WHITEPAPER

RELATIONAL TO RIAK

INTRODUCTION

This whitepaper will provide you with both background and detailed information on relational databases versus Riak. Riak is a distributed NoSQL database that helps you store, manage, and secure unstructured data in a fault-tolerant, highly available platform. Riak can be operationalized at lower costs than traditional relational databases and is easy for you to manage at scale. It scales up and out enabling you to add commodity hardware, as needed, for additional capacity. In this brief, we will look at why companies choose Riak over a relational database. We focus specifically on availability, scalability, and the key/value data model. Then we analyze the decision points that should be considered when choosing a non-relational solution and review data modeling, querying, and consistency guarantees. Finally, we end with simple patterns for building common applications in Riak using its key/value design, dealing with data conflicts that emerge in an eventually consistent system, and discuss multi-datacenter replication.

WHY MIGRATE TO RIAK?

HIGH AVAILABILITY

Strongly consistent operations provide applications with guarantees that read operations will reflect the last successful update to your database, and are an important part of relational database systems enabling operations, like transactions, that are essential for some types of applications. Strong consistency is relatively straightforward when there is one large single relational database server. Once your dataset grows beyond the capacity of a single machine, it becomes necessary to scale the database and operate in a distributed fashion. Relational databases typically address the challenge of availability with a master/replica architecture, where the topology of a cluster is comprised of a single master and multiple replicas. Under this configuration, the master is responsible for accepting all write operations and coordinating with replicas to apply the updates in a consistent manner. Read requests can either be proxied through the master or sent directly to the replicas.

But what happens when your relational master server fails? The database will favor consistency and reject write operations until the failure is resolved. This can lead to a window of write unavailability, which is unacceptable in many application designs. Most master/replica architectures recognize that a master is a single point of failure and will perform automatic master failover, where a replica will be elected as a new master when failure of the master is detected.

For many of today's applications and platforms, high availability is the most important requirement. Downtime can cost you millions.

In contrast, Riak is a masterless system designed for high availability, even in the event of hardware failures or network partitions. Any server (termed a "node" in Riak) can serve any incoming request, and all data is replicated across multiple nodes. If a node experiences an outage, other nodes will continue to service read and write requests. Further, if a node becomes unavailable to the rest of the cluster, a neighboring node will take over the responsibilities of the missing node. The neighboring node will pass new or updated data (termed "objects") back to the original node once it rejoins the cluster. This process is called "hinted handoff" and ensures that read and write availability is maintained automatically to minimize your operational burden when nodes fail or comeback on-line.

For many of today's applications and platforms, high availability has a direct impact on revenue. A few examples include: cloud services, online retail, shopping carts, gaming and betting, and advertising. Further, lack of availability can damage user trust and result in a poor user experience for many social media and chat applications, websites, and mobile applications. Riak provides the high availability needed for your critical applications. It provides a better solution than relational databases when availability is a key requirement.

High availability is also closely linked to scalability.

MINIMIZING THE COST OF SCALE

Relational databases scale-out by increasing the server and storage capacity. Special, expensive versions of the database may technically distribute load across multiple machines, but they rely on shared storage that often becomes a bottleneck. These systems are not designed to run on commodity hardware. An Oracle RAC system can cost millions to store a mere 20TB of data, a tiny number in the realm of Big Data where petabytes, exabytes, and zettabytes are the common units of measure. There are current Riak customers who ingest over 20TB of data a day.

Another way databases scale is to use sharding. Sharding distributes data across several database servers. A common example of this would be putting your user data for differing geographical regions (e.g., US and EU) on different machines, or using an alphabetical or numerical order to split data. While this seems simple, sharding is complex and inherently inflexible.

First, writing and maintaining custom sharding logic increases the overhead of operating and developing an application on the database. Significant growth of data, or traffic, typically means significant, often manual, re-sharding projects. Determining how to intelligently split the dataset without negatively impacting performance, operations, and development presents a substantial challenge — especially when dealing with "big data," rapid scale, or peak loads. Rapidly growing applications frequently outpace an existing sharding scheme and, when the data in a shard grows too large, the shard must again be split. While several "auto"-sharding technologies have emerged in recent years, these methods are often imprecise and still require manual intervention. Sharding can often lead to "hot spots" in the database where a few physical machines become responsible for storing and serving a disproportionately high amount of both data and requests. Hot spots can lead to unpredictable latency and degraded performance.

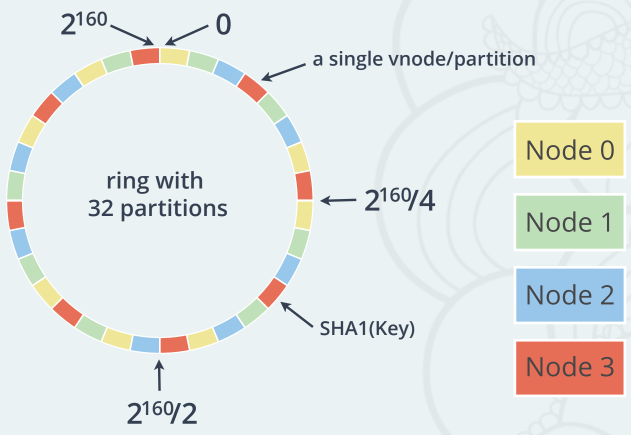
In Riak, data is automatically distributed evenly across nodes using consistent hashing. Consistent hashing ensures data is evenly distributed around the cluster and new nodes can be added with automatic, minimal reshuffling of data. This significantly decreases risky hot spots in the database and lowers the operational burden of scaling.

Scaling a relational database to handle more data and usage can be prohibitively expensive. Riak scales near linearly using commodity hardware.

Riak stores data using a simple key/value model. Data entries in Riak are referred to as "objects." Key/value pairs are logically grouped together in namespaces called "buckets." As you write new keys to Riak, the object's bucket/key pair is hashed. The resulting value maps onto a 160-bit integer space. This integer space can be conceptualized as a ring that is used to determine where data is placed on physical machines.

Riak tokenizes the total key space into a fixed number of equally sized partitions (default is 64). Each partition owns the given range of values on the ring and is responsible for all buckets and keys that, when hashed, fall into that range. A virtual node (a "vnode") is the process that manages each of these partitions. Physical machines evenly divide responsibility for vnodes.

THE RIAK "RING" ARCHITECTURE



Hashing and shared responsibility for keys across nodes ensures that data in Riak is evenly distributed. When machines are added, data is rebalanced automatically with no downtime. New machines take responsibility for their share of data by assuming ownership of some of the partitions; existing cluster members hand off the relevant partitions and the associated data. The new node continues claiming partitions until data ownership is equal. The ring state is shared around the cluster by means of a "gossip protocol." Whenever a node changes its claim on the ring, it announces (i.e., "gossips") this change to other nodes so that those nodes can respond appropriately. Nodes also periodically re-announce what they know in case any nodes happened to miss previous updates.

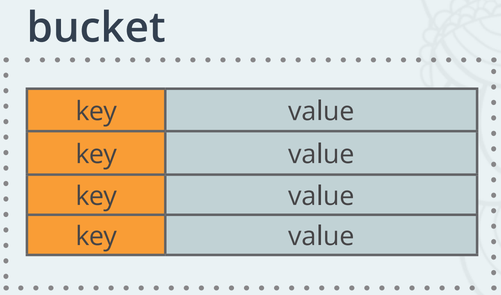
> **The relational data model is complex and inflexible for storing massive amounts of unstructured data.**

Riak is easier to manage, doesn't require sharding, and runs on commodity hardware. This provides you with significantly reduced costs of scale over traditional relational databases.

SIMPLE DATA MODELS

With the tremendous growth in data — especially big data and new types of gaming, social, and mobile applications — there is a growing need to store unstructured data (data that does not fit easily in the rigid data model of relational systems). This unstructured data is more effectively stored in a simple and powerful key/value store. Riak stores all types of objects. The objects are stored on disk as binaries. As a result, developing code that interacts with this simple, straightforward design can be accomplished quickly and easily. As your application changes and you add new features there is no need to make complex schema changes. This is especially ideal for applications where rapid iterations are required. A key/value platform provides more flexibility and simplicity than a traditional relational database. Riak provides multi-model support, including key/value, search, and object storage, ensuring you can easily store and retrieve documents, images, videos, and much more.

RIAK KEY/VALUE PAIRS STORED IN A BUCKET



With Riak there are no join operations since the key/value model has no concept of columns and rows. Riak is queried via HTTP requests, via the protocol buffers API, or through various client libraries. There are several powerful querying options:

- **Riak Search:** Integration with Apache Solr provides full-text search and support for Solr's client query APIs.
- **Indexes:** Secondary Indexes (2i) give developers the ability to tag an object stored in Riak with one or more query values. Indexes can be either integers or strings and can be queried by either exact matches or ranges of values.
- **MapReduce:** Developers can leverage Riak MapReduce for tasks like filtering documents by tag, counting words in documents, and extracting links to related data. It offers support for JavaScript and Erlang.

OPERATIONAL & DEVELOPMENT CONSIDERATIONS

POWERFUL DATA MODELING IN RIAK

The table below illustrates key/value mappings for common application types. Values in Riak are stored on disk as binaries — JSON or XML documents, images, text, etc.

Application Type	Key	Value
Session	User/Session ID	Session Data
Advertising	Campaign ID	Ad Content
Logs	Date	Log File
Sensor	Date, Date/Time	Sensor Updates
User Data	Login, eMail, UUID	User Attributes
Content	Title, Integer	Text, JSON/XML/HTML Document, Images, etc.

Consider, for example, one of Riak's common use cases, storing user and session data. In a relational database, the "users' table is well know and, basically, provides a unique identifier per user. It then provides a series of identifying information about that user as individual columns such as:

- First name
- Last name
- Interests
- Counter of Site Visits
- Paid Account Identifier

This data can then be used to correlate or count paid users, common interests, etc. via a series of SQL queries against the row/column structure of the users table.

Riak, in contrast, provides flexibility in how this data can be modeled based upon your application requirements. You can create a Users bucket with the UserName (or Unique Identifier) as the key and a JSON object storing all user attributes as the value. You can also leverage the power of Riak Data Types by creating a map data type for each user, storing:

- User's first and last name strings in registers
- User Interests as a set
- User visits in a counter
- A paid account identifier in a flag

Riak, in contrast, provides flexibility in how this data can be modeled based upon your application requirements.

One of the best ways to enable application interaction with objects (a key/value pair) in Riak is to provide structured bucket and key names for the objects. This approach often involves wrapping information about the object in the object's location data itself.

For example, appending a timestamp, UUID, or Geographical coordinate to a key's name allows for fine-grained queryability via simple lookup to locate and retrieve a specific set of information. Leveraging the same naming mechanism as created for users (UniqueID as the key) enables, in a separate session's bucket, storing the UUID append with a timestamp as the key and the session data (in binary format) as the object. In this way, using the same UUID, it is possible to obtain both user and session data stored in different buckets and in different formats.

RESOLVING DATA CONFLICTS

In any system that replicates data, conflicts can arise (e.g., if two clients update the same object at the exact same time or if not all updates have yet reached hardware that is experiencing lag). As discussed earlier, Riak is "eventually consistent" — while data is always available, not all replicas may have the most recent update at the same time, causing brief periods (generally on the order of milliseconds) of inconsistency while all state changes are synchronized.

However, Riak provides features to detect and help resolve the statistically small number of incidents when data conflicts occur. When a read request is performed, Riak looks up all replicas for that object. By default, Riak will return the most updated version. You can also resolve conflicts using Riak Data Types to simplify the burden of producing data convergence at the application level by absorbing the complexity into Riak itself.

Further, when an outdated object is discovered as part of a read request, Riak will automatically update the out-of-sync replica to make it consistent. Read Repair, a self-healing property of the database, will even update a replica that returns a "not-found" in the event that a node loses it due to physical failure.

Riak also includes "Active Anti-Entropy," which is an automatic self-healing property that runs in the background. Rather than waiting for a read request to trigger a replica repair (as with Read Repair), Active Anti-Entropy constantly uses a hash tree exchange to compare replicas of objects and automatically repairs or updates any that are divergent, missing, or corrupt. This can be beneficial for large clusters storing "stale" data.

MULTI-DATACENTER OPERATIONS

Multi-site replication is critical for many of today's platforms and applications. Not only does replication across multiple clusters provide geographic data locality (the ability to serve global traffic at low-latencies), but it can also be an integral part of a disaster recovery or backup strategy. You can also use multi-site replication to maintain secondary data stores, both for failover as well as for performing intensive computation without disrupting production load.

Multi-site replication in Riak works differently than the typical approach of relational databases. In Riak's multi-datacenter replication, one cluster acts as a "primary cluster." The primary cluster handles replication requests from one or more "secondary clusters" (generally 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-datacenter capabilities are "masterless."

In multi-datacenter replication, there are two primary modes of operation — full-sync and real-time. In full-sync mode, a complete synchronization occurs between primary and secondary cluster(s). In real-time mode, continual, incremental synchronization occurs and replication is triggered by new updates. Full-sync is performed upon initial connection of a secondary cluster, and then periodically (by default, every 6 hours). Full-sync is also triggered if the TCP connection between primary and secondary clusters is severed and then recovered.

Data transfer is unidirectional (primary->secondary). However, bidirectional synchronization can be achieved by configuring a pair of connections between clusters.

> *"The new consolidated platform will capture 2.25 billion weather data points 15 times per hour. As with any large-scale, algorithmic-type modeling, the more data you have, the better the weather predictions will be. NoSQL won out over relational databases primarily for its scalability, but Riak was chosen for its simplicity and ease of administration at high scale. It won out over Apache Cassandra, MongoDB and Hadoop."*
>
> — The Weather Company

CONCLUSION

Modeling data in any non-relational solution requires a different way of thinking about the data itself. Rather than an assumption that all data cleanly fits into a structure of rows and columns, the data domain can be overlaid on the core Key/Value store (Riak) in a variety of ways. There are, however, distinct tradeoffs and benefits to understand.

Relational Databases have:

- Tables
- Foreign keys and constraints
- ACID
- Sophisticated query planners
- Declarative query language (SQL)

Riak has:

- A Key/Value model where the value is any unstructured data
- More data redundancy that provides better availability
- Eventual consistency
- Simplified query capabilities
- Riak Search

What you will gain:

- More flexible, fluid designs
- More natural data representations
- Scaling without pain
- Reduced operational complexity

Picking the right database for your team requires a careful understanding of:

- The requirements of your application or platform
- What developer productivity means to you
- The operational conditions you need
- How different database solutions support your goals

Riak is designed to deliver maximum data availability, to scale linearly using commodity hardware, and to provide powerful yet simple data models for storing massive amounts of unstructured data.

NEXT STEPS

You can read more about Riak and Riak Enterprise, including more in-depth technical details for developers and operators, at the documentation portal. Riak is available open source for download at http://basho.com/resources/downloads/.

If you are interested in Riak Enterprise, please contact us. We would love to talk to you about your possible use case and how Riak can work for you.

For more information visit www.basho.com or follow us on Twitter at www.twitter.com/basho.