MockData: load generation based on read replay with integrity check for Xcache

David Smith (david.smith@cern.ch) on behalf of IT-DI-LCG, UP team.

10 Dec 2019, DOMA Access meeting

Overview

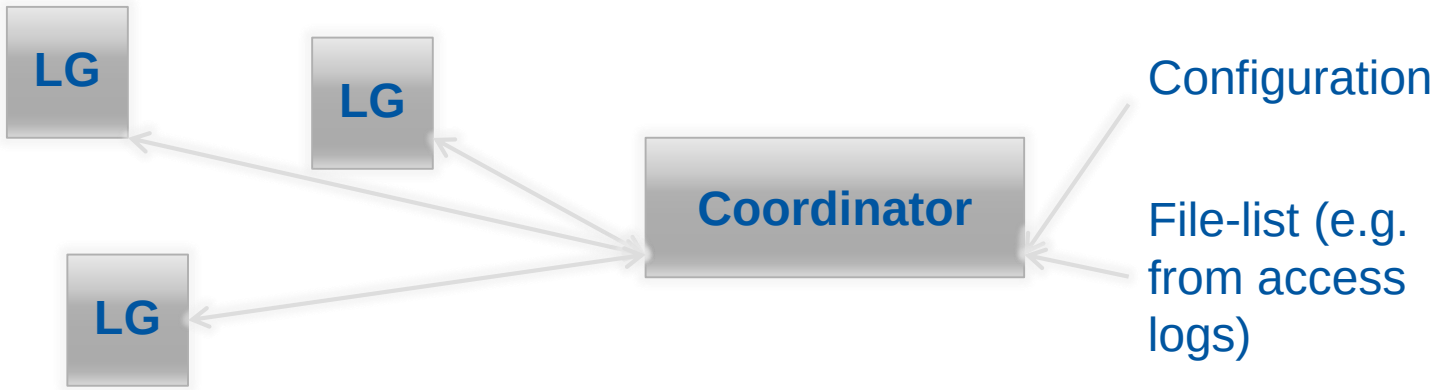
- What is MockData/The Pieces/MockData & Xcache
- The file-list
- About the files
- Assigning files: Scaled interval or predictable rate
- Load-generator: transferring files
- An example: log messages
- Network traffic plots from test runs
- Other transfer options
- If you would like to try MockData
- To do

What is MockData

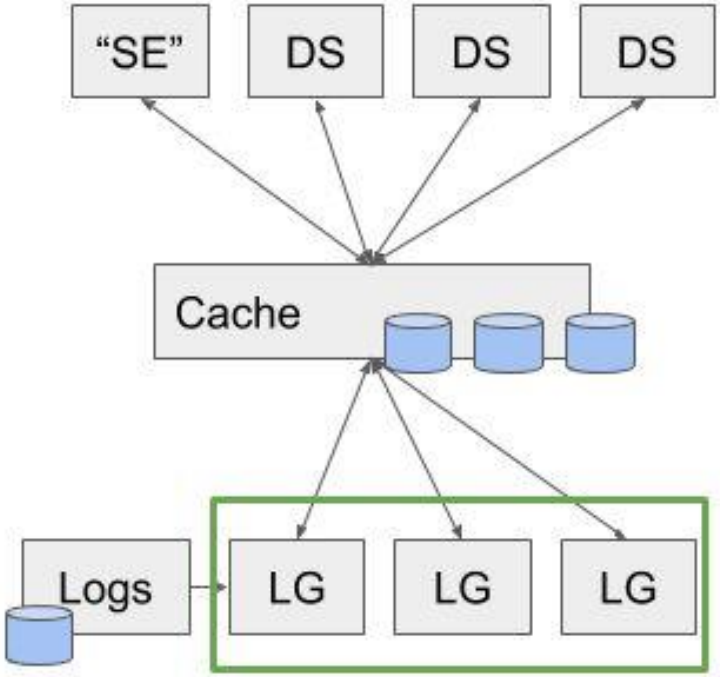
- Has a server and multi-client architecture
- A server (*coordinator*) sends configuration information and a list of file names
- clients (*load-generator*) make XRootD transfers of files
  - Number of clients (and therefore number of hosts) can be raised to provide useful load level
- An XRootD plugin allows an XRootD server (*data-source*) to present an unlimited number of files, each with individual, random-looking but predictable content without needing to store the files

The pieces

- MockData: coordinator and load-generator
- (data-source with plugin not shown)



MockData and Xcache



The file-list

- File list looks like:

```
# time                filename                                  filesize
#
1527793220 DAOD_HIGG2D1.14249006._000128.pool.root.1 4140091969
1527793250 DAOD_HIGG2D1.14249006._000125.pool.root.1 4787931809
1527793258 DAOD_HIGG2D1.14249006._000135.pool.root.1 3977269205
…
```

- Must be time ordered
- For testing have been using a list taken from ATLAS Rucio trace files from a Tier-2 over 1 month
- 1.43M files, 1.50PB, 0.84M unique files

About the files

- Files can be sent to the Xcache from a server using a MockData plugin which generates content based on filename & size. This allows:
  - Sending many different files to the Xcache without having to store all files
  - The load-generator can verify the chunks of the files received by regenerating them locally.
  - The plugin at the server can introduce a configurable delay in fulfilling the requests, thereby approximating a latency between server and cache
- Actually the plugin is optional: A regular storage could be used, if the files exist there. But the integrity check will not be done.

Assigning files: Scaled interval

- The coordinator assigns out files in turn
- Can reproduce the time interval between file start of each file transfer, with an optional scaling factor. e.g.

mockdata-coordinator –f 1.0

```
1527793220 DAOD_HIGG2D1.14249006._000128.pool.root.1 4140091969
1527793250 DAOD_HIGG2D1.14249006._000125.pool.root.1 4787931809
…
```

- Starts the first transfer, then 30 seconds later starts the second, or with a factor of 2.0 would start it 15 seconds later.. The test finishes when the file-list is finished

Assigning files: Predictable Rate

- Alternatively targets an average transfer rate:

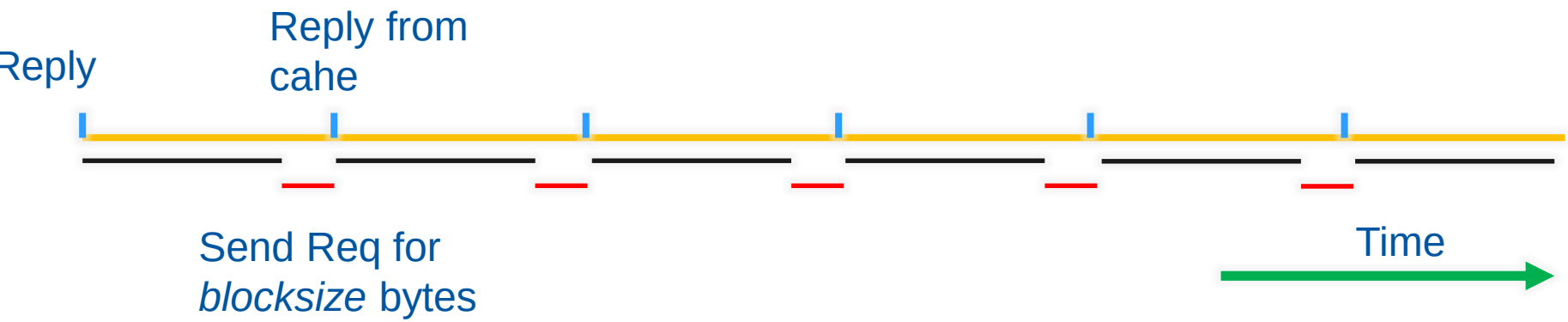
mockdata-coordinator –f 100,500,5000

- Means start assigning files so that the initial rate should be 100MB/s, rising to 500MB/s over 5000s, after which the test finishes.
- This essentially ignores the “time” parameter in the file-list, but preserves the order

Load-generator: transferring files

- Once assigned a file, the load-generator transfers it by making read requests with delays between them to match the desired duration.
- The overall duration of the transfer is a part of the configuration and can either be an absolute time or a depend on the filesize (i.e. a rate)

Load-generator: transferring files



Load-generator sleeping

Load-generator waiting for an XRootD reply

Try to target configured overall transfer time by adjusting sleeping time so that the sum of sleep+request waiting = desired duration

A summarizing quantity called the transfer efficiency is also reported:

transfer efficiency = (desired duration)/(desired duration + sum[request wait]) * 100%

The actual duration of the transfer can exceed the desired duration (overrun) if the reply waiting time is too large

An example: log messages

- One file in the file-list

1567502522 myfile1.txt 1024

- And run the coordinator:

mockdata-coordinator –f 1.0

The command will write log lines to stdout:

Nov 29 11:47:24.370539 pmpe01 [INFO] Starting replay, reference time 1575024459
Nov 29 11:47:24.370640 pmpe01 [INFO] Setting the startpoint timestamp from the filelist to 1567502522

The above means the first file will start to transfer at 1575024459 (a unix timestamp in about 15 seconds time), and it originally had a timestamp of 1567502522

Assignment made and load-generator commits to start:

Nov 29 11:47:34.730485 pmpe01 [INFO] Assigning id=0 originalStartTime=1567502522 filename=myfile1.txt intended_duration=0.000102 allowed_overrun=1.000000/600.000000 approx_start=5.000000 lgidstr=pmpe01.cern.ch#31790 dutycycle=0.000000 prev_nassigned_lg=0 nassigned_total=1 assignRateFiveMinAvgMBs=0.000000

Load-generator reported transfer finished:

Nov 29 11:47:39.375672 pmpe01 [INFO] Received success for myfile1.txt id=0 lgidstr=pmpe01.cern.ch#31790 result_rc=0 result_summary=Final stats for fid=0 fileUrl=xroot://caaaa000@pmpe06.cern.ch//mockdata/myfile1.txt_1024_0 intended_duration=0.000102 nbissued=1024 nread/vecread_bytes=1024/0 nread/vecread_calls=1/0 duration=0.004259 accessMap=|**********| seekPdfFit=0 nseekbytes=0 nseeks=0 seekedFracPerSeek=0 seeksPerMB=0.000000 Vnchunks/Vnread=0 fsize=1024 lastServer=caaaa000@pmpe15.cern.ch:1095 xrdReqCallbackTime=0.000353 xrdReqOverlapFactor=1.000000 deliveryEff=22.493284 globalRateEstimateMBps=0.000068

An example (log line focus 1)

Nov 29 11:47:34.730485 pmpe01 [INFO] Assigning id=0 originalStartTime=1567502522 filename=myfile1.txt intended_duration=0.000102 allowed_overrun=1.000000/600.000000 approx_start=5.000000 lgidstr=pmpe01.cern.ch#31790 dutycycle=0.000000 prev_nassigned_lg=0 nassigned_total=1 assignRateFiveMinAvgMBs=0.000000

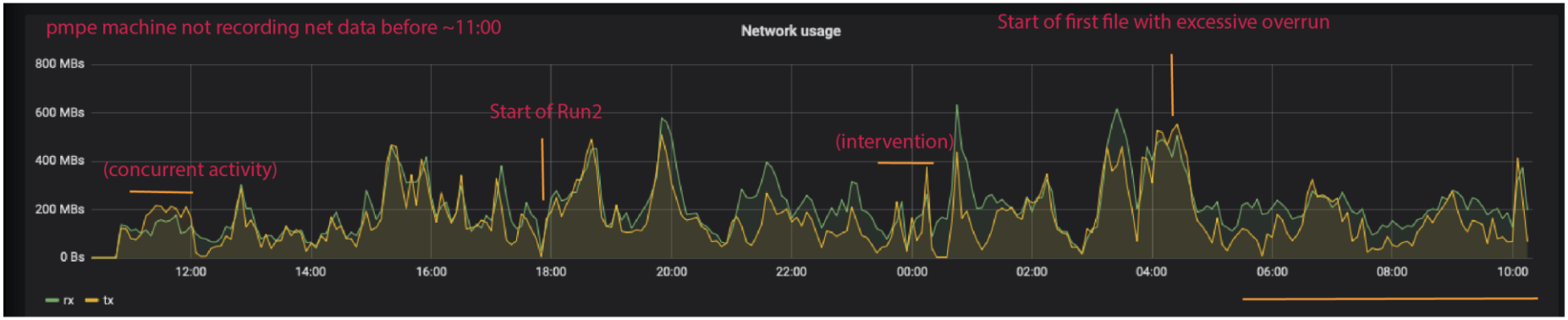
The first file ("myfile1.txt") has been assigned to the load-generator running at pmpe01 pid31790. The load-generator will start it in about 5 seconds, and it should complete in about 0.0001s (it's a very small file!) The file must be received within an overrun time which is 1.00 * intended duration or 600s, whichever is larger. Overrunning transfers will be stopped and considered failed.

An example (log line focus 2)

Nov 29 11:47:39.375672 pmpe01 [INFO] Received success for myfile1.txt id=0 lgidstr=pmpe01.cern.ch#31790 result_rc=0 result_summary=Final stats for fid=0 fileUrl=xroot://caaaa000@pmpe06.cern.ch//mockdata/myfile1.txt_1024_0 intended_duration=0.000102 nbissued=1024 nread/vecread_bytes=1024/0 nread/vecread_calls=1/0 duration=0.004259 accessMap=|**********| seekPdfFit=0 nseekbytes=0 nseeks=0 seekedFracPerSeek=0 seeksPerMB=0.000000 Vnchunks/Vnread=0 fsize=1024 lastServer=caaaa000@pmpe15.cern.ch:1095 xrdReqCallbackTime=0.000353 xrdReqOverlapFactor=1.000000 deliveryEff=22.493284 globalRateEstimateMBps=0.000068

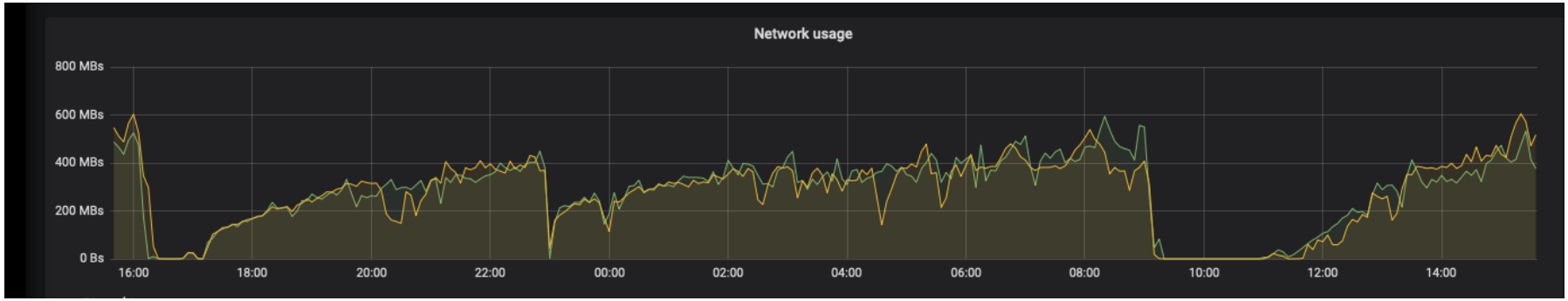
The file myfile1.txt completes transfer ok. It was served by the xcache node at pmpe15:1095.

Example network traffic plot



- This plot shows (green) incoming traffic to load-generators and (yellow) going traffic from XRootD server
- This was a scaled-replay mode: The peaks are mostly a reflection of the original variability of requests at the original site (at 10 to 20% scaled rate)

Example network traffic plot 2



This was a rate-target mode: The traffic rate profile is expected to approach a linear slope. The above contains 3 separate runs.

- Linearity visible in the low traffic level portions
- However deviation is soon evident, I suspect to be limits at this cache instance during this test. (For further investigation)

Other transfer options

- There are a number of aspects of the transfers which can be controlled. They are written in *file & access profiles* given to the coordinator and can be matched on a per file basis using a regular expression against the filename.
- These include:
  - The delay to introduce at the XRootD server (10ms in current tests)
  - Fraction of the file to read: (1.0 currently; may be >1.0)
  - Intended duration (absolute or a rate: e.g. 10MB/s)
  - Allowed overrun time (fraction or absolute)
  - Number of IO requests to be outstanding at any time
  - Number of file requests to XrdCl login (e.g. stream) and number of XrdCl substreams per stream
  - Fraction of reads which should be VectorRead and number of chunks for each vector read.
  - Binned probability distributions describing the seek distance between each request (as a fraction of the file length: -1.0 to +1.0) and the chunk read size (between arbitrary low/high).
    - Tests so far used a sequential access (0 seek) and a distribution of small chunk sizes, average ~25KB.

If you would like to try:

- The repository is readable if you have a cern account:
- https://gitlab.cern.ch/dhsmith/mockdata
- This includes a README.md (rendered at the gitlab page above) with a quick-start guide.
- Build for centos7 in RPMs is available within the CI/DI link (look for tag v1.0.0-1/build artifacts) or here
- http://cern.ch/~dhsmith/MockData/v1.0.0-1/
- Sample file-list may be available (contact me).
- Package includes tool to request random files

To Do

- I will aim to fix any bugs and add useful features
- The quick-start guide should allow you to get going, but more detailed documentation of all the options and operation may be needed for your study
- Please contact me if you have problems or questions: I'll setup a bug or feature tracker if there's sufficient volume of requests

david.smith@cern.ch