MONGOOSE 2.0

WHAT IS MONGOOSE

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

- 1M of concurrent connections
- 1M of operations per second
- 1M of items which may be processed multiple times in the circular load mode
- 1M of items which may be stored in the storage mock

TOP FEATURES

1. Distributed Mode
2. Rich Metrics Reporting
3. Different operation types

   (Create, Update, Append, Read, Delete)
4. Abstract Load Engine

   (load with objects, files, containers, directories, etc)
5. Cloud Storages support

   (S3, Atmos, Swift)
6. Flexible Load Limitation

   (by count, time, rate)
7. Custom Content Generation and Verification
8. Circular Load Mode
9. Dynamic Configuration Parameters
10. Custom Item Naming Schemes

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=10.248.236.69,10.248.236.68,10.248.236.67,10.248.
236.66
-Dstorage.addrs=10.247.235.65,10.247.235.64,10.247.235.63,10.247.235.
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

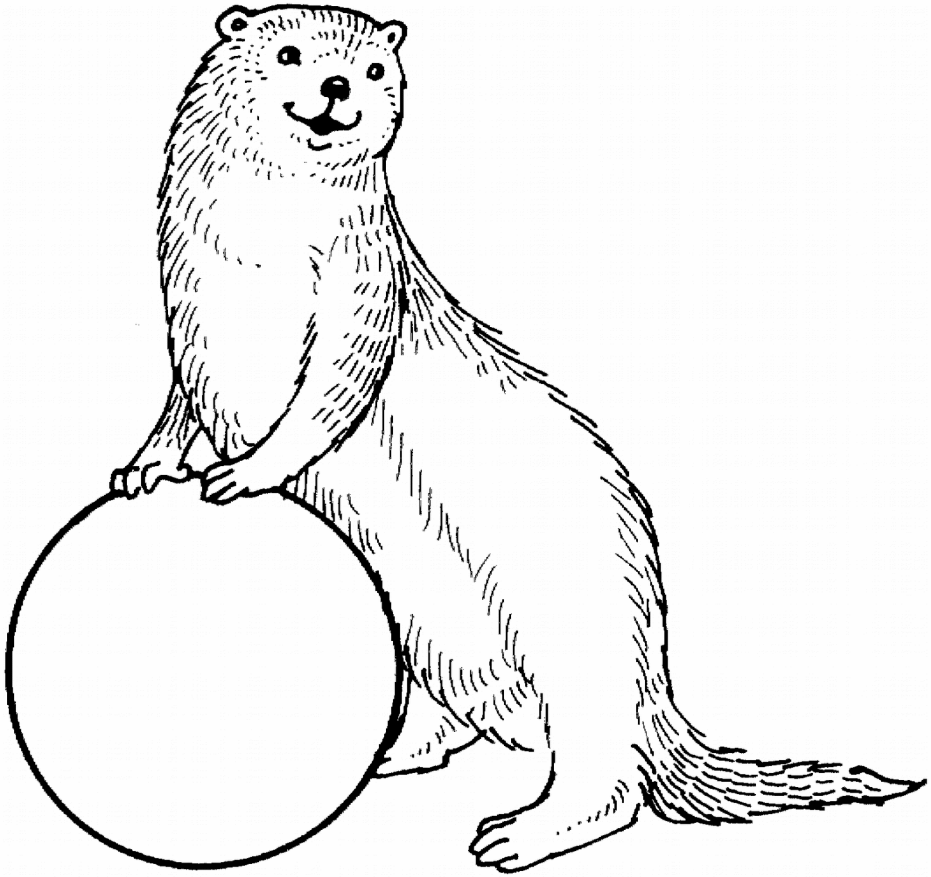
- Be able to execute custom scenarios
- Make the tool use-case oriented
- Support various mixed load cases
- Support weighted load case
- Include rich set of example scenarios into the distribution

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
- GUI for 2.x is not ready yet



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
Precondition	Same as "Load" but doesn't produce the metrics output CSV files
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*
Command	Execute a shell command

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
        }
    ]
}
```

SHELL COMMAND JOB EXAMPLE

```
...
"jobs" : [
     {
          ...
     }, {
          "type" : "command",
          "value" : "sleep 5m"
     }, {
          ...
     }
]
```

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*

OTHER V2.0 FEATURES

1. Copy Mode
   - *Allows to copy files/directories/objects from the source container/directory to the destination one.*
   - *Implemented as an extension of the "Write" load type.*
2. Load Limit By Total Size

   It was possible to limit a load job by an item count, a time and a rate in the previous versions. There are the new requirement to make it possible to limit by total processed size. For example, a load job should stop after writing 1TB of a data to the storage.

THE ROADMAP FOR 2.X

- GUI Enhancements
- Partial Read
- Centera API Support

Q & A

Thank you


EMC²
®