

Czech Technical University in Prague, Faculty of Information Technology

MIE-PDB: **Advanced Database Systems**

<http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-2-MIE-PDB/>

Lecture 8

Key-Value Stores: RiakKV

Martin Svoboda

svoboda@ksi.mff.cuni.cz

21. 4. 2017



Charles University, Faculty of Mathematics and Physics

NDBI040: Big Data Management and NoSQL Databases

Lecture Outline

RiakKV

- Data model
- HTTP interface
- **CRUD operations**
- **Link walking**
- Data types
- **Search 2.0**
- Internal details

Riak Key-Value Store



RiakKV

Key-value store

- <http://basho.com/products/riak-kv/>
- Features
 - Open source, incremental scalability, high availability, operational simplicity, decentralized design, automatic data distribution, advanced replication, fault tolerance, ...
- Developed by **Basho Technologies**
- Implemented in **Erlang**
 - General-purpose, concurrent, garbage-collected programming language and runtime system
- Operating system: **Linux**, Mac OS X, ... (not Windows)
- Initial release in 2009

Data Model

Riak database system structure

Instance (\rightarrow bucket types) \rightarrow **buckets** \rightarrow **objects**

- **Bucket** = **collection of objects** (logical, not physical collection)
 - Each object must have a unique key
 - Various properties are set at the level of buckets
 - E.g. default replication factor, read / write quora, ...
- **Object** = **key-value pair**
 - **Key** is a Unicode string
 - **Value** can be anything (text, binary object, image, ...)
 - Each object is also associated with **metadata**
 - E.g. its **content type** (text/plain, image/jpeg, ...),
 - and other internal metadata as well

Data Model

Design Questions

How **buckets, keys and values** should be designed?

- Complex objects containing various kinds of data
 - E.g. one key-value pair holding information about all the actors and movies at the same time
- **Buckets with different kinds of objects**
 - E.g. distinct objects for actors and movies, but all in one bucket
 - **Structured naming convention for keys** might help
 - E.g. actor_trojan, movie_medvidek
- **Separate buckets for different kinds of objects**
 - E.g. one bucket for actors, one for movies

Riak Usage: Querying

Basic **CRUD** operations

- Create, Read, Uppdate, and Delete
- Based on **key look-up**

Extended functionality

- **Links** – relationships between objects and their traversal
- **Search 2.0** – full-text queries accessing values of objects
- **MapReduce**
- ...

Riak Usage: API

Application interfaces

- **HTTP API**
 - All the user requests are submitted as **HTTP requests** with an appropriately selected **method** and specifically constructed **URL, headers, and data**
- Protocol Buffers API
- Erlang API

Client libraries for a variety of programming languages

- Official: Java, Ruby, Python, C#, PHP, ...
- Community: C, C++, Haskell, Perl, Python, Scala, ...

Riak Usage: HTTP API

cURL tool

- Allows to **transfer data from / to a server using HTTP** (or other supported protocols)

Options

- `-X command, --request command`
 - **HTTP request method to be used** (GET, ...)
- `-d data, --data data`
 - **Data to be sent** to the server (implies the **POST method**)
- `-H header, --header header`
 - **Extra headers** to be included when sending the request
- `-i, --include`
 - **Include received headers** when printing the response

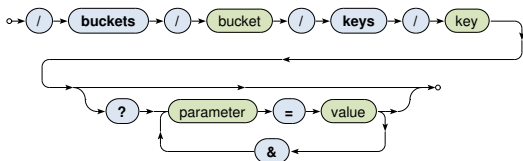
CRUD Operations

Basic operations on objects

- **Create**: POST or PUT methods
 - **Inserts a key-value pair** into a given bucket
 - Key is specified manually, or will be generated automatically
- **Read**: GET method
 - **Retrieves a key-value pair** from a given bucket
- **Uppdate**: PUT method
 - **Updates a key-value pair** in a given bucket
- **Delete**: DELETE method
 - **Removes a key-value pair** from a given bucket

CRUD Operations

URL pattern of HTTP requests for all the CRUD operations



Optional parameters (depending on the operation)

- r, w : read / write quorum to be attained
- ...

CRUD Operations

Create and Update

Inserts / updates a key-value pair in a given bucket

- **PUT** method
 - Should be used when a **key is specified explicitly**
 - Transparently **inserts / updates** a given object
- **POST** method
 - When a **key is to be generated automatically**
 - Always **inserts** a new object
- Buckets are created transparently whenever needed

Example

```
curl -i -X PUT
  -H 'Content-Type: text/plain'
  -d 'Ivan Trojan, 1964'
  http://localhost:8098/buckets/actors/keys/trojan
```

CRUD Operations

Read

Retrieves a **key-value pair** from a given bucket

- Method: **GET**

Example

Request

```
curl -i -X GET
  http://localhost:8098/buckets/actors/keys/trojan
```

Response

```
...
Content-Type: text/plain
...
```

```
Ivan Trojan, 1964
```

CRUD Operations

Delete

Removes a key-value pair from a given bucket

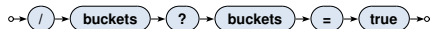
- Method: **DELETE**
- If a given object does not exist, it does not matter

Example

```
curl -i -X DELETE  
http://localhost:8098/buckets/actors/keys/trojan
```

Bucket Operations

Lists all the buckets (buckets with at least one object)



```
curl -i -X GET http://localhost:8098/buckets?buckets=true
```

```
Content-Type: application/json
```

```
{  
  "buckets" : [ "actors", "movies" ]  
}
```

Bucket Operations

Lists all the keys within a given bucket

- Not recommended since it is a very expensive operation



```
curl -i -X GET http://localhost:8098/buckets/actors/keys?keys=true
```

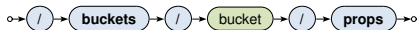
```
Content-Type: application/json
```

```
{  
  "keys" : [ "trojan", "machacek", "schneiderova", "sverak" ]  
}
```


Bucket Operations

Setting and retrieval of **bucket properties**

- Properties
 - `n_val`: replication factor
 - `r`, `w`, ...: read / write quora and their alternatives
 - ...
- Requests
 - GET method: **retrieve** bucket properties
 - PUT method: **set** bucket properties



Example

```
{  
  "props" : { "n_val" : 3, "w" : "all", "r" : 1 }  
}
```

Links and Link Walking

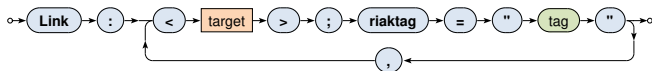
Links

- **Links** are metadata that establish one-way relationships **between objects**
 - Act as lightweight pointers between individual key-value pairs
 - I.e. represent and **extension to the pure key-value data model**
- Each link...
 - is defined at the source object
 - is associated with a **tag** (sort of link type)
- Multiple links can lead from / to a given object
- Source and target may not belong to the same bucket
- Motivation: **new way of querying**:
 - **Link walking** – navigation between objects

Links and Link Walking

Links: how are links defined?

- **Special Link header** is used for this purpose
- Multiple separate link headers can be provided, as well as multiple links within one header



Example

```
curl -i -X PUT
-H 'Content-Type: text/plain'
-H 'Link: </buckets/actors/keys/trojan>; riactag="tactor"'
-H 'Link: </buckets/actors/keys/machacek>; riactag="tactor"'
-d 'Medvídek, 2007'
http://localhost:8098/buckets/movies/keys/medvidek
```


Links and Link Walking

Link walking: parameters

- *Bucket*
 - Only objects from (exactly one) **target bucket** are found
 - `_` when not limited to any particular bucket
- *Tag*
 - Only links of a given **tag** are considered
 - `_` when not limited
- *Keep*
 - 1 when the objects should be included in the **result**
 - 0 otherwise
 - `_` means yes for the very last step, no for all the other

Links and Link Walking

Examples

Find all the actors that appeared in *Medvídek* movie

```
curl -i -X GET
  http://localhost:8098/buckets/movies/keys/medvidek
    /actors,tactor,1
```

```
Content-Type: multipart/mixed; boundary=...
```

Find all the movies in which appeared actors from *Medvídek* movie
(assuming that the corresponding actor → movie links also exist)

```
curl -i -X GET
  http://localhost:8098/buckets/movies/keys/medvidek
    /actors,tactor,0/movies,tmovie,1
```

Data Types

Motivation

- Riak began as a **pure key-value store**
 - I.e. was completely agnostic toward the contents of values
- However, if **availability is preferred to consistency**, mutually conflicting replicas might exist
 - Such **conflicts can be resolved at the application level**,
 - but this is often (only too) difficult for the developers
- And so the concept of **Riak Data Types** was introduced
 - When used (it is not compulsory), **Riak is able to resolve conflicts automatically** (and so eventual consistency is achieved)

Data Types

Available **data types**

- Register, flag, counter, set, and map
- Based on a generic concept of **CRDT** (*Convergent Replicated Data Types*)
- Cover (just) a few common scenarios
- Each applies specific **conflict resolution rule**

Implementation details

- Beside the **current value**, necessary **history of changes** is also internally stored so that conflicts can be judged

Data Types

Register

- Allows to store **any binary value** (e.g. string, ...)
- Convergence rule: **the most chronologically recent value wins**
- Note: registers can only be stored within maps

Flag

- **Boolean values:** enable (true), and disable (false)
- Convergence rule: **enable wins over disable**
- Note: flags can also be stored only within maps

Counter

- Operations: increment / decrement by a given integer value
- Convergence rule: **all increments and decrements by all actors are eventually applied**

Data Types

Set

- **Collection of unique binary values**
- Operations: addition / removal of one / multiple elements
- Convergence rule: **addition wins over removal** of elements

Map

- **Collection of fields with embedded elements** of any data type (including other nested maps)
- Operations: addition / removal of an element
- Convergence rule: **addition / update wins over removal**

Search 2.0

Riak **Search 2.0** (Yokozuna)

- **Full-text search engine**
 - Allows us to **find and query objects using full-text index structures based on the contents of the value parts**
- Based on **Apache Solr**
 - Distributed, scalable, failure tolerant, real-time search platform

Principles

- **Riak object to be indexed is transformed to a Solr document**
 - Various **extractors** are used for this purpose
- The resulting Solr document...
 - contains **fields** that are actually indexed by and within Solr
 - its contents must be described by a **schema**

Search 2.0: Extractors

Extractor

- Its goal is to **parse the value part** and **produce fields to index**
- Extractors are chosen automatically based on MIME types

Available extractors

- **Common predefined extractors**
 - Plain text, XML, JSON, *noop* (unknown content type)
- **Built-in extractors for Riak Data Types**
 - Counter, map, set
- **User-defined custom extractors**
 - Implemented in Erlang, registered with Riak

Search 2.0: Extractors

Plain text extractor (text/plain)

- Single field with the whole content is extracted

Example

Input Riak object

```
Ivan Trojan, 1964
```

Output Solr document

```
[  
  { text, <<"Ivan Trojan, 1964">> }  
]
```

Search 2.0: Extractors

XML extractor (text/xml, application/xml)

- One field is created for each element and attribute
- Dot notation is used to compose names of nested items

Example

Input Riak object

```
<?xml version="1.0" encoding="UTF-8" ?>
<actor year="1964">
  <name>Ivan Trojan</name>
</actor>
```

Output Solr document

```
[
  { <<"actor.name">>, <<"Ivan Trojan">> },
  { <<"actor.@year">>, <<"1964">> }
]
```

Search 2.0: Extractors

JSON extractor (application/json)

- Similar principles as for XML documents are applied

Example

Input Riak object

```
{  
  name : "Ivan Trojan",  
  year : 1964  
}
```

Output Solr document

```
[  
  { <<"name">>, <<"Ivan Trojan">> },  
  { <<"year">>, <<"1964">> }  
]
```

Search 2.0

Automatic fields

- A few technical fields are automatically added as well
- E.g. `_yz_rb` (containing **bucket name**), `_yz_rk` (**key**), ...

Solr index **schema**

- **Describes how fields should be indexed within Solr**
- Default schema available (`_yz_default`)
 - Suitable for debugging,
but custom schemas should be used in production

Field analysis and indexation

- E.g.:
 - Values of **fields are split into terms**
 - **Terms are normalized, stop words removed, ...**
 - Triples (token, field, document) are then indexed

Search 2.0: Index Creation

How is index created?

- Index must be created and then also associated with a bucket
- Each index servers to a single bucket only

Example

```
curl -i -X PUT
  -H 'Content-Type: application/json'
  -d '{ "schema" : "_yz_default" }'
  http://localhost:8098/search/index/iactors
```

```
curl -i -X PUT
  http://localhost:8098/search/index/iactors
```

```
curl -i -X PUT
  -H 'Content-Type: application/json'
  -d '{ "props" : { "search_index" : "iactors" } }'
  http://localhost:8098/buckets/actors/props
```

Search 2.0: Index Usage

Generic pattern for search queries

- Parameters
 - q – **search query** (correctly encoded)
 - wt – Solr response writer to be used to compose response
 - start and rows – pagination of matching objects
 - ...



Search 2.0: Index Usage

Available search functionality

- **Wildcards**
 - E.g. `name:Iva*`, `name:Iva?`
- **Range queries**
 - E.g. `year:[2010 TO *]`
- **Logical connectives** and parentheses
 - AND, OR, NOT
- **Proximity searches**
- ...

Architecture

Sharding + peer-to-peer replication architecture

- Any node can serve any **read** or **write** user request
- **Physical nodes** run (several) **virtual nodes (vnodes)**
 - Nodes can be added and removed from the cluster dynamically
- **Gossip protocol**
 - Each node periodically sends its current view of the cluster, its state and changes, bucket properties, ...

CAP properties

- AP system: **availability + partition tolerance**

Consistency

BASE principles

- **Availability is preferred to consistency**
- Default properties of buckets
 - n_val : replication factor
 - r : read quorum
 - w : write quorum (node participation is sufficient)
 - dw : write quorum (write to durable storage is required)
- Specific options of requests override the bucket properties

However, **strong consistency can be achieved**

- When quora set carefully, i.e.:
 - $w > n_val/2$ for write quorum
 - $r > n_val - w$ for read quorum

Causal Context

Conflicting replicas are unavoidable (with eventual consistency)

⇒ how are they resolved?

- **Causal context** = data and mechanisms necessary in order to resolve the conflicts
- **Low-level techniques**
 - Timestamps, vectors clocks, dotted version vectors
 - They can be used to resolve conflicts **automatically**
 - Might fail, then we must make the choice by ourselves
 - Or we can resolve the conflicts **manually**
 - Siblings then need to be enabled (`allow_mult`)
= multiple versions of object values
- User-friendly **CRDT data types** with built in resolution
 - Register, flag, counter, set, map

Causal Context

Vector clocks

- Mechanism for **tracking object update causality** in terms of logical time (not chronological time)
- **Each node has its own logical clock** (integer counter)
 - Initially equal to 0
 - Incremented by 1 whenever any event takes place
- **Vector clock = vector of logical clocks of all the nodes**
 - Each node maintains its local copy of this vector
 - **Whenever a message is sent, the local vector is sent as well**
 - **Whenever a message is received, the local vector is updated**
 - Maximal value for each individual node clock is taken

Riak Ring

Replica placement strategy

- Consistent hashing function
 - Consistent = does not change when cluster changes
 - Domain: pairs of a **bucket name and object key**
 - Range: **160-bit integer space** = Riak Ring

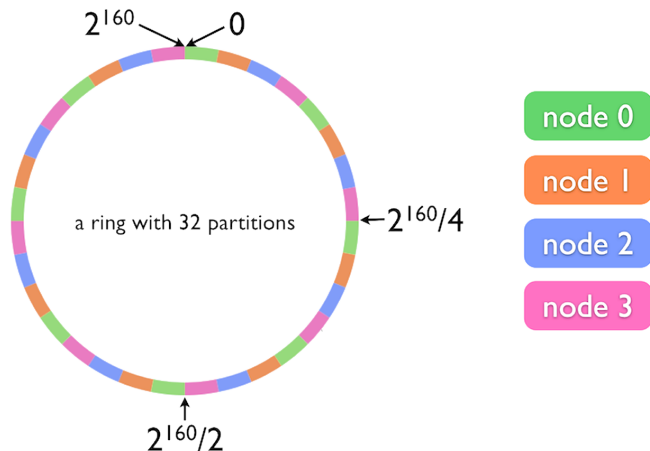
Riak Ring

- The whole ring is split into equally-sized disjoint partitions
 - Physical nodes are mutually interleaved
 - ⇒ reshuffling when cluster changes is less demanding
- **Each virtual node is responsible for exactly one partition**

Example

- Cluster with 4 physical nodes, each running 8 virtual nodes
- I.e. 32 partitions altogether

Riak Ring



Source: <http://docs.basho.com/>

Riak Ring

Replica placement strategy

- The first replica...
 - Its location is **directly determined by the hash function**
- All the remaining replicas...
 - Placed to the **consecutive partitions in a clockwise direction**

What if a virtual node is failing?

- Hinted handoff
 - Failing nodes are simply skipped, neighboring nodes temporarily take responsibility
 - When resolved, replicas are handed off to the proper locations
- Motivation: high availability

Request Handling

Read and write requests can be submitted to any node

- This node is called a **coordinating node**
- Hash function is calculated, i.e. **replica locations determined**
- **Internal requests are sent** to all the corresponding nodes
- Then the coordinating node starts to wait **until sufficient number of responses is received**
- **Result / failure is returned to the user**

But what if the cluster changes?

- The value of the hash function does not change, only the partitions and their mapping to virtual nodes change
- However, the Ring knowledge a given node has might be obsolete!

Lecture Conclusion

RiakKV

- **Highly available distributed key-value store**
- **Sharding with peer-to-peer replication architecture**
- **Riak Ring** with consistent hashing for replica placement

Query functionality

- Basic **CRUD operations**
- **Link walking**
- **Search 2.0** full-text based on Apache Solr