1. Mapreduce

I will be using a standalone riak server on 8098 for the mapreduce exercises

Stop ring servers ./riak stop in each /bin folder

Start standalone server from rel/riak/bin

*A quick populator script in Ruby installed 10,000-room hotel.*

*STYLES = single double queen king suite*

*bucket = rooms*

*ro.data = {'style' => style, 'capacity' => capacity}*

*)*

Let’s create mapreduce functions for our Riak dataset.

A neat feature of Riak’s mapreduce is that you can run the map() function alone and see what all the results are mid-run (assuming you even want to run a reduce).

Let’s look at the results for rooms 101, 102, and 103 only.

The map setting needs the language we’re using and the source code; only then do we actually write the JavaScript map function (the function is just a string, so we always need to escape any characters accordingly).

Using the @- command in cURL keeps the console’s standard input open until receiving CTRL+D.

This data will populate the HTTP body sent to the URL, which we post to the /mapred command (look carefully—the URL is /mapred, not /riak/mapred).

 In the next commands paste the first line and hit enter, then the code, then finally ctrl-d

**curl -X POST -H "content-type:application/json"** [**http://localhost:8098/mapred**](http://localhost:8098/mapred) **--data @-**

{

"inputs":[  
 ["rooms","101"],["rooms","102"],["rooms","103"],["rooms","104"],["rooms","105"]

], "query":[

{"map":{  
 "language":"javascript",  
 "source":

"function(v) {

/\* From the Riak object, pull data and parse it as JSON \*/

var parsed\_data = JSON.parse(v.values[0].data);

var data = {};

/\* Key capacity number by room style string \*/

**data[parsed\_data.style] = parsed\_data.capacity;**

return [data];

}" }}

] }

CTRL-D

1. *Stored Functions*

Remember to use <your initials>my\_functions/

**curl -X PUT -H "content-type:application/json"** [**http://localhost:8098/riak/pggmy\_functions/map\_capacity**](http://localhost:8098/riak/pggmy_functions/map_capacity) **--data @-**

function(v) {

var parsed\_data = JSON.parse(v.values[0].data);

var data = {};

**data[parsed\_data.style] = parsed\_data.capacity;**

return [data];

}

Ctrl D

With your function safely stored, we’ll run the function by pointing to the new bucket and key containing the function.

**curl -X POST -H "content-type:application/json"** [**http://localhost:8098/mapred**](http://localhost:8098/mapred) **--data @-**

{

"inputs":[  
 ["rooms","101"],["rooms","102"],["rooms","103"]

], "query":[

{"map":{  
 "language":"javascript",  
 "bucket":"pggmy\_functions",  
 "key":"map\_capacity"

}} ]

}

Ctrl D

10 Reduce

This time, we add the reduce function.

**curl -X POST -H "content-type:application/json"** [**http://localhost:8098/mapred**](http://localhost:8098/mapred) **--data @-**

{

"inputs":"rooms",  
 "query":[

{"map":{  
 "language":"javascript",  
 "bucket":"pggmy\_functions",  
 "key":"map\_capacity"

}},  
 {"reduce":{

"language":"javascript",  
 "source":

"function(v) {  
 var totals = {};  
 for (var i in v) {

for(var style in v[i]) {  
 if( totals[style] )

totals[style] += v[i][style];

else

totals[style] = v[i][style];

} }

return [totals];  
 }"

}} ]

}

Ctrl D

Running this on all rooms should return total counts of capacity, keyed by room style