1. Use AWS cloud formation templates with
   Instance types:
   - i2-2xlarge for private agents
   - m3-2xlarge for public agents

   Use 8 private agents and 4 public agents for now.
2. ssh into master and edit:
   - `/opt/mesosphere/etc/mesos-master`
   - and change
   - `MESOS_ROLES=slave_public`
   - `MESOS_WEIGHTS=slave_public=1`

     into
   - `MESOS_ROLES=slave_public,arangodb`
   - `MESOS_WEIGHTS=slave_public=1,arangodb=1`

     Then simply kill the mesos-master process, it will restart automatically.
3. Open
   - the Mesosphere dash board (<master-url>/)
   - the Mesos dash board (<master-url>/mesos)
   - the Marathon dash board (<master-url>/marathon)

4. Install `dcos` cli on your laptop (follow instructions on the Mesosphere dash board)
5. Deploy ArangoDB cluster with:

   `dcos package install --options=config.json arangodb`

   with the following file `config.json`:

```
{ "arangodb": {
    "async-replication": true,
    "nr-dbservers": 8,
    "nr-coordinators": 8,
    "framework-cpus": 0.5,
    "role": "arangodb",
    "principal": "pri",
    "minimal-resources-agent":
"mem(*):512;cpus(*):0.5;disk(*):512",
    "minimal-resources-dbserver":
"mem(*):8192;cpus(*):3;disk(*):8192",
    "minimal-resources-secondary":
"mem(*):8192;cpus(*):1;disk(*):8192",
    "minimal-resources-coordinator":
"mem(*):8192;cpus(*):3;disk(*):8192",
    "secondaries-with-dbservers":true,
    "docker-image": "arangodb/arangodb-mesos:devel"
  }
}
```

6. Find out the endpoints (internal IP addresses) of the coordinator tasks:

`dcos arangodb endpoints`

which gives something like

```
URL of ArangoDB web frontend:
  http://52.24.224.199/service/arangodb/
Coordinators running on:
  http://10.0.2.24:1027
  http://10.0.2.25:1027
  http://10.0.3.125:1027
  http://10.0.3.127:1027
  http://10.0.2.23:1028
  http://10.0.2.26:1027
  http://10.0.3.126:1027
  http://10.0.2.27:1027
```

Edit the list of coordinators in `doit.js`, the corresponding section must look like:

```
var coordinators = [
  "http://10.0.2.24:1027",
  "http://10.0.2.25:1027",
  "http://10.0.3.125:1027",
  "http://10.0.3.127:1027",
  "http://10.0.2.23:1028",
  "http://10.0.2.26:1027",
  "http://10.0.3.126:1027",
  "http://10.0.2.27:1027"
];
```

7. Deploy load servers

```
curl -X POST <master-url>/service/marathon/v2/apps -d @load.json \
        -H "Content-type: application/json"
```

with file `load.json`:

```
{
  "id": "loadserver",
  "cpus": 7.5,
  "mem": 8192.0,
```

```
  "ports": [],
  "requirePorts": false,
  "instances": 4,
  "args": [],
  "env": {},
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "neunhoef/waiter",
      "network": "HOST",
      "forcePullImage": true
    }
  },
  "acceptedResourceRoles": [
    "slave_public"
  ]
}
```

This will deploy 4 load servers to the public slaves.

8. I use a single ArangoDB instance somewhere, whose endpoint is hard-wired into the `doit.js` script:

```
var jobnr = 0;
var o = { _key: "job",
          TIME: 30,
          COMPLEXITY: 20,
          RESULT_URL:
"http://104.155.62.222:8529/_api/document?collection=results",
          NRSHARDS: 8,
          clients: 8,
          job: ""+jobnr};

var coordinators = [
  "http://10.0.2.24:1027",
  "http://10.0.2.25:1027",
  "http://10.0.3.125:1027",
  "http://10.0.3.127:1027",
  "http://10.0.2.23:1028",
  "http://10.0.2.26:1027",
  "http://10.0.3.126:1027",
  "http://10.0.2.27:1027"
];

var print = require("internal").print;
```

```
var wait = require("internal").wait;
var time = require("internal").time;

var hosts = [];
var ports = [];

for (var c in coordinators) {
  cc = coordinators[c];
  var pos = cc.indexOf("http://");
  if (pos === 0) {
    pos = cc.indexOf(":", 7);
    if (pos !== -1) {
      hosts.push(cc.substr(7,pos-7));
      ports.push(cc.substr(pos+1));
    }
  }
}

hosts = "(" + hosts.join(" ") + ")";
ports = "(" + ports.join(" ") + ")";

print("Hosts: "+hosts)
print("Ports: "+ports)

o.HOSTS = hosts;
o.PORTS = ports;

db = require("internal").db;
work = db.work;
done = db.done;
work.truncate();
done.truncate();

function init () {
  jobnr += 1
  o.job = "JOB"+jobnr
  o.name = "INIT";
  o.multiple = 0;
  o.READPERCENTS = 0;
  work.insert(o);
  print("Submitted init job");
  var t = time();
  while (done.count() !== 1) {
    print("Not finished.");
    wait(5);
    if (time() - t > 300) {
      print("Aborting!");
```

```
      break;
    }
  }
  work.truncate();
  print("Result:"+JSON.stringify(done.any()));
  done.truncate();
}

function helper (name, multiple, readpercents) {
  o.name = "" + o.clients + "_" + name + "_" + multiple;
  jobnr += 1
  o.job = "JOB"+jobnr
  print("Doing "+o.name+"...");
  o.multiple = multiple;
  o.READPERCENTS = readpercents;
  work.insert(o);
  print("Submitted job "+o.job);
  var t = time();
  var ok = true;
  while (done.count() < o.clients) {
    print("Not finished, have " + done.count() + " answers.");
    wait(3);
    if (time() - t > 60) {
      print("Aborting!");
      ok = false;
      break;
    }
  }
  work.truncate();
  print("Result:"+JSON.stringify(done.toArray()));
  print("Have all "+done.count()+" answers.");
  done.truncate();
  return ok;
}

function doit (from, limit) {
  helper("WARMUP", 1, 100);

  for (var i = from; i <= limit; i++) {
    if (! helper("RD", i, 100)) {
      return;
    }
  }

  for (var i = from; i <= limit; i++) {
    if (! helper("WR", i, 0)) {
      return;
```

```
    }
  }

  for (var i = from; i <= limit; i++) {
    if (! helper("RW", i, 50)) {
      return;
    }
  }
}
```

We adjust die number of clients (= number of coordinators), the number of shards (number of ArangoDB nodes).

9. Attach an `arangosh` to the single ArangoDB server via

```
arangosh --server.endpoint tcp://104.155.62.222:8529
```

and do

```
require("internal").load("doit.js")
init()
doit(20, 21)
```

20 is the minimum number of connections per load server to try to each coordinator and 21 is the maximum. For example, with 20 (and 4 load servers and 8 coordinators), each coordinator will get 80=4*20 incoming connections and each load server will thus open 160=20*8 outgoing connections. The optimal number is 80-100 incoming connections per coordinator, since we run 60 worker threads.

10. Evaluate the results with this script called `eval.js`:

```
function sum(l) {
  var s = 0,i;
  for (i = 0; i < l.length; i++) {
    s += l[i];
  }
  return s;
}

function avg(l) {
  return sum(l)/l.length;
}
```

```
l = [];
clients=8;
from=20
k=21
for (i = from; i <= k; i++) {
  l.push(clients+"_RD_"+i);
}
for (i = from; i <= k; i++) {
  l.push(clients+"_WR_"+i);
};
for (i = from ; i <= k; i++) {
  l.push(clients+"_RW_"+i);
}
l

ll = {}
for (x in l) {
  ll[l[x]] = {
    read: sum(db._query('FOR r IN results FILTER r.testName ==
"arangoReadWriteCluster_'+l[x]+'" RETURN
r.nrReads').toArray())/30.0,
    write: sum(db._query('FOR r IN results FILTER r.testName ==
"arangoReadWriteCluster_'+l[x]+'" RETURN
r.nrWrites').toArray())/30.0,
    latency: avg(db._query('FOR r IN results FILTER r.testName ==
"arangoReadWriteCluster_'+l[x]+'" RETURN r.median').toArray()),
    number: db._query('FOR r IN results FILTER r.testName ==
"arangoReadWriteCluster_'+l[x]+'" RETURN
r.median').toArray().length,
    errors: sum(db._query('FOR r IN results FILTER r.testName ==
"arangoReadWriteCluster_'+l[x]+'" RETURN r.nrErrors').toArray())
  };
}
ll
```

```
arangosh --server.endpoint tcp://104.155.62.222:8529
```

and do

```
require("internal").load("eval.js")
```

adjust the clients and from and k numbers.