Using MapReduce

Contents

1. Introduction
2. Features
3. When to Use MapReduce
4. When Not to Use MapReduce
5. How it Works
6. Examples
7. Further Reading

Introduction

MapReduce (M/R) is a technique for dividing work across a distributed system. This 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 lives, 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. **Riak MapReduce is intended for batch processing, not real time querying.**

Features

- Map phases execute in parallel with data locality
- Reduce phases execute in parallel on the node where the job was submitted

- Javascript MapReduce support
- Erlang MapReduce support

When to Use MapReduce

- 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, as do Search and Secondary Indexes
- 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

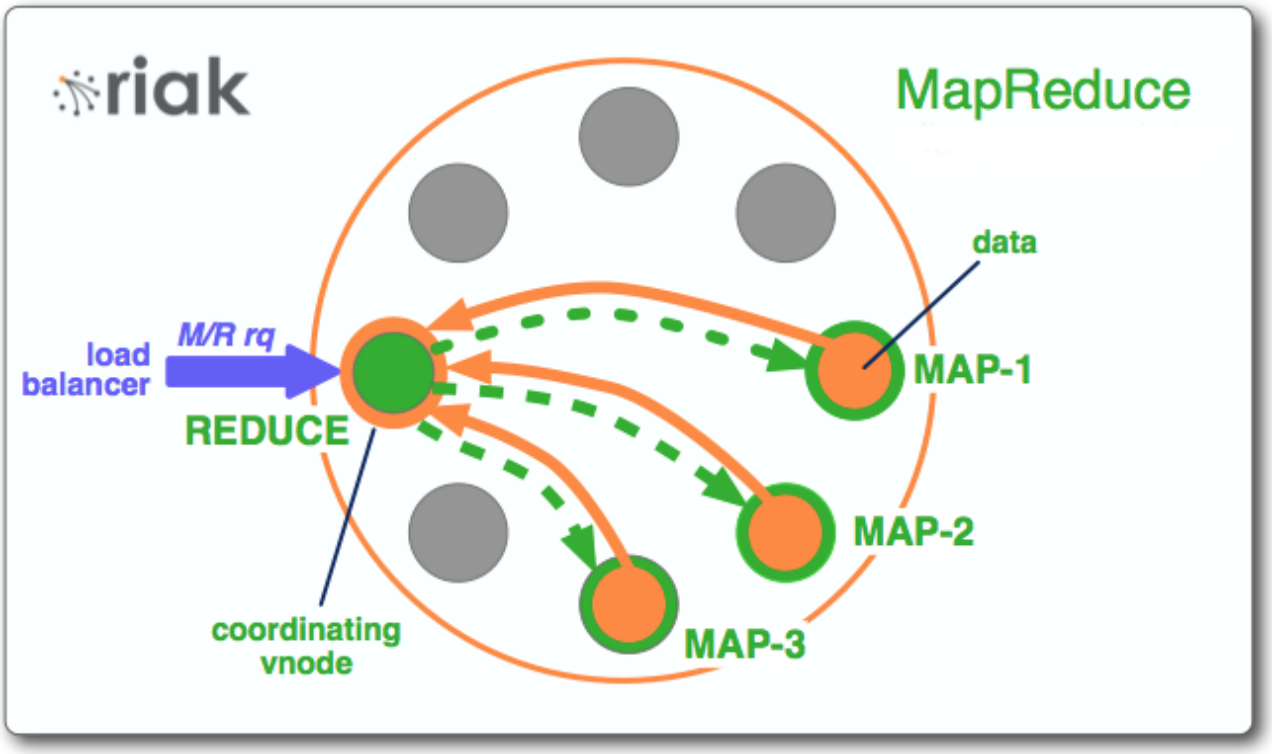
- When you want to query data of an entire bucket. MapReduce uses a list of keys, which can place a lot of demand on the cluster.
- When you want latency to be as predictable as possible.

How it 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:

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



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).

Examples

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”.

Data object input commands:

HTTP

```
$ curl -XPUT http://localhost:8098/buckets/training/keys/foo
$ curl -XPUT http://localhost:8098/buckets/training/keys/bar
$ curl -XPUT http://localhost:8098/buckets/training/keys/baz
$ curl -XPUT http://localhost:8098/buckets/training/keys/bam
```

MapReduce script and deployment:

HTTP

```
curl -XPOST http://localhost:8098/mapred \
  -H 'Content-Type: application/json' \
  -d '{
    "inputs":"training",
    "query":[{"map":{"language":"javascript",
    "source":"function(riakObject) {
      var val = riakObject.values[0].data.match(/pizza/g);
      return [[riakObject.key, (val ? val.length : 0 )]];
    }"}}]}'
```

Output

The output is the key of each object, followed by the count of the word “pizza” for

that object. It looks like:

```
Text
[["foo",1],["baz",0],["bar",4],["bam",3]]
```

Recap

We run a Javascript MapReduce function against the `training` bucket, which takes each `riakObject` (a JavaScript representation of a key/value) and searches the text for the word "pizza". `val` is the result of the search, which includes zero or more regular expression matches. The function then returns the `key` of the `riakObject` along with the number of matches.

Further Reading

- Advanced MapReduce: Details on Riak's implementation of MapReduce, different ways to run queries, examples, and configuration details
- Using Key Filters: Pre-processing MapReduce inputs from a full bucket query by examining the key

These May Also Interest You

- Five-Minute Install
- Advanced Secondary Indexes
- Replication Properties
- Advanced Commit Hooks
- Advanced MapReduce
- Advanced Search