Couchbase for Python developers

HotCode 2013

What is Couchbase?

Couchbase

- http://www.couchbase.com
- Open source NoSQL database technology for interactive web and mobile apps
- Easy scalability
- Consistent high performance
- Flexible data model

History. Membase

- Started in June 2010 as Membase
  - Simplicity and speed of Memcached
  - But also provide storage, persistence and querying capabilities
  - Initially created by NorthScale with co-sponsors Zynga and NHN

History. Merge with CouchOne

- In February 2011 Membase merged with CouchOne
  - CouchOne company with many developers of CouchDB
- After merge company named Couchbase
- In January 2012 Couchbase 1.8 released

History. Couchbase 2.0

- On December 2012 Couchbase 2.0 released
  - New JSON document store
  - Indexing and querying data
  - Incremental MapReduce
  - Cross datacenter replication

In total

- CouchBase is
  - Memcached interface for managing data
  - CouchDB based interface for indexing and querying data
  - Scalability (clustering, replications) out of the box

Why we need another NoSQL solution?

What we have?

- Key-value cache in RAM
  - Redis
  - Memcached
- Eventually-consistent key-value store
  - Cassandra
  - Dynamo
  - Riak

What we have?

- Document store
  - CouchDB
  - MongoDB
  - {{ name }}DB

Why we need another?

- Cause developers can
- This is cool to be a NoSQL storage author
- There should be a serious NoSQL storages for business
- But really, I don’t know
- I still prefer Redis + PostgreSQL more than Couchbase :)

CouchBase as cache storage

Couchbase is Memcached

- All Memcached commands* work with Couchbase
  - set/add/replace
  - append/prepend
  - get/delete
  - incr/decr
  - touch
  - stats
- *except of flush

How it works?

- Connect to Memcached socket using binary protocol
- Authenticate with or without password
- Send Memcached command as request
- Receive response from Memcached

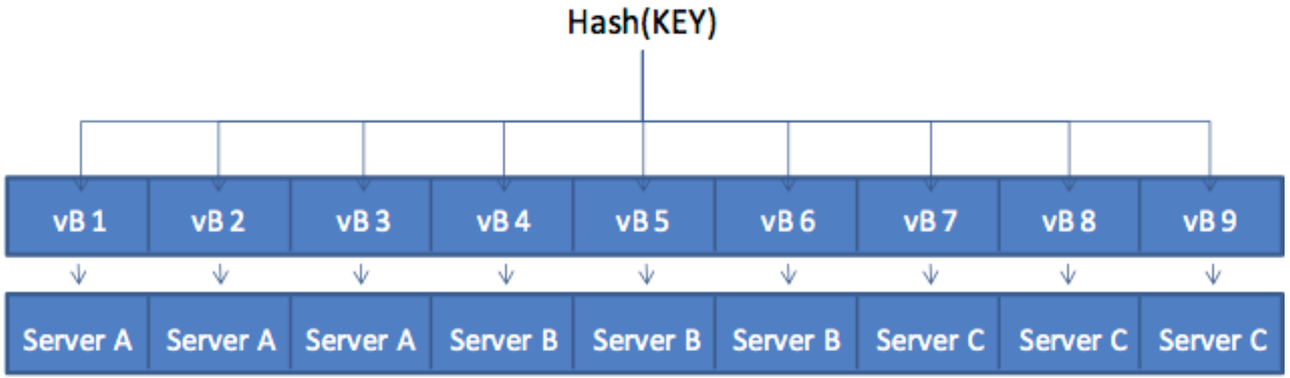
Differences

- Additional commands
  - lock/unlock
- vBucketID value **should** present in each key request

What is vBucket?

- http://dustin.github.io/2010/06/29/memcached-vbuckets.html
- http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-introduction-architecture-vbuckets.html
- vBucket is computed subset of all possible keys
- vBucket system used for distributing data and for supporting replicas
- vBucket mapping allows server know the fastest way for getting/setting data
- Couchbase vBucket implementation differs from standard Memcached implementation

vBucket mapping

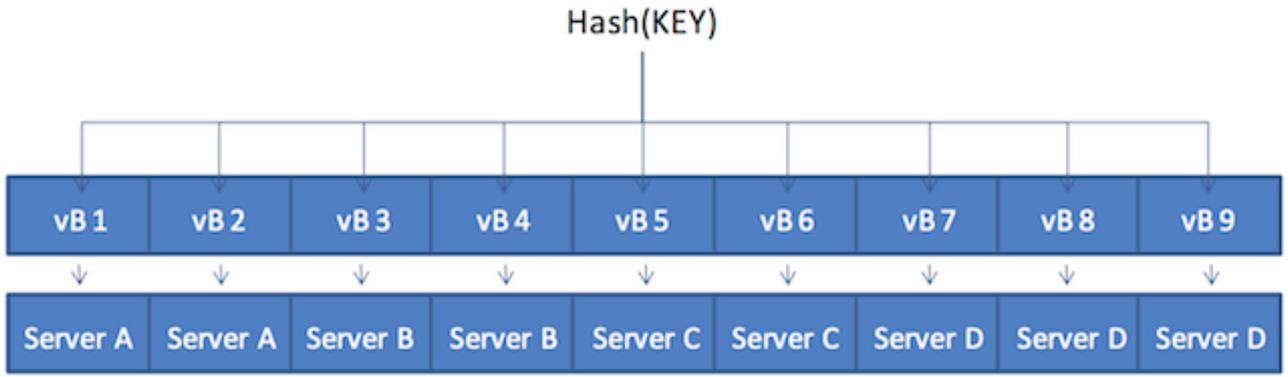


9 vbuckets, 3 clusters

if hash(KEY) == vB 8:

    read key from server C

vBucket mapping after new node added



9 vbuckets, 4 clusters

if hash(KEY) == vB 8:

    read key from server D

CouchBase as document storage

Same between CouchDB and Couchbase

- NoSQL document database
- JSON as document format
- Append-only file format
- Same approach for indexing and querying
- CouchDB replication technology is base for Couchbase cross datacenter replication

Differences

- Couchbase cache-based database in terms of read/write performance
- CouchDB disk-based database
- Couchbase has a built-in clustering system that allows data spread over nodes
- CouchDB is a single node solution with peer-to-peer replication
- Each solution has their admin interfaces (Couchbase server admin, Futon)

In total

- **Couchbase is not CouchDB**

Indexing and querying data

- Say hello to views!
- First you need to create design document (via REST API or admin UI)
  - Design document is collection of views
- Next you need to create view and provide map/reduce functions
  - View should contain map function and could contain reduce function
- After Couchbase server indexing data by executing map functions and stores result in sorted order

View map functions

```
# Returns all document IDs from bucket
function(doc, meta) {
   emit(meta.id, null);
}

# Returns only JSON document IDs
function(doc, meta) {
    if (meta.type == “json”) {
        emit(meta.id, null);
    }
}

# Multiple returns in one view
function(doc, meta) {
    if (meta.type == “json”) {
        if (doc.name) {
            emit(doc.name, null);
        }
        if (doc.nameLong) {
            emit(doc.nameLong, null);
        }
    }
}
```

Querying indexed data

- Querying available via REST API
- Can filter results by exact key
- Or using range (startkey, endkey)
- All key requests should use JSON format
- Also can group results, reverse results, paginate results

View reduce functions

- When you need data to be summarized or reduced
- Reduce function could be defined in view or
  send while querying view via REST API
- Built-in functions
  - _count
  - _sum
  - _stats

Overview of Python clients

couchbase < 0.9

- Very slow official Python client
- Supports Memcached commands and REST API
- Not ready for any usage
- ~800 ops/sec on my machine

couchbase >= 0.9

- Python client based on C libcouchbase library
- Only supports Memcached commands
- No REST API support
- ~8000 ops/sec on my machine

couchbase >= 0.9

```
from couchbase import Couchbase
couchbase = Couchbase()
client = couchbase.connect(bucket, host, port, username, password)
client.set('key', 'value')
assert client.get('key').value == 'value'

from couchbase import FMT_JSON
client.set('doc', {'key': 'value'}, FMT_JSON)
assert client.get('key').value == {'key': 'value'}
```

mcdonnell

- Experimental Python client created by myself
- Not open sourced yet :(
- Supports Memcached commands and REST API
- Has Django cache backend and Celery result backend
- ~6000 ops/sec on my machine

mcdonnell

```
from mcdonnell.bucket import Bucket
client = Bucket('couchbase://username:password@host:port/bucket')
client.set('key', 'value')
assert client.get('key') == 'value'

from mcdonnell.constants import FLAG_JSON
client.set('doc', {'key': 'value'}, flags=FLAG_JSON)
assert client.get('key') == {'key': 'value'}
```

mcdonnell

```
import types
from mcdonnell.bucket import Bucket
client = Bucket('couchbase://username:password@host:port/bucket')
results = client.view('design', 'view')
assert isinstance(results, types.GeneratorType)

from mcdonnell.ddocs import DesignDoc
ddoc = DesignDoc('design_name')
ddoc.views['view_name'] = map_function
ddoc.upload()
```

How we use CouchBase in real life in GetGoing?

What we have?

- 3 m2.xlarge EC2 instances
- 45 GB RAM
- 1.43 TB disk

For what reasons?

- Search results cache (with ttl)
- User cache (persistance)
- Hotels data document storage (persistance)

Clustering, replications and other NoSQL buzzwords

Clustering

- New node could be added/deleted via command line or REST API
- After node added/deleted cluster should be rebalanced
- Can auto-failover broken nodes, but only for 3+ total nodes in cluster

Cross datacenter replication

- Could replicate to other Couchbase cluster
- Could replicate to other storage, like ElasticSearch

Couchbase -> ElasticSearch replication

- ElasticSearch is search engine built on top of Apache Lucene, same to Solr, but more REST API friendly
- Couchbase has official transport to replicate all cluster data to ElasticSearch
- This enables fulltext search over cluster data

Other

- Couchbase has experimental geo views support
- Couchbase has two editions: community and enterprise

Questions?

I am **Igor Davydenko**

http://igordavydenko.com

http://github.com/playpauseandstop