A TECHNICAL OVERVIEW OF RIAK KV ENTERPRISE

INTRODUCTION

Basho Riak® KV Enterprise is a highly resilient NoSQL database. It ensures your most critical data is always available and that your Big Data applications can scale. Riak KV can be operationalized at lower costs than both relational and other NoSQL databases, especially at scale. Running on commodity hardware, Riak KV is operationally easy to use with the ability to add and remove capacity on demand without data sharding or manually restructuring your cluster.

Since Riak KV was built to address the problem of data availability with ease of scale, it is a good fit whenever downtime is unacceptable and scalability is critical. Riak KV is designed to survive network partition and hardware failures. Many leading companies, from large enterprises to small startups, are using Riak KV for mission-critical applications in a variety of use cases and verticals.

Riak KV stores data as a combination of keys and values, and is a fundamentally content-agnostic database. You can use it to store anything you want – JSON, XML, HTML, documents, binaries, images, and more. Keys are simply binary values used to uniquely identify a value.

Riak KV integrates with Apache Spark, Redis Caching, Apache Solr, and Apache Mesos to reduce the complexity of integrating and deploying other Big Data technologies.

EXAMPLE USE CASES

MANAGING USER AND SESSION DATA

PERSONALIZATION

SHOPPING CART AND WALLET DATA

CHAT AND MESSAGING

COMMUNICATIONS

MOBILE PLATFORMS

OPERATIONAL ANALYTICS

LOG STORAGE

MULTI-SITE BUSINESS CONTINUITY

FRAUD DETECTION

DOCUMENT STORE

RIAK KV ARCHITECTURE

MASTERLESS

At its core, Riak KV is a key/value database, built from the ground up to safely distribute data across a cluster of servers, called nodes.

A Riak KV cluster is a group of nodes that are in constant communication to ensure data availability and partition tolerance.

Riak KV has a masterless architecture in which every node in a cluster is capable of serving read and write requests. All nodes are homogeneous with no single master or point of failure. Any node selected can serve an incoming request, regardless of data locality, providing data availability even when hardware, or the network itself, is experiencing failure conditions.

THE RIAK RING

The basis of Riak KV's masterless architecture, replication, and fault tolerance is the Ring. This Ring is a managed collection of partitions that share a common hash space. The hash space is called a Ring because the last value in the hash space is thought of as being adjacent to the first value in the space. Replicas of data are stored in the "next N partitions" of the hash space, following the partition to which the key hashes.

The Ring is also used as shorthand for the "Ring State." The Ring State is a data structure that gets communicated and stays in sync between all the nodes, so each node knows the state of the entire cluster. If a node gets a request for an object managed by another node, it consults the Ring State and forwards the request to the proper nodes, effectively proxying the request as the coordinating node. If a node is taken offline permanently or a new server is added, the other nodes adjust, balancing the partitions around the cluster, then updating the Ring State. You can read more about nodes, vnodes, clusters, the Ring, and Ring State here.

NODES AND VNODES

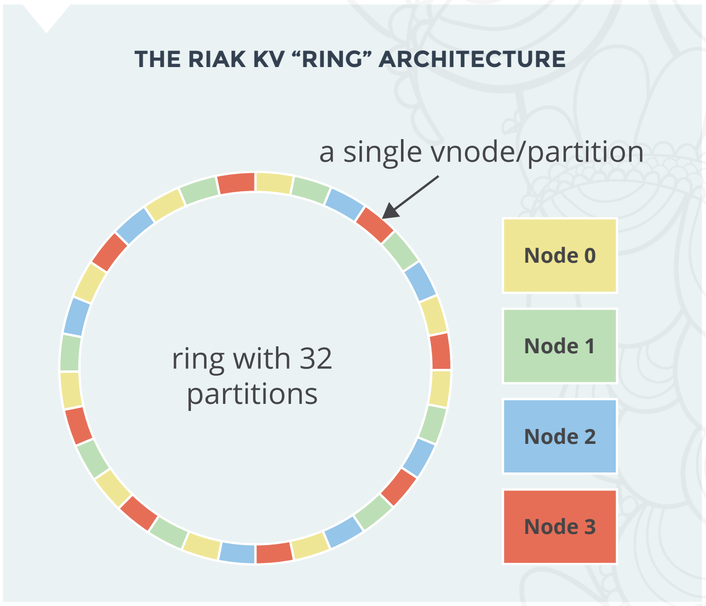
Each node in a Riak KV cluster manages one or many virtual nodes, or vnodes. Each vnode is a separate process which is assigned a partition of the Ring, and is responsible for a number of operations in a Riak cluster, from the storing of objects, to handling of incoming read/write requests from other vnodes, to interpreting causal context metadata for objects.

This uniformity of Riak KV vnode responsibility provides the basis for Riak KV's fault tolerance and scalability. If your cluster has 64 partitions and you are running three nodes, two of your nodes will have 21 vnodes, while the third node holds 22 vnodes.

The concept of vnodes is important as we look at data replication. No single vnode is responsible for more than one replica of an object. Each object belongs to a primary vnode, and is then replicated to neighboring vnodes located on separate nodes in the cluster.

INTELLIGENT REPLICATION

Riak KV automatically distributes data across nodes in a Riak KV cluster and yields a near-linear performance increase as you add capacity. Data is distributed evenly across nodes using consistent hashing. Consistent hashing is a special kind of hashing such that when a hash table is resized and consistent hashing is used, only K/n keys need to be remapped on average, where K is the number of keys, and n is the number of slots.



In contrast, in most traditional hash tables, a change in the number of array slots causes nearly all keys to be remapped.

When you add (or remove) machines, data is rebalanced automatically in a non-blocking operation. Riak KV will continue servicing read and write requests, allowing the ability to perform scaling operations without the need for downtime.

Any new machines that are added to a cluster claim a portion of the overall dataset, which is then redistributed, with the resulting cluster status shared across all vnodes. Consistent hashing and vnodes ensure horizontal scale across N servers.

Replication ensures that there are multiple copies of data across multiple servers. By default, Riak KV makes 3 replicas, each belonging to a unique vnode. The primary benefits derived from replication are high availability and low latency. If a server in the cluster becomes unavailable, other servers that contain the replicated data remain available to serve requests and the whole system remains highly available.

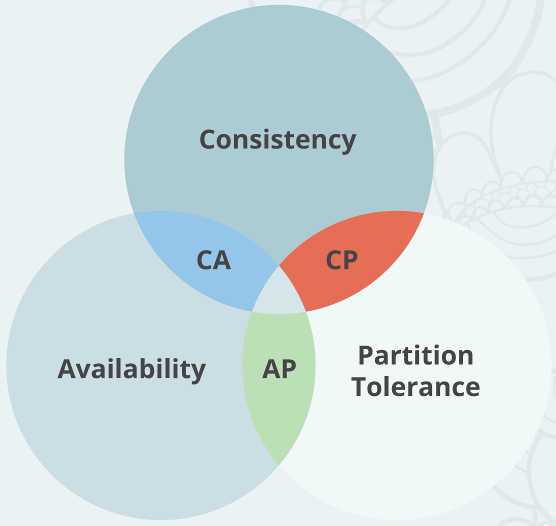
To further understand Riak KV's intelligent replication, it is helpful to understand how data is distributed across the cluster. Riak KV chooses one vnode to exclusively host a range of keys, and the other vnodes host the remaining non-overlapping ranges. With partitioning, the total capacity can increase by simply adding commodity servers.

Since replication improves availability and partitions allow us to increase capacity, Riak KV combines both partitions and replication to work together. Data is partitioned as well as replicated across multiple nodes to create a horizontally scalable system.

EVENTUAL/RELAXED CONSISTENCY

One of the important differences and key advantages of some NoSQL systems is the concept of relaxed consistency. Relaxed consistency, also known as eventual consistency, means that not all of the assigned nodes for a transaction in a distributed system need to have that transaction confirmed before the distributed system considers that transaction to be complete. This allows for a higher degree of workload concurrency and data availability.

The CAP theorem, also known as Brewer's theorem, defines a natural tension and trade-offs between three core operational capabilities in distributed systems and database infrastructure **— Consistency, Availability, and Partition Tolerance.**



Riak KV is a tunable AP system. By default, Riak KV replicas are "eventually consistent," meaning that while data is always available, not all replicas may have the most recent update at the exact same time, causing brief periods—generally on the order of milliseconds—of inconsistency while all state changes are synchronized. Riak KV is designed to deliver maximum data availability, so as long as your client can reach one Riak KV server, it can write data.

HANDLING VERSION CONFLICTS AND RECOVERING DATA CONSISTENCY

In any system that replicates data, inconsistencies can arise. For example, when two clients update the same object at the exact same time, or when not all updates have yet reached hardware that is experiencing heavy load or network delay.

When you make a read request, Riak KV looks up all replicas for that object. By default, Riak KV addresses any inconsistencies by returning the most recently updated version, determined by looking at the object's Dotted Version Vector, or DVV. DVVs are metadata attached to each data replica when it is created. They are extended each time a data replica is updated in order to keep track of data versions.

There are two processes which continually check and repair any divergent replica sets, which are the results of replication failure (e.g. disk failure, data corruption, etc.). With read repair, Riak KV will automatically update the out-of-sync replica to make it consistent during a read operation. A more advanced capability, called Active Anti Entropy, addresses colder data in the cluster. AAE is a background process which continually compares merkle trees across replica sets to determine discrepancies. Both are key to preventing the need for manual operator intervention under failure scenarios.

NODE OR NETWORK FAILURE RECOVERY

Due to replication and eventual consistency, it then becomes important to understand the concept of quorums, which define the success or failures of reads and writes. A quorum is defined by the number of replicas that you define. By default, a quorum is half of all the total replicated nodes plus one more (a majority).

You can set a value to give you control over how many replicas must respond to a read or write request for the request to succeed ("R value" for reads and a "W value" for writes). If this value is not specified, Riak KV defaults to requiring a quorum, where the majority of nodes must respond.

A "strict quorum" is where a request would fail (and deliver an error message to the client) if the required number of primary nodes cannot accept requests. Using the strict quorum would work for most requests. However, at any moment, a node could go down or the network could partition, triggering the unavailability of the required number of nodes. Therefore, to provide high availability, Riak KV defaults to what is known as a sloppy quorum, meaning that if any primary (expected) node is unavailable, the next available node in the cluster will accept requests. That node will then update the primary node when it comes back online. This ability to easily handle node failures is known as a hinted handoff.

Hinted handoff ensures that if a node goes down, a neighboring node will take over its storage operations. When the failed node returns, the updates received by the neighboring node are handed back to it. This ensures that availability for writes and updates is maintained automatically, minimizing the operational burden of failure conditions.

RIAK KV DEVELOPMENT CONCEPTS

READING AND WRITING DATA

Riak KV stores data as a combination of keys and values in buckets. Keys are simply binary values used to uniquely identify a value. A value is the data that is associated with a key and stored. Values can be numbers, strings, binaries, or almost any data. Buckets are used to define a virtual namespace for storing Riak KV "objects." In Riak KV, an "object" is a nickname for the combination of a bucket, key, and value.

Key/Value stores are conceptually like hash tables, where values are stored and accessed by an immutable key. Riak KV functions like a very large hash space, also known as a hash table, a map, a dictionary, or an object.

Most of the operations you'll perform with Riak KV will be setting or retrieving the value of a key (reading and writing key/value pairs). Other operations, such as advanced (SOLR) querying and bucket configuration, can be done via the Riak KV HTTP API or via the Riak KV Protocol Buffers Client API.

BUCKETS AND BUCKET TYPES

Buckets in Riak KV provide logical namespaces so that identical keys in different buckets will not conflict. Buckets allow for cleaner key naming, and have other benefits, such as custom properties. Since all keys must belong to a bucket, there is no global namespace. A unique key in Riak KV is defined by bucket/key.

There is also a level of classification that exists above buckets, called "bucket types." Bucket types are groups of buckets with a similar set of properties. The benefit of bucket types is that a group of distinct buckets can share properties. This has practical implications such that with bucket types, and the communication mechanism that accompany them, there's no limit to your bucket count. Since every bucket of a type inherits the common properties, you can make across-the-board changes trivially, making managing multiple buckets easier.

Additionally, Riak Spark Connector (described below) has a special query type, called "full bucket read," which allows bulk retrieval of the whole bucket, without providing the keys — unlike other queries.

RIAK KV SEARCH AND SECONDARY INDEXING

Riak KV exposes several other functionalities for searching and accessing data including MapReduce, full-text search, and secondary indexing.

Riak KV provides Riak KV Search, a full-text search engine that indexes objects on write and provides an easy, robust query language plus integration with Apache Solr. Riak KV Search is ideal for indexing content like posts, user bios, articles, and other documents, as well as indexing JSON data. For more information on Riak KV Search, see the documentation on Riak KV Search.

Secondary indexing allows you to tag objects in Riak KV with one or more queryable values. These "tags" can then be queried by exact or range value for integers and strings. Secondary indexing is great for simple tagging and for searching Riak KV objects for additional attributes. Find out more about Secondary Indexing here.

RIAK DISTRIBUTED DATA TYPES

Riak Distributed Data Types provide the developer a rich set of data types to build distributed applications. These include: Flags, Registers, Counter, Sets, Maps, and HyperLogLog. These data types enable you to perform operations, such as counting the number of users, or whether a user has signed up for a specific pricing plan. Counters, sets, and maps can be used as bucket-level data types or types that you interact with directly. Maps are the most versatile of the Riak Data Types, because all other data types can be embedded within them (including maps themselves). This enables the creation of complex, custom data types from a few basic building blocks. Using counters, sets, and maps within maps is similar to working with those types at the bucket level. Using these data types, Riak automatically handles any read conflict resolution on the server side so developers can simplify their client-side coding.

Although Riak Distributed Data Types function differently from other Riak KV objects in some respects, when you're using Search you can think of them as normal Riak KV objects with special metadata attached (metadata that you don't need to worry about as a developer). Riak Data Types have a deep integration with Riak Search. Riak's counters, sets, and maps can be indexed and have their contents searched just like other Riak objects without having to create custom schemas.

Riak KV currently implements the following Data Types:

- **Flags** — Flags behave much like Boolean values, except that instead of true/false flags, the values are enable/disable.
- **Registers** — Registers are essentially named binaries (like strings). Any binary value can act as the value of a register.
- **Counters** — Counters can only be a positive or negative integer, or zero.
- **Sets** — Sets are collections of unique binary values, such as strings.
- **Maps** — Maps are the richest of the Riak KV Data Types, because, within the fields of a map, you can nest any of the five Data Types, including maps themselves.
- **HyperLogLog** — HyperLogLog is used to approximately count unique elements within a data set or stream, for example counting unique IP addresses or User IDs.

“ *The new advanced Data Types are a game changer for us, because it makes it simple for us to manage our data model at a scale that supports over a billion devices all around the world.* ”

– Weston Jossey, Head of Operations, Tapjoy

RIAK KV SUPPORTED PROGRAMMING LANGUAGES

There are a diverse group of client libraries for Riak KV that support both the HTTP API and Protocol Buffer APIs.

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – like XML, but smaller, faster, and simpler. Riak KV Enterprise includes a complete set of development tools to help you get productive more quickly.

- **Basho Supported Libraries** — Java, Ruby, Python, PHP, Erlang, .NET, Node.js
- **Community Libraries** — C, Clojure, Go, Perl, Scala, R

SPARK CONNECTOR

Modern Big Data applications need to process data in real time to reveal patterns, trends, and associations. The Spark Connector efficiently retrieves data from Riak KV to Spark for in-memory distributed processing, then the results can be stored back in Riak KV as needed. The ability to persist these results to Riak KV provides flexibility for future data processing or analysis.

The Spark Connector leverages secondary indexes to perform the queries while partitioning the results across a number of Spark Worker nodes for fast and reliable distributed in-memory processing. With Riak KV, Spark cluster deployments are as simple as specifying where the code should be deployed. Both static information (configuration) and dynamic information (port numbers, etc.) are managed at the time of installation.

RIAK KV ENTERPRISE

Riak Enterprise is a commercially distributed product built on Riak (Apache 2.0-licensed) that extends Riak's capabilities with Multi-cluster Replication, SNMP monitoring, JMX-Monitoring, and 24x7 support.

Multi-cluster Replication

Riak KV Enterprise offers Multi-cluster Replication features so that data stored in Riak KV can be replicated for backup-clusters, analysis clusters, or for multiple datacenters. With Riak KV Enterprise, data can be replicated across the datacenter or across geographic areas, providing disaster recovery, data locality, compliance with regulatory requirements, the ability to "burst" peak loads into public cloud infrastructure, and more.

In Multi-cluster Replication, one cluster acts as a primary, or source, cluster. The primary cluster handles replication requests from one or more secondary clusters (often located in datacenters in other regions or countries). If the datacenter with the primary cluster goes down, a secondary cluster can take over as the primary cluster. In this sense, Riak's multi-cluster capabilities are "masterless."

There are a number of use cases where Multi-cluster Replication is essential to fulfilling the business requirements of your data store including: Primary Cluster with Failover, Active - Active Cluster, Availability Zones, Secondary Analytics Clusters, and Public Cloud use cases. For more detailed discussion of the various Multi-cluster Replication use cases, take a look at the Multi-cluster Replication Technical Overview.

NEXT STEPS

To get started with Riak KV, you can download the open source at http://basho.com/download.

If you are evaluating Riak KV Enterprise, please contact us. We would be happy to arrange a tech talk with your team, or answer any questions about our product and how customers are using it in production to meet their business goals. If you want to try Riak KV Enterprise, we can provide a free developer trial that you can set up on your own hardware and evaluate on your own time. Finally, our Professional Services Team can assist you in planning, setting up, and optimizing your multi-datacenter strategy.

ABOUT BASHO TECHNOLOGIES

Basho, the creator of the world's most resilient databases, is dedicated to developing disruptive technology that simplifies enterprises' most critical distributed systems data management challenges. Basho has attracted one of the most talented groups of engineers and technical experts ever assembled devoted exclusively to solving some of the most complex distributed systems challenges presented by Big Data and IoT.

Basho's database, Riak® KV, the industry leading distributed NoSQL database, is used by fast growing Web businesses and by one-third of the Fortune 50 to power their critical Web, mobile and social applications. Built on the same foundation, Basho introduced Riak TS, which is the first enterprise-ready NoSQL database specifically optimized to store, query and analyze time series data. Basho also provides Riak integrations for a variety of Big Data technologies like Apache Spark, Redis, Mesos, and Apache Solr.

For more information visit Basho.com which is full of interesting use cases, customer case studies and product detail, or docs.basho.com for technical documentation.