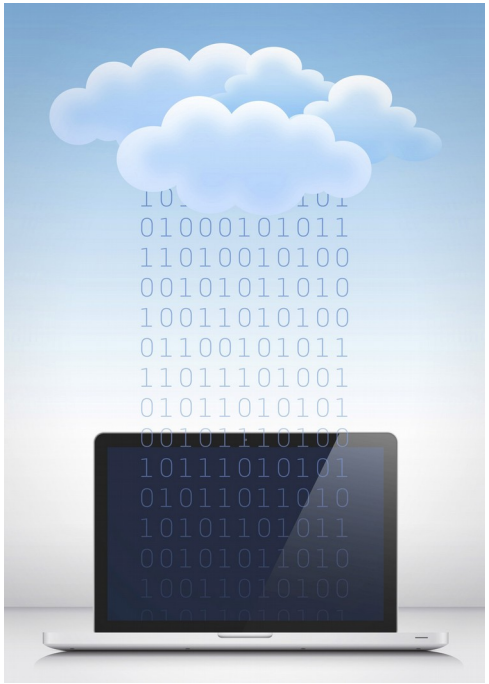
MONGOOSE 2.0

WHAT IS MONGOOSE

Mongoose is a benchmarking tool initially designed for cloud storage performance testing



TOP 10 FEATURES

1. Easily Scaleable to up to Million Of Connections
2. Distributed Mode
3. Reporting:
   - *Item lists for reusing*
   - *Statistics for the rates and timings*
   - *High-resolution timings for each operation*
4. Custom Content Generation
5. Unlimited Content Updating ability
6. Content Verification
7. Cloud storage support:
   - *Amazon S3*
   - *EMC Atmos*
   - *OpenStack Swift*
8. Filesystem Operations Support
9. Web GUI
10. Docker Integration

CURRENT USABILITY ISSUES

- Not enough flexible — *requires Java programming to implement a custom scenario*
- Error-prone and complicated scenario configuration — *the usual way to run from the CLI looks like*:

```
java
-Dload.server.addrs=10.248.236.69,10.248.236.68,10.248.236.67,10.2
48.236.66
-Dstorage.addrs=10.247.235.65,10.247.235.64,10.247.235.63,10.247.2
35.62 -Dload.threads=100 -Ddata.size=16MB
-Drun.id=mySimpleReadTest1 -Dapi.type.s3.bucket=bucket1
-Dscenario.type.single.load=read -Ditem.src.fpath=mongoose-
1.4.0/log/mySimpleWriteTest1/items.csv -jar mongoose-
1.4.0/mongoose.jar client
```

NEW REQUIREMENTS

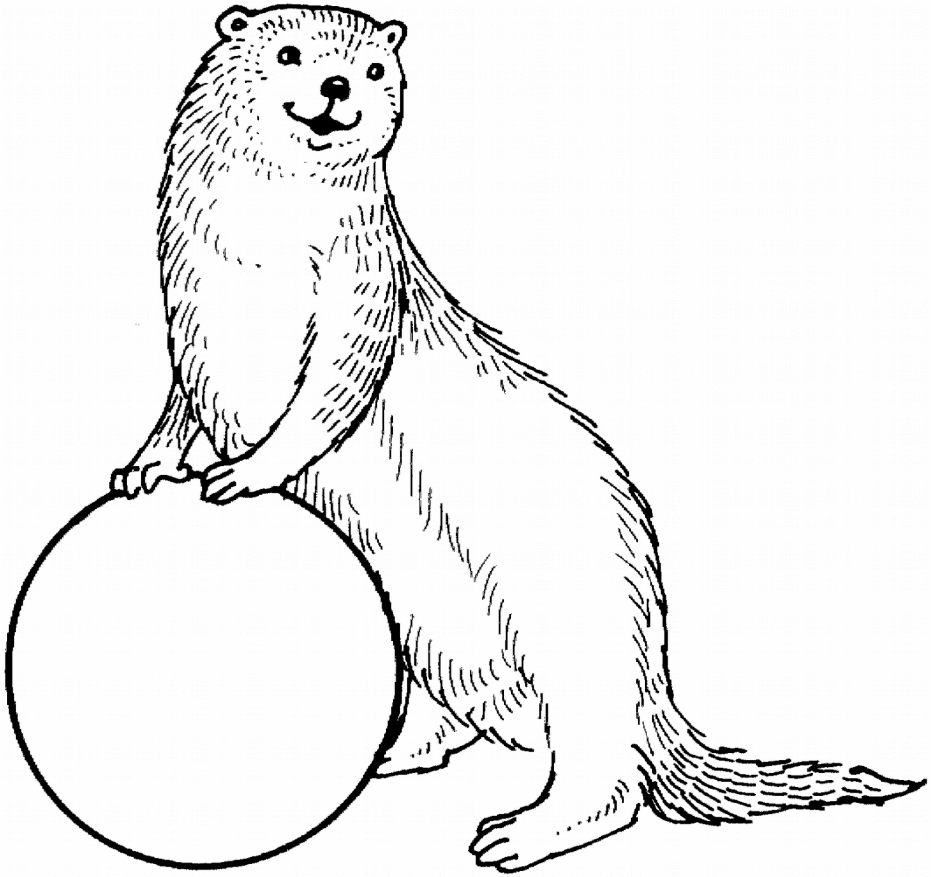
- Make the tool “scriptable”
- Make the tool use-case oriented
- Support various mixed load cases
- Support weighted load case
- Include rich set of example scenarios into the distribution

THE APPROACH

- Use JSON scenario files on input
- Aggregate the load jobs into the job containers
- Make job containers executable in parallel and sequentially
- Make job containers and single load jobs configurable individually

TRADE-OFFS

- No backward compatibility with 1.x versions
- More general ”WRD” load type notation instead of ”CRUD”



HOW TO RUN A SCENARIO FILE

- Specify the scenario file as a CLI argument:

```
java [<DEFAULTS_OVERRIDING>] -jar mongoose.jar [<MODE>] -f
<PATH/TO/SCENARIO.json>
```

- Or pass the scenario content on the standard input:

```
cat <PATH/TO/SCENARIO.json> | java [<DEFAULTS_OVERRIDING>]
-jar mongoose.jar [<MODE>]
```

THE SCENARIO FILE OVERVIEW

The root node is always a job container:

```
{
    "type" : <JOB_CONTAINER_TYPE>
    ...
    ...
}
```

JOB CONTAINER TYPES

Load	• Single load job • Cannot include another job containers
Rampup	• Multiple load jobs • Cannot include another job containers
Parallel	• Should include other job containers • Nested job containers are executed *in parallel*
Sequential	• Should include other job containers • Nested job containers are executed *sequentially*
Sleep	Pause for some specified time

JOB CONTAINER CONFIGURATION

- Any job container can contain an optional "config" node:

```
{
    "type" : <JOB_CONTAINER_TYPE>
    "config" : {
        // here are the configuration hierarchy
    }
}
```

- The layout of the "config" subtree is the same as for default configuration

CONFIGURATION EXAMPLE

```
{
    "type" : "load"
    "config" : {
        "storage" : {
            "addrs" : [
                "192.168.0.1", "192.168.0.2", "192.168.0.3"
            ]
        }
    }
}
```

SEQUENTIAL JOBS EXECUTION EXAMPLE

```
{
    "type" : "sequential"
    "jobs" : [
        {
            // 1st job container
        }, {
            // 2nd job container
        }
    ]
}
```

PARALLEL JOBS EXECUTION EXAMPLE

```
{
    "type" : "parallel"
    "jobs" : [
        {
            // 1st job container
        }, {
            // 2nd job container
        }
    ]
}
```

SLEEP JOB EXAMPLE

```
...
"jobs" : [
     {
          ...
     }, {
          "type" : "sleep",
          "value" : "10m"
     }, {
          ...
     }
]
```

HELLO WORLD SCENARIO EXAMPLE

```
{
    "type" : "load"
}
```

Will use the default configuration values:

- Use S3 API and port 9020
- Use 1 connection to the default single node @ 127.0.0.1
- Use *Write* load type
- No load limits (infinite load job)
- Use 1MB as the size of the data items
- Use a container (bucket) created automatically

WEIGHTED LOAD EXAMPLE

```
1.        "type" : "load",
2.        "config" : {
3.           "item" : {
4.              "src" : {
5.                 "file" : [
6.                    null,
7.                    "/tmp/precreated-items-list.csv"
8.                 ]
9.              }
10.          },
11.          "load" : {
12.             "type" : [
13.                "write=20%", "read=80%"
14.             ]
15.          }
16.       }
```

- Performs both write and read operations
- 20% of operations are Write ones
- 80% of operations are Read ones

WHERE TO GO NEXT

1. Refer to the Mongoose wiki for the *configuration layout* and the detailed scripting engine specification
2. Much *more example scenarios* is available in the Mongoose distribution
3. Ask via email *Mongoose.Support@emc.com*

THE ROADMAP FOR 2.X

- Copy mode for the Write load type
- Local Results Persistence
- GUI Enhancements
- Partial Read
- Centera API Support

Q & A

Thank you


EMC²
®