

Riak: key-value store

Software Languages Team
University of Koblenz-Landau
Ralf Lämmel and Andrei Varanovich

Motivation

- File system for the web: store everything (json, images, blobs, etc.) and make it accessible via HTTP with REST

What's Riak?

- Fault-tolerant
- Low-latency
- High-throughput
- HTTP + JSON

Key concepts

- ***Buckets, Keys***: data is stored and referenced by bucket/key pairs
- ***Values***: can be of any data type. Each value has a unique key.

Resources and Resource Identifiers

- The key abstraction of information in REST is a ***resource***.
- Each resource has a ***resource identifier***.

Examples of identifiers

- <http://example.com/customers/1234>
- <http://example.com/orders/2007/10/776654>
- <http://example.com/products/4554>
- <http://example.com/processes/salary-increase-234>

Remember:

Hypertext Transfer Protocol

http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

- **GET** Request representation for resource
- **HEAD** Like GET but without response body
- **PUT** Upload representation for resource
- **POST** Submit data for resource
- **OPTIONS** Query for available methods
- **CONNECT** Facilitate SSL-encrypted communication
- **DELETE** Delete specified resource
- **TRACE** Return request as it arrived at server
- **PATCH** Partial modification of resource

RESTful Web Service HTTP methods

- **Collection** URI, such as <http://example.com/companies/>
- GET: **List** the URIs and perhaps other details of the collection's members
- PUT: **Replace** the entire collection with another collection.
- POST: **Create** a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.
- DELETE: **Delete** the entire collection.

RESTful Web Service HTTP methods

- ***Element*** URI, such as `http://example.com/companies/32`
- **GET: Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
- **PUT: Replace** the addressed member of the collection, or if it doesn't exist, **create** it.
- **POST:** Treat the addressed member as a collection in its own right and **create** a new entry in it.
- **DELETE: Delete** the addressed member of the collection.

curl -v http://debeka.uni-koblenz.de:8091/riak/test

```
* About to connect() to 127.0.0.1 port 8098 (#0)
* Trying 127.0.0.1... connected
> GET /riak/test HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: 127.0.0.1:8098
> Accept: */*
>
< HTTP/1.1 200 OK
< Vary: Accept-Encoding
< Server: MochiWeb/1.1 WebMachine/1.9.0 (someone had painted it blue)
< Date: Tue, 11 Sep 2012 10:56:22 GMT
< Content-Type: application/json
< Content-Length: 422
<
* Connection #0 to host 127.0.0.1 left intact
* Closing connection #0
{"props":{"name":"test","allow_mult":false,"basic_quorum":false,"big_vclock":50,"chash_keyfun":
{"mod":"riak_core_util","fun":"chash_std_keyfun"},"dw":"quorum","last_write_wins":false,"linkfun":
{"mod":"riak_kv_wm_link_walker","fun":"mapreduce_linkfun"},"n_val":3,"notfound_ok":true,"old_vclock":
86400,"postcommit":[],"pr":0,"precommit":[],"pw":0,"r":"quorum","rw":"quorum","small_vclock":
50,"w":"quorum","young_vclock":20}}
```

simple GET

properties


Response codes

Normal status codes:

- 201 Created (when submitting without a key)
- 200 OK
- 204 No Content
- 300 Multiple Choices

Error codes:

- 400 Bad Request
- 412 Precondition Failed if one of the *conditional request* headers



.....
: If-None-Match, If-Match, If-Modified-Since, :
: and If-Unmodified-Since invoke conditional request semantics, :
: matching on the ETag and Last-Modified of the existing object :
:.....

Accessing buckets

```
curl -v http://debeka.uni-koblenz.de:8091/riak/bucket/key
```

< **HTTP/1.1 404 Object Not Found**

< Server: MochiWeb/1.1 WebMachine/1.9.0 (someone had painted it blue)

< Date: Tue, 11 Sep 2012 11:00:09 GMT

< Content-Type: text/plain

< Content-Length: 10

<

not found



HTTP 404

Data storage URL pattern

`http://SERVER:PORT/riak/BUCKET/KEY`

Storing data

```
curl -v -d 'this is a test' -H  
"Content-Type: text/plain"  
http://debeka.uni-koblenz.de:  
8091/riak/test
```

POST Request

Bucket: *test*

Key: *auto-generated*

Storing binaries

```
curl -v -X PUT http://debeka.uni-koblenz.de:8091/riak/img/KoblenzTotale.jpg  
-H "Content-Type: image/jpeg"  
--data-binary @KoblenzTotale.jpg
```

Open in browser: <http://debeka.uni-koblenz.de:8091/riak/img/KoblenzTotale.jpg>



Key



bucket

MIME content type

- **application/json**: JavaScript Object Notation [JSON](#); Defined in [RFC 4627](#)
- **application/pdf**: Portable Document Format, [PDF](#) has been in use for document exchange on the Internet since 1993; Defined in [RFC 3778](#)
- **application/zip**: [ZIP](#) archive files; Registered[\[7\]](#)
- **image/jpeg**: [JPEG](#) JFIF image; Defined in [RFC 2045](#) and [RFC 2046](#)
- **text/html**: [HTML](#); Defined in [RFC 2854](#)
- **video/mpeg**: [MPEG-1](#) video with multiplexed audio; Defined in [RFC 2045](#) and [RFC 2046](#)

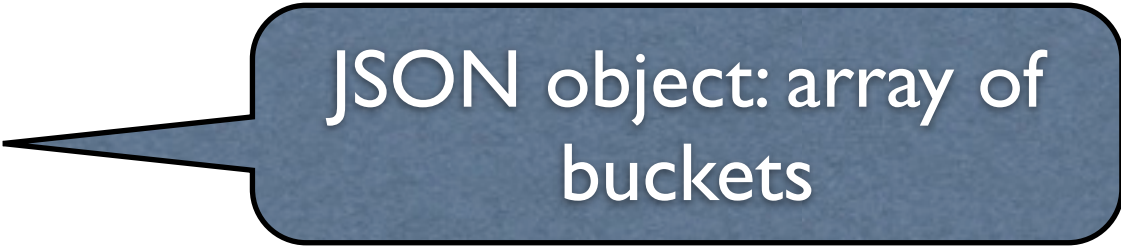
Full list:

http://en.wikipedia.org/wiki/Internet_media_type

View list of buckets

<http://debeka.uni-koblenz.de:8091/buckets?buckets=true>

```
{  
  buckets:  
  [ "img" ]  
}
```



JSON object: array of buckets

Delete object

```
curl -v -X DELETE
```

```
http://debeka.uni-koblenz.de:8091/riak/img/KoblenzTotale.jpg
```

What about relationships between objects? In Riak they are called “Links”

HTTP Header with POST request

Links are **metadata** that establish one-way relationships between objects in Riak:

```
Link: </riak/bucket/key>; riaktag="tag"
```

```
.....  
: Link: </riak/list/1>; riaktag="previous", </riak/list/3>; riaktag="next" :  
.....
```

multiple links

DEMO

I0I companies:riak

Manager

http://debeka.uni-koblenz.de:8091/riak/meganalysis_employees/klaus

HEADERS:

'Link': '</riak/meganalysis_depts/dev1>; riaktag="manages"'
'content-type': 'application/json'

DATA:

```
{"salary": 23456, "name": "Klaus",  
  "address": {"city": "Boston", "country": "USA"}}
```

Department

http://debeka.uni-koblenz.de:8091/riak/meganalysis_depts/dev1

HEADERS:

'Link': '</riak/meganalysis_employees/klaus>; riaktag="employs"' **'content-type':** 'application/json'

DATA:

```
{"name": "Dev1"}
```

Sub-department

http://debeka.uni-koblenz.de:8091/riak/meganalysis_depts/development

HEADERS:

'Link': '</riak/meganalysis_employees/ray>; riaktag="employs",
</riak/meganalysis_depts/dev1>; riaktag="has_subunit"',

'content-type': 'application/json'

Querying data

All sub-departments and employees:

```
curl http://debeka.uni-koblenz.de:8091/riak/meganalysis_depts/development/
```

All sub-departments only:

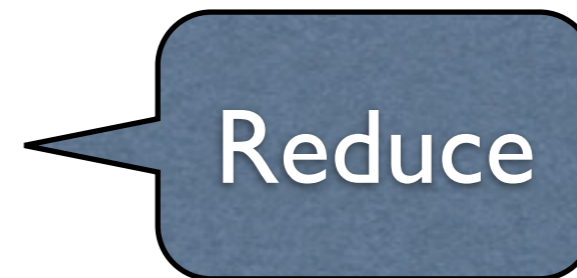
```
curl http://debeka.uni-koblenz.de:8091/riak/meganalysis_depts/development/  
has_subunit,
```

```
GET /riak/bucket/key/[bucket],[tag],[keep]
```

- Bucket - a bucket name to limit the links to
- Tag - a "riaktag" to limit the links to
- Keep - 0 or 1, whether to return results from this phase

Total

```
totalcode = {  
  "inputs" : "meganalysis_employees",  
  "query" : [ .....  
    {"map" : {  
      "language" : "javascript",  
      "source" : ""function(v) {  
        var parsedData = JSON.parse(v.values[0].data);  
        return [{'salary' : parsedData.salary}];  
      }""  
    }  
  },  
  {"reduce" : {"language" : "javascript",  
    "source" : ""function(mappedVals) {  
      var sums = {'salary' : 0};  
      for (var i in mappedVals) {  
        sums.salary += mappedVals[i].salary;  
      }  
      return [sums];  
    }""}}]  
}
```



```
header = {"content-type" : "application/json"}  
host = "http://localhost:8091/mapred"
```


Cut

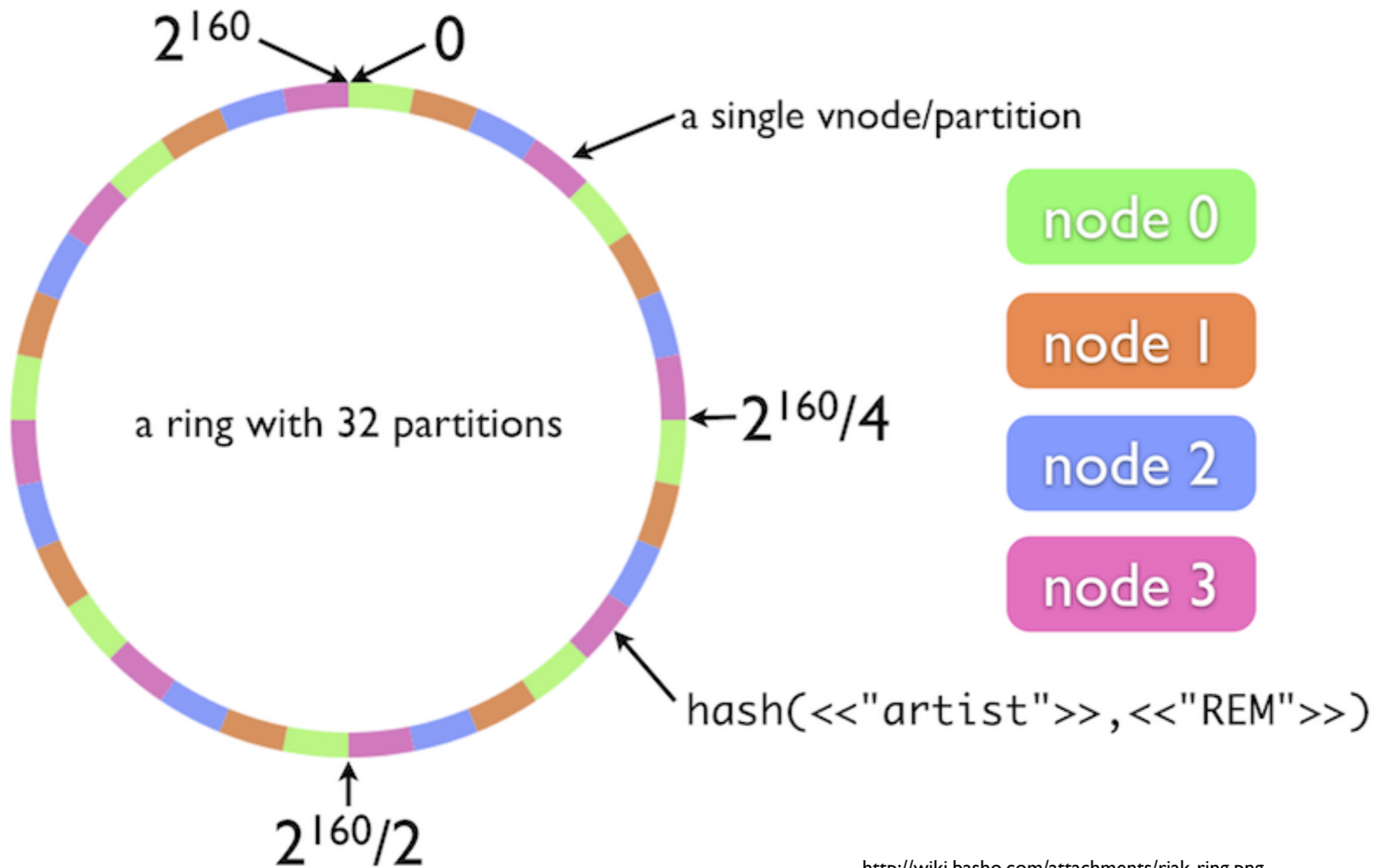
```
cutcode = {  
  "inputs" : "meganalysis_employees",  
  "query" : [  
    {"map" : {  
      "language" : "javascript",  
      "source" : ""function(v) {  
var parsedData = JSON.parse(v.values[0].data);  
parsedData.salary = parsedData.salary / 2.0;  
return [{"key" : v.key, "value" : parsedData}];  
}""}}]}]
```

Returns an array of updated objects

Need to iterate over this array
and PUT the object back to Riak

No in-place
updates!

Clustering



<http://wiki.basho.com/attachments/riak-ring.png>

Summary

You learned about ...

- using Riak as a key-value storage,
- storing the relational data in Riak, and
- using MapReduce for data aggregation.

Resources

- **Dynamo: Amazon's Highly Available Key-value Store:**

<http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf>

- **Riak documentation:**

<http://wiki.basho.com/>