

CSC570 NoSQL Databases

Lab 2, Part 2: RIAK and Mapreduce

RIAK and Mapreduce:

In part 2 of our look at RIAK we will use its mapreduce framework to perform more powerful queries than the standard key-value paradigm can normally provide. RIAK's link walking combined with mapreduce creates powerful querying capabilities for simple key-value databases.

We can also investigate the server architecture of Riak and how its approach to server layout provides flexibility in consistency or availability.

If you have turned off your RIAK VM while performing the labs, you will need to get your 4-node cluster back up and running. RIAK assumes that the nodes have dropped as each one was shutdown.

Power on your RIAK VM, open a terminal window, get to your `/riak-1.2.1/dev` directory and start each node. Since you have never removed these servers from the cluster, once they have restarted, they should rejoin the cluster. You can verify this by checking the status:

```
$ dev1/bin/riak-admin member-status
```

You should see your four-node cluster valid and equally sharing the workload:

```
student@student-virtual-machine:~/riak-1.2.1/dev$ dev1/bin/riak-admin member-status
===== Membership =====
Status      Ring      Pending   Node
-----
valid       25.0%     --        'dev1@127.0.0.1'
valid       25.0%     --        'dev2@127.0.0.1'
valid       25.0%     --        'dev3@127.0.0.1'
valid       25.0%     --        'dev4@127.0.0.1'
-----
Valid:4 / Leaving:0 / Exiting:0 / Joining:0 / Down:0
student@student-virtual-machine:~/riak-1.2.1/dev$
```

MapReduce

MapReduce (M/R) is a technique for dividing work across a distributed system. It takes advantage of the parallel processing power of distributed systems, and also reduces network bandwidth as the algorithm is passed around to where the data is stored, rather than a potentially huge dataset transferred to a client algorithm. Developers can use MapReduce for things like filtering documents by tags, counting words in documents, and extracting links to related data.

In Riak, MapReduce is one method for non-key-based querying. MapReduce jobs can be submitted through the HTTP API or the Protocol Buffers API. (It should be mentioned that Riak MapReduce is intended for batch processing, not real-time querying.)

When to Consider Using a MapReduce Solution

- When you know the set of objects you want to MapReduce over (the bucket-key pairs)
- When you want to return actual objects or pieces of the object – not just the keys, (Search & Secondary Indexes are good for keys)

- When you need utmost flexibility in querying your data. MapReduce gives you full access to your object and lets you pick it apart any way you want.

When Not to Use MapReduce

- When you want to query data of an entire bucket. MapReduce uses a list of keys, which can place a lot of demand on the cluster.
- When you want latency to be as predictable as possible.

How MapReduce Works

The MapReduce framework helps developers divide a query into steps, divide the dataset into chunks, and then run those step/chunk pairs in separate physical hosts.

There are two steps in a MapReduce query in RIAK:

1. Map – data collection phase. Map breaks up large chunks of work into smaller ones and then takes action on each chunk.
2. Reduce – data collation or processing phase. Reduce combines the many results from the map step into a single output (*this step is optional*).

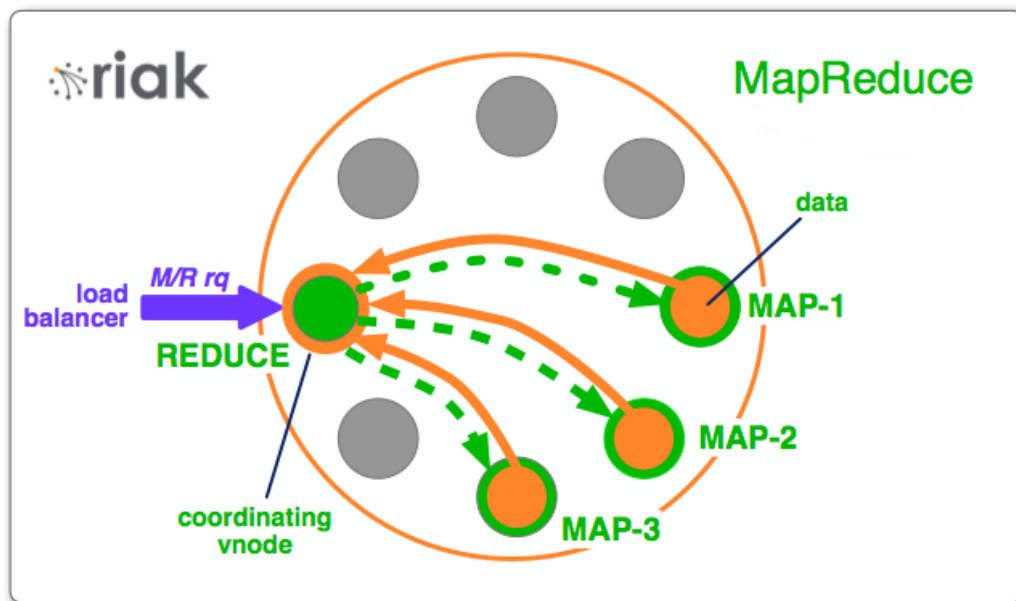


Figure 1: from <http://docs.basho.com/riak>

Riak MapReduce queries have two components:

- A list of inputs
- A list of phases

The elements of the input list are bucket-key pairs. The elements of the phases list are chunks of information related to a map, a reduce, or a link function.

The client makes a request to Riak. The node the client contacts to make the request becomes the coordinating node for the MapReduce job. The MapReduce job consists of a list of phases— either a map or a reduce. The map phase consists of a function and a list of objects the function will operate on, bucketed by the object's key. The coordinator uses the list to route the object keys and the function with a request for the vnode to run that function over those particular objects.

After running the map function, the results are sent back to the coordinating node. The coordinating node concatenates the list and then passes that information over to a reduce phase on the same coordinating node (assuming reduce is the next phase in the list).

Our first mapreduce

In this example we will create four objects with the text “pizza” sometimes repeated. Javascript MapReduce will be used to count the occurrences of the word “pizza”.

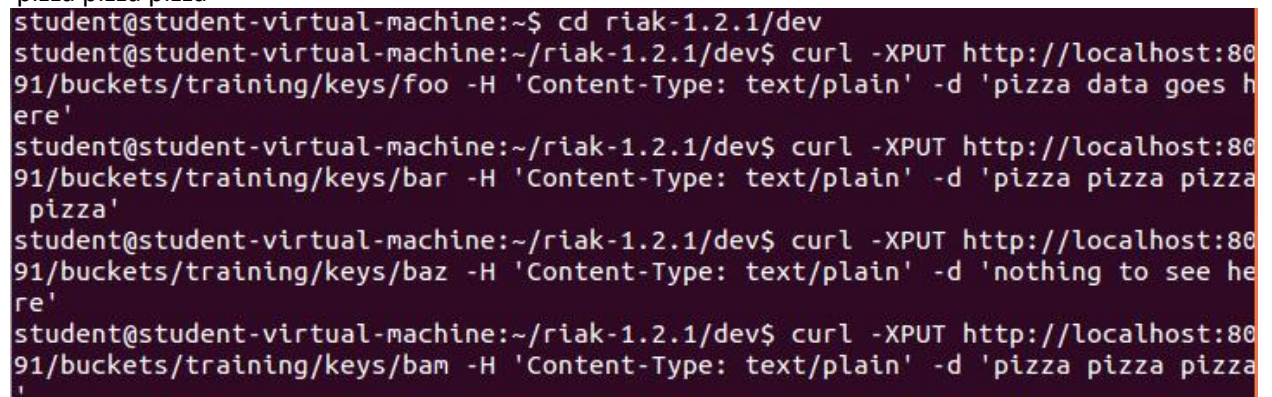
1. Populate your buckets: In your /riak-1.2.1/dev directory type:

```
$ curl -XPUT http://localhost:8091/buckets/training/keys/foo -H 'Content-Type: text/plain' -d 'pizza data goes here'
```

```
$ curl -XPUT http://localhost:8091/buckets/training/keys/bar -H 'Content-Type: text/plain' -d 'pizza pizza pizza pizza'
```

```
$ curl -XPUT http://localhost:8091/buckets/training/keys/baz -H 'Content-Type: text/plain' -d 'nothing to see here'
```

```
$ curl -XPUT http://localhost:8091/buckets/training/keys/bam -H 'Content-Type: text/plain' -d 'pizza pizza pizza'
```



```
student@student-virtual-machine:~$ cd riak-1.2.1/dev
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -XPUT http://localhost:8091/buckets/training/keys/foo -H 'Content-Type: text/plain' -d 'pizza data goes here'
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -XPUT http://localhost:8091/buckets/training/keys/bar -H 'Content-Type: text/plain' -d 'pizza pizza pizza pizza'
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -XPUT http://localhost:8091/buckets/training/keys/baz -H 'Content-Type: text/plain' -d 'nothing to see here'
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -XPUT http://localhost:8091/buckets/training/keys/bam -H 'Content-Type: text/plain' -d 'pizza pizza pizza'
```

2. Enter the mapreduce script:

```
curl -XPOST http://localhost:8091/mapred \
-H 'Content-Type: application/json' \
-d '{
  "inputs": "training",
  "query": { "map": { "language": "javascript",
    "source": "function(riakObject) {
      var m = riakObject.values[0].data.match(/pizza/g);
```

```

return [[riakObject.key, (m ? m.length : 0 )]];
}]]]]}

```

```

student@student-virtual-machine:~/riak-1.2.1/dev$ curl -XPOST http://localhost:8091/mapred \
> -H 'Content-Type: application/json' \
> -d '{
>   "inputs": "training",
>   "query": [{"map": {"language": "javascript",
>   "source": "function(riakObject) {
>     var m = riakObject.values[0].data.match(/pizza/g);
>     return [[riakObject.key, (m ? m.length : 0 )]];
>   }]]]]}'

```

3. This will return the key of each object, followed by the count of the word “pizza”

```

[["baz",0],["bar",4],["foo",1],["bam",3]]
student@student-virtual-machine:~/riak-1.2.1/dev$

```

What the mapreduce script did was use the 'training' bucket as the input data; called a Javascript MapReduce function; the function took each riakObject that exists in the 'training' bucket and searched the text for the word “pizza”; 'm' is the result of the search, which includes zero or more results that match “pizza”; the function then returned the key of the riakObject and the number of matches.

A Larger example: Google stock data

Rarely will you work with tiny data sets like our first example. Typically, you’ll create (or extract) a large data file, upload it onto your cluster, then analyze it.

1. Download the goog.csv file from <http://csc570e.uis.edu/csc561/> and save it in your /riak-1.2.1/dev directory
2. Download the Erlang script, load_data.erl, from <http://csc570e.uis.edu/csc561/> and save it in your /riak-1.2.1/dev directory
3. The load script will not be executable until to change its mode. Do this by entering:

```
$ chmod +x load_data.erl
```

```
student@student-virtual-machine:~/riak-1.2.1/dev$ chmod +x load_data.erl
```

4. Load the data into your cluster by executing the script by entering:

```
$ ./load_data.erl goog.csv
```

```

student@student-virtual-machine:~/riak-1.2.1/dev$
student@student-virtual-machine:~/riak-1.2.1/dev$ ./load_data.erl goog.csv

```

You should see the data scroll as it is read into the cluster:

```
Inserting: 2004-08-27
Inserting: 2004-08-26
Inserting: 2004-08-25
Inserting: 2004-08-24
Inserting: 2004-08-23
Inserting: 2004-08-20
Inserting: 2004-08-19
student@student-virtual-machine:~/riak-1.2.1/dev$
```

You can open the loading script and take a look at the code, I am not going into much detail on it because I want you focus on the actual mapreduce.

Before we can execute more C\complex Mapreduce...

HTTP Query Syntax

Before we run some MapReduce queries of our own on the sample data, we should review a bit about how to write the queries and how they are executed.

MapReduce queries are issued over HTTP via a POST to the “/mapred” resource. The body should be “application/json” of the form:

```
{"inputs":[...inputs...],"query":[...query...]}
```

Inputs

The list of input objects is given as a list of 2-element lists of the form [Bucket,Key] or 3-element lists of the form [Bucket,Key,KeyData].

You may also pass just the name of a bucket ({"inputs":"mybucket",...}), which is equivalent to passing all of the keys in that bucket as inputs (i.e. “a map/reduce across the whole bucket”). You should be aware that this triggers the somewhat expensive “list keys” operation, so you should use it sparingly.

Query

The query is given as a list of phases, each phase being of the form {PhaseType:{...spec...}}. Valid PhaseType values are “map”, “reduce”, and “link”.

Every phase spec may include a “keep” field, which must have a Boolean value: “true” means that the results of this phase should be included in the final result of the map/reduce, “false” means the results of this phase should be used only by the next phase. Omitting the “keep” field accepts its default value, which is “false” for all phases except the final phase (Riak assumes that you were most interested in the results of the last phase of your map/reduce query).

Map

Map phases must be told where to find the code for the function to execute, and what language that function is in. Function source can be specified directly in the query by using the “source” spec field. Function source can also be loaded from a pre-stored Riak object by providing “bucket” and “key” fields in the spec.

Examples:

```
{"map":{"language":"javascript","source":"function(v) { return [v]; }","keep":true}}
```


would run the Javascript function given in the spec, and include the results in the final output of the m/r query.

```
{"map":{"language":"javascript","bucket":"myjs","key":"mymap","keep":false}}
```

would run the Javascript function declared in the content of the Riak object under “mymap” in the “myjs” bucket, and the results of the function would not be included in the final output of the m/r query.

```
{"map":{"language":"erlang","module":"riak_kv_mapreduce","function":"map_object_value"}}
```

would run the Erlang function “riak_kv_mapreduce:map_object_value/3”.

Reduce

Reduce phases look exactly like map phases, but are labeled “reduce”.

Link

Link phases accept “bucket” and “tag” fields that specify which links match the link query. The string “” (underscore) in each field means “match all”, while any other string means “match exactly this string”. If either field is left out, it is considered to be set to “” (match all).

Example:

```
{"link":{"bucket":"foo","keep":false}}
```

would follow all links pointing to objects in the “foo” bucket, regardless of their tag.

Submitting MapReduce queries from the shell

To run a query from the shell, here’s the curl command to use:

```
curl -X POST http://127.0.0.1:8091/mapred -H "Content-Type: application/json" -d @-
```

After pressing return, paste your jobcode in, press return again, and then Ctrl-D to submit it.

This way of running MapReduce queries is not specific to our lab; it is a code shortcut that comes in very handy to run quick fire-and-forget queries from the command line. With a client library, most of the dirty work of assembling the JSON that’s sent to Riak will be done for you but this should help you for this lab

Query 1: return the entire data set

So, to use the above, lets create a simple map-only job that returns the entire data set:

1. Type in the line of code above , press return

```
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -X POST http://127.0.0.1:8091/mapred -H "Content-Type:application/json" -d @-
```

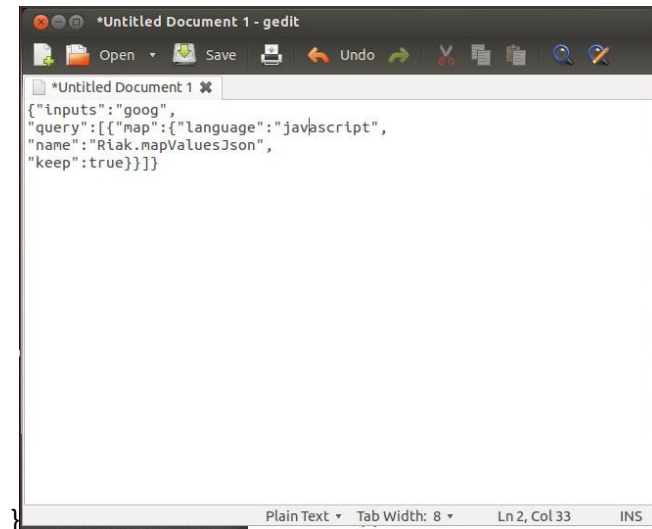
2. In a text editor (gedit installed on your machine), type out your specific query, ie, the data body of our request after the –d @-

```
{"inputs":"goog",  
  "query":{"map":{"language":"javascript",  
                  "name":"Riak.mapValuesJson",
```

```

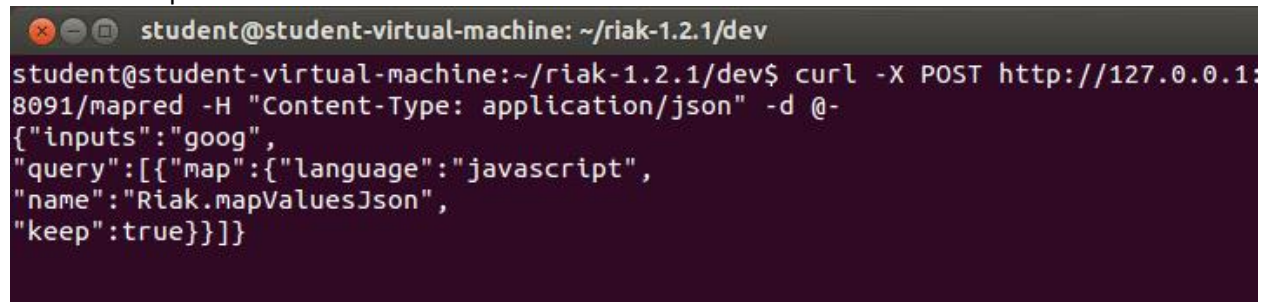
    "keep":true}}
  ]
}

```



Riak.mapValuesJson is a built-in function that parses and extracts our object as Json data.

3. Now select your text from your text editor and paste it after the `-@` on the command line of your terminal and press enter



4. Enter Control `-D` (press `<Ctrl><d>`) to execute. This will return the entire contents of our data set

```

student@student-virtual-machine: ~/riak-1.2.1/dev
: "2006-09-13", "Open": 395.15, "High": 406.76, "Low": 395.1, "Close": 406.57, "Volume": 9768200, "Adj. Close": 406.57}, {"Date": "2006-03-02", "Open": 364.28, "High": 381.1, "Low": 362.2, "Close": 376.45, "Volume": 18330300, "Adj. Close": 376.45}, {"Date": "2009-08-07", "Open": 455.67, "High": 459.42, "Low": 454.99, "Close": 457.1, "Volume": 2543100, "Adj. Close": 457.1}, {"Date": "2007-06-18", "Open": 506.18, "High": 516, "Low": 504.24, "Close": 515.2, "Volume": 4835900, "Adj. Close": 515.2}, {"Date": "2009-07-07", "Open": 408.24, "High": 409.19, "Low": 395.98, "Close": 396.63, "Volume": 3259300, "Adj. Close": 396.63}, {"Date": "2004-12-28", "Open": 192.11, "High": 193.55, "Low": 191.01, "Close": 192.76, "Volume": 4145800, "Adj. Close": 192.76}, {"Date": "2009-11-09", "Open": 555.45, "High": 562.58, "Low": 554.23, "Close": 562.51, "Volume": 2649900, "Adj. Close": 562.51}, {"Date": "2006-11-02", "Open": 467.5, "High": 473.73, "Low": 466.38, "Close": 469.91, "Volume": 5236700, "Adj. Close": 469.91}, {"Date": "2005-07-07", "Open": 289.39, "High": 295.8, "Low": 288.51, "Close": 295.54, "Volume": 10672100, "Adj. Close": 295.54}, {"Date": "2005-09-16", "Open": 304.02, "High": 304.5, "Low": 299.87, "Close": 300.2, "Volume": 7579800, "Adj. Close": 300.2}, {"Date": "2005-12-14", "Open": 417.04, "High": 419.73, "Low": 415.49, "Close": 418.96, "Volume": 6630400, "Adj. Close": 418.96}, {"Date": "2006-05-05", "Open": 397.6, "High": 400.68, "Low": 391.78, "Close": 394.3, "Volume": 6065000, "Adj. Close": 394.3}, {"Date": "2005-05-19", "Open": 240.34, "High": 241.17, "Low": 238.27, "Close": 239.18, "Volume": 9716500, "Adj. Close": 239.18}, {"Date": "2008-08-05", "Open": 467.59, "High": 480.08, "Low": 466.33, "Close": 479.85, "Volume": 3584500, "Adj. Close": 479.85}, {"Date": "2005-11-04", "Open": 389.98, "High": 391.79, "Low": 385.45, "Close": 390.43, "Volume": 8824900, "Adj. Close": 390.43}, {"Date": "2006-03-15", "Open": 350.77, "High": 352.3, "Low": 340.53, "Close": 344.5, "Volume": 12768800, "Adj. Close": 344.5}]student@student-virtual-machine:~/riak-1.2.1/dev$

```

Query 2: simple subset

A simple map-only job that returns the values for the first week of January 2010.

The first week of January in 2010 was the 4th-8th. We are querying a K-V DB, so our input must be the keys for the dates we want: "2010-01-04", "2010-01-05", "2010-01-06", "2010-01-07", "2010-01-08".

```

{"inputs":["goog","2010-01-04"],
  ["goog","2010-01-05"],
  ["goog","2010-01-06"],
  ["goog","2010-01-07"],
  ["goog","2010-01-08"]],
  "query":{"map":{"language":"javascript","name":"Riak.mapValuesJson","keep":true}}}
}

```

```

student@student-virtual-machine: ~/riak-1.2.1/dev
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -X POST http://127.0.0.1:8091/mapred -H "Content-Type: application/json" -d @-
{"inputs":["goog","2010-01-04"],
  ["goog","2010-01-05"],
  ["goog","2010-01-06"],
  ["goog","2010-01-07"],
  ["goog","2010-01-08"]],
  "query":{"map":{"language":"javascript","name":"Riak.mapValuesJson","keep":true}}}
}

[{"Date":"2010-01-04","Open":626.95,"High":629.51,"Low":624.24,"Close":626.75,"Volume":1956200,"Adj. Close":626.75},{"Date":"2010-01-05","Open":627.18,"High":627.84,"Low":621.54,"Close":623.99,"Volume":3004700,"Adj. Close":623.99},{"Date":"2010-01-06","Open":625.86,"High":625.86,"Low":606.36,"Close":608.26,"Volume":3978700,"Adj. Close":608.26},{"Date":"2010-01-07","Open":609.4,"High":610,"Low":592.65,"Close":594.1,"Volume":6414300,"Adj. Close":594.1},{"Date":"2010-01-08","Open":592,"High":603.25,"Low":589.11,"Close":602.02,"Volume":4724300,"Adj. Close":602.02}]student@student-virtual-machine:~/riak-1.2.1/dev$

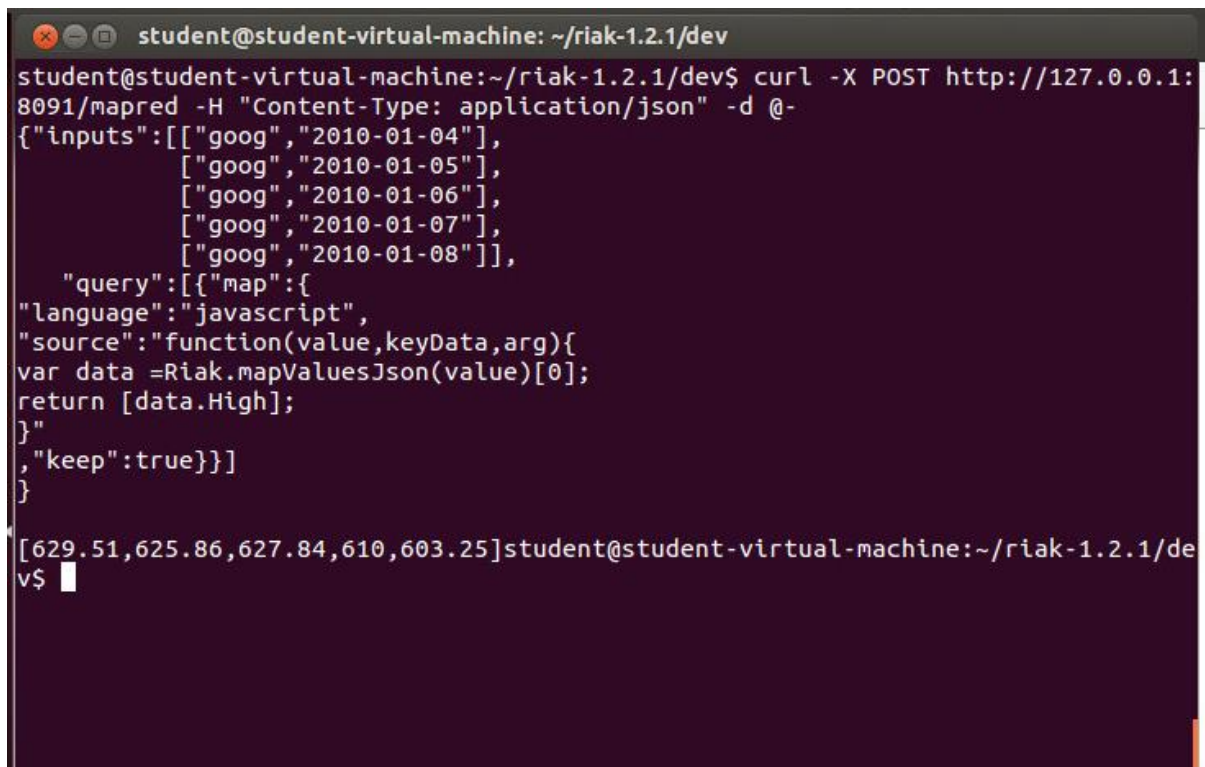
```


Query 3: map subset

A map-only job that returns the “High” sell values for the first week of January 2010.

For this, we will build on our query for the 1st week of January 2010. Our input will still be the keys for the dates, but now we’ll add a step to the map that will run a function to return the “High” sell values only.

```
{
  "inputs": [
    ["goog", "2010-01-04"],
    ["goog", "2010-01-05"],
    ["goog", "2010-01-06"],
    ["goog", "2010-01-07"],
    ["goog", "2010-01-08"]
  ],
  "query": [
    {
      "map": {
        "language": "javascript",
        "source": "function(value, keyData, arg) {
          var data = Riak.mapValuesJson(value)[0];
          return [data.High];
        }",
        "keep": true
      }
    }
  ]
}
```



A terminal window with a dark background and light text. The title bar shows 'student@student-virtual-machine: ~/riak-1.2.1/dev'. The command executed is 'curl -X POST http://127.0.0.1:8091/mapred -H "Content-Type: application/json" -d @-'. The input is a JSON object with 'inputs' and 'query' fields. The output is an array of five numbers: [629.51, 625.86, 627.84, 610, 603.25].

```
student@student-virtual-machine: ~/riak-1.2.1/dev
student@student-virtual-machine:~/riak-1.2.1/dev$ curl -X POST http://127.0.0.1:
8091/mapred -H "Content-Type: application/json" -d @-
{"inputs": [
  ["goog", "2010-01-04"],
  ["goog", "2010-01-05"],
  ["goog", "2010-01-06"],
  ["goog", "2010-01-07"],
  ["goog", "2010-01-08"]
],
"query": [
  {
    "map": {
      "language": "javascript",
      "source": "function(value, keyData, arg) {
        var data = Riak.mapValuesJson(value)[0];
        return [data.High];
      }",
      "keep": true
    }
  }
]
}
[629.51,625.86,627.84,610,603.25]student@student-virtual-machine:~/riak-1.2.1/de
v$
```

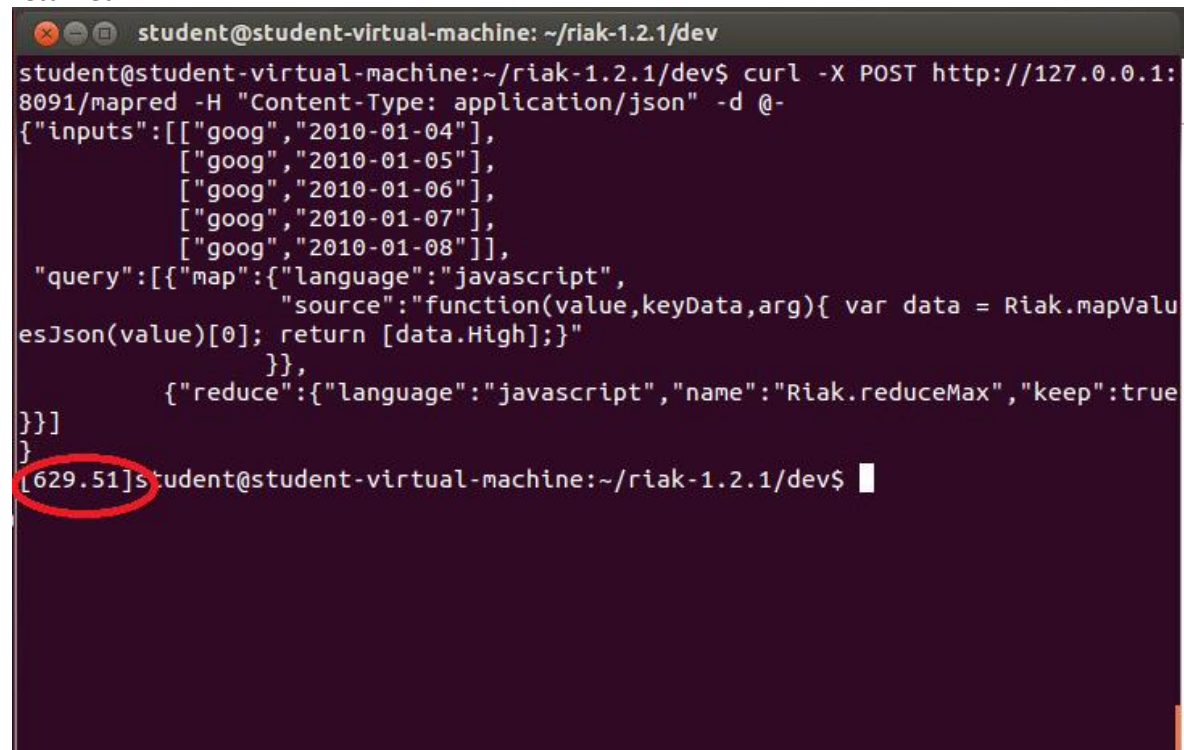
Query 4: A map-reduce query

Now we'll add reduce to create a job that returns the maximum high sell value in the first week of January. We used the built-in function `Riak.mapValuesJson` to map our data, now we'll use another built-in, `Riak.reduceMax`, to extract and return the highest "High" value:

```
{
  "inputs": [
    ["goog", "2010-01-04"],
    ["goog", "2010-01-05"],
    ["goog", "2010-01-06"],
    ["goog", "2010-01-07"],
    ["goog", "2010-01-08"]
  ],
  "query": {
    "map": {
      "language": "javascript",
      "source": "function(value, keyData, arg) {
        var data = Riak.mapValuesJson(value)[0];
        return [data.High];
      }"
    },
    "reduce": {
      "language": "javascript",
      "name": "Riak.reduceMax",
      "keep": true
    }
  }
}
```

Notice we moved the `"keep": true` down to our `"reduce"` so that the intermediate step results from the map phase are not displayed.

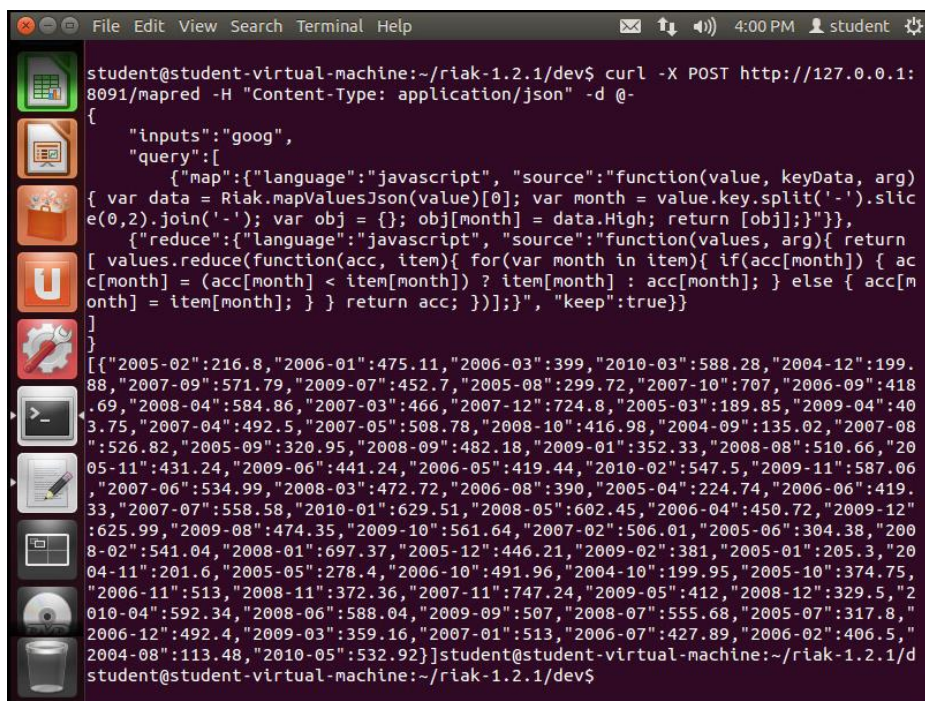
When we run this, we only get the maximum of our "High" values from the first week in January, 2010 returned.

A terminal window with a dark background. The title bar shows 'student@student-virtual-machine: ~/riak-1.2.1/dev'. The command prompt is 'student@student-virtual-machine:~/riak-1.2.1/dev\$'. The command entered is 'curl -X POST http://127.0.0.1:8091/mapred -H "Content-Type: application/json" -d @-'. The output is a JSON object, identical to the one in the previous code block. The final line of the output, '[629.51]', is circled in red. The prompt after the output is 'student@student-virtual-machine:~/riak-1.2.1/dev\$'.

Query 5: A complex map-reduce query

Now, let's extend this to create a more complicated map-reduce job that requires both a complex map and a complex reduce. Our goal is to create a query that will collect the max highs by month.

```
{ "inputs": "goog",
  "query": [
    { "map": { "language": "javascript",
      "source": "function(value, keyData, arg){
        var data = Riak.mapValuesJson(value)[0];
        var month = value.key.split('-').slice(0,2).join('-');
        var obj = {};
        obj[month] = data.High; return [obj]; }"},
    { "reduce": { "language": "javascript",
      "source": "function(values, arg){
        return [ values.reduce(function(acc, item){
          for(var month in item){
            if(acc[month]) {
              acc[month] = (acc[month] < item[month]) ? item[month] : acc[month]; }
            else { acc[month] = item[month]; }
          }
          return acc; }]); }, "keep": true}"} ]
  }, "keep": true } }
```



The screenshot shows a terminal window with a dark background. The prompt is `student@student-virtual-machine:~/riak-1.2.1/dev$`. The command executed is `curl -X POST http://127.0.0.1:8091/mapred -H "Content-Type: application/json" -d @-`. The output is a large JSON array of month-year pairs and their corresponding 'High' values, such as `[{"2005-02":216.8,"2006-01":475.11,"2006-03":399,"2010-03":588.28,"2004-12":199.88,"2007-09":571.79,"2009-07":452.7,"2005-08":299.72,"2007-10":707,"2006-09":418.69,"2008-04":584.86,"2007-03":466,"2007-12":724.8,"2005-03":189.85,"2009-04":403.75,"2007-04":492.5,"2007-05":508.78,"2008-10":416.98,"2004-09":135.02,"2007-08":526.82,"2005-09":320.95,"2008-09":482.18,"2009-01":352.33,"2008-08":510.66,"2005-11":431.24,"2009-06":441.24,"2006-05":419.44,"2010-02":547.5,"2009-11":587.06,"2007-06":534.99,"2008-03":472.72,"2006-08":390,"2005-04":224.74,"2006-06":419.33,"2007-07":558.58,"2010-01":629.51,"2008-05":602.45,"2006-04":450.72,"2009-12":625.99,"2009-08":474.35,"2009-10":561.64,"2007-02":506.01,"2005-06":304.38,"2008-02":541.04,"2008-01":697.37,"2005-12":446.21,"2009-02":381,"2005-01":205.3,"2004-11":201.6,"2005-05":278.4,"2006-10":491.96,"2004-10":199.95,"2005-10":374.75,"2006-11":513,"2008-11":372.36,"2007-11":747.24,"2009-05":412,"2008-12":329.5,"2010-04":592.34,"2008-06":588.04,"2009-09":507,"2008-07":555.68,"2005-07":317.8,"2006-12":492.4,"2009-03":359.16,"2007-01":513,"2006-07":427.89,"2006-02":406.5,"2004-08":113.48,"2010-05":532.92}]`. The terminal window has a sidebar with icons for various applications and a top bar with file management and system status icons.

One thing to notice about the returned dataset – it is not in order. RIAK mapped the input to the available servers and the results were returned to the coordinating node to aggregate and pass to the reduce step as they became available. When your data set becomes large, it will be returned as it is processed (almost guaranteeing that it will not be in-order). If you need data in some sorted order, you would need to use secondary indexes and code them into your reduce phase.

Writing compound/complex queries for RIAK, like the example above, is beyond the scope of this course. But the example above is used to illustrate how such queries are written and what the results returned will look like. Most RIAK applications would then pipe these results to web pages or web apps to clean up their presentation.

CONTINUE TO NEXT PAGE FOR WORK TO SUBMIT

Work to Submit

The script lab2b/lab2b.sh with the loading of the data and the return of the entire dataset will be provided in your repo as an example, along with the goog.csv and the load_data.erl files. Append your commands for tasks 1 and 2 to lab2b/lab2b.sh.

Be sure to do a 'git pull' to sync your repo.

Note: The erlang file on the repo is slightly different than the file on <http://csc570e.uis.edu/csc561> due to the hostname difference.

1. Write a map-only query for our google data that finds all the days where the low was less than \$450.00.

The map phase function you are going to want to use is

```
function(value, keyData, arg) {  
    var data = Riak.mapValuesJson(value)[0];  
    if(data.Low && data.Low < 450.00)  
        return [value.key];  
    else  
        return [];  
}
```

2. Write a map-only query for our google data that finds all of the days where the close was lower than the open. (You will need to write your own map phase function. You can modify the one given for 1.)
3. To submit your lab, push a lab2b.sh script with the appropriate curl commands to the drone Lab2b folder.