Play it Again, Sam – record and replay tools for load testing

David Collier-Brown
davecb@spamcop.net

“Play it Again, Sam”

- The real quote is

  Ilsa: "Play it once, Sam, for old times' sake."

  Rick: “Play it"

The Old Way

- Invent a synthetic test
- Buy H-P LoadRunner licenses
- Get misleading answers
  - The H-P product is evil
  - And *inventing* the truth isn't a good plan
- Switch to Jmeter
- Which is free and good
  - But keep trying to invent a valid test load ...

The Samba Team did it Better

- They carefully created a load script from a debug=10 log
- Laboriously!
  - And just once...

But it's way easier now

- Everything uses REST
  - And Apache, and Nginx, and so on
- They log one line per request-response pair
- So capture those and replay at 1x, 2x, 10x

Nginx example

- Same format as Apache (standardized)

  10.76.2.1 - - [28/Nov/2017:06:55:04 -0500] "GET /81eae740-a93a-467c-90d5-c555db9dc8a7 HTTP/1.1" 200 3994 "-" "Dalvik/1.6.0 (Linux; U; Android 4.4.4; Nexus 7 Build/KTU84P)"

- And as a stretch goal, can also log the $request_time

I reformat it with awk to this

#date time latency xferTime think bytes url rc op

2017-Nov-28 06:54:52 0 0 0 12578 /81eae740-a93a-467c-90d5-c555db9dc8a7 200 GET

- That's identical to my output format
  - > An output can be used as an input for reruns
  - > Or compared for improvement/degradation

And it looks like this in use

```
$ runLoadTest --rest --tps 80 --for 80 \
  log.csv http://10.92.10.201:80  |\
tee raw.csv
#yyy-mm-dd hh:mm:ss latency xfertime
thinktime bytes url rc op
2017-11-10 01:20:03.170 0.049963 0.000045
0 100 fc9b8674-3c3f-459a-a597-
bf4b1cc8ec00 200 GET
```

Code:

- This is a much simpler version of a convoluted C program

The implementation is pipe-oriented

- Write input to pipe
- Start consumers, 10 more every 10 seconds
- Consumers send a GET, time it and mostly just discard whatever comes back

Main loop

```
go workSelector(f, filename, fromTime, forTime, pipe)       // which pipes to ...
go generateLoad(pipe, tpsTarget, progressRate, startTps, baseURL) // .. alive
for {
        select {
        case <-alive:
                processed++
        case <-time.After(time.Second * conf.Timeout):
                log.Printf("%d records processed\n", processed)
                log.Printf("No activity after %d seconds, halting normally.\n",
                        conf.Timeout)
                return
        }
}
```

This is called “building a pipeline that ends here” (see also leafless.ca)

Increasing load

```
// add to the workers until we have enough
log.Printf("now at %d requests/second\n", rate)
for range time.Tick(time.Duration(conf.StepDuration) * time.Second) {
     //start another progressRate of workers
     rate += progressRate
     if rate > tpsTarget {
          // OK, we're past the range, quit.
          log.Printf("completed maximum rate, starting %d sec cleanup\n",
                    conf.Timeout)
          break
     }
     for i := 0; i < progressRate; i++ {
          go worker(pipe)
     }
     log.Printf("now at %d requests/second\n", rate)
}
// let them run for a cycle and shut down
time.Sleep(time.Duration(10 * float64(time.Second)))
close(closed) // We're done
```

Workers

```
func worker(pipe chan []string) {
     var r []string

     if conf.Debug {
          log.Print("started a worker\n")
     }
     // wait a random fraction of one second before looping, for randomness.
     time.Sleep(time.Duration(random.Float64() * float64(time.Second)))

     for range time.Tick(1 * time.Second) { // nolint
          select {
          case <-closed:
               if conf.Debug {
                    log.Print("pipe closed, no more requests to process.\n")
               }
               return
          case r = <-pipe:
               //log.Printf("got %v\n", r)
                go op.Get(r[pathField], r[returnCodeField])

            }
     }
}
```

How to use the program

- Debug the system under test
- Debug your date
- Do some sanity checks, then
- Run an increasing-load test until the target system falls over

First, debug the victim

- loadGenerator -v --rest --tps 1 --for 1 ./samples.csv http://localhost:5280
- This is super verbose
- If the return is not a 2XX, it prints the body

A bad connection

#yyy-mm-dd hh:mm:ss latency xfertime thinktime bytes url rc op

2018-01-06 16:43:59.654 0.000895 0.000000 0 0 /15b00a26-9ba3-4649-8477-c48bcab90dc7 444 GET expected=200

2018/01/06 16:43:59 restOps.go:197: error getting http response, Get http://localhost:5280///15b00a26-9ba3-4649-8477-c48bcab90dc7: dial tcp 127.0.0.1:5280: getsockopt: connection refused

Request:

GET ///15b00a26-9ba3-4649-8477-c48bcab90dc7 HTTP/1.1

Host: localhost:5280

User-Agent: Go-http-client/1.1

Cache-Control: no-cache

Accept-Encoding: gzip

Response: <nil>

Body: <nil>

2017/12/04 19:46:33 runLoadTest.go:115: No activity after 35 seconds, halting normally.

A success

```
#yyy-mm-dd hh:mm:ss latency xfertime thinktime bytes url rc
2017/11/11 21:11:20 runLoadTest.go:194: starting, at 1 requests/second
2017/11/11 21:11:20 runLoadTest.go:137: Loaded 1 records, closing input
2017/11/11 21:11:22 restOps.go:189:
Request:
GET /zaphod-beebelbrox.jpg HTTP/1.1
Host: calvin
User-Agent: Go-http-client/1.1
Cache-Control: no-cache
Accept-Encoding: gzip

Response headers:
   Length: 122944
   Status code: 200 È OK
   Last-Modified : [Fri, 11 Aug 2017 13:59:57 GMT]
   Accept-Ranges : [bytes]
   Server : [nginx/1.10.3 (Ubuntu)]
   Content-Type : [image/jpeg]
   Content-Length : [12530]
   Date : [Sun, 12 Nov 2017 02:11:47 GMT]
   Connection : [keep-alive]
   Etag : ["598db85d-30f2"]
Response contents:
HTTP/1.1 200 OK
Content-Length: 122944
Accept-Ranges: bytes
Connection: keep-alive
Content-Type: image/jpeg
Date: Sun, 12 Nov 2017 02:11:47 GMT
Etag: "598db85d-30f2"
Last-Modified: Fri, 11 Aug 2017 13:59:57 GMT
Server: nginx/1.10.3 (Ubuntu)

Body:
```

```
```

followed by many lines of gibberish from viewing a gif as text.

Then run from end to end

- Instead of `--for 1`, we run through the whole file at some convenient speed.
- If the system is expected to handle 100 request/second (TPS), try running at `--tps 100 --crash`, and see if you can get a clean run from beginning to end.
- Any error will put the verbose switch on, and --crash will stop on error

Then you can try a load test

- Once you have a test that will run from end to end at a moderate load,try a test with a load varying from small to perhaps ten time the maximum
- 10x so you find the point at which the response time curve turns upwards in the classic hockey-stick, "_/".

Demo:

- Run a smoke-test
- Then the increasing-load test
- Look at the raw data
- Then convert it to one-second samples
- And plot response time against increasing load to get a "_/" graph
- Finally, look at the low-load region you're actually going to use
  - Unless, of course, there is no such region (;-))

Then the increasing-load test

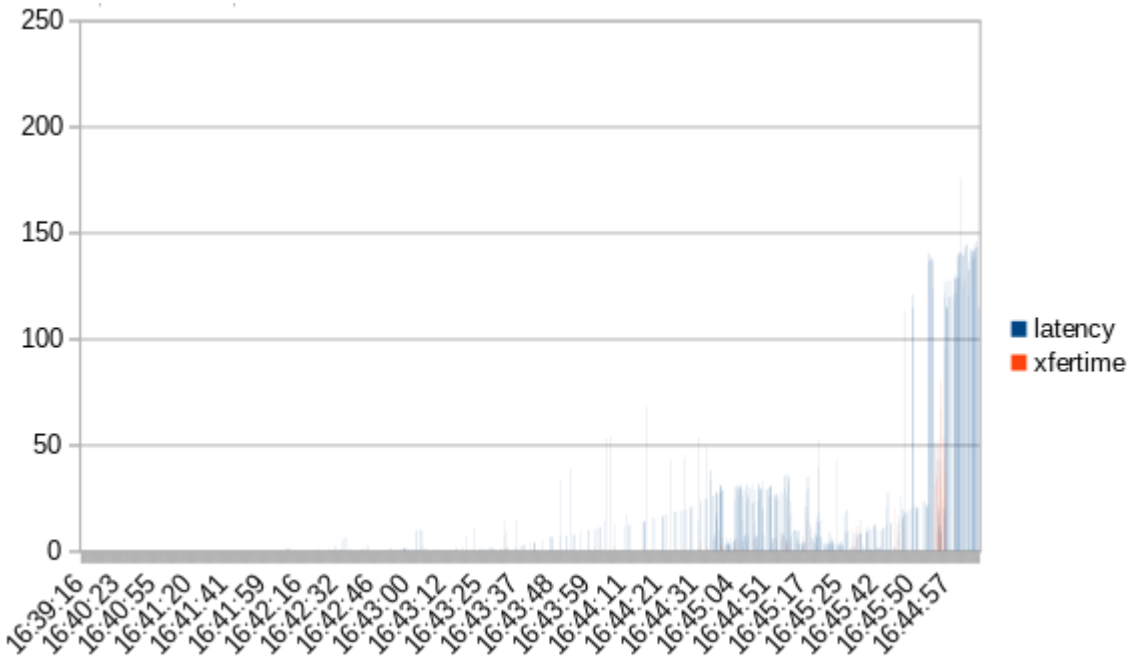
- The data looks like this…

```
#yyy-mm-dd    hh:mm:ss latency  xfertime thinktimebytes    url  rc   op

2017-12-10    16:39:08.511  0.002729 0.000288 0    12530    /15b00a26-
9ba3-4649-8477-c48bcab90dc7 200  GET

2017-12-10    16:39:08.533  0.000648 0.000323 0    178  /8318b57f-c1fa-
4587-8dbd-2b78cee5d20b 404  GET
```

- Just a sequence of boring lines, right?

But if you graph it

- You can see signs of the hockey-stick



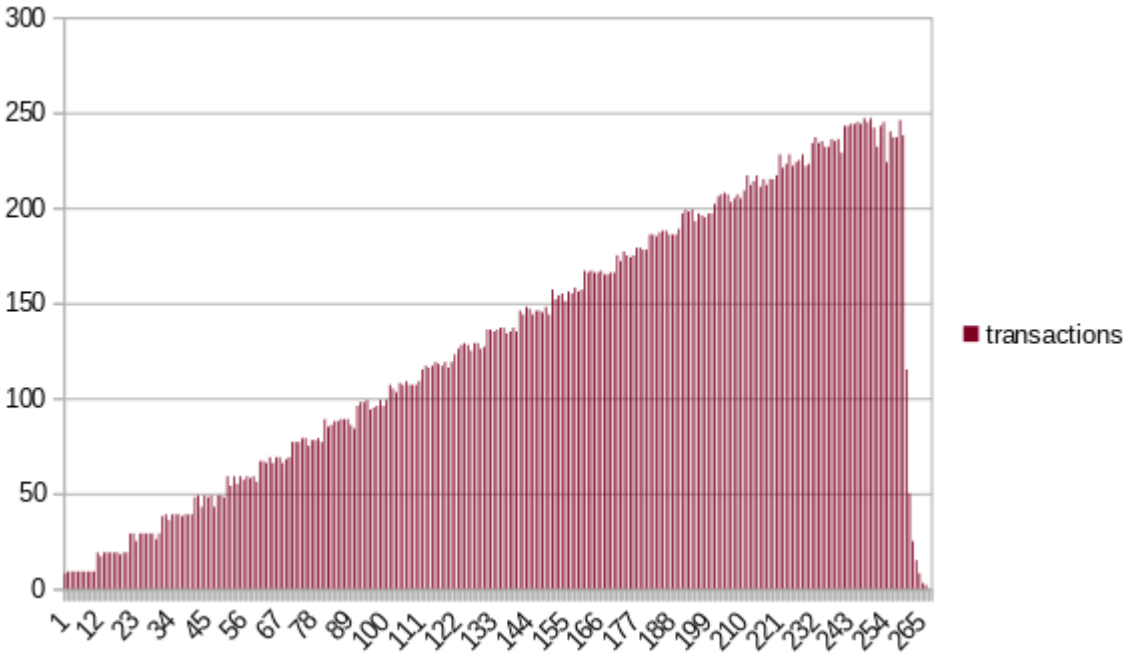
If you convert into one-second samples

- You can also see the request rate (TPS)

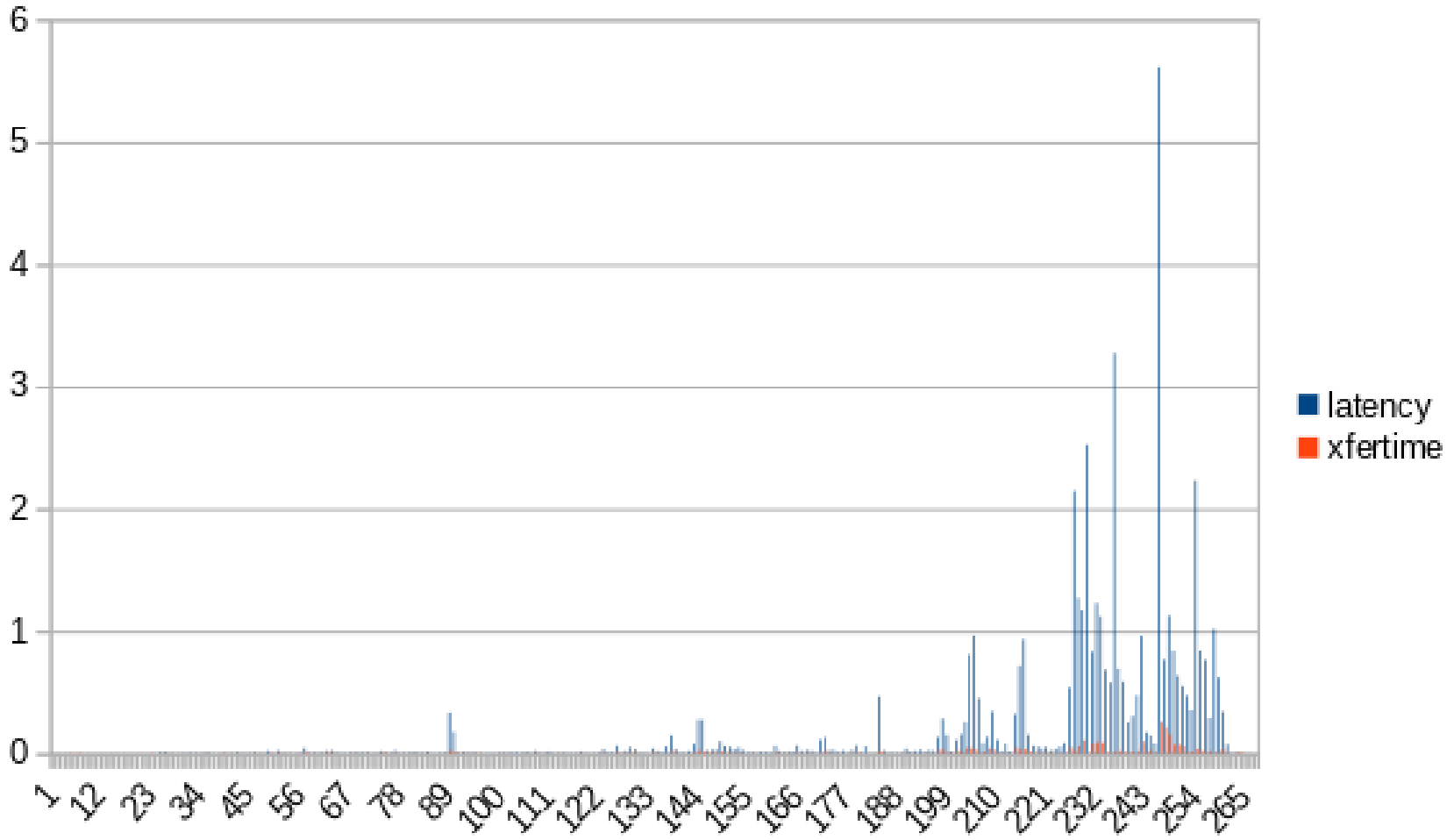
#date	time	latency	xfertime	thinktime	bytes	transactions	
2017-12-10	16:08:51	0.001429	0.002411	0	175108	8
2017-12-10	16:08:52	0.001542	0.002063	0	210979	9
2017-12-10	16:08:53	0.001012	0.000515	0	54809	9
2017-12-10	16:08:54	0.001011	0.000862	0	77291	9
2017-12-10	16:08:55	0.00104	0.004892	0	439226	9
2017-12-10	16:08:56	0.00111	0.001067	0	114574	9
2017-12-10	16:08:57	0.001122	0.006129	0	629454	9
2017-12-10	16:08:58	0.001019	0.000302	0	27544	9
2017-12-10	16:08:59	0.00097	0.000919	0	90388	9
2017-12-10	16:09:00	0.001177	0.001893	0	123507	9
2017-12-10	16:09:01	0.000984	0.000335	0	39037	19
2017-12-10	16:09:02	0.000918	0.001949	0	219558	17
2017-12-10	16:09:03	0.00098	0.002238	0	269115	19
2017-12-10	16:09:04	0.001097	0.003237	0	384714	19

Graphed, you start to see more ...

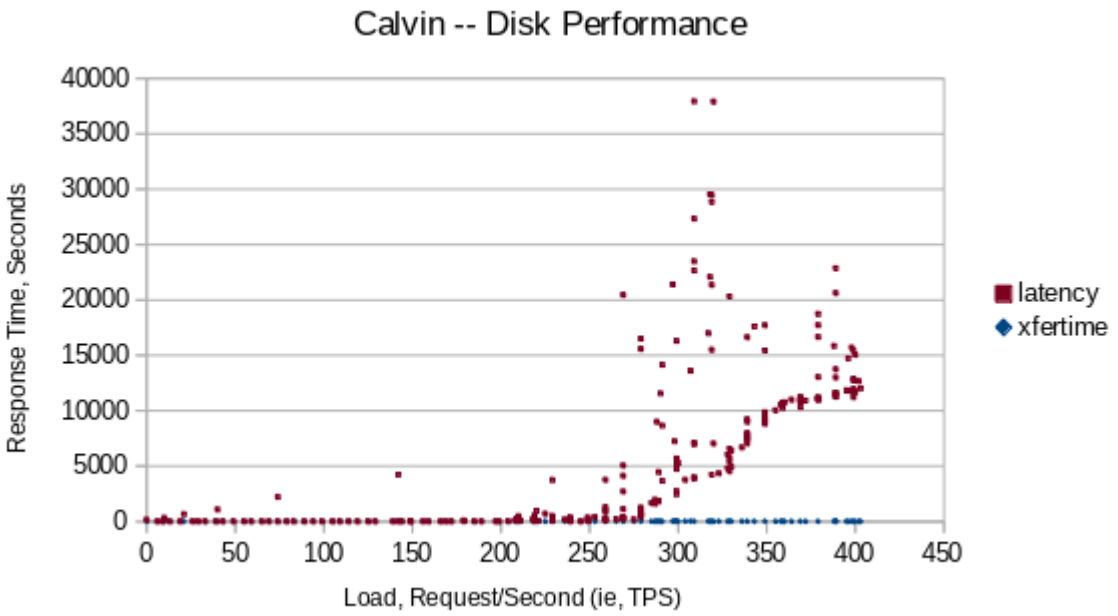
- Transaction rate rises steadily



Response times spike

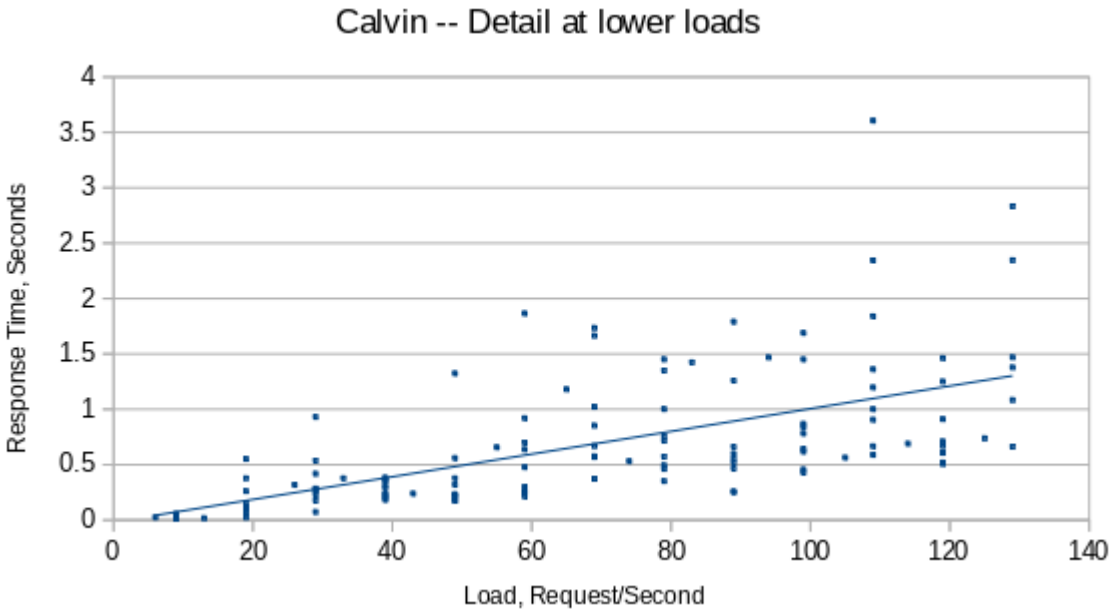


Plot as a scattergram



- The latency greatly increases after 250 requests per second.

Low-load detail



- Below that it gently increases, from a quite pleasant 0.4s at 40 requests/second to an ugly 1.25 seconds at 120.

Draw conclusions

- Down in the low range we expect, it's linear and pretty quick
- At "normal overload" it's ordinarily slow
- At > 250 TPS, it finally hits the wall

- If I want to build a ceph array of those disks, that's the information I need.
- Ditto if it were any other server.

Apply this to your problems

- It's pretty much always not the speed of am individual device, but instead the *number* of devices.
- Apply a real production load to a test server, then crank it up.
- Then scale horizontally

  [Note that doesn't work if your problem only scales vertically, like a relational DB]

What else to use a load generator for?

- Profiling –
  - Where does the program spend it's time when it's actually doing a production load?
  - That's not the same as while doing unit tests!
- Code coverage –
  - What part of the code is unreachable by paying customers?
  - Why are we paying for unused stuff?

What else...

- Debugging –
  - I find LOTS of stuff
  - And you probably can use it for fuzz testing by creating data with /dev/random (;-))
- Pre-production –
  - Feed a copy of the production load by doing a tail -f on the production logs and feeding that to the generator
- Pre-migration/pre-caching –
  - In one case where we did lazy fetching from another service, replaying prod pre-loaded us

Links

- Gitub repo
  - https://github.com/davecb/Play-it-Again-Sam
- Tutorials
  - https://github.com/davecb/Play-it-Again-Sam/blob/master/Running_Record-Reply_Tests.md
  - https://leaflessca.wordpress.com/2017/11/12/play-it-again-sam-a-load-testing-tool/
- Man Page
  - https://github.com/davecb/Play-it-Again-Sam/blob/master/cmd/runLoadTest/runLoadTest.md