basho

No more fighting with your siblings

Riak Data Types (CRDTs) remove the stress

EUC Stockholm 2016 Tutorial session

Magnus Kessler

Customer Services Engineer @ Basho

In this session…

- An introduction to Riak
- Hands-on with Riak
- An introduction to CRDTs
- Hands-on with CRDTs in Riak
- Conclusion / Questions

… first, let's get a hands-on setup

- Everybody should already have vagrant, virtualbox and git.
- `vagrant box add bento/centos-7.2`
- `git clone https://github.com/kesslerm/crdt_tutorial.git`
- `cd crdt_tutorial`
- `git submodule init && git submodule update`
- `vagrant up`

basho

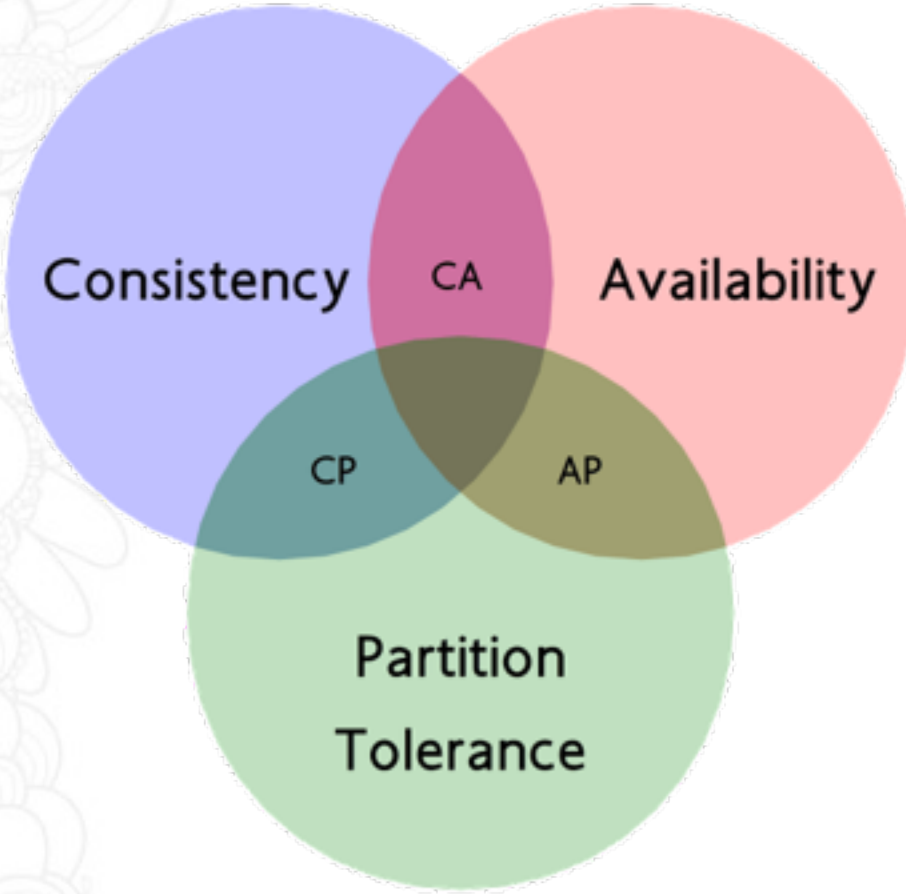
Riak KV Introduction

What is Riak KV

- a distributed Key-Value store
- highly available
- fault-tolerant
- horizontally scalable
- low-latency
- simple to operate

The CAP Theorem

In a distributed system, when faced with network partitioning, you can not have full strong consistency and complete availability at the same time.



basho

Riak Objects

- Riak objects associate Values with a namespace and metadata
- Riak is agnostic to the content of the Value
- Bucket-types and Buckets allow to namespace objects
- Metadata is magic… :)

A Cluster of Equals

- A Riak cluster consists of one or more nodes (typically 5 or more)
- All Riak nodes are equal, there are no special nodes

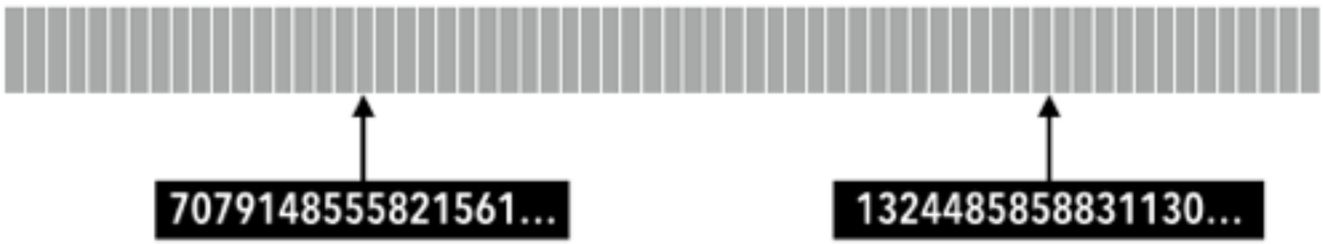
Consistent Hashing

- Each object is assigned an address, calculated from a SHA1 hash of its bucket-type, bucket, and key name.

basho

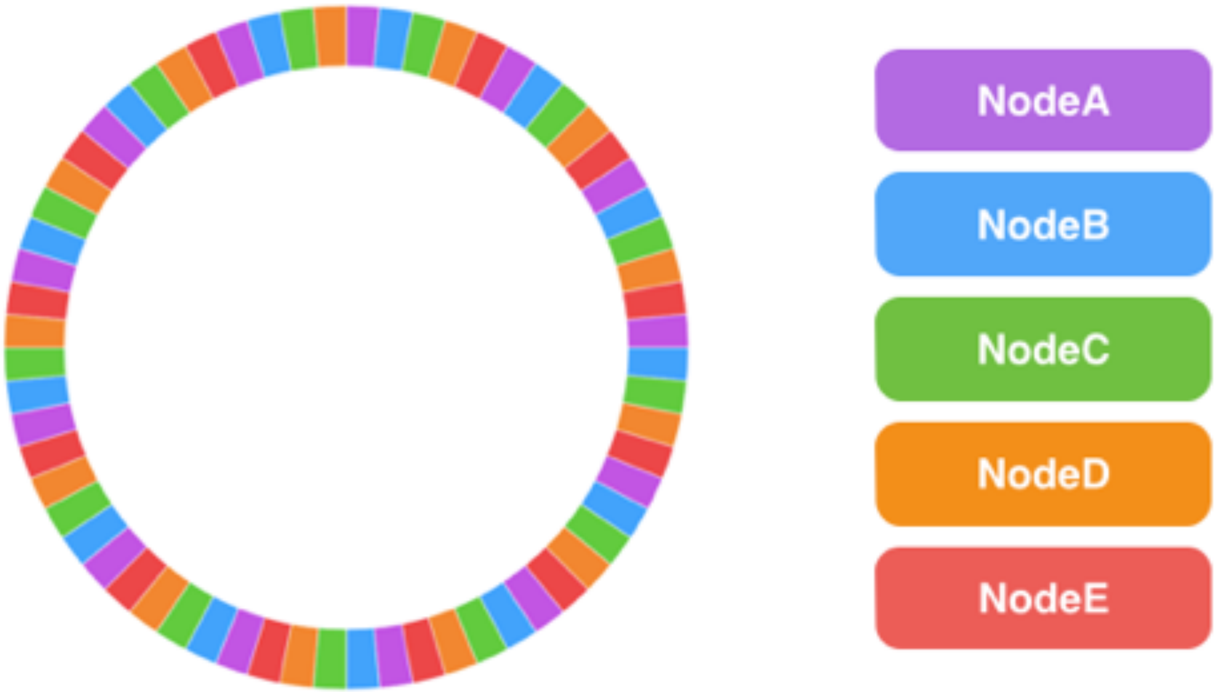
Consistent Hashing

- The Hash space is divided into equal chunks, called partitions



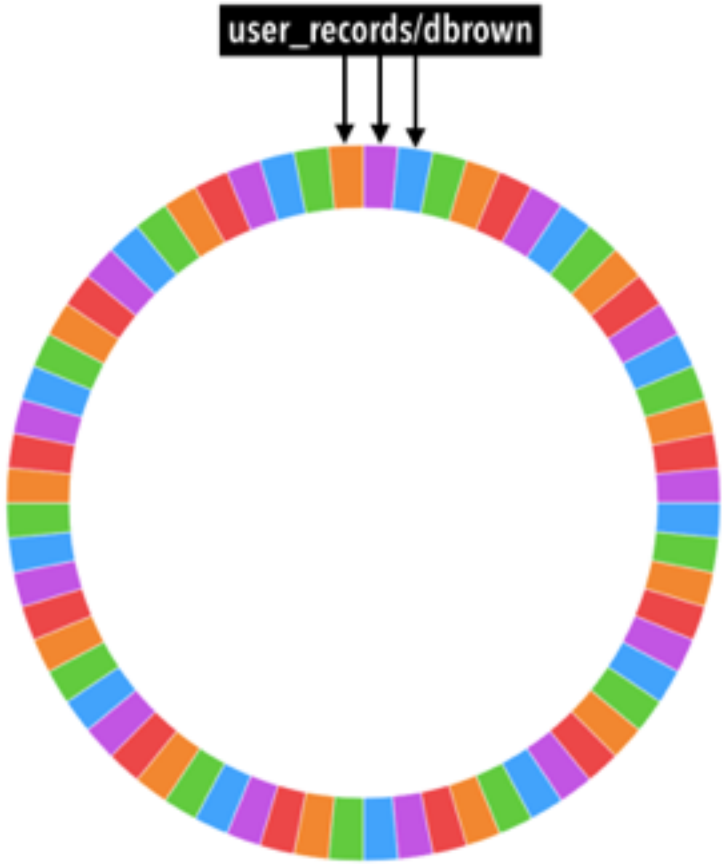
Consistent Hashing

- Partitions (a.k.a VNodes) are assigned to Nodes. Data is distributed throughout the cluster



Consistent Hashing

- Replicas of data objects guarantee availability



Conflict Resolution

Conflict Resolution

- Network partition and concurrent actors modifying the same data can cause data divergence.
- Two main mechanisms available in Riak
  - Timestamps - The copy with the most recent timestamp wins
  - Causal Context or Version Vectors - “Sibling” copies of data are retained and client or server code provides a merge function that resolves siblings to the correct state.

Conflict Resolution - Strategies

- Pick one at random
- Pick the last written one
- Use the highest / lowest value
- Use the union of all siblings
- Use domain specific knowledge to merge more complicated objects
- What's your favourite?

Let’s play!

Time for some hands-on experience with
a simple Riak use case: Counters

Lessons learned - Counters

- Even increment-by-one counters can be tricky
- Time stamp based resolution may miss concurrent updates
- Time stamp based resolution misses updates during a network split
- Simplistic sibling resolution can work well, but corner cases arise from network splits

More Hands-On

Let’s get physical with Sets

Lessons learned - Sets

- Sets are well supported in Erlang
- A G(row-only)-Set is straightforward to implement
- Removing elements can prove tricky with concurrent updates

CRDTs

CRDTs

- Conflict-free Replicated Data Types
- Based on 3 properties:
  - Operations are *commutative*: x # y == y # x
  - Operations are *associative*: (x # y) # z == x # (y # z)
  - Operations are *idempotent*: x # x == x
- Two flavours:
  - Operations based, a.k.a Commutative Replicated Data Types (CmRDTs)
    - *commutative* and *associative* operations, but individual updates are not *idempotent* and require assistance from the distribution mechanism.
  - State based, a.k.a Convergent Replicated Data Types (CvRDTs)
    - State elements are defined so that all 3 properties are satisfied.
- Riak's server side implementation uses CvRDTs, but the client APIs expose an operations based interaction model.
- For more information see
  - https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type (quite readable!)
  - "A comprehensive study of Convergent and Commutative Replicated Data Types", M. Shapiro, et. al. https://hal.inria.fr/inria-00555588

CRDTs in Riak

- Counters
- Sets
- Maps
- Flags† (boolean, true/enabled wins)
- Registers† (opaque binary value, last write wins)

CRDTs in action

Let's revisit our previous examples

CRDT - Map Datatype in Erlang

```
% Create a new Map object
Map = riakc_map:new(),
% Registers nested in map for personal info
Map1 = riakc_map:update({<<"first_name">>, register},
                        fun(R) -> riakc_register:set(<<"Sigge">>, R) end, Map),
Map2 = riakc_map:update({<<"phone_number">>, register},
                        fun(R) -> riakc_register:set(<<"5551234567">>, R) end, Map1).
% Counters can be nested, too
Map3 = riakc_map:update({<<"page_visits">>, counter},
                        fun(C) -> riakc_counter:increment(1, C) end, Map2),
% Flags must be nested in a map
Map4 = riakc_map:update({<<"enterprise_customer">>, flag},
                        fun(F) -> riakc_flag:disable(F) end, Map3),
% Sets can be nested in a map
Map5 = riakc_map:update({<<"interests">>, set},
                        fun(S) -> riakc_set:add_element(<<"robots">>, S) end, Map4),
Map6 = riakc_map:update({<<"interests">>, set},
                        fun(S) -> riakc_set:add_element(<<"opera">>, S) end, Map5),
Map7 = riakc_map:update({<<"interests">>, set},
                        fun(S) -> riakc_set:add_element(<<"motorcycles">>, S) end, Map6),
```

basho

CRDT - Map Datatype in Erlang

```
% Nested elements can be updated at any time
Map8 = riakc_map:update({<<"interests">>, set},
                        fun(S) -> riakc_set:del_element(<<"opera">>, S) end, Map7),
Map9 = riakc_map:update({<<"interests">>, set},
                        fun(S) -> riakc_set:add_element(<<"indie pop">>, S) end, Map8),
% Nested Maps!
Map9 = riakc_map:update(
    {<<"annika_info">>, map},
    fun(M) -> riakc_map:update(
        {<<"first_name">>, register},
        fun(R) -> riakc_register:set(<<"Annika">>, R) end, M) end,
    Map8),
Map10 = riakc_map:update(
    {<<"annika_info">>, map},
    fun(M) -> riakc_map:update(
        {<<"phone_number">>, register},
        fun(R) -> riakc_register:set(<<"5559876543">>, R) end, M) end,
    Map9).
% and even maps within maps. Don't drink too much of the cool-aid!
```

CRDT - Map Datatype in Erlang

```
% finally, write local state back to the server
{ok, Pid} = riakc_pb_socket:start_link(“localhost”, 8087),
riakc_pb_socket:update_type(Pid, {<<“maps”>>, <<“customers”>>}, <<“sigge”>>,
                            riakc_map:to_op(Map10)).

%% For updates of existing objects, make sure to fetch the object first with
%% riakc_pb_socket:fetch_type
```

Conclusion

Conclusion

- Key concepts of Riak
- Experimented with the behaviour of Riak KV vis-a-vis concurrent updates and network partitions
- Limitations of naïve implementations of counters and sets
- CRDTs as implemented in Riak can cope with failures

Questions?

Thank You!

We're Hiring!

UK Client Services Engineer

Developer Advocate EMEA

bashojobs.theresumator.com

Visit our Stand

Experience our Riak TS demo and be entered to win a Scalextric set!

Get your invitation to our IoT Riak TS Roadshow

basho