NoSQL project instructions for the course Advanced databases

Academic year 2014/2015

Instructions are divided into two parts:

1. **Tutorial on basic Riak functionalities**
2. **Assignments for students to complete on their own**

1. Tutorial on basic Riak functionalities

Start the virtual machine and connect to the machine.

Starting Riak cluster:

From folder /usr/riak/riak-1.4.2/dev start at least 3 Riak nodes:

```
> ulimit -n 4096
> dev1/bin/riak start
> dev2/bin/riak start
> dev3/bin/riak start
> dev4/bin/riak start
> dev5/bin/riak start
```

Check if they are operational:

```
> ps aux | grep beam
```

Use files dev*/etc/app.config to discover ports used by the nodes.
Look for configuration line of type: {http, [{"192.168.56.12", PORTNUMBER}]}
Use:

```
> netstat -a -n | grep 12:100
```

to make sure that Riak is listening at those ports.

Connect nodes into a cluster:

```
> dev2/bin/riak-admin cluster join dev1@192.168.56.12
```

(From here onward it is not relevant which node you are connecting to, they are all equal):

```
> dev3/bin/riak-admin cluster join dev2@192.168.56.12
> dev4/bin/riak-admin cluster join dev3@192.168.56.12
> dev5/bin/riak-admin cluster join dev4@192.168.56.12
```

above instruction only prepare the ground, but do not actually create the cluster.
Check out cluster plan using the following instruction:

```
> dev1/bin/riak-admin cluster plan
```

and finally, create the cluster (once again, it doesn't matter which node we use):

```
> dev2/bin/riak-admin cluster commit
```

Check out node statistics with:

```
> curl http://192.168.56.12:10018/stats
```

Look for connected_nodes and ring_members.

Try also:

```
> curl http://192.168.56.12:10018/ping
```

Insert, update and delete of values

Insert, read and delete a value, i.e.:

```
    > curl -v -X PUT http://192.168.56.12:10018/riak/testbucket/helloworld -H
"Content-Type: text/html" -d "<html><body>Hello world\!</body></html>"
```

What http code will last instruction return?
What does that http code stand for?
Read from all nodes (you can go without -v):

```
    curl -v http://192.168.56.12:10018/riak/testbucket/helloworld
    curl -v http://192.168.56.12:10028/riak/testbucket/helloworld
    curl -v http://192.168.56.12:10038/riak/testbucket/helloworld
    curl -v http://192.168.56.12:10048/riak/testbucket/helloworld
    curl -v http://192.168.56.12:10058/riak/testbucket/helloworld
```

What do you conclude from this?

Find out a default replication factor for a bucket (look for "n_val"):

```
  > curl http://192.168.56.12:10018/buckets/testbucket/props
```

Add one more value to bucket testbucket2:

```
  > curl -v -X PUT http://192.168.56.12:10018/riak/testbucket2/helloworld -H
"Content-Type: text/html" -d "<html><body>Hello world\!</body></html>"
```

Get a list of all existing buckets (you can add a few more)

```
  > curl http://192.168.56.12:10018/riak?buckets=true
```

Add one more value to bucket testbucket2:

```
  > curl -v -X PUT http://192.168.56.12:10018/riak/testbucket2/goodday -H
"Content-Type: text/html" -d "<html><body>Today is a good day.</body></html>"
```

and one more without a key (Riak will generate a key). Note that you need to use POST instead of PUT:

```
  > curl -v -X POST http://192.168.56.12:10018/riak/testbucket2 -H "Content-
Type: text/html" -d "<html><body>Generated key.</body></html>"
```

What is http response code?
What is the generated key?
Read that value (switch the generated key with your own).

```
  > curl http://192.168.56.12:10018/riak/testbucket2/Jb3lPf8FJzv8IPfE6ZvwbWwcDnJ
```

Get all keys in bucket testbucket2:

```
  > curl http://192.168.56.12:10018/riak/testbucket2?keys=true
```

Finally, delete one value from bucket testbucket2, i.e. (in the following instructions switch the key with your own):

```
    > curl -v -X DELETE
http://192.168.56.12:10018/riak/testbucket2/Jb3lPf8FJzv8IPfE6ZvwbWwcDnJ
```

What is the response code?
Try to read the deleted value.
Try to delete the same value again (repeat the instruction).

Access from an outside computer

Use outside (host) computer (after installing Curl) to add a value with an arbitrary key using Curl program (as before).
View it using an internet browser.

Insert an image into Riak (place an image into a working folder and switch “slika.png“ with image filename in the following instruction):

```
    > curl -X PUT http://192.168.56.12:10028/riak/images/slika.jpg -H "Content-
type: image/png" --data-binary @slika.jpg
```

Read it using internet browser.

Besides images, it is convenient to store other types of files in this way. For example, while solving a second problem, you can store .js file in this way:

```
    > curl -v -X PUT http://192.168.56.12:10018/riak/nmbp/nmbp_map.js -H
"Content-Type: text/javascript" --data-binary @nmbp_map.js
```

Stopping and restarting Riak

Stop Riak cluster:

```
    > dev5/bin/riak stop
    > dev4/bin/riak stop
    > dev3/bin/riak stop
    > dev2/bin/riak stop
    > dev1/bin/riak stop
```

Restart it:

```
  > dev5/bin/riak start
    > dev4/bin/riak start
    > dev3/bin/riak start
    > dev2/bin/riak start
    > dev1/bin/riak start
```

Execute:

```
    > curl http://192.168.56.12:10018/stats
```

and look for „connected_nodes“.
Do you need to create the cluster again?
Try, for example:

```
    > dev2/bin/riak-admin cluster join dev1@192.168.56.12
```

Try once more:

```
    > curl http://192.168.56.12:10018/riak/testbucket2?keys=true
```

to make sure that the values are preserved.

Vector clocks

Create a new bucket by setting a property allow_mult = true (default is false) which allows creating sibling values.

Riak supports two bucket property formats, old and new one.
(http://docs.basho.com/riak/latest/dev/references/http/set-bucket-props/):

```
    PUT /riak/bucket                        # Old format
    PUT /buckets/bucket/props       # New format
```

```
    > curl -v -X PUT http://192.168.56.12:10018/buckets/dogovor/props -H
"Content-Type: application/json" -d '{"props":{"allow_mult":true}}'
```

Get all properties and check if they are correctly set:

```
  > curl -v  http://192.168.56.12:10018/buckets/dogovor/props
```

Let assume that Ljilja, Krešo and Jasna want to come to an agreement on grading a student Igor's homework assignment.

(a) Krešo suggests grade one:

```
    > curl -v -X PUT http://192.168.56.12:10018/buckets/dogovor/keys/igor -H
"Content-Type: application/json" -H "X-Riak-ClientId: kreso" -d '{"grade":"1"}'
```

Get the value and write a vector clock:

```
    > curl -v http://192.168.56.12:10018/buckets/dogovor/keys/igor
```

i.e. X-Riak-Vclock: a85hYGBgzGDKBVIcKlYHQoIWzJiQwZTImMfKsCZA6gxfFgA=
(your clock will surely be different)

(b) Ljilja read Krešo's suggestion and suggests grade 2 instead (with the knowledge that someone already suggested a negative grade). Including VClock in the request, Ljilja lets Riak know which (previous) version of the values she saw:

```
    > curl -i -X PUT http://192.168.56.12:10018/buckets/dogovor/keys/igor \
        -H "X-Riak-ClientId: ljilja" \
        -H "X-Riak-Vclock: a85hYGBgzGDKBVIcKlYHQoIWzJiQwZTImMfKsCZA6gxfFgA=" \
        -H "Content-Type: application/json" \
        -d '{"grade" : 2}'
```

We can read the value again to make sure that the grade is now 2.

```
       > curl -v http://192.168.56.12:10018/buckets/dogovor/keys/igor
```

Check the vector clock?
Compare its length to previous one.

(c) Let's suppose that Jasna read the grade before Ljilja changed it (while the grade was 1). Jasna (after Ljilja already changed grade to 2) accepts 1 and executes:

```
    > curl -i -X PUT http://192.168.56.12:10018/buckets/dogovor/keys/igor \
        -H "X-Riak-ClientId: jasna" \
        -H "X-Riak-Vclock: a85hYGBgzGDKBVIcKlYHQoIWzJiQwZTImMfKsCZA6gxfFgA=" \
        -H "Content-Type: application/json" \
        -d '{"grade" : 1}'
```

How should the system react?
Read the value:

```
    > curl -v http://192.168.56.12:10018/buckets/dogovor/keys/igor
```

Where are the values?
What is the HTTP code?

Check the vector clock?
Compare its length to previous one.

Here, a **conflict appeared and needs to be resolved**.
Both versions (siblings) can be read by adding -H "Accept: multipart/mixed":

```
   > curl -v http://192.168.56.12:10018/buckets/dogovor/keys/igor -H "Accept:
multipart/mixed"
```

or you can read them one by one, using sibling keys, i.e. (switch with your own key):

```
 > curl -v
http://192.168.56.12:10018/buckets/dogovor/keys/igor?vtag=2vUo9hVift7TTf3q0LkG5h
  > curl -v
http://192.168.56.12:10018/buckets/dogovor/keys/igor?vtag=3dFQPS7ZQXoU9MUlXrYYdy
```

Finally, **Ljilja** (luckily for Igor) **resolves the conflict** with (switch vector clock with your own):

```
    > curl -i -X PUT http://192.168.56.12:10018/buckets/dogovor/keys/igor \
        -H "X-Riak-ClientId: ljilja" \
        -H "X-Riak-Vclock:
a85hYGBgymDKBVIccw9K7wlaMGNKBlMiYx4rQ2yQ1Bk+qJSK1YEQoNQEoBQzUEo0GCiVBQA=" \
        -H "Content-Type: application/json" \
        -d '{"grade" : 2}'
```

Let's check that only one value remains (no more siblings):

```
    > curl -v http://192.168.56.12:10018/buckets/dogovor/keys/igor
```

Try out (or at least read through) a basic Riak MapReduce example:
http://docs.basho.com/riak/latest/dev/using/mapreduce/

This concludes a basic tutorial on working with Riak.
For all questions consult Riak documentation: http://docs.basho.com/riak/latest/

2. Assignments for students to complete on their own

Students need to complete two assignments described in the following text.

1. NoSQL1: Minimal portal based on Riak (10 points)

Construct a web portal which displays N most recent (npr. N=10) articles.

> **?** Figure out how to do that, i.e. use the appropriate data structure.
> E.g.: assuming the database contains 10,000 articles it **is not a good strategy** to fetch all 10,000 articles to the client, sort them, and take the top ten.

Each article should roughly have the following form:

<table>
<tr><td>Title</td><td rowspan="3">Image</td></tr>
<tr><td>Text</td></tr>
<tr><td>Author</td></tr>
</table>

Implement article comments: add a text field for submitting comments beneath every article.
You do not need to construct an interface for entering new articles (but are welcome to do so if you want). You can enter new articles using Curl.
For example:

2. NoSQL2: MapReduce queries (3 + 7 = 10 points)

Write a:
(a) (3 points) MapReduce query returning a list of articles sorted descending by the number of comments.

(b) (7 points) MapReduce query that for each author returns 10 most used words. Use the concept of "word" in the simplest possible way (a series of letters separated with space, coma of full stop).

You don't need to perform any lexical transformations (like stemming). You do not need to return the 11th word if it has the same number of uses as the 10th one.

Incomplete solutions will be awarded partial points (i.e. all word used by an author instead of 10 most used ones).

Query result can be displayed using the same web portal, or the query can be executed from the console.

Some advice:

- Try out JavaScript code using Chrome console, and then store it in a Riak bucket.
- M/R javascript examples are available from the previous Riak version's docs (1.4): http://docs.basho.com/riak/1.4.1/dev/advanced/mapreduce/
- Debugging Riak JavaScript MapReduce (taken from Riak pages):

Debugging Javascript MapReduce Phases

There are currently two facilities for debugging MapReduce phases. If there was an exception in the Javascript VM you can view the error in the `log/sasl-error.log` file. In addition to viewing exceptions you can write to a specific log file from your map or reduce phases using the ejsLog function.
Javascript:

```
ejsLog('/tmp/map_reduce.log', JSON.stringify(value))
```

Note that when used from a map phase the ejsLog function will create a file on each node on which the map phase runs. The output of a reduce phase will be located on the node you queried with your MapReduce function.

For both assignments:

Project solutions should be structured as follows:

```
|-NoSql1
   |- main1.txt
   |- (...)               // other files, i.e. images, text, json etc.
   |- (projekt.zip) // zip webapp of the project if solved in that way.
|-NoSql2
   |- main2.txt
   |- (...) // other files, i.e. js files.
```

In both cases main?.txt file should contain step by step descriptions and instructions needed to be executed to complete the assignment (something like instructions from part 2 of this document)

When presenting the assignment solutions, students could be asked to:

- Explain a part of the solution (main.txt)
- Explain any of the concepts from these instructions (i.e. vector clock)
- Demonstrate that a solution works (an i.e. add a new article to the portal)
- make some minor modification to the solution (i.e. a small modification of a M/R query)
- etc.