Riak MapReduce Settings

Contents

1. Configuring MapReduce
2. Configuration Tuning for Javascript
3. Configuration for Riak 1.0
4. Configuration Tuning for Reduce Phases

Configuring MapReduce

MapReduce (M/R) is always enabled, but configurable through theapp.config file as follows under `riak_kv`

Erlang

```
{riak_kv, [
```

`mapred_name` is the URL directory used to submit M/R requests to Riak. By default `mapred`, making the command path, for example: `http://localhost:8091/mapred`

Erlang

```
    {mapred_name, "mapred"},
```

`mapred_2i_pipe` indicates whether 2i MapReduce inputs are queued in parallel in their own pipe (`true`), or serially through a helper process (`false` or undefined).

***Note**: Set to `false` or leave undefined during a rolling upgrade from 1.0.*

Erlang

```
    {mapred_2i_pipe, true},
```

Each of these entries control how many Javascript virtual machines are available for executing map, reduce, pre- and post-commit hook functions.

This is largely relevant only if you are writing JavaScript M/R jobs.

Erlang

```
        {map_js_vm_count, 8 },
        {reduce_js_vm_count, 6 },
        {hook_js_vm_count, 2 },
```

`js_max_vm_mem` is the maximum amount of memory, in megabytes, allocated to the Javascript VMs. If unset, the default is 8MB.

This is largely relevant only if you are writing JavaScript M/R jobs.

Erlang

```
        {js_max_vm_mem, 8},
```

`js_thread_stack` is the maximum amount of thread stack, in megabytes, allocated to the Javascript VMs. If unset, the default is 16MB.

***Note**: This is not the same as the C thread stack.*

Erlang

```
        {js_thread_stack, 16},
```

`js_source_dir` should point to a directory containing Javascript source files which will be loaded when Riak initializes Javascript VMs.

Erlang

```
        %{js_source_dir, "/tmp/js_source"},
```

Configuration Tuning for Javascript

If you load larger JSON objects in your buckets there is a possibility you might encounter an error like the following:

Json

```
{"lineno":465,"message":"InternalError: script stack space
```

You can increase the amount of memory allocated to the Javascript VM stack by editing your app.config. The following will increase the stack size from 8MB to 32MB:

Erlang

```
{js_thread_stack, 8}
```

becomes

Erlang

```
{js_thread_stack, 32},
```

In addition to increasing the amount of memory allocated to the stack you can increase the heap size as well by increasing the `js_max_vm_mem` from the default of 8MB. If you are collecting a large amount of results in a reduce phase you may need to increase this setting.

Configuration for Riak 1.0

Riak 1.0 is the first release including the new MapReduce subsystem known as Riak Pipe. By default, new Riak clusters will use Riak Pipe to power their MapReduce queries. Existing Riak clusters that are upgraded to Riak 1.0 will continue to use the legacy MapReduce system unless the following line is added to the riak_kv section of each node's app.config:

Erlang

```
%% Use Riak Pipe to power MapReduce queries
{mapred_system, pipe},
```

> Warning: Do not enable Riak Pipe for MapReduce processing until all nodes in the cluster are running Riak 1.0.

Other than speed and stability of the cluster, the choice of MapReduce subsystem (Riak Pipe or legacy) should be invisible to your client. All queries should have the same syntax and return the same results on Riak 1.0 with Riak Pipe as they did on earlier versions with the legacy subsystem. If you should find a case where this is not true, you may revert to using the legacy subsystem by either removing the aforementioned line in your app.config or by changing it to read like this:

Erlang

```
%% Use the legacy MapReduce system
{mapred_system, legacy},
```

Configuration Tuning for Reduce Phases

If you are using Riak 1.0 and the Riak Pipe subsystem for MapReduce queries, you have additional options for tuning your reduce phases.

Batch Size

By default, Riak will evaluate a reduce function every time its phase receives 20 new inputs. If your reduce phases would run more efficiently with more or fewer new inputs, you may change this default by adding the following to the riak_kv section of your app.config:

Erlang

```
%% Run reduce functions after 100 new inputs are received
{mapred_reduce_phase_batch_size, 100},
```

You may also control this batching behavior on a per-query basis by using the static argument of the phase specification. When specifying phases over HTTP, the JSON configuration for evaluating the function after 150 new inputs looks like this:

Json

```
{"reduce":
  {...language, etc. as usual...
   "arg":{"reduce_phase_batch_size":150}}}
```

In Erlang, you may either specify a similar mochijson2 structure for the phase argument, or use the simpler proplist form:

Erlang

```
{reduce, FunSpec, [{reduce_phase_batch_size, 150}], Keep}
```

Finally, if you want your reduce function to be evaluated only once, after all inputs are received, use this argument instead:

Json

```
{"reduce":
  {...language, etc. as usual...
   "arg":{"reduce_phase_only_1":true}}}
```

Similarly, in Erlang:

Erlang

```
{reduce, FunSpec, [reduce_phase_only_1], Keep}
```

Warning: A known bug in Riak 1.0.0 means that it is possible a reduce function may run more often than specified if handoff happens while the phase is accumulating inputs. This bug was fixed in 1.0.1.

Pre-Reduce

If your reduce functions can benefit from parallel execution, it is possible to request that the outputs of a preceding map phase be reduced local to the partition that produced them, before being sent, as usual, to the final aggregate reduce.

Pre-reduce is disabled by default. To enable it for all reduce phases by default, add the following to the riak_kv section of your app.config:

Erlang

```
%% Always pre-reduce between map and reduce phases
{mapred_always_prereduce, true}
```

Pre-reduce may also be enabled or disabled on a per-phase basis via the Erlang API for map phases implemented in Erlang. To enable pre-reduce, for any map phase followed by a reduce phase, pass a proplist as its static phase argument and include the following flag:

Erlang

```
{map, FunSpec, [do_prereduce], Keep}
```

Warning: A known bug in Riak 1.0.0 prevents per-phase pre-reduce from being enabled over HTTP. This bug also prevents per-phase pre-reduce from being enabled for Javascript phases. Use the global app.config flag for these cases. This bug was fixed in 1.0.1.

These May Also Interest You

- Advanced MapReduce
- Key Filters Reference
- Using Key Filters
- Using MapReduce