decides who is Master (Quorum)
- only allowed to write / read from Master
- Replica Sets coming soon...

crash

- reindex MongoDB on Crash Fault
- may take several hours
- increased MTTR

CouchDB 0.11.0

- Key Value Store (Document oriented)
- HTTP interface, JSON objects
- Apache project
- written in Erlang
- http://couchdb.apache.org/
- http://wiki.apache.org/couchdb/FrontPage

Experiments

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

For this purpose we set up two machines â Aliceâ & â Bobâ (+ "Charles"). Both run a vanilla Debian Squeeze Release in a Virtual Box.

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. Now Bob joins the network. To be check if Bob already has all data we frequently tried to read the 1000th entry from Bob. If this was possible we assumed that Bob was 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 second

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+Bob are synchronized and connected. Now we deactivated Bob's network and after that put 1000 data records into Alice. In this case it is impossible for Bob to have the fresh data, because a network split happened. We turned on the network and checked again, how long it takes for Bob to receive all data entries (by querying for the 1000th entry).

Results (Replicating after network split):

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

In MongoDB it's not possible to read from the Slave, when configured as Replica Pair.

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 "Charles" was used as Arbiter (Quorum Device). MongoDB choose the first one (Alice) to be the Master and Bob the Slave. Now we pushed 1000 data records into Alice.

Then we stopped Alice in 3 ways:

- removed the network connection
- stopped it gracefully
- used: kill -9

After that we checked how long it takes Bob to recognize that Alice had disappeared and Bob becomes Master on its own.

Results: 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 it did not notice the network split, if we just removed the network connection. We assume that this is because of a timeout on the tcp layer.

Sources

- Eric Brewer: â Towards Robust Distributed Systemsâ
  http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- Gilbert, Lynch: â Brewerâ s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Servicesâ
- Werner Vogels: â Eventual Consistentâ
- W. Vogels et all: â Dynamo: Amazonâ s highly Available Key-Value Storeâ
- Lakshman, Malik: â Cassandra - A Decentralized Structured Storage Systemâ