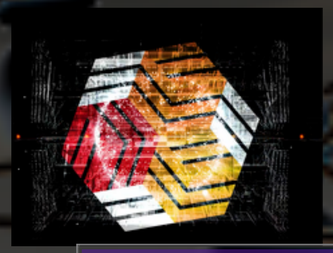
IO SHAPING
in EOS
IT Storage & Data Management Group - HEPiX Online 2022

Andreas-Joachim Peters
CERN IT-SD for the EOS team



Contents

- **Introduction**
  - why did we implement **IO Shaping***?

    * meta-data and data traffic shaping

- Five types of **IO Shaping**
  1. IO Types
  2. IO Priorities
  3. Bandwidth Policies
  4. Filesystem Overload Protection
  5. Meta-data rate & thread pool limits

- **Summary**

Introduction (1)

- **EOS** instances are used by a **large user community - EOS is a large shared resource**
  - the **criticality** of individual access types **varies** a lot

    example:
    - data INGRES from online DAQ systems requires highest priority (real-time critical)
    - background scanning to verify file checksum is a low-priority task, which should back off for most other use cases
  - **real-time driven** applications require minimal IO fluctuations

    example:
    - the transfer time for an online system can vary within the given time budget, but large tails in transfer times have to be avoided
  - **meta-data performance** degrades for with thousand clients interactive usage “batch DOS”

Introduction (2)

- To provide a certain **service guarantee** to core-activities we need handles to influence data and meta-data rates per user, group, per client or by activity
  - batch vs interactive
  - online vs offline

The Policy Model

- **IO data policies** can be defined in five ways for readers and writers separately

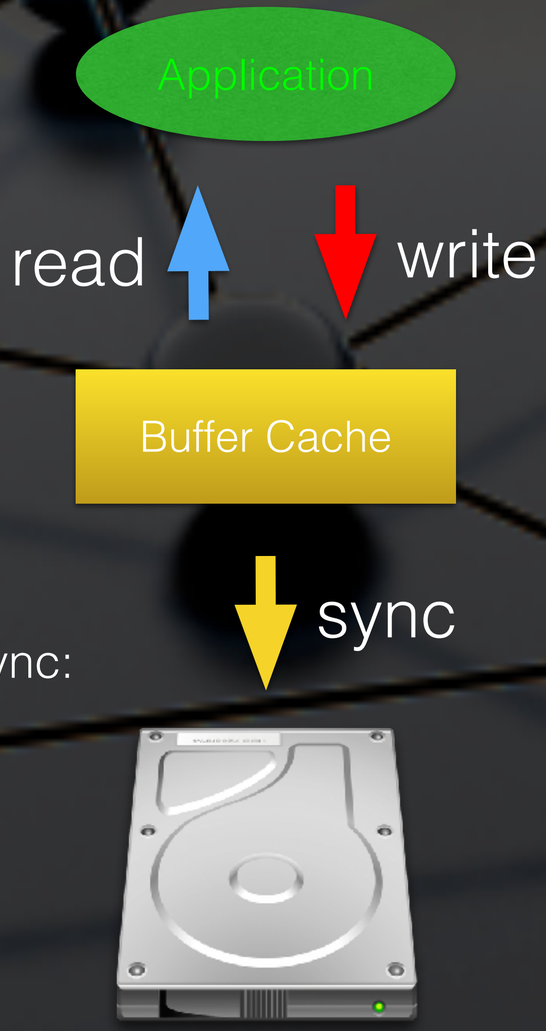
1. **by space -** applies to everybody using a target space (=pool)
2. **by group -** valid for all users in a given group
3. **by user** - valid for a single user
4. **by application** - valid for a tagged client application
   - applications are identified in the URI via `?eos.app=applicationname`
5. **via** an **extended attribute** on the **parent directory** of a given file

The evaluation order is from 1 to 5 e.g. space policies are overwritten by an application policy

1. IO Types

Buffered vs direct IO …

buffered IO

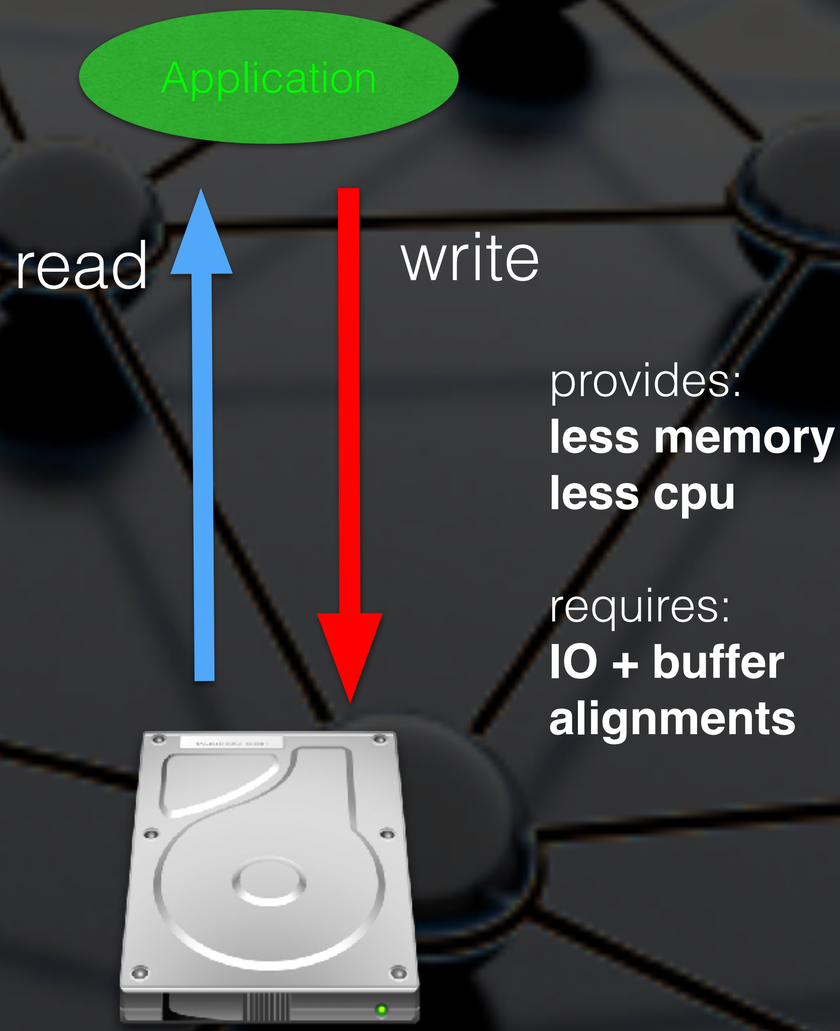


provides:
caching
read-ahead
write-back

requires:
memory
cpu

pitfalls without fsync:
data safety
ENOSPC
IO stalling

direct IO



provides:
less memory
less cpu

requires:
IO + buffer alignments

The three **EOS IO Types**

- **direct IO** - implemented in EOS OSS plug-in
  - uses two file descriptor
    1. **direct IO** for IO which fullfills alignment
    2. **buffered IO + fdatasync + posix_fadvice** for writes which does not fulfil alignment
       [might be changed to O_SYNC for simplification]

- **sync** = **dsync** - use synch. IO
  - use file descriptor with **O_SYNC**

- **csync** - sync on close
  - write via buffer cache but **fdatasync** on close without the client calling sync directly

- *option to be added:* ***direct IO + csync***

Selecting **IO Types**

Evaluation Order

instance default:
“**eos config default space.policy.iotype:r=direct**”

space specific:
“**eos config erasure space.iotype:w=sync**”

application specific default for app ‘foo’:
“**eos config default space.iotype:r.app:foo=direct**”

group specific for group ‘z2’:
“**eos config erasure space.iotype:w.group:z2=csync**”

directory enforced :
“**eos attr set sys.forced.iotype:w=direct /eos/daq/**”

Impact of direct IO

- measured that direct IO **improves** maximum **WRITE** performance of standalone XRootD server with standard CERN disk server from **7 GB/s to 9 GB/s**
- direct IO **increases** instance performance for **WRITE** workloads
- using direct IO **reduces** performance **tails for WRITE** workloads
- direct IO **reduces** instance performance for **READ** workloads

2. IO Priorities

IO Priorities

- IO priorities currently available only with **CFQ/BFQ** scheduler on LINUX - support in deadline scheduler coming
- three levels: idle (**idle:0**), best-effort (**be:0-7**) , real-time (**rt:0-7**)

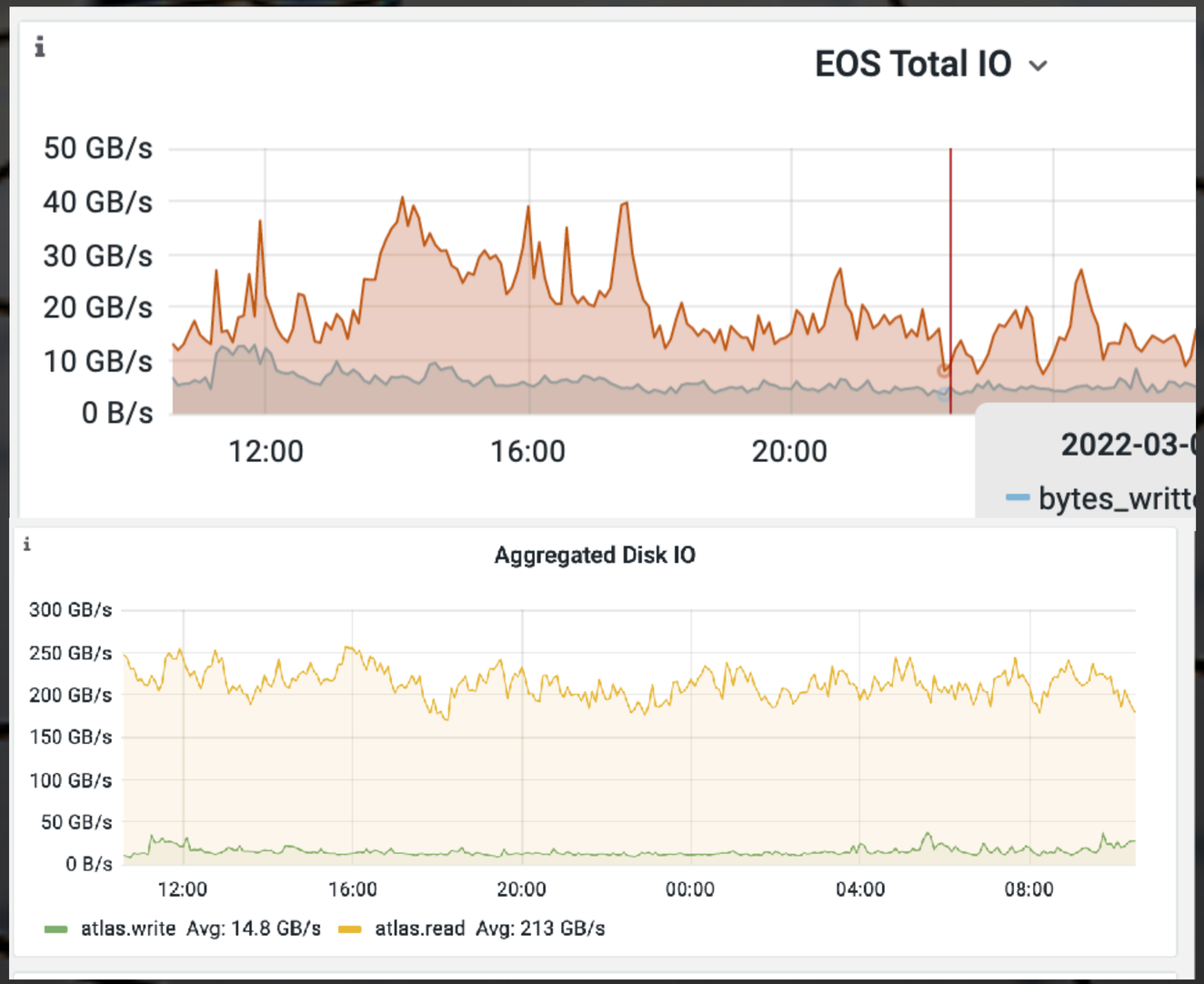
real-time 0 >> real-time 7 >> best-effort 0 >> best-effort 7 >> idle

default IO priority is **be:4**

- the EOS background scan for checksum verification is runnning with best-effort **be:7**

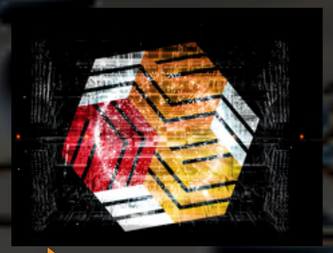
- **realtime** supported from EOS V4.8.79 on (+ 5.0.15)
- **IO** priority works only for **read + direct IO write**

IO Priorities



IO originating from applications in EOSATLAS

Disk IO measured during the same period mainly background scanning



Selecting **IO Priority**

a

via CGI: **“root://myeos?eos.iopriority=be:1”** if user has the ‘operator’ role (member of operator in VID interface)

Evaluation Order

b

instance default:
“**eos config default space.policy.iopriority:w=rt:0**”

space specific:
“**eos config erasure space.policy.iopriority:w=be:2**”

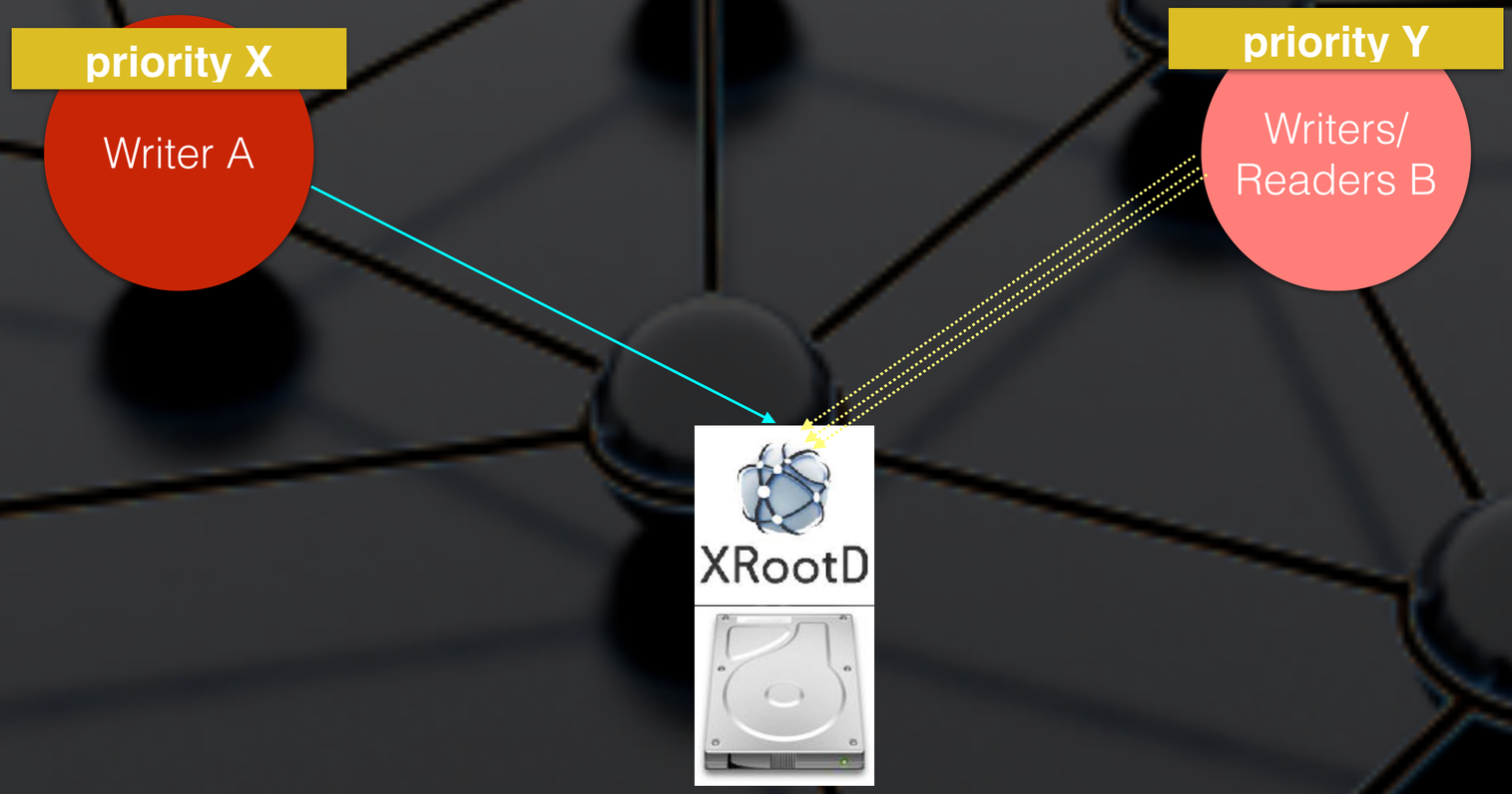
application specific default for app ‘foo’:
“**eos config default space.iopriority:w.app:foo=be:6**”

application specific for group ‘z2’:
“**eos config erasure space.iopriority:w.group:z2=idle:0**”

directory enforced :
“**eos attr set sys.forced.iopriority:w=be:1 /eos/daq/**”

IO Priority - How well does it work?

In this artificial measurement we measure how streams with a predefined IO priority compete against each other writing to a single disk over network depending on the priority setting. The single disk can deliver around 190 MB/s sequential IO.



IO Priority - How well does it work?

Competing Streams

	Writer A [MB/s]	Writer B [MB/s]	Reader C [MB/s]
rt:0 -	192	0	
rt:0 rt:0	96	96	
rt:1 be:4	166	16	
rt:1 be:7	170	15	
rt:0 be:7	172	13	
rt:0 rt:7	140	35	
be:0 be:7	143	32	
rt:0 be:0	169	15	
rt:0 2Xbe:0	160	18	
rt:0 3Xbe:0	158	22	
rt:0 4Xbe:0	150	25	
rt:0 10Xbe:0	131	50	
rt:0 R(10Xbe:0)	162		13

one rt:0: baseline performance

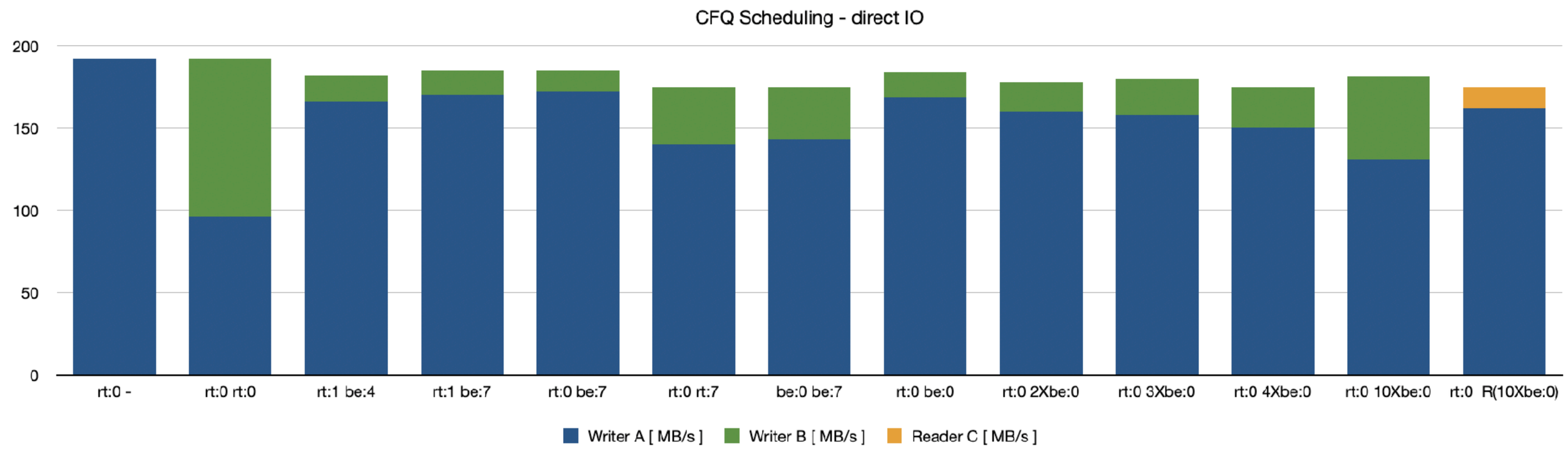
one rt:0 against one rt:0-equal share

one rt:1 against one be:4-rt dominant

one rt:1 against 10 be:0 reader-rt dominant

IO Priority - How well does it work?

realtime streams are very well **suppressing best-effort** streams



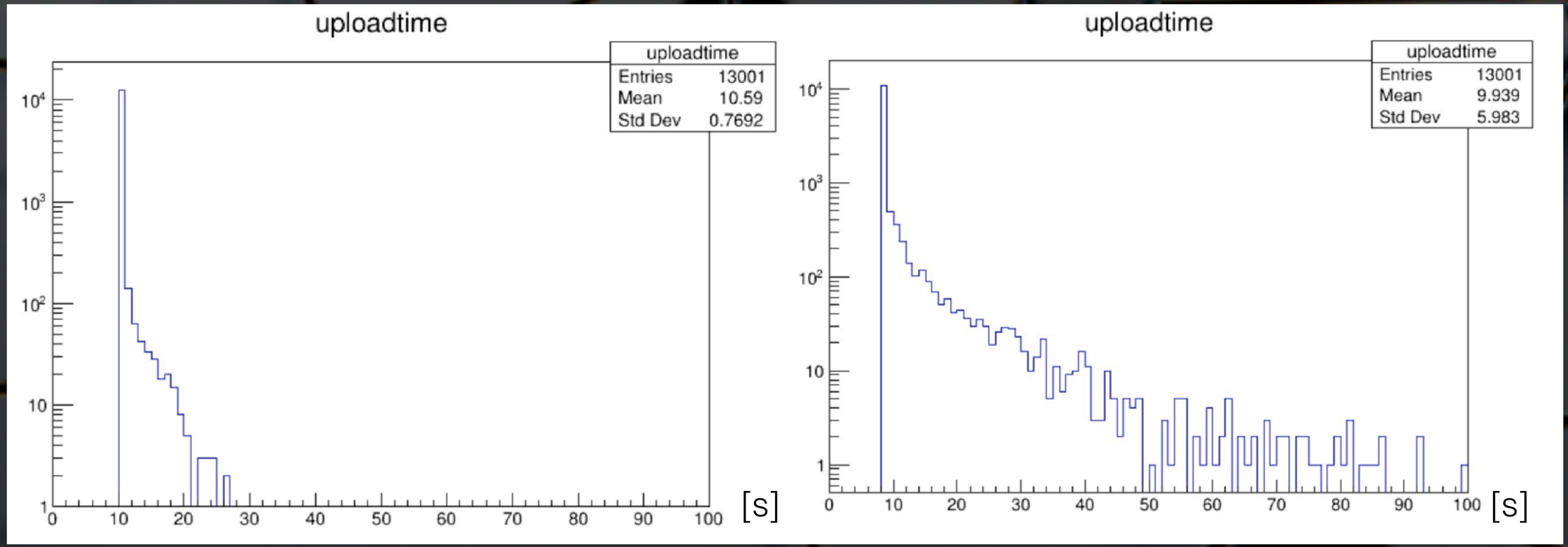
3. Bandwidth Regulation

Bandwidth Regulation

- during various benchmarks we have verified that **IO tails are reduced** when clients run with limited bandwidth
- an upper bandwidth limit can be set for **xrdcp** and **eoscp** (see help of these commands)
- we have added **bandwidth regulation** to EOS **server-side** to be able to set this limit on the instance itself such that we don't have to tell applications/people to use a reasonable setting and to be able to change this settings on the fly
- the bandwidth policy is defined *on open* and then maintained until the file gets closed

Bandwidth Regulation

Impact on performance tails
left with bandwidth limitation - **right** without

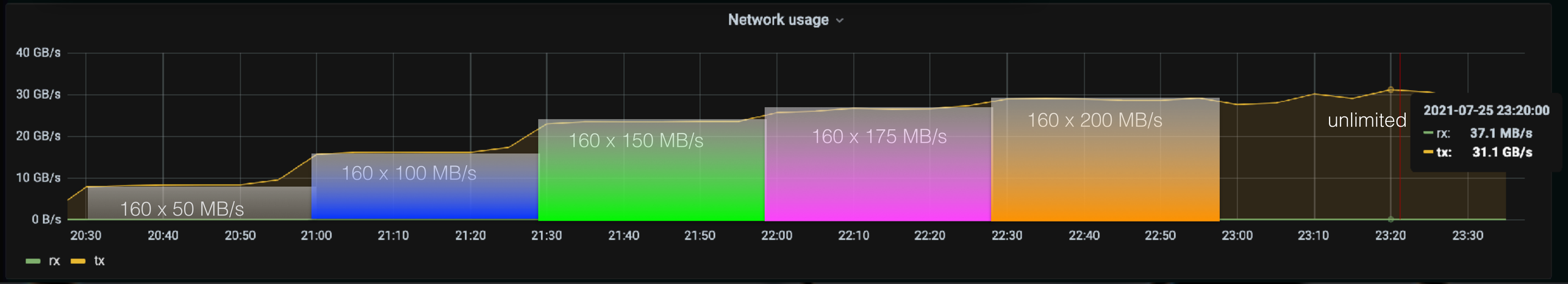


Bandwidth Regulation

Test changing bandwidth limit on the fly
and effect on instance IO performance and tails

bw policy	50	100	150	175	200	unlimited
avg [s]	42	22	15	13.4	12.3	11.9
95 perc	43.7	23.3	16.6	15.6	15.7	40.7
99 perc	44.2	24.5	18.1	20.2	25.4	104
Total BW GB/s	8	16	23	26	28	30

```
success: configured policy in space='default' as policy.bandwidth='50'
success: configured policy in space='default' as policy.bandwidth='100'
success: configured policy in space='default' as policy.bandwidth='150'
success: configured policy in space='default' as policy.bandwidth='175'
success: configured policy in space='default' as policy.bandwidth='200'
success: removed space policy 'policy.bandwidth'
```



Selecting **IO Bandwidth**

The bandwidth parameter unit is MB/s

Evaluation Order

via CGI: **“**root://myeos?eos.iobw=100**”**

instance default:
“**eos config default space.policy.bandwidth:w=250”**

space specific:
“**eos config erasure space.policy.bandwidth:r=150**”

space+application specific for app ‘foo’:
“**eos config erasure space.bandwidth:r.app:foo=50**”

Limitations:
- bandwidth settings are only once defined on open and then valid until a file gets closed

4. Filesystem Scheduling Overload

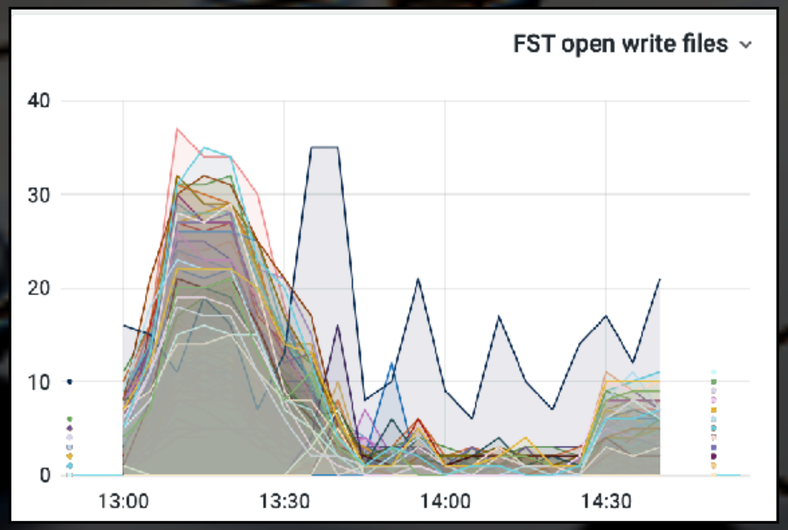
Filesystem Overload

- during data challenges and benchmarking we have observed **black-hole effects on filesystem**

- certain filesystem aggregate streams over time and the overall instance performance is defined/degraded by few overloaded filesystems due to non-linear behaviour transitioning from seq. to random IO performance

- to avoid this we have added an **overload status to filesystem**, which can be triggered when a threshold of max. readers or writers is reached on a filesystem - in this case no new file creations will be scheduled on an overloaded filesystem

Filesystem Overload



 # define when a filesystem is marked as overload

eos space config default **space.max.ropen=200**

eos space config default **space.max.ropen=50**

when the max. number of streams is reached,
the scheduler stops scheduling on this filesystems!

```
EOS Console [root://localhost] |/eos/ajp/> fs ls
```

host	port	id	path	schedgroup	geotag	boot	configstatus	drain	active	health
ajp.cern.ch	1095	17	/ceph/edeeecc1-6aaf-4672-a657-ff8910ca9ed3/fst.00/	cephfs.0	ajp	opserror	drain	failed	online	no smartctl
ajp.cern.ch	1095	18	/ceph/edeeecc1-6aaf-4672-a657-ff8910ca9ed3/fst.01	cephfs.0	ajp	opserror	drain	failed	online	no smartctl
ajp.cern.ch	1095	1	/data/01	default.0	ajp	booted	rw	nodrain	overload	no smartctl
ajp.cern.ch	1095	2	/data/02	default.0	ajp	booted	rw	nodrain	online	no smartctl
ajp.cern.ch	1095	3	/data/03	default.0	ajp	booted	ro	nodrain	online	no smartctl
ajp.cern.ch	1095	4	/data/04	default.0	ajp	booted	ro	nodrain	overload	no smartctl
ajp.cern.ch	1095	11	/data/05	default.0	ajp	booted	ro	nodrain	online	no smartctl
ajp.cern.ch	1095	12	/data/06	default.0	ajp	booted	ro	nodrain	online	no smartctl
ajp.cern.ch	1095	13	/data/07	default.0	ajp	booted	ro	nodrain	online	no smartctl
ajp.cern.ch	1095	14	/data/08	default.0	ajp	booted	ro	nodrain	online	no smartctl
ajp.cern.ch	4001	5	/rain/1/	rain.0			rw	nodrain	offline	
ajp.cern.ch	4002	6	/rain/2/	rain.0			rw	nodrain	offline	
ajp.cern.ch	4003	7	/rain/3/	rain.0			rw	nodrain	offline	
ajp.cern.ch	4004	8	/rain/4/	rain.0			rw	nodrain	offline	

5. Meta-data rate and thread pool limits

Meta-Data Overloads

- **What is the problem?**
  - EOS provides file access via redirection from a central namespace service [MGM]
  - the MGM is a multithreaded application and a typical production scenario is

    `n(clients) >> n(threads@mgm)`

    e.g. 30k clients >> 4096 threads

  - when an individual **user launches batch jobs**, it happens often that several **thousand jobs start** at the same time for this user
    - some jobs act like a DOS attack on the namespace service even if they access only few files via /eos
    - this is particular **problematic in** non-physics instances like **CERNBOX**, where people rely on interactive usability

- **What can we do?**
  - rate-limit meta-data access by user
  - limit the number of worker threads per user

Namespace Statistics

eos ns stat displays rates for all MGM operations - can be broken down by user [-a]

who	command	sum	5s	1min	5min	1h	exec(ms)	sigma(ms)	99p(ms)	max(ms)
all	Access	229.06 M	335.75	304.76	361.76	349.10	-NA-	-NA-	-NA-	-NA-
all	AccessControl	295	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	AdjustReplica	121.99 K	0.25	0.12	0.08	0.09	2.47	11.93	89.30	76.92
all	AttrGet	5.26 M	10.00	9.47	8.36	9.22	0.21	1.01	9.84	3.03
all	AttrLs	399.95 M	509.75	488.20	611.89	649.06	0.04	0.01	0.08	0.07
all	AttrRm	549	0.00	0.00	0.00	0.00	0.47	2.41	24.32	2.64
all	AttrSet	100.29 K	0.00	0.02	0.03	0.15	1.28	4.07	22.16	19.70
all	Cd	1	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	Checksum	0	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	Chmod	60.51 K	0.00	0.17	0.10	0.19	0.50	1.27	11.64	5.57
all	Chown	3.55 K	0.00	0.02	0.01	0.01	-NA-	-NA-	-NA-	-NA-
all	Commit	32.90 M	118.75	93.51	94.69	62.74	0.72	0.63	4.27	2.98
all	CommitFailedFid	0	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	CommitFailedNamespace	0	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	CommitFailedParameters	0	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	CommitFailedUnlinked	512.84 K	0.00	0.00	0.00	0.42	-NA-	-NA-	-NA-	-NA-
all	ConversionDone	0	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	ConversionFailed	0	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	CopyStripe	6.65 K	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	DrainCentralFailed	1.89 K	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	DrainCentralStarted	2.81 M	0.00	0.00	0.00	0.04	-NA-	-NA-	-NA-	-NA-
all	DrainCentralSuccessful	2.81 M	0.00	0.00	0.00	0.04	-NA-	-NA-	-NA-	-NA-
all	Drop	20.76 M	12.75	7.95	18.15	9.93	0.34	0.34	2.16	1.89
all	DropStripe	16	0.00	0.00	0.00	0.00	0.40	0.50	2.16	1.10
all	DumpMd	827	0.00	0.00	0.00	0.00	-NA-	-NA-	-NA-	-NA-
all	EAccess	31.45 M	49.00	46.05	49.43	29.92	-NA-	-NA-	-NA-	-NA-

…

Access Limits

eos access allows to view and set access rate limits [Hz]

```
[ 01 ]                  rate:user:*:Chmod => 500
[ 02 ]                  rate:user:*:Chown => 500
[ 03 ] rate:user:*:Eosxd::ext::LS-Entry => 10000
[ 04 ]      rate:user:*:Eosxd::ext::SET => 50
[ 05 ] rate:user:*:Eosxd::int::FillFileMD => 8000
[ 06 ]              rate:user:*:OpSetFile => 300
[ 07 ]                   rate:user:*:Open => 500
[ 08 ]         rate:user:*:OpenDir-Entry => 20000
[ 09 ]               rate:user:*:OpenProc => 200
[ 10 ]               rate:user:*:OpenRead => 500
[ 11 ]              rate:user:*:OpenWrite => 300
[ 12 ]                     rate:user:*:Rm => 100
[ 13 ]                   rate:user:*:Stat => 2000
[ 14 ]     rate:user:foo:Eosxd::ext::LS => 1
[ 15 ] rate:user:foo:Eosxd::ext::LS-Entry => 10
[ 16 ]            rate:user:bar:AttrLs => 50
[ 17 ]           rate:user:bar:Stat => 50
```

Application of Limits

eos ns stat | grep Stall shows functions where limits are applied and their rates

```
[root@eoshome-i01 (mgm:master mq:master) ~]$ eos ns stat | grep Stall
 all OpenStalled                            0     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
 all Stall                                  0     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
 all Stall::AttrLs                         44     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
 all Stall::Eosxd::ext::LS-Entry    80.57 K     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
 all Stall::Open                     1.74 M    10.00     0.00     0.00     0.23     -NA-     -NA-     -NA-     -NA-
 all Stall::OpenProc               100.59 K     0.00     0.00     0.00     0.68     -NA-     -NA-     -NA-     -NA-
 all Stall::OpenRead               474.33 K     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
 all Stall::Rm                      41.17 K     0.00     0.00     0.00     0.01     -NA-     -NA-     -NA-     -NA-
 all Stall::Stat                     7.06 K     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
 all Stall::threads::77965              119     0.00     0.00     0.00     0.00     -NA-     -NA-     -NA-     -NA-
```

To figure out who is being stalled, one can add the -a option:

eos ns stat -a | grep Stall

Thread Limits

eos ns stat shows the current status of the thread pool, its limits and who is using it

uid	threads	sessions	limit	stalls	stalltime	status
0	1	3	500	0	22	user-OK
2	1	0	500	0	1	user-OK
83***	1	0	500	0	1	user-OK
120***	1	0	500	0	1	user-OK

The thread pool limit is configured as a rate limit using the access interface

eos access ls l grep threads

```
[ 16 ]                          threads:* => 500
```

Thread Limits

There are three types of thread limit rules:

1. **wild-card** rules for all users
2. **specific** user rules
3. **global** thread limit for user requests

```
threads:* => 500
threads:cmsprod => 2000
threads:max => 3900
```

The global thread limit is useful to reserve a given amount of threads to EOS components. The global limit only applies for **uid>3** and excludes the restic backup

Evalutation

How well do meta-data rate & thread-pool limits work?

- service **degradation** has been significantly **decreased**
- we still saw **few episodes** where the configured limits where not sufficient to avoid degradation

- in these cases we ban a user until he corrects his workflow:

```
eos access ban user overloadingusername
```

Summary

- The presented five configuration handle provide a very powerful tool to service administrators to influence the resource sharing and favour time critical workflows - we will apply IO priorities for online workflows

- Artificial tests show, that when network usage is the dominant bottleneck, IO priorities don't help and we need a **dynamic bandwidth policy**
  - we need to change stream weights in real-time to throttle network usage
  - therefore we will add dynamic on-the-fly network (bandwidth) throttling by `user|group|application`, which changes stream bandwidth during individual transfers and a possible optimiser thread for automatisation

- The best would be not to require them, however this can only work if storage resources are never saturated - which is true for the average but not peak usage

https://eos-docs.web.cern.ch/using/policies.html


CERN storage technology
used at the Large Hadron Collider (LHC)
EOS Open Storage
Thank you!
Question or Comments?
eos.web.cern.ch