**My Notes on Working with Antidote**

# Links

Little Riak Core Book:

<https://marianoguerra.github.io/little-riak-core-book/>

Basho riak\_core:

<https://github.com/basho/riak_core/wiki>

# Installation

**Erlang 21.3:**

$ LANG=C ./configure --with-ssl=/usr/local/Cellar/openssl/1.0.2s --prefix=/Users/shamouda/opt/otp\_21.3

$ make

$ make install

**Antidote Make Compile problem – solved by switching to the riak\_migrate branch where eleveldb is not used:**

g++ -I /Users/shamouda/GitRepo/antidote/\_build/default/lib/eleveldb/c\_src/system/include -I. -I./include -DOS\_MACOSX -DLEVELDB\_PLATFORM\_POSIX -DSNAPPY -DLEVELDB\_VSN="2.0.19" -O2 -g -DNDEBUG -fPIC tools/sst\_scan.cc -o sst\_scan -L . -lleveldb -L/Users/shamouda/GitRepo/antidote/\_build/default/lib/eleveldb/c\_src/system/lib -lsnappy

===> Compiling /Users/shamouda/GitRepo/antidote/c\_src/eleveldb.cc

===> warning: include path for stdlibc++ headers not found; pass '-stdlib=libc++' on the command line to use the libc++ standard library instead [-Wstdlibcxx-not-found]

/Users/shamouda/GitRepo/antidote/c\_src/eleveldb.cc:25:10: fatal error: 'new' file not found

#include <new>

^~~~~

1 warning and 1 error generated.

make: \*\*\* [compile] Error 1

# Testing

* Rebar comes with built in support for eunit:
  + rebar3 eunit -> compiles any file containing the macros TEST or EUNIT
* Rebar comes with built in support for common tests
  + rebar3 ct -> compiles any file ending with \_SUITE.erl

# Riak Core Components

Operating elements:

* Vnode
* Vnode master
* Ring Manager

Date:

* Ring
* Node

# Riak Functions

Get the ring

{ok, Ring} = riak\_core\_ring\_manager:get\_my\_ring()

Get the indices of the ring

Indices = riak\_core\_ring:my\_indices(Ring)

Returns the local node

node()

Sync spawn command (index, command, vnode\_master). The vnode master name is:

the module name of the vnode plus \_master

riak\_core\_vnode\_master:sync\_spawn\_command({Index, node()}, {list\_keys}, simple\_kv\_key\_store\_vnode\_master)

# Coding Task 1: Refactoring the Fold calls

So lists:foldl is a relative low level operation on lists. You usually use it if you want to compute an aggregate over a list, e.g. if you want to compute the sum of some parts in the list. However, in Antidote it is also often used for filtering lists (for this it's nicer to use lists:filter ) or transforming items in a list (lists:map).

it's not always that straight-forward, for example here <https://github.com/AntidoteDB/antidote/blob/master/src/materializer_vnode.erl#L532-L534>  
the foldl is actually used for accumulation, but it can still be simplified as vectorclock:min already takes a list and could be used

There are also some places where foldl is used but the Accumulator is not used at all. In that case you can use lists:foreach

# Questions

meta\_data\_sender.erl

Note that there is one of these state machines per physical node.

How is that configured?

# Adding partial replication to the simple kv store

riak\_core:join('simple\_kv2@127.0.0.1').

simple\_kv\_key\_store\_vnode:qget('hello').

simple\_kv\_key\_store\_vnode:qput('hello', 12312).

recruitment

# Using RAFT inside the Logging VNode

1. Riak core will eventually start a failed vnode.
2. The init function is called when the vnode starts.
3. We will need to modify init such that it agrees with the other replicas on the missed part of the log to copy it to the lagging behind node (the restarting node)

Notes on code:

1. The read functions use a PrefList of size one, but the append functions take a NodeIndex. This is an inconsistency. We need the append to take PrefList so that we can have multiple replicas.
2. Define a function append\_replicate(PrefList, …)