
GPFS performance session

Sven Oehme – oeemes@us.ibm.com



Latency numbers you NEED to know *

L1 cache reference	1 ns
L2 cache reference	5 ns
Acquire/release mutex	100 ns
Main Memory reference	100 ns
Send 2k byte over verbs	10,000 ns
Send 2k bytes over 1 Gbps network	60,000 ns
Read 1 MB sequentially from Memory	250,000 ns
Read 1 MB sequentially from network	5,000,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send network packet from SJC,CA-> FRA,DE -> SJC,CA	150,000,000 ns

*These numbers are rounded and don't claim to be 100% accurate

Bandwidth numbers you **NEED** to know *

Gigabit	100 MB/sec
10 Gbit	1,000 MB/sec
40 Gbit QDR IB	4,000 MB/sec
56 Gbit FDR IB	5,600 MB/sec
GEN-2 PCI Slot	3,000 MB/sec
GEN-3 PCI Slot	6,000 MB/sec
NL SAS drive 100% Seq Read/Write	100 MB/sec
NL SAS drive 100% 2MB Random Read/Write	75 MB/sec
SSD 100% Seq Read	300 MB/sec
SSD 100% Seq Write	200 MB/sec

*These numbers are rounded and don't claim to be 100% accurate

IOPS numbers you NEED to know *

NL SAS drive 100% 4k Random iops	100
10k SAS drive 100% 4k Random iops	200
SSD 100% 4k random reads	20,000
SSD 100% 4k random writes	4,000

*These numbers are rounded and don't claim to be 100% accurate

How to collect a GPFS trace for performance analysis

GPFS 3.5 TL3 provides a new low overhead Tracing facility - in memory tracing
We had a in Memory tracing before, but it had still large overhead and was now replaced by a new technique

To set Cluster wide in Memory tracing, run :

```
mmtracectl --set --trace=def --tracedev-write-mode=overwrite --tracedev-overwrite-buffer-size=1g
```

You can now turn on tracing :

```
mmtracectl --start
```

Stop tracing, while leave the settings in place :

```
mmtracectl --stop
```

Or turn tracing off and reset to non in-memory tracing :

```
mmtracectl --off
```

Analyze traces – create reports

As an example my traces are stored in /var/log/gpfstraces/
And my GPFS clone directory is /xcat/oehmes/gpfs-clone

Now run the following commands :

```
# /xcat/oehmes/gpfs-clone/tools/trcio -W -T -F -L -d /var/log/gpfstraces/ -o trcio.output
Reading directory /var/log/gpfstraces
Found 1 files matching ['trcrpt.*']
Processing 1 of 1: trcrpt.130714.16.56.30.sonas04n1.gz
bad trace entry: Using invariant time cycle counter at 2499.996000Mhz to calculate timestamps
bad trace entry: Detected CPU running at 2435.000000 MHz at 1373833975.470050 (10381609864903222).
bad trace entry: Detected CPU running at 2435.000000 MHz at 1373833975.470059 (10381609864925144).
bad trace entry: Detected CPU running at 2499.000000 MHz at 1373835982.360727 (10386627083572363).
bad trace entry: Note that this is a change in CPU speed. (if prior TOD times are within a second of each
other this is a spurious error)
bad trace entry: Detected CPU running at 2498.000000 MHz at 1373835982.362344 (10386627087612460).
bad trace entry: Detected CPU running at 2499.000000 MHz at 1373841806.311670 (10401186937643757).
bad trace entry: Detected CPU running at 2396.000000 MHz at 1373841806.311677 (10401186937660535).
Writing trcio.output
```

Analyze traces – look at the reports

Processed 1 trace files:
 ['0.000000', '13893.154279', 'trcrpt.130714.16.56.30.sonas04n1.gz']
 Total elapsedTime: 13893.154 sec

Total Time (seconds)	Count	Ops/sec	--Time-per-operation-(milli-sec)--				-Cumul.-Med.			
			min	avg	90%	max	(ms)	pcnt		
2415.654	146023	10.5	0.037	16.543	23.771	150.453	18.346	36%	read	'*'
606.029	33875	2.4	0.011	17.890	19.114	1407.950	22.622	4%	send	'*'
30.767	983	0.1	0.039	31.299	99.342	227.816	99.340	10%	write	'*'
16.758	1032	0.1	0.073	16.238	37.210	255.746	35.009	15%	nsd	'*'
0.206	758	0.1	0.001	0.272	0.132	13.952	9.006	1%	dmtx	'*'
0.156	5585	0.4	0.003	0.028	0.033	0.541	0.027	30%	recv	'*'
0.017	36746	2.6	0.000	0.000	0.001	0.013	0.001	22%	tsc	'*'
0.005	476	0.0	0.003	0.011	0.023	0.038	0.013	23%	vnop	'*'
0.001	578	0.0	0.000	0.001	0.002	0.006	0.002	35%	lock	'*'

Total Time (seconds)	Count	Ops/sec	--Time-per-operation-(milli-sec)--				-Cumul.-Med.			
			min	avg	90%	max	(ms)	pcnt		
1645.859	88177	6.3	0.712	18.665	24.245	115.608	19.345	40%	read	'vdiskBuf'
686.816	54824	3.9	0.037	12.528	19.139	150.453	15.294	32%	read	'unknown'
343.764	27898	2.0	0.022	12.322	19.428	93.996	15.429	31%	send	'nspdMsgReadWrite'
262.045	308	0.0	684.291	850.795	1384.841	1407.950	705.956	39%	send	'nspdMsgDiscover'
82.962	2976	0.2	19.182	27.877	33.172	100.497	27.288	43%	read	'data'
23.358	200	0.0	59.854	116.790	194.596	227.816	118.109	35%	write	'data'
16.758	1032	0.1	0.073	16.238	37.210	255.746	35.009	15%	nsd	'processRequest'
5.451	308	0.0	11.072	17.698	20.362	83.842	17.108	41%	write	'vdiskEVLog2'
0.746	369	0.0	0.040	2.023	0.113	57.654	26.016	2%	write	'vdiskFWLog'
0.706	35	0.0	17.121	20.176	20.518	25.195	20.281	48%	write	'vdiskRGDesc'
0.387	12	0.0	26.997	32.260	37.030	37.036	36.234	50%	write	'logData'
0.162	24	0.0	0.004	6.752	9.955	13.952	9.577	29%	dmtx	'VBufFreeMutex <VdiskScrubWorkerThread>'

.....

sdstat Plugin for GPFS

With GPFS 3.5 PTF12 we will add a new sample code plugin for sdstat to the GPFS rpms on Linux

In the meanwhile who has access to the git repository can use it , files are :

```
./ts/util/dstat_gpfsops.py.dstat.0.6  
./ts/util/dstat_gpfsops.py.dstat.0.7
```

The extension of the file (0.6 and 0.7) are for the 2 incompatible plugin versions of dstat.
0.6 will work on all older Linux version prior to RHEL 6.1 and version 0.7 will work on all newer versions

0.6 and 0.7 are the versions of dstat reported by `dstat -version`

The version of the plugin needs to be copied into `/usr/share/dstat/` (on RHEL 6.X) and renamed to `dstat_gpfsops.py` like

```
cp ./ts/util/dstat_gpfsops.py.dstat.0.7 /usr/share/dstat/dstat_gpfsops.py
```

After that you can add the plugin to the dstat output by running :

```
dstat -c -n -d -M gpfsops --nocolor
```

This will show cpu , network, disk and GPFS default stats on a single line at 1 second granularity

In order to enable vfs statistics you need to run :

```
mmfsadm vfsstats enable
```

On each node in the cluster (or add to mmfsup file in `/vsr/mmfs/etc/`)

sdstat Plugin for GPFS

Dstat class to display selected gpfs performance counters returned by the mmpmon "vfs_s", "ioc_s", "vio_s", "vflush_s", and "lroc_s" commands.

The set of counters displayed can be customized via environment variables:

DSTAT_GPFS_WHAT

Selects which of the five mmpmon commands to display.

It is a comma separated list of any of the following:

```
"vfs":      show mmpmon "vfs_s" counters
"ioc":      show mmpmon "ioc_s" counters related to NSD client I/O
"nsd":      show mmpmon "ioc_s" counters related to NSD server I/O
"vio":      show mmpmon "vio_s" counters
"vflush":   show mmpmon "vflush_s" counters
"lroc":     show mmpmon "lroc_s" counters
"all":      equivalent to specifying all of the above
```

Example:

```
DSTAT_GPFS_WHAT=vfs,lroc dstat -M gpfsops
```

will display counters for mmpmon "vfs_s" and "lroc" commands.

The default setting is "vfs,ioc", i.e., by default only "vfs_s" and NSD client related "ioc_s" counters are displayed.

For more details on further customization see the dstats_gpfsops.py file

sdstat Plugin for GPFS

Just show VFS level Counters :

```
# DSTAT_GPFS_WHAT=vfs dstat -c -n -d -M gpfsops --nocolor
WARNING: Option -M is deprecated, please use --gpfsops instead
/usr/bin/dstat:1672: DeprecationWarning: os.popen3 is deprecated. Use the subprocess module.
  pipes[cmd] = os.popen3(cmd, 't', 0)
----total-cpu-usage---- -net/total- -dsk/total- -----gpfs-vfs-ops-----
usr sys idl wai hiq siq| recv  send| read  writ| cr   del  op/cl   rd    wr  trunc  fsync  looku  gattr  sattr  other
  0  0  98   1   0   0|   0    0 | 13M   45M|  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 990B 5434B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 740B 4996B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 684B 4590B|   0  16k|  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 628B 5012B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 916B 5346B|   0  60k|  0   0   0     0    0   0     0     0     0     0
```

Just show the vDISK counters :

```
# DSTAT_GPFS_WHAT=vio dstat -c -n -d -M gpfsops --nocolor
WARNING: Option -M is deprecated, please use --gpfsops instead
/usr/bin/dstat:1672: DeprecationWarning: os.popen3 is deprecated. Use the subprocess module.
  pipes[cmd] = os.popen3(cmd, 't', 0)
----total-cpu-usage---- -net/total- -dsk/total- -----gpfs-vio-----
usr sys idl wai hiq siq| recv  send| read  writ| ClRea ClShW ClMdW ClPFT ClFTW FlUpW FlPFT Migrt Scrub  LgWr
  0  0  98   1   0   0|   0    0 | 13M   45M|  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0|1454B 5986B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 564B 5170B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 684B 4900B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 864B 4654B|   0    0 |  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 740B 4980B|   0  64k|  0   0   0     0    0   0     0     0     0     0
  0  0 100   0   0   0| 680B 4574B|   0    0 |  0   0   0     0    0   0     0     0     0     0
```

Perf top – Tool to find the high CPU contender

If you start perf top without parameters, it gives you a top CPU consuming processes of the system in real time and show a relative % compared to others.

```

23.06% mmfsd          [.] rsD10T2_8_9_low_vector_cksum(void**, CK64State*, int)
16.42% mmfsd          [.] rsD10T2_8_9_high_vector_cksum(void**, CK64State*, int)
 9.22% libmlx4-rdmav2.so [.] 0x00000000000003592
 3.27% [mmfslinux]     [k] cxiGetPagePtrs
 2.82% [kernel]       [k] _spin_lock_irqsave
 1.36% [kernel]       [k] schedule
 1.21% [kernel]       [k] _spin_unlock_irqrestore
 1.09% mmfsd          [.] VTrackDesc::cleanBuffersAndBitmaps(VIORequest*)
 1.02% [kernel]       [k] _spin_lock
 0.99% [kernel]       [k] fget_light
 0.93% [mmfslinux]     [k] cxiStartIO
 0.80% mmfsd          [.] VDataBuf::vPrebuildBufferTrailer(int)
 0.73% mmfsd          [.] NotGlobalMutexClass::acquire()
 0.72% mmfsd          [.] Checksum16::calc16(void const*, int)
 0.63% [kernel]       [k] fput

```

...

Perf top – Tool to find the high CPU contender

You can further zoom into a process (in this example mmfsd) and see a breakdown of cpu chewing functions :

Samples: 269K of event 'cycles', Event count (approx.): 126730391715, Thread: mmfsd(344422), DSO: mmfsd

```

39.51%  [.] rsD10T2_8_9_low_vector_cksum(void**, CK64State*, int)           < GNR checksum code
28.37%  [.] rsD10T2_8_9_high_vector_cksum(void**, CK64State*, int)
 1.92%  [.] VTrackDesc::cleanBuffersAndBitmaps(VIORequest*)
 1.34%  [.] VDataBuf::vPrebuildBufferTrailer(int)
 1.28%  [.] NotGlobalMutexClass::acquire()
 1.25%  [.] Checksum16::calc16(void const*, int)
 1.00%  [.] VTrackDesc::buildBufferTrailers(BufBitmap const&, VDataBuf**)
 0.91%  [.] IOBundle::queueIOBuffer(VDataBuf*, int, int, int)
 0.86%  [.] VIORequest::performPromotedFTWrite()
 0.85%  [.] verbs::verbsServer_i(int, RpcContext*, NodeAddr, int, unsigned int, int, int, nsdRdmaRmr_s*, int, iovec*, long long, long long)
 0.84%  [.] VDataBuf::vGetDataAddrAtOffset(int, int) const
 0.82%  [.] ChunkTab::findChunk(char*, unsigned long)
 0.81%  [.] VDataBuf::vBuildBufferTrailer(int)
 0.77%  [.] verbs::verbsDtoThread_i(int)
 0.74%  [.] VDataBuf::vHold()
 0.69%  [.] IncrementalChecksumState::ickAccumulate(void const*, int)
 0.66%  [.] VTrackDesc::prepareToBuildTrailers(VIORequest*)
 0.66%  [.] VTrackDesc::vtUpdateTrailerVersions(VIORequest*)
 0.66%  [.] ThCond::wait(int, char const*)
 0.63%  [.] VIORequest::vioReshapeFreeBuffers(int)
 0.63%  [.] VMemHandle::mGetVAddr() const
 0.60%  [.] VTrackDesc::vtProcessWriteIOBundleStatus(IOBundle*, VIORequest*, BufBitmap*)
...

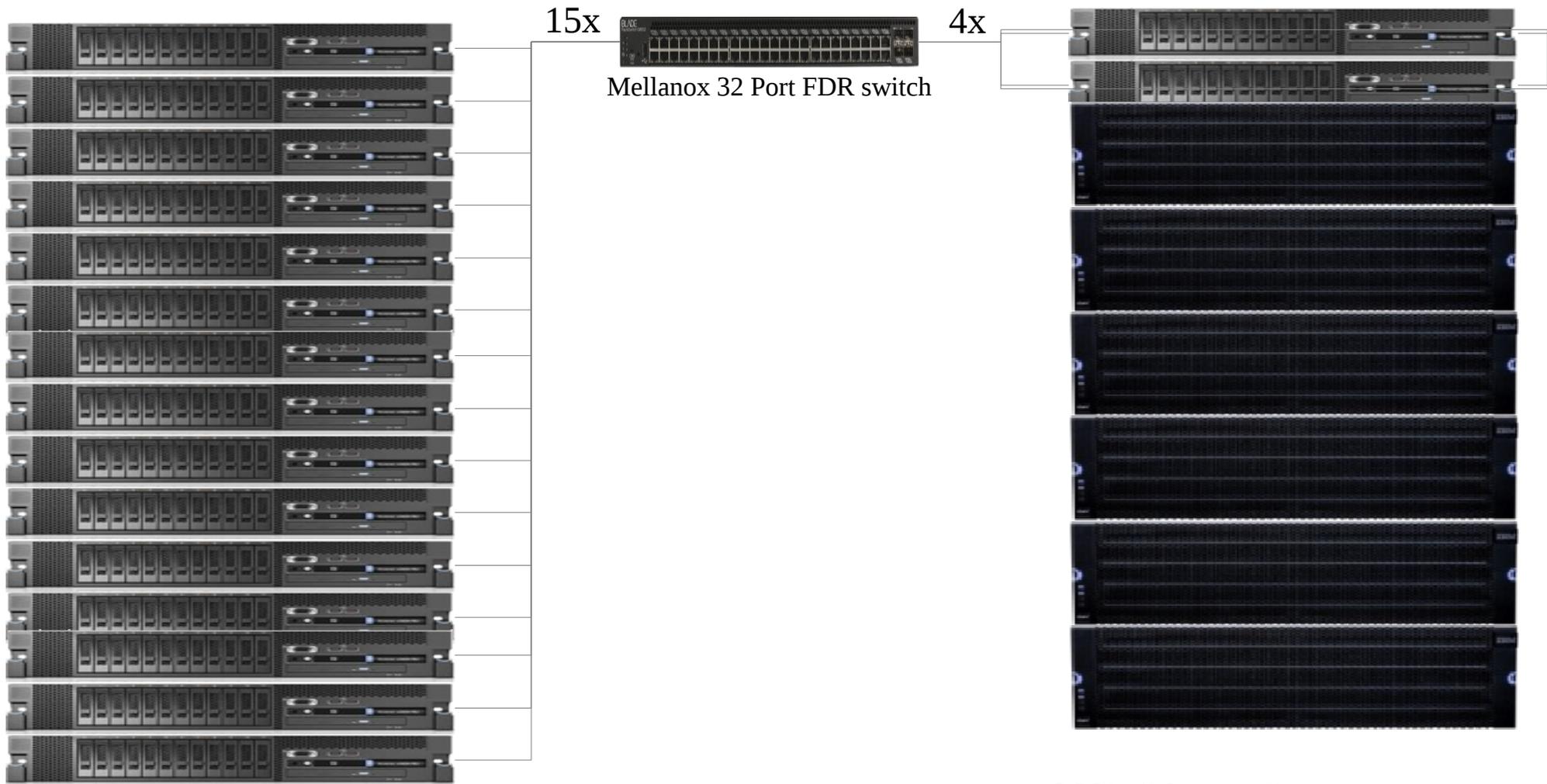
```

Performance data

Non of the following Performance numbers should be reused for sales or contract purposes.

Some of the numbers produced are a result of very advanced tuning and while achievable, not very easy to recreate at customer systems without the same level of effort

A word of caution : The achieved numbers depends on the right Client configuration and good Interconnect and can vary between environments. They should not be used in RFI's as committed numbers, rather to demonstrate the technical capabilities of the Product in good conditions



15 x3550-M3 Server each with
16 GB of Memory (6 gb Pagepool)
1 FDR Port
1 x 6 core CPU

1 GSS24/26 depending on the test.
2 FDR Ports connected per Server
GPFS 3.5.0.7 GA code level

Apply these numbers to practice

Creating a single 10 Gbyte File from one Client using a GEN-2 FDR IB card

```
# /usr/local/bin/gpfsperf create seq -n 10G -r 8m /ibm/fs2-8m/test-10g-write
/usr/local/bin/gpfsperf create seq /ibm/fs2-8m/test-10g-write
  recSize 8M nBytes 10G fileSize 10G
  nProcesses 1 nThreadsPerProcess 1
  file cache flushed before test
  not using data shipping
  not using direct I/O
  offsets accessed will cycle through the same file segment
  not using shared memory buffer
  not releasing byte-range token after open
  no fsync at end of test
  Data rate was 3268199.54 Kbytes/sec, iops was 398.95, thread utilization 0.984
  Record size: 8388608 bytes, 10737418240 bytes to transfer, 10737418240 bytes transferred
  CPU utilization: user 3.68%, sys 3.87%, idle 92.45%, wait 0.00%
```

Why didn't it run at 5.6 GB/sec ? GEN - 2

Apply these numbers to practice

Reading a single block random from this 10 Gbyte File while it is not cached anymore

```
# /usr/local/bin/gpfsperf read rand -n 8m -r 8m /ibm/fs2-8m/test-10g-write
/usr/local/bin/gpfsperf read rand /ibm/fs2-8m/test-10g-write
  recSize 8M nBytes 8M fileSize 10G
  nProcesses 1 nThreadsPerProcess 1
  file cache flushed before test
  not using data shipping
  not using direct I/O
  offsets accessed will cycle through the same file segment
  not using shared memory buffer
  not releasing byte-range token after open
  Data rate was 220322.52 Kbytes/sec, iops was 26.89, thread utilization 0.989
  Record size: 8388608 bytes, 8388608 bytes to transfer, 8388608 bytes transferred
  CPU utilization: user 0.00%, sys 0.00%, idle 100.00%, wait 0.00%
[root@clients.sonascl16 mpi]# mmdiag --iohist
```

=== mmdiag: iohist ===

I/O history:

I/O start time	RW	Buf type	disk:sectorNum	nSec	time ms	Type	Device/NSD ID	NSD server
10:04:45.979047	R	data	10:40527921152	16384	33.646	cli	C0A70402:51E1B12C	192.167.4.2

Apply these numbers to practice

Reading 8 blocks sequentially from this 10 Gbyte File while it is not cached anymore

```
# /usr/local/bin/gpfsperf read seq -n 64m -r 8m /ibm/fs2-8m/test-10g-write
/usr/local/bin/gpfsperf read seq /ibm/fs2-8m/test-10g-write
recSize 8M nBytes 64M fileSize 10G
nProcesses 1 nThreadsPerProcess 1
file cache flushed before test
not using data shipping
not using direct I/O
offsets accessed will cycle through the same file segment
not using shared memory buffer
not releasing byte-range token after open
Data rate was 533933.24 Kbytes/sec, iops was 65.18, thread utilization 0.901
Record size: 8388608 bytes, 67108864 bytes to transfer, 67108864 bytes transferred
CPU utilization: user 0.68%, sys 0.68%, idle 98.64%, wait 0.00%
[root@clients.sonascl16 mpi]# mmdiag --iohist
```

=== mmdiag: iohist ===

I/O history:

I/O start time	RW	Buf type	disk:sectorNum	nSec	time ms	Type	Device/NSD ID	NSD server
10:06:24.664878	R	data	11:69057658880	16384	27.807	cli	C0A70402:51E1B12D	192.167.4.2
10:06:24.664878	R	data	10:81416355840	16384	34.372	cli	C0A70402:51E1B12C	192.167.4.2
10:06:24.704199	R	data	7:153683312640	16384	26.884	cli	C0A70401:51E1B11C	192.167.4.1
10:06:24.701967	R	data	12:155233386496	16384	32.577	cli	C0A70402:51E1B12E	192.167.4.2
10:06:24.704210	R	data	8:86150365184	16384	32.199	cli	C0A70401:51E1B11D	192.167.4.1
10:06:24.737230	R	data	9:170315186176	16384	30.527	cli	C0A70401:51E1B11E	192.167.4.1
10:06:24.741580	R	data	8:71319928832	16384	30.542	cli	C0A70401:51E1B11D	192.167.4.1
10:06:24.737237	R	data	10:144131506176	16384	37.358	cli	C0A70402:51E1B12C	192.167.4.2
10:06:24.739233	R	data	12:9777807360	16384	36.375	cli	C0A70402:51E1B12E	192.167.4.2
10:06:24.739203	R	data	11:91428995072	16384	37.607	cli	C0A70402:51E1B12D	192.167.4.2
10:06:24.741588	R	data	7:171111170048	16384	40.440	cli	C0A70401:51E1B11C	192.167.4.1

Apply these numbers to practice

Reading 8 blocks sequentially from this 10 Gbyte File while it is **STILL** cached

```
# /usr/local/bin/gpfsperf read seq -n 64m -r 8m /ibm/fs2-8m/test-10g-write
/usr/local/bin/gpfsperf read seq /ibm/fs2-8m/test-10g-write
recSize 8M nBytes 64M fileSize 10G
nProcesses 1 nThreadsPerProcess 1
file cache flushed before test
not using data shipping
not using direct I/O
offsets accessed will cycle through the same file segment
not using shared memory buffer
not releasing byte-range token after open
  Data rate was 3286165.76 Kbytes/sec, iops was 401.14, thread utilization 0.980
  Record size: 8388608 bytes, 67108864 bytes to transfer, 67108864 bytes transferred
  CPU utilization: user 0.00%, sys 4.08%, idle 95.92%, wait 0.00%
[root@clients.sonascl16 mpi]# mmdiag --iohist
```

=== mmdiag: iohist ===

I/O history:

I/O	start	time	RW	Buf	type	disk:sectorNum	nSec	time	ms	Type	Device/NSD ID	NSD	server
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
#													

Apply these numbers to practice

Reading the whole file sequentially

```
# /usr/local/bin/gpfsperf read seq -n 10g -r 8m /ibm/fs2-8m/test-10g-write
/usr/local/bin/gpfsperf read seq /ibm/fs2-8m/test-10g-write
recSize 8M nBytes 10G fileSize 10G
nProcesses 1 nThreadsPerProcess 1
file cache flushed before test
not using data shipping
not using direct I/O
offsets accessed will cycle through the same file segment
not using shared memory buffer
not releasing byte-range token after open
Data rate was 2330376.56 Kbytes/sec, iops was 284.47, thread utilization 1.000
Record size: 8388608 bytes, 10737418240 bytes to transfer, 10737418240 bytes transferred
CPU utilization: user 3.01%, sys 3.24%, idle 93.75%, wait 0.00%
```

=== mmdiag: iohist ===

I/O history:

I/O start time	RW	Buf type	disk:sectorNum	nSec	time ms	Type	Device/NSD ID	NSD server
10:34:02.345925	R	data	8:167131250688	16384	34.668	cli	C0A70401:51E1B11D	192.167.4.1
10:34:02.345925	R	data	9:107683823616	16384	37.434	cli	C0A70401:51E1B11E	192.167.4.1
10:34:02.387112	R	data	11:69057658880	16384	30.356	cli	C0A70402:51E1B12D	192.167.4.2
10:34:02.384108	R	data	10:81416355840	16384	35.265	cli	C0A70402:51E1B12C	192.167.4.2
10:34:02.387031	R	data	12:155233386496	16384	34.376	cli	C0A70402:51E1B12E	192.167.4.2
10:34:02.422283	R	data	8:86150365184	16384	31.123	cli	C0A70401:51E1B11D	192.167.4.1
10:34:02.424731	R	data	10:144131506176	16384	29.495	cli	C0A70402:51E1B12C	192.167.4.2
10:34:02.422251	R	data	7:153683312640	16384	32.435	cli	C0A70401:51E1B11C	192.167.4.1
10:34:02.424731	R	data	9:170315186176	16384	34.049	cli	C0A70401:51E1B11E	192.167.4.1
.....								

Apply these numbers to practice

See that the data was prefetched which is why the response time per request is lower :

```
mmfsadm dump iohist
```

I/O history:

I/O start time	RW	Buf type	disk:sectorNum	nSec	time ms	tag1	tag2	Disk UID	typ
NSD server context		thread							
10:34:47.148582	R	data	9:107683823616	16384	30.611	2295808	1	C0A70401:51E1B130	cli
192.167.4.1	Prefetch	PrefetchWorkerThread							
10:34:47.148590	R	data	8:167131250688	16384	51.180	2295808	0	C0A70401:51E1B12A	cli
192.167.4.1	MBHandler	FileBlockReadFetchHandlerThread							
10:34:47.204880	R	data	11:69057658880	16384	27.887	2295808	3	C0A70401:51E1B12D	cli
192.167.4.2	Prefetch	PrefetchWorkerThread							
10:34:47.202549	R	data	10:81416355840	16384	36.348	2295808	2	C0A70401:51E1B12E	cli
192.167.4.2	Prefetch	PrefetchWorkerThread							
10:34:47.204888	R	data	12:155233386496	16384	34.017	2295808	4	C0A70401:51E1B12C	cli
192.167.4.2	Prefetch	PrefetchWorkerThread							
10:34:47.244035	R	data	10:144131506176	16384	32.866	2295808	8	C0A70401:51E1B12E	cli
192.167.4.2	Prefetch	PrefetchWorkerThread							
10:34:47.241839	R	data	7:153683312640	16384	35.732	2295808	5	C0A70401:51E1B128	cli
192.167.4.1	Prefetch	PrefetchWorkerThread							
10:34:47.241839	R	data	8:86150365184	16384	37.547	2295808	6	C0A70401:51E1B12A	cli
192.167.4.1	Prefetch	PrefetchWorkerThread							
10:34:47.246567	R	data	12:9777807360	16384	33.634	2295808	10	C0A70401:51E1B12C	cli
192.167.4.2	Prefetch	PrefetchWorkerThread							
10:34:47.246567	R	data	11:91428995072	16384	36.871	2295808	9	C0A70401:51E1B12D	cli
192.167.4.2	Prefetch	PrefetchWorkerThread							
10:34:47.243946	R	data	9:170315186176	16384	43.591	2295808	7	C0A70401:51E1B130	cli
192.167.4.1	Prefetch	PrefetchWorkerThread							

Benchmark execution and results

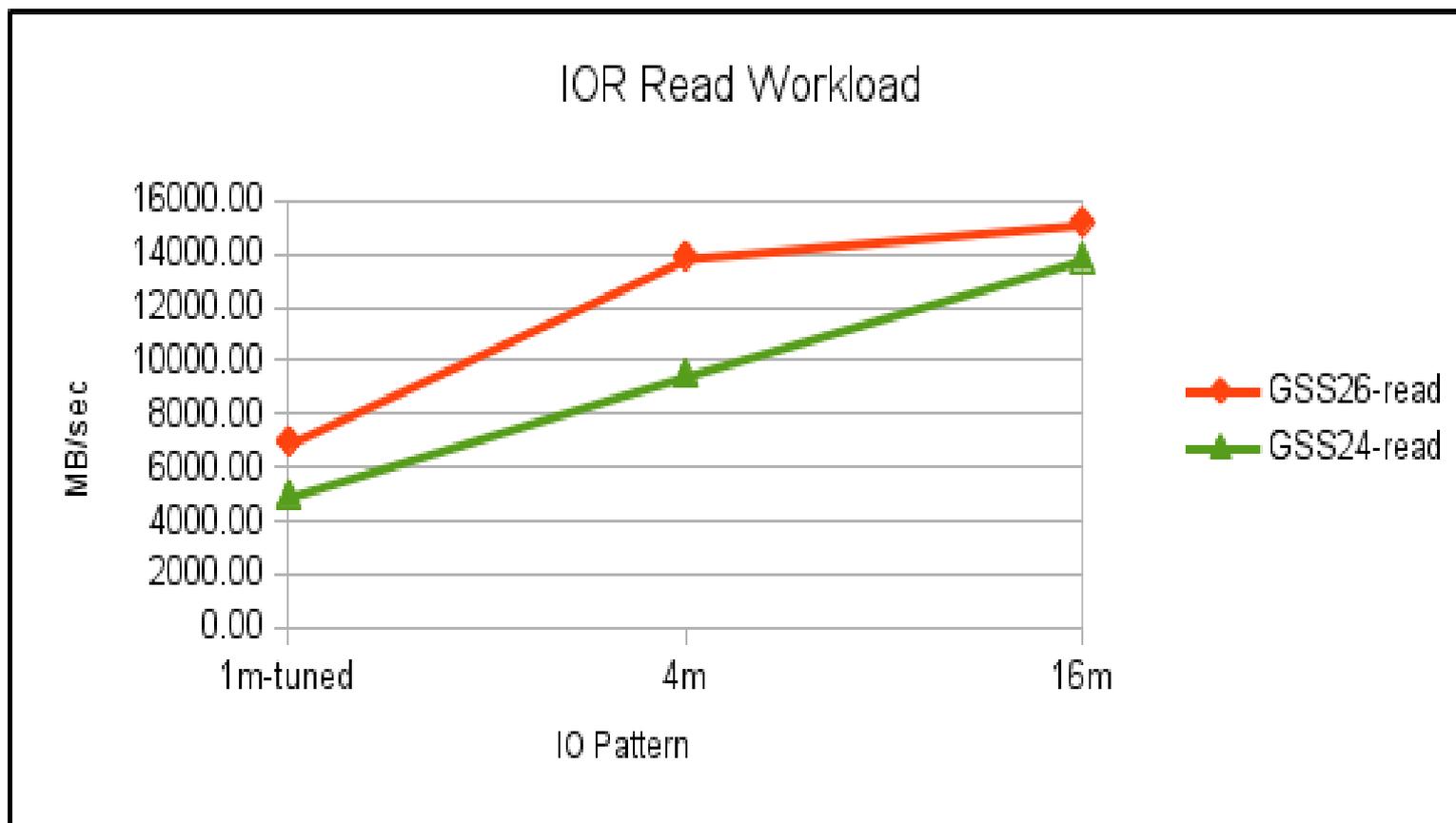


```
ior -i 2 -p -d 10 -w -r -e -t 16m -b 32G -o /ibm/fs2-16m/shared/ior//iorfile

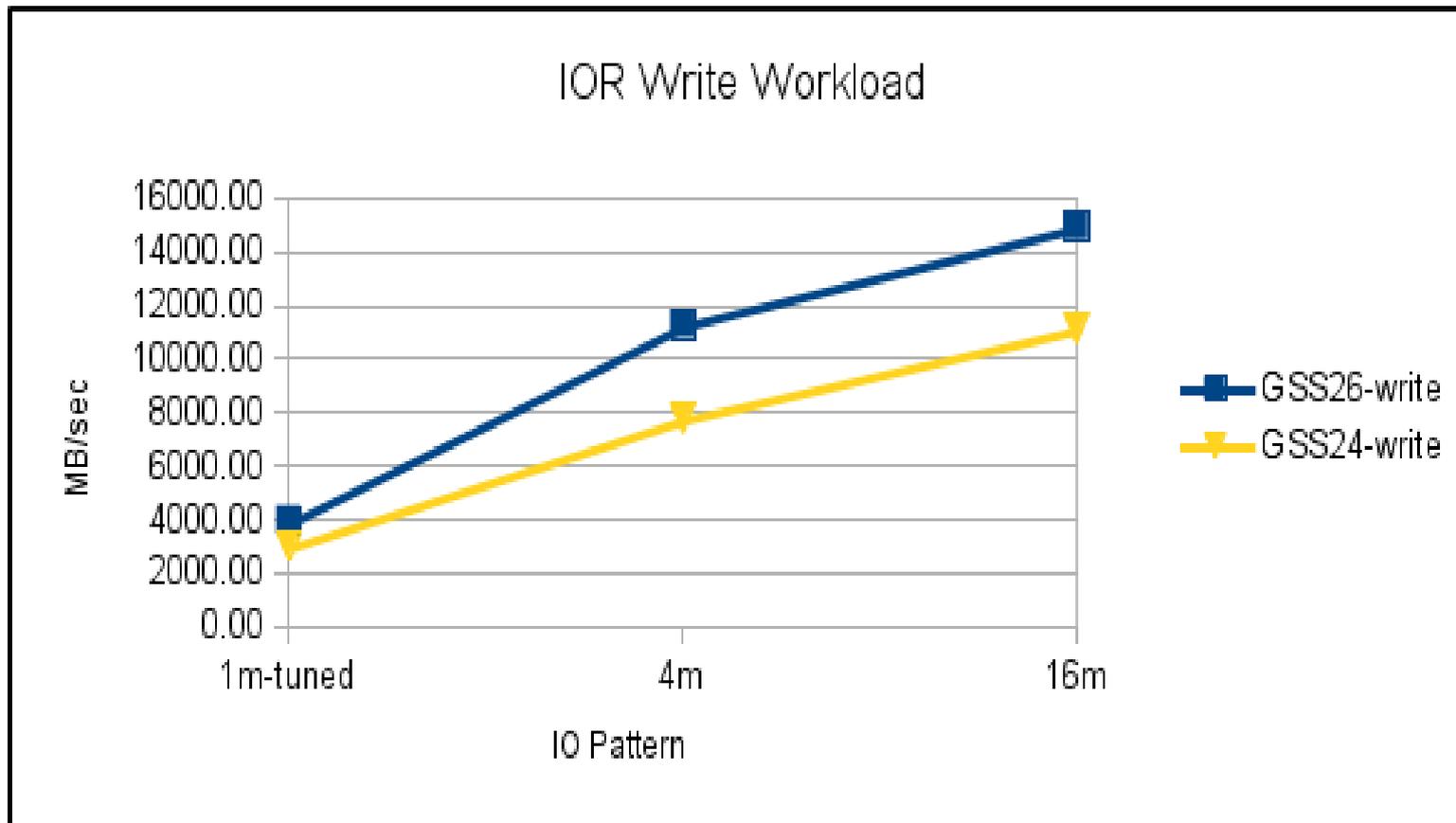
-i N repetitions -- number of repetitions of test
-d N interTestDelay -- delay between reps in seconds
-w writeFile -- write file
-r readFile -- read existing file
-e fsync -- perform fsync upon POSIX write close
-t N transferSize -- size of transfer in bytes (e.g.: 8, 4k, 2m, 1g)
-b N blockSize -- contiguous bytes to write per task (e.g.: 8, 4k, 2m, 1g)
-o S testFile -- full name for test
```

Operation	1m	4m	16m
GSS26-write (MB/sec)	3957.30	11302.19	14970.40
GSS26-read (MB/sec)	6987.58	13915.36	15193.71
GSS24-write (MB/sec)	3023.23	7799.26	11148.37
GSS24-read (MB/sec)	4987.02	9515.66	13875.70

A word of caution : The achieved numbers depends on the right Client configuration and good Interconnect and can vary between environments. They should not be used in RFI's as committed numbers, rather to demonstrate the technical capabilities of the Product in good conditions



A word of caution : The achieved numbers depends on the right Client configuration and good Interconnect and can vary between environments. They should not be used in RFI's as committed numbers, rather to demonstrate the technical capabilities of the Product in good conditions



A word of caution : The achieved numbers depends on the right Client configuration and good Interconnect and can vary between environments. They should not be used in RFI's as committed numbers, rather to demonstrate the technical capabilities of the Product in good conditions

GPFS Parameters explained

General Parameter

Modern Servers have multiple Memory regions that are attached to a given socket. by default Linux allocates data for a given process from only 1 NUMA Region This Parameter tells GPFS to round robin across all regions to not run into a out of memory condition when you reached the limit of one of the regions while the remaining still have plenty of memory left.

mmchconfig numaMemoryInterleave=yes

page pool defines the amount of physical memory that should be pinned by GPFS at startup. it is used in various places of the code, but from a Performance perspective its required to cache data and metadata objects (indirect blocks, directory blocks).

mmchconfig pagepool=38g

Defines the maximum number of Bufferdescriptors. for data block (full block or fragment) or directory block you want to hold in the cache you need to have exactly 1

mmchconfig maxBufferDescs=2m

Percentage of page pool used for file prefetching needs to be less than the default of 20% since most of the page pool was given to GNR.

mmchconfig prefetchPct=5

Allow largest possible GPFS block size and GNR vdisk track size

mmchconfig maxblocksize=16m

Number of recent IOs whose target address and response times are recorded. Default 512.

mmchconfig ioHistorySize=64k

defines no of Multiclass / Non-critical worker threads to be started

mmchconfig maxGeneralThreads=1280

maxMBpS affects the depth of prefetching for sequential file access. It should be set at least as large as the maximum expected hardware bandwidth.

mmchconfig maxMBpS=16000

GPFS Parameters explained

Housekeeping / cache related settings

`syncIntervalStrict` defines if we should only follow the `syncInterval` (default 30) value rather than the main interval of the OS triggered sync, which happens on linux every 5 seconds. this has a very big positive impact on workloads with buffered writes.

`mmchconfig syncIntervalStrict=yes`

These are all about cleaning "files" so `OpenFile` objects can be stolen and re-used. To steal an `OpenFile` object the whole file (data & metadata) must be flushed.

`flushedDataTarget`: no of `OpenFile` objects where data have been flushed already

`flushedInodeTarget`: no of `OpenFile` objects where data & metadata have been flushed

`maxFileCleaners`: no threads flushing data and/or metadata

`mmchconfig flushedDataTarget=1024`

`mmchconfig flushedInodeTarget=1024`

`mmchconfig maxFileCleaners=1024`

These are cleaning data buffers, so sync doesn't have to flush data blocks

`mmchconfig maxBufferCleaners=1024`

Number of GPFS log buffers. Having lots of these allows the log to absorb bursts of log appends. For systems with large page pools (1 G or more), log buffers are the size of the metadata block size, and there is a separate set of such buffers for each file system. Default 3.

`mmchconfig logBufferCount=20`

GPFS log flush controls. When the log becomes `logWrapThresholdPct`, the log flush code is activated to flush dirty objects so the log records that describe their updates can be discarded. This percentage defaults to 50%, and although there is some code to allow changing it, modifying this value is not supported by `mmchconfig`. Log wrap will start `logWrapThreads` flush threads (default 8), which will flush enough dirty objects so the recovery start position can be moved forward by `logWrapAmountPct` percent (default 10%).

`mmchconfig logWrapAmountPct=2`

`mmchconfig logWrapThreads=128`

GPFS Parameters explained

Number of active allocation regions for disk allocation. Larger numbers can improve allocation performance, but high numbers should not be used for large clusters. Default is 4.

mmchconfig maxAllocRegionsPerNode=32

Size of the pool of threads that completes file deletions in the background. Default is 4.

mmchconfig maxBackgroundDeletionThreads=16

Maximum number of threads that prefetch inode tokens of deleted files to speed up file creates. Default is 8.

mmchconfig maxInodeDeallocPrefetch=128

Maximum number of simultaneous local GPFS requests. Default 48.

mmchconfig worker1Threads=1024

maxFilesToCache should be set fairly large to assist with local workload. It can be set very large in small client clusters, but should remain small on clients in large clusters to avoid excessive memory use on the token servers. The stat cache is not effective on Linux, so it should always be small.

mmchconfig maxFilesToCache=128k

mmchconfig maxStatCache=512

Maximum number of threads that prefetch inode tokens of deleted files to speed up file creates. Default is 8.

mmchconfig maxInodeDeallocPrefetch=128

Pre-steal some page pool space to reduce the latency of acquiring a free buffer. preStealCount is the option to specify a hard number vs Pct. the way it works is if set to 10000 , 5000 go to 32k, 2500 to 16k, 1250 to 8k ,

mmchconfig preStealCount=1000

mmchconfig preStealPct=1

GPFS Parameters explained

`syncBackgroundThreads` define how many threads in parallel are allowed to run to flush data during regular sync intervals. Default 16.

`syncWorkerThreads` no of threads in parallele to flush data during explicit sync (sync command, or `crsnapshot`, or `umount`, ...)

`mmchconfig syncBackgroundThreads=64`

`mmchconfig syncWorkerThreads=256`

These Settings influence the inode Prefetch behaviour for "`ls -l`"

`InodePrefectFirstDirblock` set to "yes" to have inode prefetch read the first block of each subdir as well. Defaults to no.

`InodePrefetchThreshold` defines how many stat's we wait for before start prefetching inodes, default is 5, make it smaller to start inode prefetch sooner.

`InodePrefetchWindow` define how close together in time the stat's have to be to trigger inode prefetch, default is 0.5 seconds which means the 5 stat's all have to within half a second of each other, otherwise we'll ignore them. you need to make it larger to trigger inode prefetch even if stat's are coming in more slowly units are in milli seconds.

e.g., setting it to 2500 will make the window be 2.5 seconds

`mmchconfig InodePrefectFirstDirblock=yes`

`mmchconfig InodePrefetchThreshold=5`

`mmchconfig InodePrefetchWindow=500`

General number of inode prefetch threads to use. Default 8.

`mmchconfig worker3Threads=32`

`pitWorkerThreadsPerNode` specify how much threads do restripe, data movement, etc ...

Default is `threadsPerNode = MIN(16, (numberOfDisks * 4)/numberOfNodes + 1)` so 16, or less if there are fewer than about four LUNs

`mmchconfig pitWorkerThreadsPerNode=16`

GPFS Parameters explained

PrefetchAggressiveness defines how aggressive to prefetch data

0 means never prefetch

1 means prefetch on 2nd access if sequential

2 means prefetch on 1st access at offset 0 or 2nd sequential access anywhere else

3 means prefetch on 1st access anywhere

In 3.3, the default was 3 (prefetchOnFirstAccess), which means it would always prefetch immediately, even if the first access is in the middle of the file.

In GPFS 3.4, the default is 2 (prefetchNormal), which means if you start reading at the beginning of the file, it will start prefetching immediately, but if you start reading somewhere in the middle of the file, it waits until the second read to confirm that the access is sequential before it starts prefetching. With the setting of 1 (prefetchOnSecondAccess), it will wait for a second read, even if the first read was at the beginning of the file.

since 3.5 you can specify read and write aggressiveness independent.

mmchconfig prefetchAggressiveness=2

mmchconfig prefetchAggressivenessRead=-1

mmchconfig prefetchAggressivenessWrite=-1

ignorePrefetchLUNCount tells the NSD client to not limit the numbers of requests based on the number of visible LUN's (as they can have a large number of physical disks behind them) and rather limit by the max to number of buffers and prefetch threads. Defaults to no

mmchconfig ignorePrefetchLUNCount=yes

GPFS Parameters explained

Communication Related Parameter

tscWorkerPool defines no of threads per class of receive workers

mmchconfig tscWorkerPool=64

nsdInlineWriteMax defines the maximum allowed single io size to use Inline writes.Defaults to 1k

mmchconfig nsdInlineWriteMax=32k

This needs to be set larger than the default for server nodes that may have connections to many clients, since it indirectly controls the number of TCP connections managed by each receiver thread.

mmchconfig maxReceiverThreads=32

RDMA Port configuration

mmchconfig verbsPorts='mlx4_0/1 mlx4_0/2 mlx4_1/1 mlx4_1/2'

enable RDMA in general, if this is set to disable all RDMA communication is shut off

mmchconfig verbsRdma=enable

defines minimum size of a Packet to use RDMA , also see nsdInlineWriteMax

mmchconfig verbsRdmaMinBytes=16k

Turns verbsSend on, a low level IB inline transfer method

mmchconfig verbsRdmaSend=yes

Max number of outstanding transfers at a time per connection

mmchconfig verbsRdmPerConnection=256

Max number of outstanding transfers at a time for the entire node

mmchconfig verbsRdmPerNode=1024

How much dedicated PAGEpool for verbs communication

mmchconfig verbsSendBufferMemoryMB=1024