Probabilistic CAP implementation for Latency and Consistency SLAs

Riak Cluster Setup

Version 0.1

By:

Lavanya Kandukuri
lavanya.kandukuri@sjsu.edu

Advisor: **Prof. Rakesh Ranjan**

Table of Contents

1. Version History

Version	Description
0.1	Initial Document for Cluster Setup

2. Introduction

This document details the description of a Riak Cluster Setup which is required for the implementation of PCAP (Probabilistic CAP) system.

Riak is a NoSQL database inspired from the dynamo design. It is a distributed, decentralized data storage system which is developed in Erlang. The services consuming Riak have the freedom to tradeoff between latency or consistency depending on the service level agreement with the customers. Riak has a master-less peer to peer architecture where each node in a Riak cluster is the same. This uniform nature forms the basis of Riak's fault tolerance and scalability. Data is evenly distributed across all the nodes.

Riak is eventually consistent, which means it has a default Available, Partition-tolerant(AP) database configuration. It can be configured to be strongly consistent by setting configuration variables that ensure there is a quorum during a read. Riak has a quorum approach by default in its querying model. Riak supports both HTTP or a protocol buffer API for client interaction.

Riak Architecture:

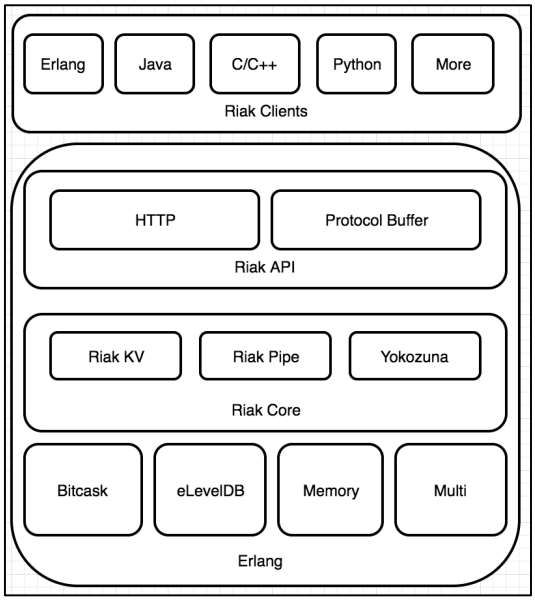

Fig.1 Riak Architecture

Riak Cluster:

Riak Cluster is a combination of multiple Riak nodes which are well connected data hosts. Consistent hashing is used to spread data across the cluster. Data is distributed evenly across the cluster which helps reduce hot spots and new nodes can be added with minimal reshuffling of data and no downtime. Riak uses SHA-1 as a hash function and treats its 160-bit value space as a ring. The ring is divided into partitions called "virtual nodes" or vnodes. Ring accommodates vnodes. Each physical node in the cluster hosts multiple vnodes. Nodes use gossip protocol to exchange their understanding on ring state.

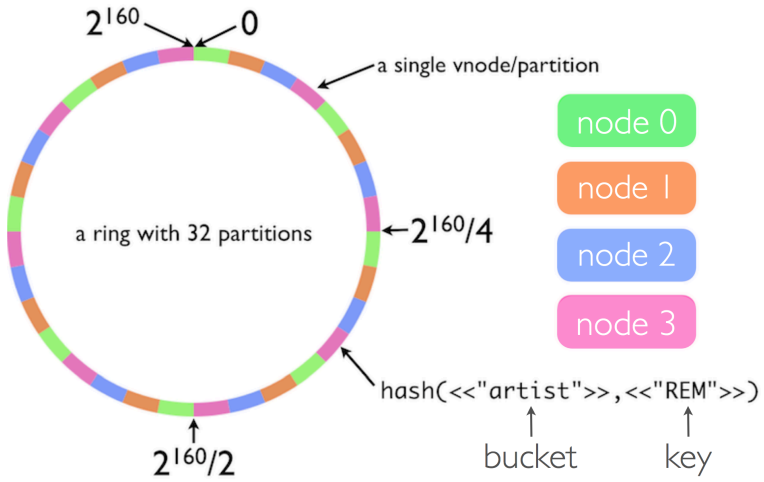

Fig.2 Riak's Ring. Adapted from 'http://docs.basho.com/riak/kv/2.2.3/learn/concepts/clusters/'

3. References

[1] Riak KV. (n.d.). Retrieved from http://docs.basho.com/.

[2] Riak Client for Java. (n.d.). Retrieved from https://github.com/basho/riak-java-client.

[3] Riak. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Riak.

[4] Rahman, M. R., Tseng, L., Nguyen, S., Gupta, I., & Vaidya, N. (2017). Characterizing and Adapting the Consistency-Latency Tradeoff in Distributed Key-Value Stores. *ACM Transactions on Autonomous and Adaptive Systems*, *11*(4), 1-36. doi:10.1145/2997654.

[5] NoSQL Database Architectural Comparison. (2017). Retrieved from https://griddb.net/en/docs/NoSQL_Database_Architectural_Comparison.pdf

[6] Vinoski, S. (2018). A Peek Inside Riak. Retrieved from http://gotocon.com/dl/goto-berlin-2013/slides/SteveVinoski_APeekInsideRiak.pdf

[7] Riak. (n.d.). Retrieved from https://aws.amazon.com/marketplace/pp/B00AMRXCQA/ref=mkt_m3_riak

4. Requirements

Requirements covered in this document are:

1. Installing Riak on local machine and running it successfully.
2. Basic configuration before setting up nodes.
3. Creating a cluster.
4. Running a Riak cluster.
5. Testing node connectivity.
6. Working on replica configuration.
7. Installing and running a Riak cluster on AWS.

5. Functional Overview

Important Terminology:

Riak bucket: Keys in Riak are separated across different virtual key spaces referred to as 'buckets'.

Riak Node: Riak node is similar to a physical server.

Riak Cluster: Riak cluster is a combination of multiple nodes.

Replication: Riak replicates data in a cluster automatically. By default, there are three replicas per object. If one node fails, others can continue to serve client requests.

Inter node communication: Nodes communicate their understanding of the ring state using "gossip protocol".

The parameters N, R, W should be taken care of before the set up.

- N is the number of replicas, which is three by default. N is set as 3 for this setup.
- R is the read quorum, i.e. number of responses required for a successful read.
- W is the write quorum, i.e. number of responses required for a successful write.

All nodes in the cluster are same – there are no masters and slaves. Riak nodes form a mesh where all nodes are aware about each other. Nodes talk to each other from time to time to check their liveliness. Hinted handoff serves the purpose of fault tolerance and scalability where fallback node holds the data for another node till it becomes available. Read Repair helps implement eventual consistency. When a node fails, or is partitioned, storage operations are taken care by adjacent nodes. When the failed node returns, responsibilities are handed back again. This is how Riak handles a failover. Since Riak allows us to simulate multi node installation on a single host, it is very good for development. Two methods have been followed for the cluster setup, one of which is a development technique where multiple nodes are hosted on a single machine.

5.1 Configuration/ External Interfaces

Configuration that needs to be done for this setup:

- **Riak:**

  Download and Install

- **Single Host:**

  1. Configuring Riak Protocol buffer.
  2. Configuring riak.conf file for data nodes.
  3. Adding nodes to the cluster.

- **AWS:**

  1. Set up required EC2 instances.
  2. Configure security group settings.
  3. Connect instances using SSH.
  4. Cluster creation using Riak commands.

5.2 Debug

For any problems with Riak KV nodes, the command **riak-debug** can be used. This command can identify and diagnose common problems. The command shouldn't be overused as it might cause the node to crash.

Logging:

All the cluster related information is logged in a riak console file. For detailed information on the actions performed, one can refer to this file.

6. Implementation

The implementation includes a task level breakdown of functional requirements such as:

1. Downloading and Setting up Riak on the system.
2. System tuning to adjust open files limit and performance tuning.

```
Lavanyas-MacBook-Pro:riak-2.2.3 lavanyakandukuri$ ulimit -n
65536
```

3. Adjusting configuration files (riak.conf) for desired ring size, node setup and backend requirements.
4. Creating and running a cluster.

Running a cluster on one host:

- Before basic cluster setup, configure the riak.conf file by adding node information. This file can be found in the installed riak version under /etc folder. Other properties like ring size, Erlang VM Tuning can also be done in this configuration file. The system should be tuned to increase Open file limit for current session.
- To verify whether riak has been installed properly, test by starting a node.

```
Lavanyas-MacBook-Pro:riak-2.2.3 lavanyakandukuri$ bin/riak start
Node is already running!
Lavanyas-MacBook-Pro:riak-2.2.3 lavanyakandukuri$ bin/riak-admin test
Successfully completed 1 read/write cycle to 'riak@127.0.0.1'

Lavanyas-MacBook-Pro:riak-2.2.3 lavanyakandukuri$ bin/riak-admin member-status
================================= Membership ==================================
Status     Ring    Pending    Node
-------------------------------------------------------------------------------
valid     100.0%      --      'riak@127.0.0.1'
-------------------------------------------------------------------------------
```

- Now start adding more nodes to the cluster. Before that, change the configuration(riak.conf) file based on the interface used to Riak.

  listener.protobuf.internal = 192.168.1.10:8087

nodename = riak@192.168.1.10

- To add the second node, use the cluster join command below:

```
Lavanyas-MacBook-Pro:riak-2.2.3 lavanyakandukuri$ bin/riak-admin cluster join riak@192.168.1.10
Success: staged join request for `riak@192.168.1.11` to `riak@192.168.1.10`
```

- Next check the cluster plan and if everything seems fine, use the cluster commit command.

```
Lavanyas-MacBook-Pro:riak-2.2.3 lavanyakandukuri$ bin/riak-admin cluster commit
Cluster changes committed
```

- For adding nodes on a single host, make copies of originally installed riak folder, set handoff port and protocol buffers to different values for each node and start the nodes with riak start command. Using cluster join command, join them plan and commit. Result of this in the planning phase is as follows:

```
================================= Membership ==================================
Status     Ring    Pending    Node
-------------------------------------------------------------------------------
valid    100.0%     20.3%    'riak@192.168.1.10'
valid      0.0%     20.3%    'riak@192.168.1.11'
valid      0.0%     20.3%    'riak@192.168.1.12'
valid      0.0%     20.3%    'riak@192.168.1.13'
valid      0.0%     18.8%    'riak@192.168.1.14'
-------------------------------------------------------------------------------
Valid:5 / Leaving:0 / Exiting:0 / Joining:0 / Down:0
```

- For checking ring members, use the command “bin/riak-admin status | grep ring_members”

Running a cluster on AWS:

- For creating a cluster on AWS, first select desired AWS region, then select the EC2 instance type, configure the security group, create a VPC and the key pair. VPC is created so that private IP addresses won’t change on restart.
- Security group settings: Click on Security Groups and choose the name of the Riak VM. Include the open ports 22(SSH), 8087 (Riak Protocol Buffers), 8098 (HTTP Interface). For the port ranges 4369, 6000-7999 and 8099, create a new Custom TCP rule.

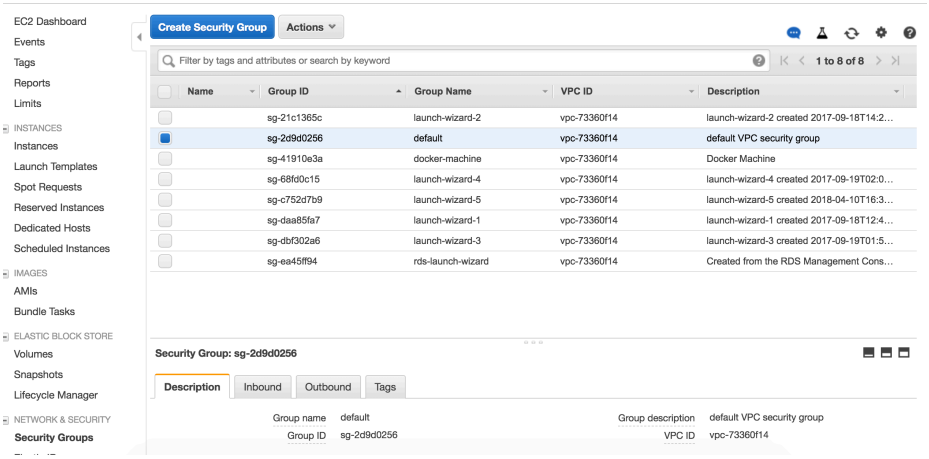

Fig 3. Security group configuration

- Launch the instances after these settings. At least 3 instances have to be launched to form a Riak cluster. 5 instances are launched in this implementation.

  The running instances (nodes) can be shown as follows:

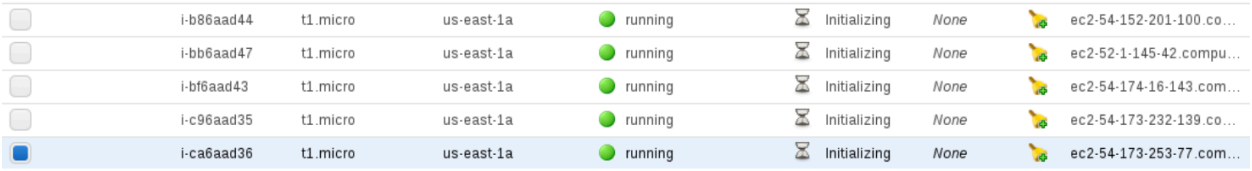

Fig.4 Running instances

- After connecting these instances using SSH, cluster is created using the commands below:

  $ curl http://169.254.169.254/latest/meta-data/local-ipv4

  $ sudo riak-admin cluster join riak@127.0.0.1

  $ sudo riak-admin cluster plan

  $ sudo riak-admin cluster commit

  $ sudo riak-admin member_status

These steps are similar to the ones shown for the creation of cluster on a single host. After the successful execution of above commands, a five-node cluster is up and running on AWS. Pictorial representation of a five-node cluster can be shown as follows:

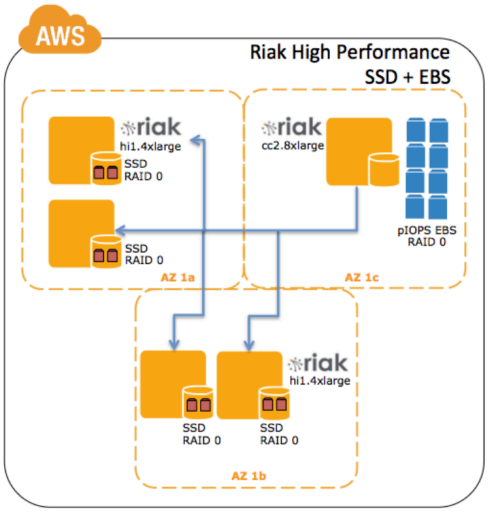

Fig. 4 Riak Cluster on AWS. Adapted from 'https://media.amazonwebservices.com/AWS_NoSQL_Riak.pdf'

7. Testing

7.1 General Approach

Logs are used to test the connectivity between nodes and to verify if they are functional in a cluster. Other types of system tests include usability testing, load testing, recovery testing and functional testing. Hardware and Software testing are performed to check if the system is working as expected by providing appropriate results. YCSB (Yahoo Cloud Serving Benchmark) is used for the performance evaluation of the overall PCAP system.

7.2 Unit Testing

Erlang's unit testing facility called Eunit is used for testing. QuickCheck is another testing model from Quviq which is used to test interactions between nodes and to test various protocols.

8. Appendix

http://basho.com/posts/technical/Why-Your-Riak-Cluster-Should-Have-At-Least-Five-Nodes/

https://github.com/basho/riak_test