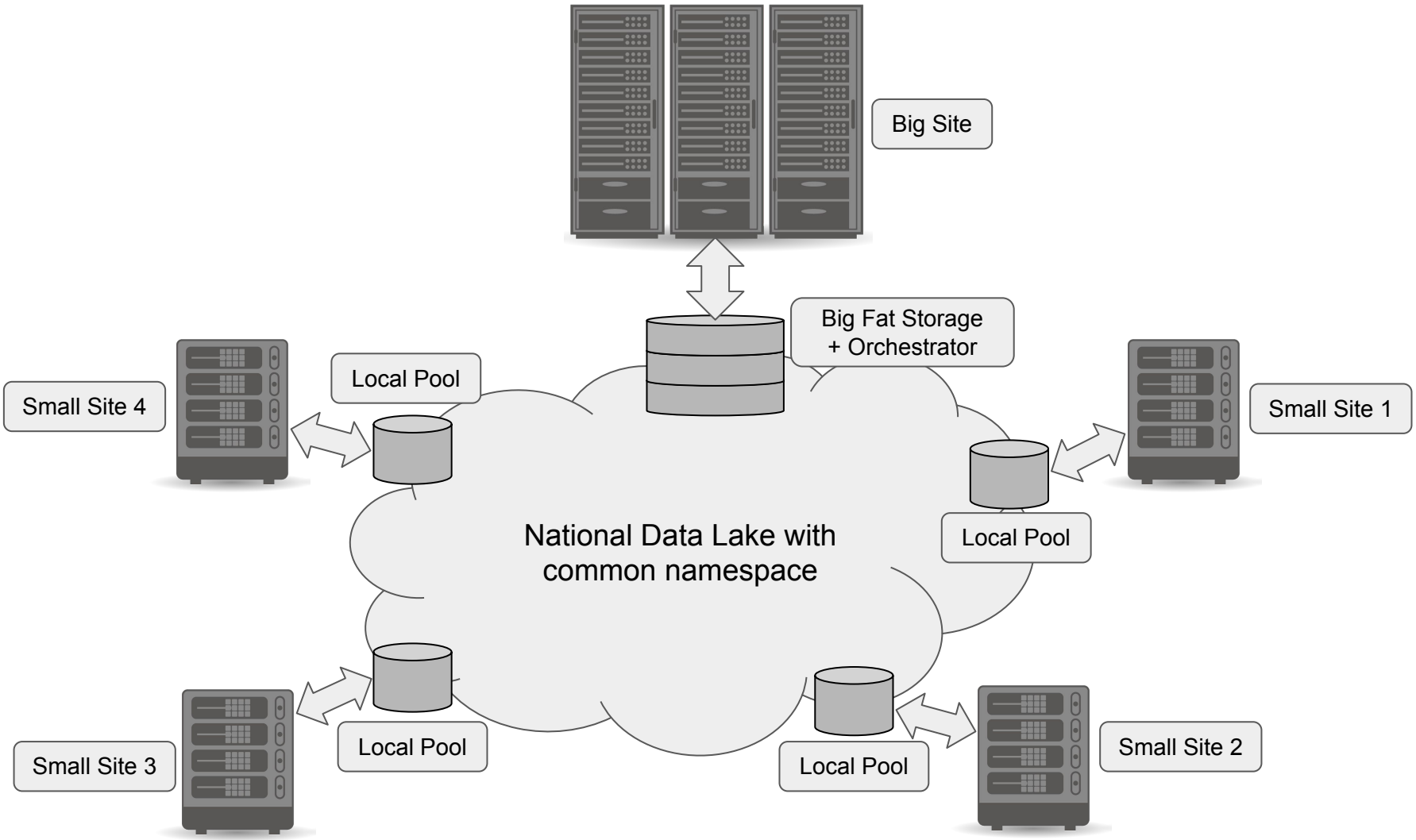
Experience of using EOS LRU and Converter engines for write buffering in the Data Lake prototype

A. Kiryanov, A. Zarochentsev

Data Lake with CPU-oriented smaller sites



Data Lake with CPU-oriented smaller sites

- Our primary goal was to optimize CPU usage for small CPU-oriented sites
  - Avoid CPU idle due to file I/O as much as possible:
  - Via caching for read-biased workloads
    - XCache-based solution
    - Outside of the scope of this talk
  - Via buffering for write-biased workloads
    - EOS-based solution
    - Use built-in features as much as possible
      - No extra site configuration
      - No extra software
      - => no problems!
- Our secondary goal (which is as much important as primary) is to minimize the deployment complexity and manpower requirements
  - Hence the inclination towards a simple all-in-one solution
  - EOS is not the only option, but we have started with it

Write buffering explained

In the “old” WLCG computing model many experiments were writing output data to the site-local SE. This solution was aimed at solving the problem of writing output data remotely from the worker node during the job execution time. There’s a couple of buts though:

1) Local SE reliability is somewhat questionable (from a life experience of a Russian sites coordinator for ALICE)
2) It’s nearly impossible to have a CPU-only site without any storage
3) This kind of storage policy may not be suitable for non-LHC workloads

In the Data Lake scenario we want to use a fast local disk pool (not a full-fledged SE) to minimize file transfer time. The job still writes directly to the Lake (local pool is transparent and optional), but it uses the fastest way possible. Because local pool is expected to be of a limited capacity and reliability, all of its data is continuously migrated to the other location in the Data Lake, but this migration happens in the background freeing the CPU time on the worker nodes.

EOS-based solution

In order to save CPU time we need to offload files to the remote reliable storage in the background, and we've found a way to achieve this using a combination of two built-in EOS features:

- LRU Engine: https://eos-docs.web.cern.ch/configuration/lru.html
- Converter Engine: https://eos-docs.web.cern.ch/configuration/converter.html

The docs say that one can create an LRU policy for a directory which will change the placement policy for all files with a given properties. So, we've come up with the following scenario:

- We create a dedicated directory in the global namespace
- Default placement policy for this directory is to store all files on the site-local pool using GeoTag
- LRU will pick up files that were created at least 10 seconds ago and change their placement policy to a new one
- New placement policy will force flles to migrate to the remote pool

Configuration details

In order to fulfill our scenario we've defined two groups:

1) Group one contains both site-local and remote pools with different GeoTags
2) Group two contains only remote pools

Baseline test were performed on a directory **/eos/jinrdvl/tests/mainpool/** with the following attributes:

```
sys.forced.group="2"
```

Then we've switched to a different directory **/eos/jinrdvl/tests/mainpoolfcache/** with the following attributes:

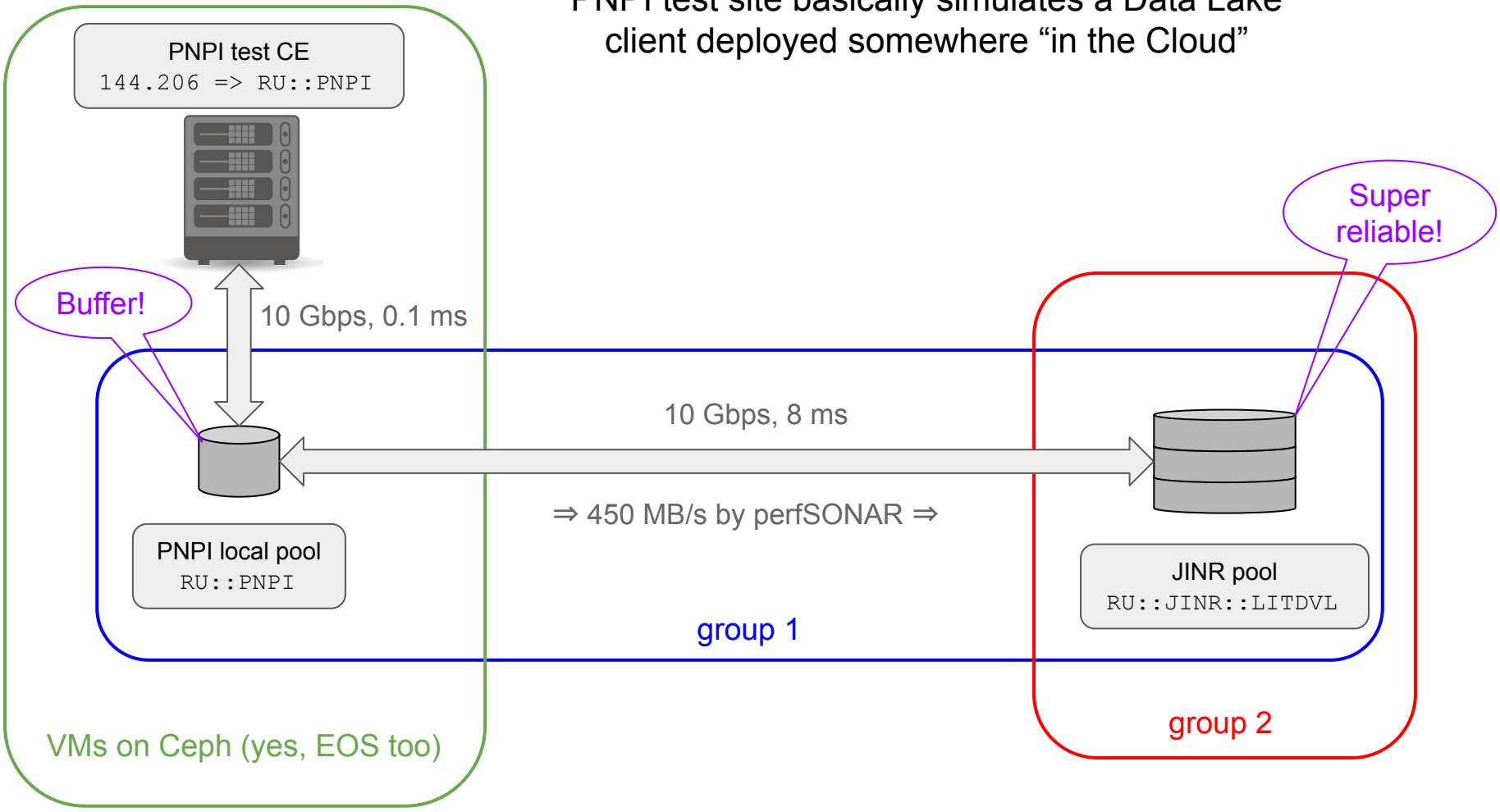
```
sys.conversion.*="gathered:RU::JINR::LITDVL"
sys.forced.group="1"
sys.forced.space="default"          ← This is needed for Converter
sys.lru.convert.match="*:10"
```

LRU was configured to run every minute

```
lru.interval := 60
```

Testbed plot

PNPI test site basically simulates a Data Lake client deployed somewhere “in the Cloud”



Oopsy daisy

Aaaaand that didn't work in the EOS 4.8.22 that we were using at the moment:

```
... [ERROR] Server responded with an error: [3009] Unable to get free physical space ...; No space left on device; (destination)
```

Apparently that version was in the middle of code rewrite, which caused incompatibility between LRU and Converter engines.

We've contacted the developers (thanks, Elvin) and got PoC tests running with the old Converter that was luckily still there:

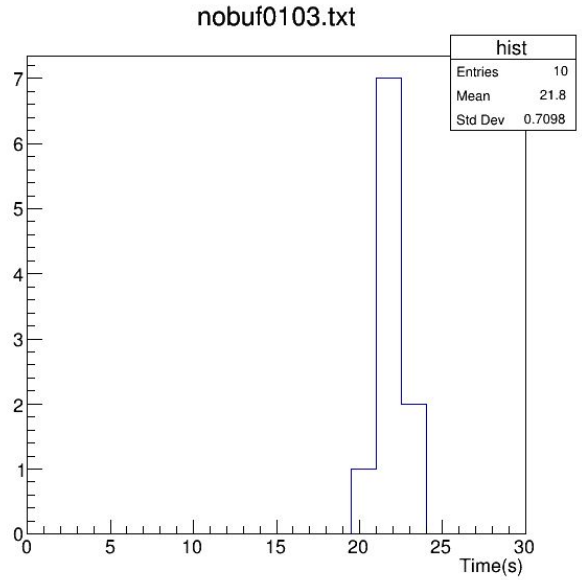
```
EOS_FORCE_DISABLE_NEW_CONVERTER=1
```

This issue was resolved in EOS 4.8.27, we're running EOS 4.8.39 now.

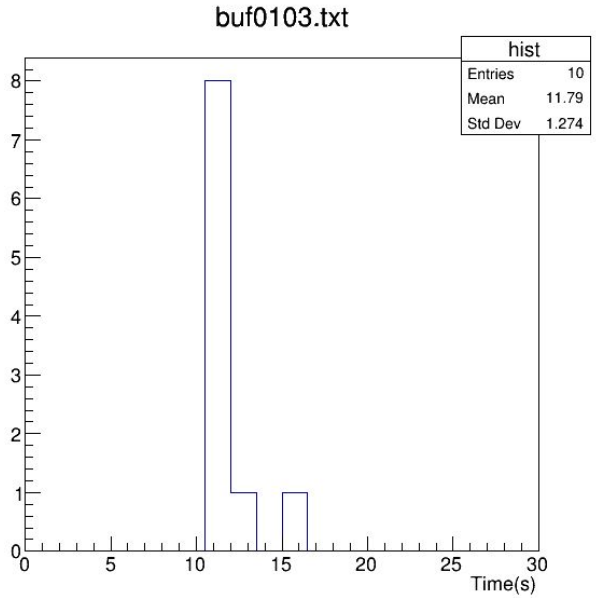
So, we've started some automated synthetic tests.

Tests and Results

Our synthetic test job generates ten 100MB files and successively uploads them into specified location using xrdcp. We were running 10 of these on different WNs at PNPI test site.



Direct upload to JINR (left plot) takes 22 seconds



Upload to PNPI Buffer (right plot) takes 12 seconds

Great success!

Post-flight analysis

- No issues whatsoever with files uploaded directly to JINR. 100/100 reached their final destination
- One hour after PNPI Buffer test was done only 57 files out of 100 were migrated (converted) to JINR (from group 1 to group 2)
- 43 files were still on PNPI FST, but looking at the conversion queue we were seeing only 34 jobs:

```
dvl-eos-m01:~ # eos ls /eos/jinrdvl/proc/conversion/ | wc -l
34
```

  And they looked kinda stale

- We've restarted the MGM service and voila, conversion queue is empty:

```
dvl-eos-m01:~ # eos ls /eos/jinrdvl/proc/conversion/ | wc -l
0
```

- But 12 files out of 100 are still left on PNPI FST

```
-bash-4.2$ dir=/eos/jinrdvl/tests/mainpoolfcache
-bash-4.2$ for i in `eos ls $dir`; do eos fileinfo $dir/$i; done | grep PNPI | wc -l
12
```

Conclusions

- Maybe it wasn’t the intended way of using LRU and Converter, but it worked!
- We’ve run into some minor issues: ~10 percent of files were not converted
- EOS codebase evolves quickly!
- When we’re finally happy with synthetic test results we’ll move onto real-life payloads from ATLAS and ALICE
- Thanks to EOS developers for such nice features!
- Thanks to Valery Mitsyn from JINR for his help with our studies!

Thanks!

Test Script

```
#!/bin/sh
DST=$1
function main {
echo $DST
for i in {1..10}; do
OUTFILE=./file100mb$RANDOM
dd if=/dev/urandom of=$OUTFILE bs=1M count=100
xrdcp -f -p $OUTFILE ${DST}/$OUTFILE
done
}
time main
```

Space Configuration

```
dvl-eos-m01:~ # eos space status default
# ------------------------------------------------------------------------------------
# Space Variables
# ....................................................................................
autorepair                          := on
balancer                            := on
balancer.node.ntx                   := 2
balancer.node.rate                  := 25
balancer.threshold                  := 20
converter                           := on
converter.ntx                       := 10
drainer.node.nfs                    := 5
drainer.node.ntx                    := 2
drainer.node.rate                   := 25
drainperiod                         := 86400
filearchivedgc                      := off
geo.access.policy.write.exact       := on
geobalancer                         := on
geobalancer.ntx                     := 10
geobalancer.threshold               := 5
graceperiod                         := 86400
groupbalancer                       := on
groupbalancer.ntx                   := 10
groupbalancer.threshold             := 5
groupmod                            := 50
groupsize                           := 50
lru                                 := on
lru.interval                        := 60
nominalsize                         := 270.00 TB
quota                               := off
scaninterval                        := 604800
scanrate                            := 100
stat.converter.active               := 0
tracker                             := off
wfe                                 := on
wfe.interval                        := 10
wfe.ntx                             := 1
```