Adopting Random Slicing as the Partitioning Algorithm in Riak Core Lite

Master's Thesis Kick-Off

Pascal Grosch

TU Kaiserslautern

July 1, 2020

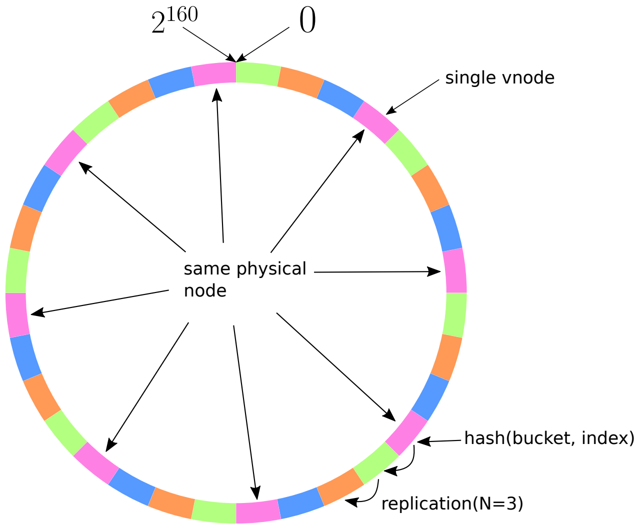
Introduction

- Riak is a distributed key-value-store
- Based on Amazon Dynamo
- Riak Core is a library for distributed platforms
- Riak Core Lite was forked to create a minimal up-to-date version

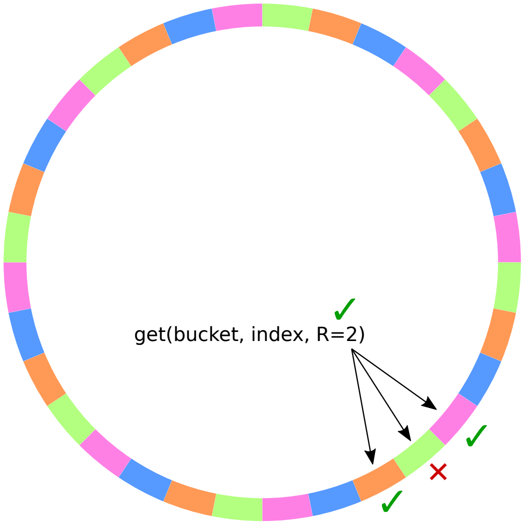
Riak Core's Consistent Hashing

- Based on Amazon Dynamo
- Visual representation as a partitioned ring
- Each partition virtual node
- Multiple vnodes per physical nodes
- bucket and index hashed with SHA-1 to partition index
- Entry replicated on N neighboring partitions
- Retrieval via hashing bucket and index and looking up N neighboring partitions
- Retrieval successful if at least R lookups are successful

Consistent Hashing - Visual Representation



Consistent Hashing - Retrieval



Constraints

As pointed out by Scott Lystig Fritchie[1]

- SHA-1 only feasible hash algorithm
- 160-Bit output sets range as 0 to $2^{160} - 1$
- Partition number is fixed at initialization
- Number of Partitions has to be power of 2
- Partition sized is fixed
- Claim assignment algorithm can lead to unbalanced workload
- No weighting of nodes with different capacities
- No "Justin Bieber's Twitter" handling
- Unchangeable replica placement policy

[1]https://www.infoq.com/articles/dynamo-riak-random-slicing/

Random Slicing

- Alternative randomized data-distribution strategy
- Partitions $[0, 1)$ range to buckets
- Hash function to real number in $[0, 1)$
- Multiple buckets can be handled by the same node

Random Slicing

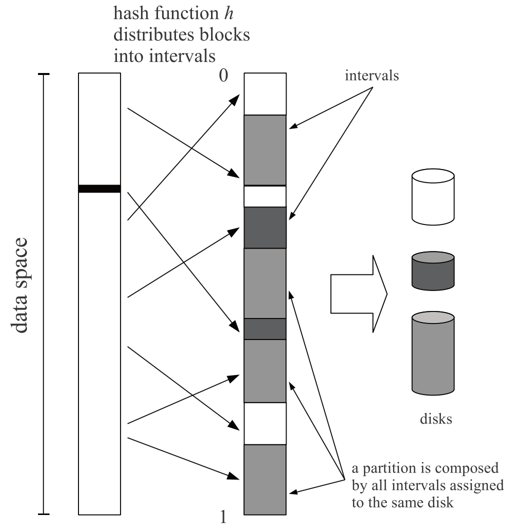
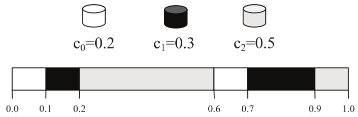


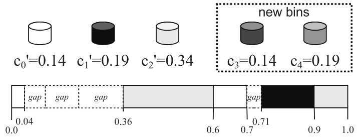
Figure: Miranda et al., "Random Slicing"

Changing Nodes

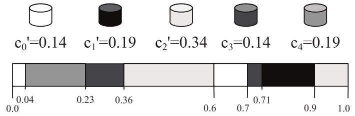
- On adding or removing nodes the new relative capacity of remaining nodes is computed
- Gaps are created by an algorithm and are assigned to new partitions
  - If necessary, existing partitions are split up and moved to gaps
  - Trade-off between computing new partition intervals and moving data



(a) Initial configuration



(b) Gap collection



(c) Final configuration

Figure: Miranda et al., "Random Slicing"

Performance Comparison

	Consistent Hashing (fixed)	Consistent Hashing (adapt.)	Random Slicing
Fairness	Poor ($\delta \uparrow$ with n)	Moderate ($\delta \approx 10\%$)	Good ($\delta \approx 0.4\%$)
Memory Usage	High ($\mu \approx 800$MB)	High ($\mu \approx 8$GB)	Low ($\mu \approx 4.5$MB)
Lookup Time	Moderate ($r \approx 50\mu s$)	High ($r \approx 98\mu s$)	Low ($r \approx 14\mu s$)
Adaptativity	Good ($\alpha \approx 7\%$)	Poor ($\alpha \approx 1172\%$)	Good ($\alpha \approx 1.63\%$)

Table: *Definitions used*: n, number of devices; δ, average deviation from ideal load; μ, worst-case memory consumption; From: Miranda et al., “Random Slicing”;

Adopting Random Slicing in Riak Core Lite

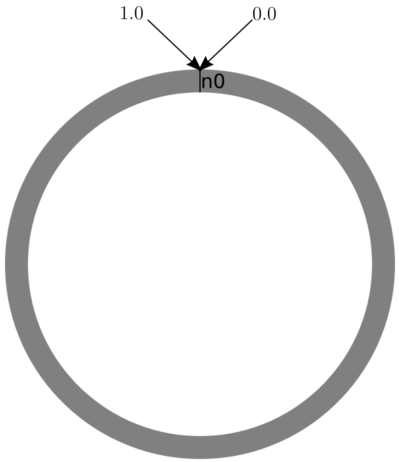
- Motivations to change the hashing algorithm:
  - less restrictions
  - better performance
  - better adaptability to different use cases
- Necessary to change the underlying model

Changing the Model

- Keeping the ring structure and adapting it may help using existing architecture
  - The ring can have arbitrary hash space and number of partitions
  - Partitions have dynamic size
- Replication strategy can differ by node and does not rely on neighbors
- Physical nodes can be weighted by their capacity

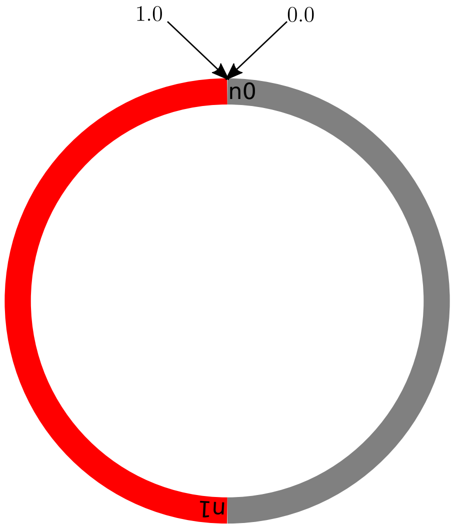
Visual Model - 1 Node

Presenting Figure 6 of Litchie's Work[2] as a ring.

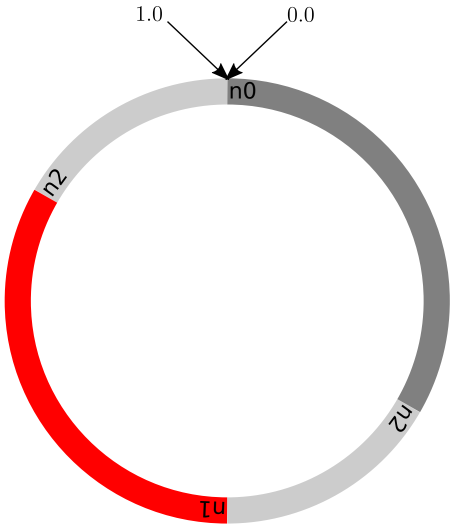


[2] Scott Lystig Fritchie. *A Critique of Resizable Hash Tables: Riak Core & Random Slicing*. URL: https://www.infoq.com/articles/dynamo-riak-random-slicing/.

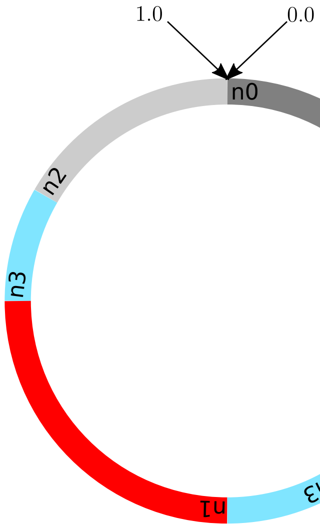
Visual Model - 2 Nodes



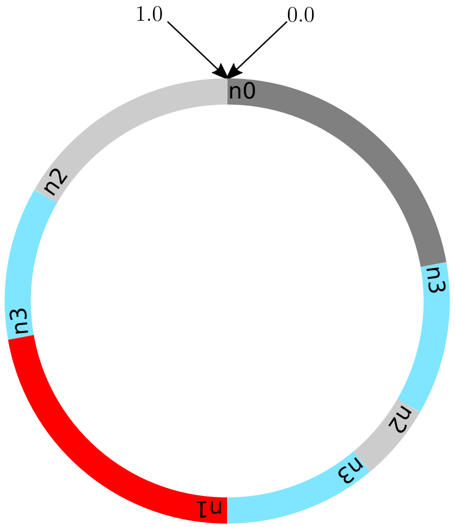
Visual Model - 3 Nodes



Visual Model - 4 Nodes



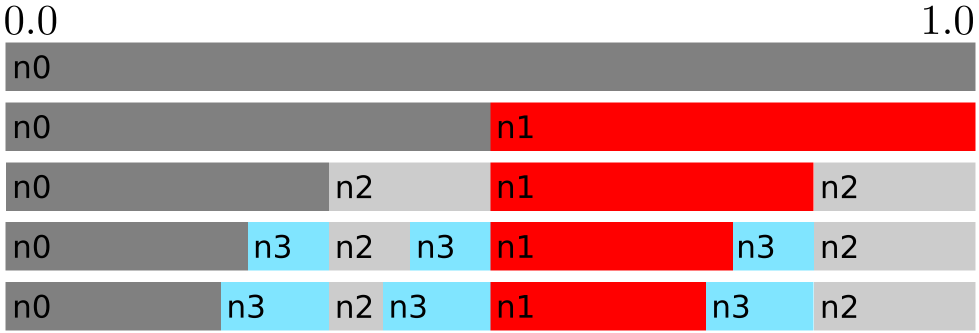
Visual Model - Adjusted Weight



Visual Model - Thoughts

- Visual model of the ring only useful if the replication strategy makes use of it
- Nodes are not evenly distributed
- Partitions are not of fixed size
- Depending on the implementation the ring is only used by name
- Simple representation as an interval as an alternative

Visual Model - Intervals



Open Problems

- What replication strategy to use
  - Cannot use the current strategy
  - Need to balance loads on nodes
  - Respect given weights
- Implementation: Keeping most of the existing architecture intact
  - The architecture might need major changes wherever it is driven by the evenly partitioned ring
- Implementation: High precision of partition bounds vs. memory usage
- Implementation: Which hash algorithm to use
- Showing the correctness of the partitioning and replication algorithm

Goals and Challenges of the Master's Thesis

- Replace Consistent Hashing with Random Slicing
  - Adapt existing implementation to Riak Core Lite
  - Define specifications of the partitioning algorithm to test and compare both variants
- Evaluate different replication strategies
  - Possibly enable setting strategy per node
  - Possibly allow nodes to be used for replication without owning a partition
- Evaluate performance improvement
- Evaluate impact of removing the ring model
- Not in scope: Actually removing the ring model