Adopting Random Slicing for Riak Core

Master's Thesis Presentation

Pascal Grosch

TU Kaiserslautern

February 17, 2021

Introduction

- cloud computing more and more prevalent
- scalability is a big quality factor
- distributing tasks to nodes influences scalability
- Riak Core provides solutions to distribute tasks to nodes

Riak Core (Lite)

- open source implementation of the Dynamo architecture
- framework for distributed systems
- generates preference lists for keys
- no actual replication mechanism
- uses variant of Consistent Hashing
- informs nodes of owner changes of keys
- Riak Core Lite as a streamlined version

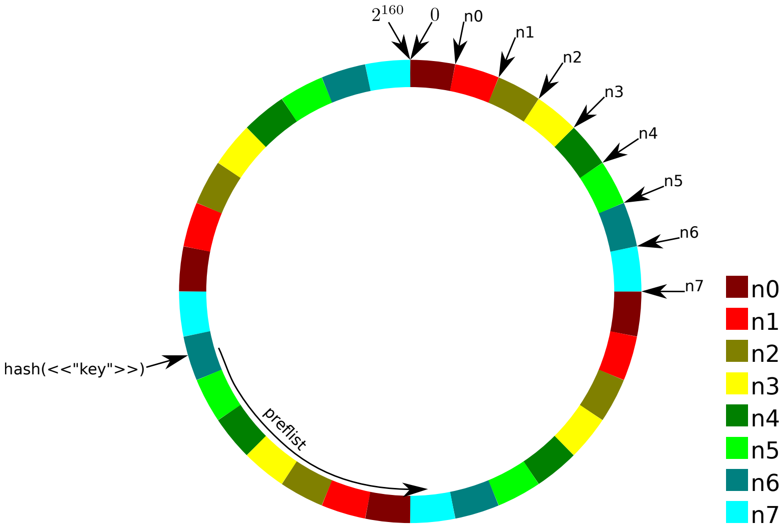
Goals

- analyze the influence of Consistent Hashing on the system structure
- replace Consistent Hashing with Random Slicing
- evaluate the performance differences

Consistent Hashing

- uses hashing to map keys to nodes
- hash space seen as ring
- nodes are hashed to ring
- keys are hashed to ring and mapped to the closest node
- many different implementations used in practice

Riak Core's Consistent Hashing



Changing the Cluster

- on adding or removing nodes partitions are reassigned
- claim algorithm responsible for load balancing and complete preference lists
- administrator can change the number of partitions on the ring

Constraints

As pointed out by Scott Lystig Fritchie[1]

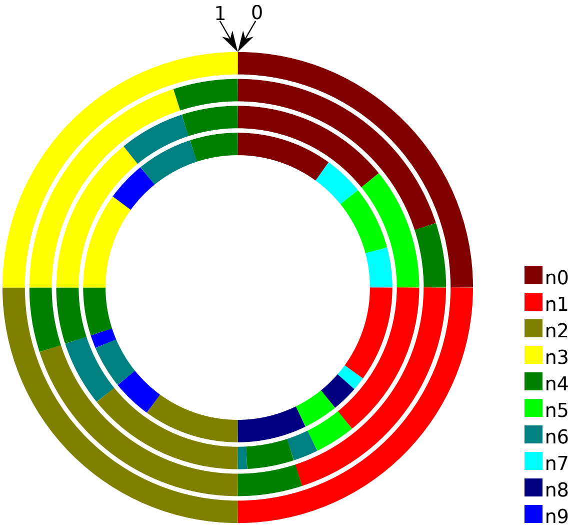
- partition number is fixed at initialization
- number of Partitions has to be power of 2
- partition size is fixed
- claim assignment algorithm can lead to unbalanced workload
- no weighting of nodes with different capacities

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

Random Slicing

- alternative randomized data-distribution strategy
- partitions $[0,1)$ range to sections
- nodes own parts of the ring according to their relative capacity
- hash function to real number in $[0,1)$
- multiple sections can be handled by the same node
- there is no fixed replication placement strategy

Changing the Cluster



Simple Replication Placement Strategies

- Random Replication
  - choose nodes randomly with their relative capacity as the probability
- Ring Rotation
  - rotate the ring counter-clockwise under the key-index by size of sections
- Ring Jumping
  - rotate the ring counter-clockwise under the key-index by size of the section the key points to
- all strategies require recomputation of replication placement after cluster changes

Adopting Random Slicing for Riak Core Lite

- system analysis shows the architecture relies on guarantees of Consistent Hashing
- recreating system architecture not in scope
- replacing Consistent Hashing as a prototype
- only basic functionality kept
- many optimizations lost
- robustness lost

Evaluation Setup

- rclref as in-memory key-value-store
- rcl_bench benchmarks operation throughput and latency
- comparing replication placement strategies and partitioning algorithms
- different cluster setups
- different workloads

Parameter	Values
Riak Core Lite Configuration	ConsistentHashing, RandomSlicing_Jumping, RandomSlicing_Random, RandomSlicing_Rotation
Cluster Configuration	3 nodes, 4 nodes, 7 nodes, dynamic
Workload	read_heavy, write_heavy

Hypotheses

- H1 Random Replication is the best replication placement strategy regarding load balancing.
- H2 Random Replication is the best replication placement strategy regarding throughput.
- H3 The divergence from optimal load balancing is only marginally higher with Random Slicing than with Consistent Hashing.
- H4 The throughput in a static cluster is only marginally smaller with Random Slicing than with Consistent Hashing.
- H5 With Random Slicing handoff operations are faster than with Consistent Hashing.

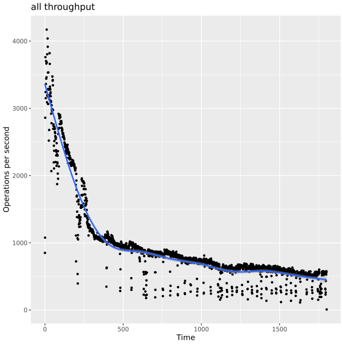
Results - Hypothesis 1

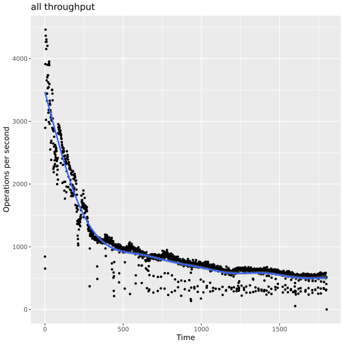
Configuration	Load Divergence Keys	Load Diversion Puts	Load Diversion Gets
Overall_RandomSlicing_Jumping	3.03%	3.66%	3.66%
Overall_RandomSlicing_Random	2.29%	2.70%	2.71%
Overall_RandomSlicing_Rotation	4.09%	3.58%	3.54%

- Random Replication has lowest load divergence in all configurations
- data supports H1

Results - Hypothesis 2

- Ring Rotation clearly has the worst throughput
- Random Replication and Ring Jumping show similiar throughput behavior in all configurations
- data neither supports nor contradicts H2
- may be a matter of scalability





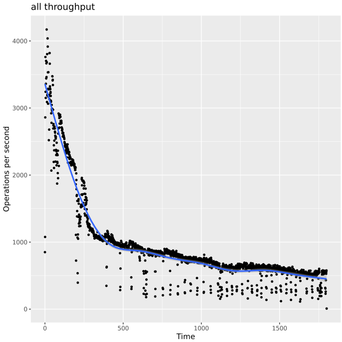
Results - Hypothesis 3

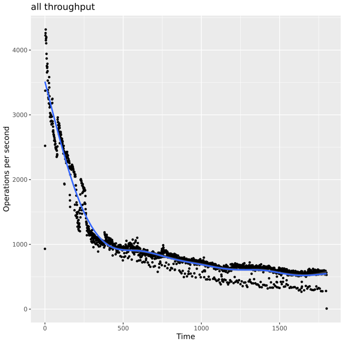
Configuration	Load Divergence Keys	Load Diversion Puts	Load Diversion Gets
Overall_RandomSlicing_Random	2.29%	2.70%	2.71%
Overall_ConsistentHashing	0.38%	3.24%	3.24%
ConsistentHashing_C2_read_heavy	0.39%	0.39%	0.39%
RandomSlicing_Random_C2_read_heavy	0.16%	0.05%	0.07%
ConsistentHashing_C2_write_heavy	0.38%	0.38%	0.40%
RandomSlicing_Random_C2_write_heavy	0.05%	0.05%	0.04%

- Overall load divergence extremely worse with Random Slicing
- data strongly contradicts H3
- for 7 nodes the load divergence with Random Slicing is less than the one with Consistent Hashing
- may be a matter of scalability

Results - Hypothesis 4

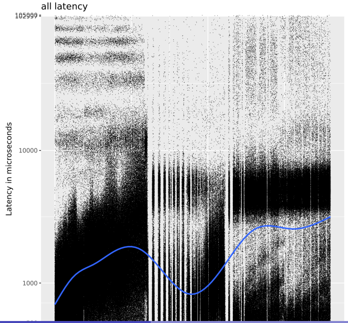
- no significant differences in throughput in any configuration
- H4 can be seen as plausible

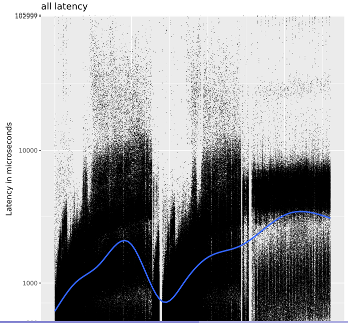




Results - Hypothesis 5

- shorter disruptions with Random Slicing than with Consistent Hashing for read-heavy workload
- larger and more disruptions with Random Slicing than with Consistent Hashing for write-heavy workload
- inconclusive results
- additional and more refined benchmarks necessary
- may be caused by cut out optimizations and edge cases





Future Work

- improve the gap collection algorithm
- Reintegrate handoff optimizations
- develop a more refined replica placement strategy
- support heterogeneous nodes
- evaluate higher scalability

Conclusion

- choice of partitioning algorithm is an architectural decision
- enough optimization options for Random Slicing to be promising