Prestazioni di accesso allo storage: confronto di diverse soluzioni hw/sw, bottleneck di rete e di applicazioni

DONVITO Giacinto (See next slide)
Università e INFN -- BARI

Contribution:

- This is a report of several activities carried on by several people
- Main contribution from:
  - Manoj Kumar Jha (Atlas)
  - Brian Bockelman (CMS)

Outlook

Top -> Down

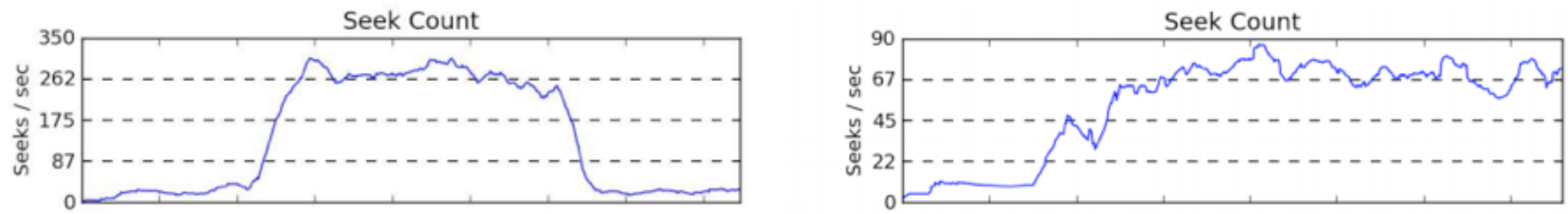
- Starting from applications:
  - Atlas
  - CMS
- Software
  - Storage management software
- Hardware:
  - Disk subsystem:
  - Network infrastructure
- Failures:
  - inefficiencies due to failure of the systems

Working on applications

Atlas Reordered AOD

- Using TTreeCache when reading the files reduces read time by ordering and predicting read requests but introduces one more memory buffer.
- Two more optimization options are to reorder the baskets which can be made according to event (entry) number or according to branch. All ATLAS files produced at Tier0 are re-ordered by event.
- Finally starting with Athena 15.9.0, which uses ROOT 5.26.00.d, we have a possibility to use the ROOT autoFlush functionality.
    - the first time that a preset amount of data has been collected (by default 30 MB) the sizes of baskets are optimized and their content is written to file. The next flush to file happens when the same number of events has been written.

Atlas Reordered AOD



a) unordered AODs b) reordered AODs

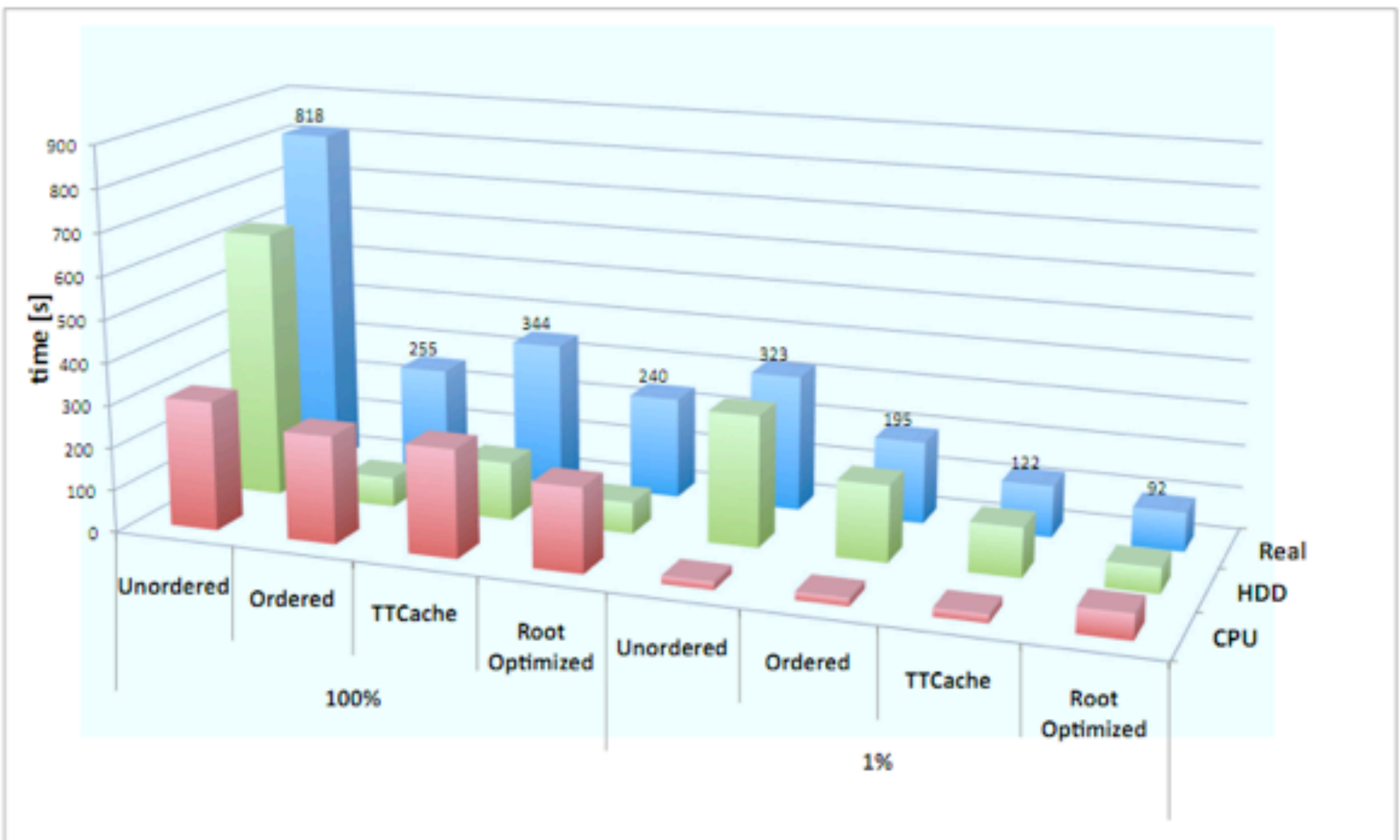


Figure 3. AOD read performance from local disk.

Basket size, zip level, and split level:

baskets of 2 kB for all branches, zip level 6, and full split (level 99) are optimal for our data

Atlas Reordered AOD

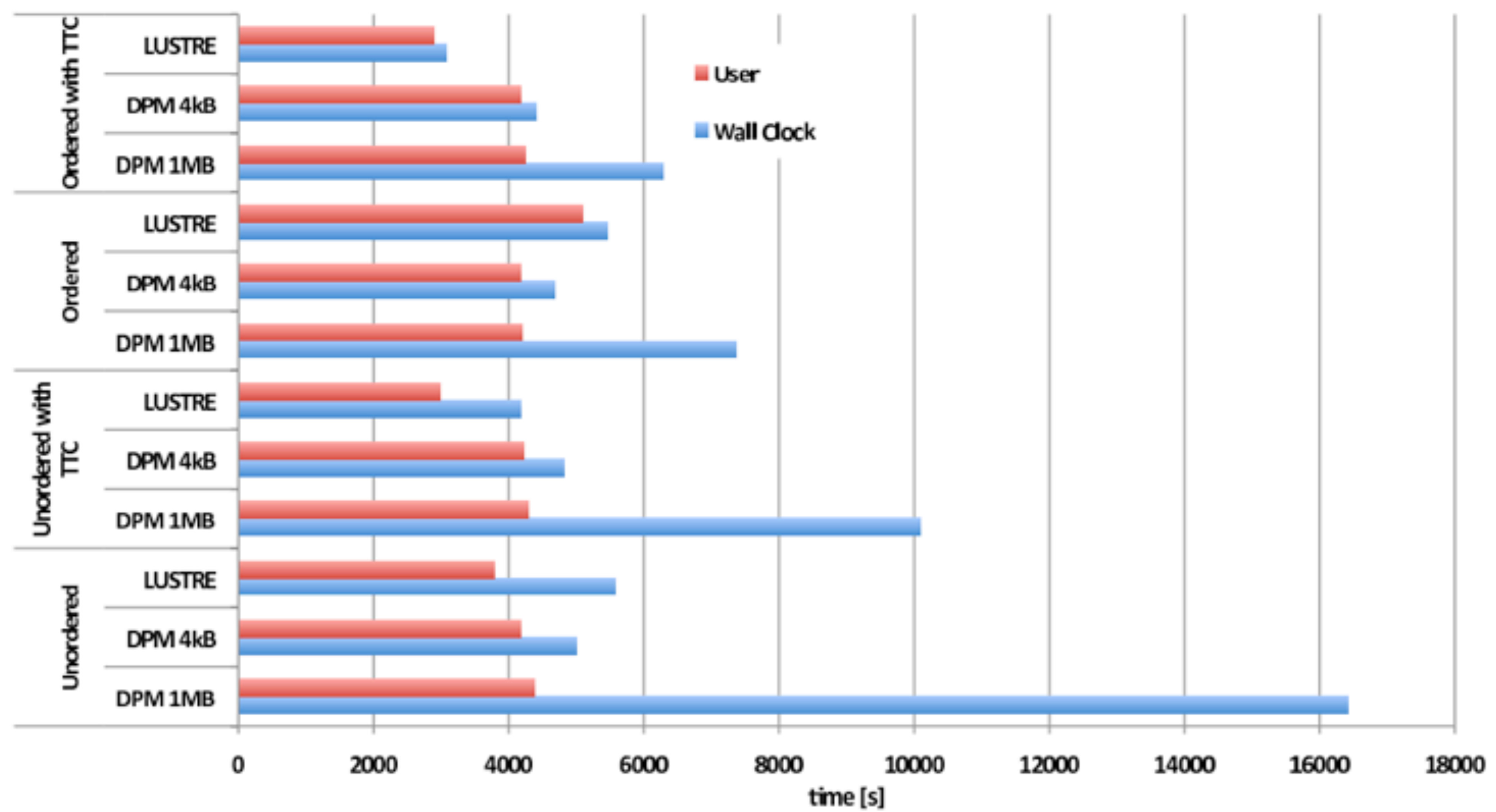
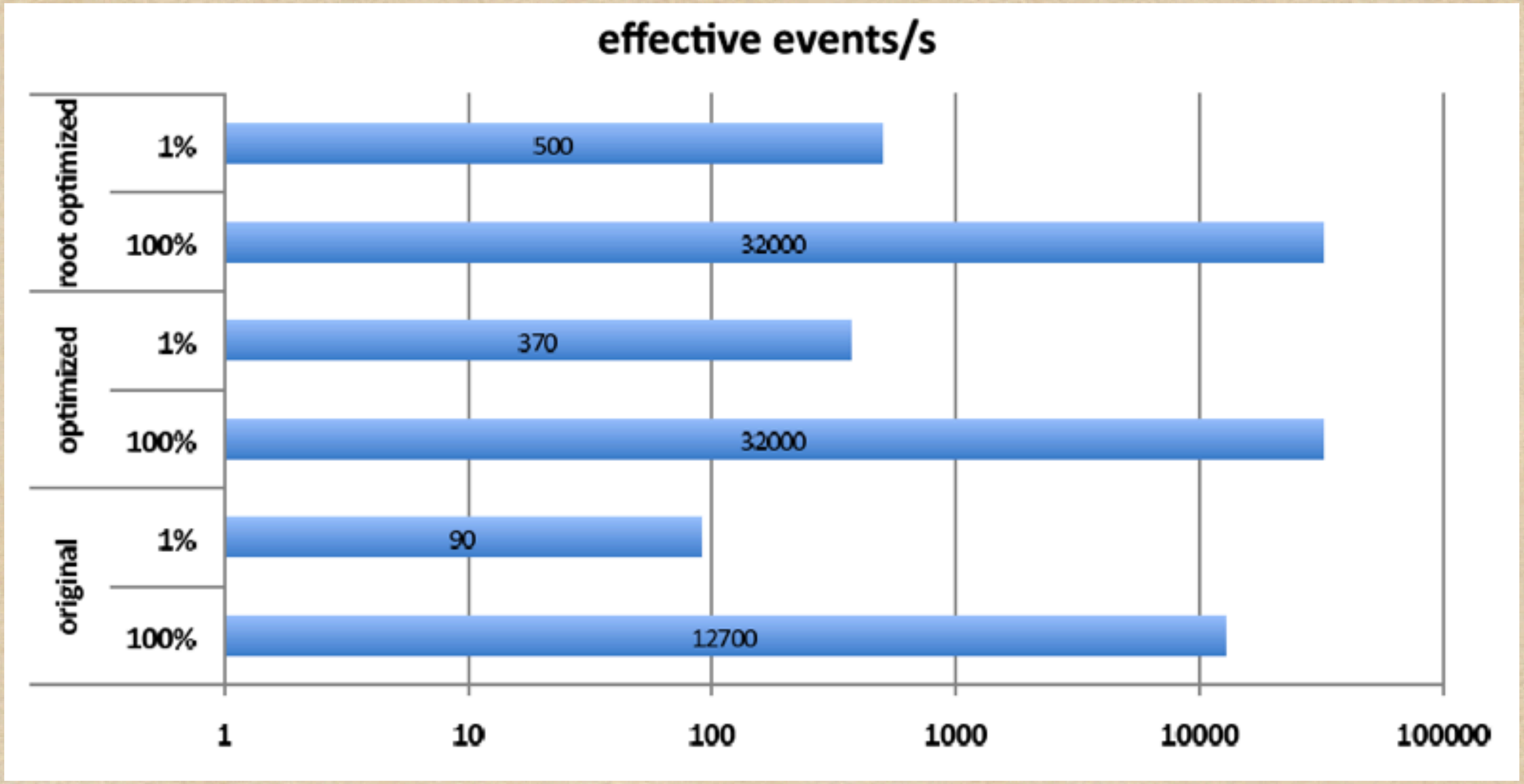


Figure 4. Comparison DPM/rfio and Lustre.

- DPM was tested in Glasgow
- Lustre file system at Queen Mary, University of London
- 1MB or 4kB of read-ahead on DPM
- TTreeCache is useful also with posix access

Atlas Reordered AOD



Few tests on EOS:

Read rate of the unoptimized data set was disk bound. Sparse reading of the randomly distributed 1% of events shows very bad performance. In the case of unoptimized files we could read at most 90 selected events per second. This is worse than reading all of the events and just discarding the un-needed ones, which would give us 127 events/s.

CMS I/O Optimization

CMSSW 3_x

- In CMSSW 3_x, buffers were fixed-sized and flushed out to disk whenever they were filled.
  - A branch with 16KB objects is flushed every event; a branch with 16 byte objects is flushed every 1024 events.
  - Compression ratios varied widely: 16KB sometimes compressed to hundreds of bytes.

CMS I/O Optimization

Consequences

- There were no locality guarantees: an event's data is spread throughout the file.
  - The reads were very small.
  - Small and random reads: just what a disk hates

CMS I/O Optimization

- State of the Art:
  - CMSSW 4_x contains ROOT 5.27/06b, which has significant I/O improvements over 3_x
  - CMSSW 4_x also contains modest I/O improvements on top of out-of-the-box ROOT I/O

CMS I/O Optimization

CMSSW 4_X_X

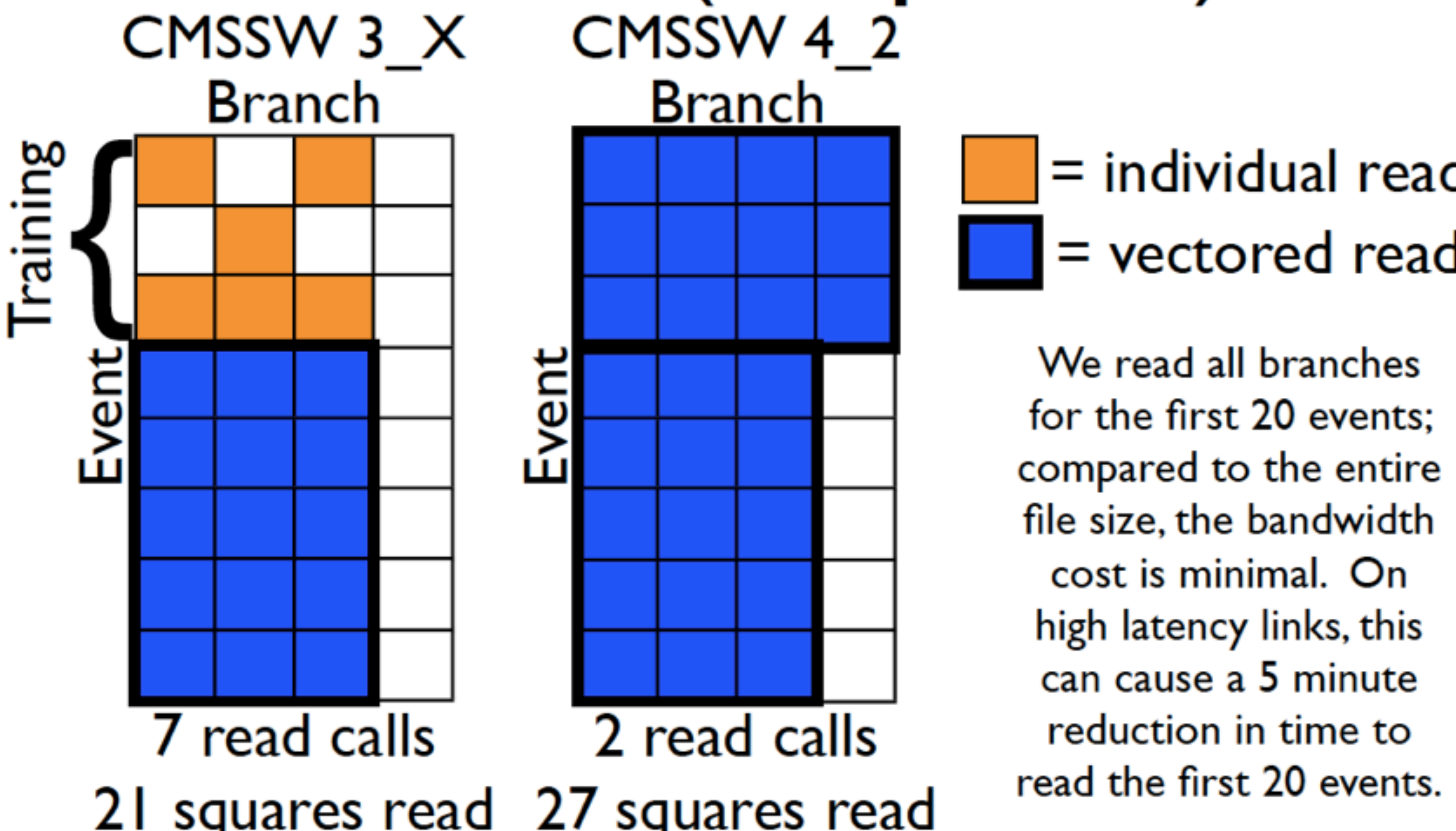
- New ROOT added:
  - Auto-flushing: All buffers are flushed to disk periodically, guaranteeing some level of locality.
  - Buffer-resizing: Buffers are resized so approximately the same number of events are in each buffer.
  - Read coalescing: "Nearby" reads (but nonconsecutive) are combined into one.

CMS I/O Optimization

CMSSW 4_X_X

- Incremental improvements through 3_x and 4_x: Only one event tree, improved event read ordering, TTreeCache became functional, caching non-event trees.
- Improved startup: While TTreeCache is being trained, we now read all baskets for the first 10 events at once. So, startup is typically one large read instead of many small ones.

Improved startup latencies (simplified)



We read all branches for the first 20 events; compared to the entire file size, the bandwidth cost is minimal. On high latency links, this can cause a 5 minute reduction in time to read the first 20 events.

ROOT Optimization results

```
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4900 0 fh=0 read 2176@64970135
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4900 sending 2176 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4a00 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4a00 0 fh=0 read 1753@65316520
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4a00 sending 1753 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4b00 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4b00 0 fh=0 read 4493@66445707
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4b00 sending 4493 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4c00 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4c00 0 fh=0 read 2010@67021064
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4c00 sending 2010 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4d00 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4d00 0 fh=0 read 4985@67315032
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4d00 sending 4985 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4e00 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4e00 0 fh=0 read 1566@68390525
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4e00 sending 1566 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4f00 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 4f00 0 fh=0 read 5748@68862703
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 4f00 sending 5748 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5000 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5000 0 fh=0 read 870@70351322
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 5000 sending 870 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5100 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5100 0 fh=0 read 2246@70484565
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 5100 sending 2246 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5200 req=3013 dlen=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5200 0 fh=0 read 2162@72141290
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdResponse: 5200 sending 2162 data bytes; status=0
110509 17:59:35 23873 donvito.2199:18@pccms64 XrootdProtocol: 5300 req=3013 dlen=0
```

CMSSW 3_x

ROOT Optimization results

```
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 13816@2086738775
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 11377@2086776641
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 4412@2086884627
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 3564@2087058046
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 16592@2087086178
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 2571@2087173866
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 12585@2087185373
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0400 fh=0 readV 2845@2087259661
110511 11:34:04 23873 root.26598:16@pccms70 XrootdResponse: 0400 sending 572878 data bytes: status=0
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 req=3025 dlen=5232
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 30283@2087579185
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 24492@2087686876
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 3135@2087778737
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 5035@2087905049
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 6038@2087924911
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 4407@2087995634
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 15128@2088161040
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 13578@2088206774
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 2759@2088256200
110511 11:34:04 23873 root.26598:16@pccms70 XrootdProtocol: 0500 fh=0 readV 4647@2088291409
....
110511 11:34:05 23873 root.26598:16@pccms70 XrootdProtocol: 0400 0 fh=0 read 1024@32000512
110511 11:34:05 23873 root.26598:16@pccms70 XrootdResponse: 0400 sending 1024 data bytes; status=0
110511 11:34:05 23873 root.26598:16@pccms70 XrootdProtocol: 0400 req=3013 dlen=0
110511 11:34:05 23873 root.26598:16@pccms70 XrootdProtocol: 0400 0 fh=0 read 1024@32046080
110511 11:34:05 23873 root.26598:16@pccms70 XrootdResponse: 0400 sending 1024 data bytes; status=0
110511 11:34:05 23873 root.26598:16@pccms70 XrootdProtocol: 0400 req=3013 dlen=0
511 11:34:05 23873 root.26598:16@pccms70 XrootdProtocol: 0400 0 fh=0 read 1024@32134144
```

CMSSW 4_x

ROOT Optimization results

1	7	91	1	0	1	0	832k	99k	17M	0	0	2[illegible]22	2643
0	0	100	0	0	0	0	0	22k	29k	0	0	1339	2230
0	0	100	0	0	0	0	8192B	25k	46k	0	0	1347	2248
0	0	99	0	0	0	0	232k	29k	34k	0	0	1416	2276
1	0	99	0	0	0	0	48k	24k	83k	0	0	1392	2474
0	0	99	0	0	0	0	0	29k	56k	0	0	1391	2305
0	0	100	0	0	0	0	32k	30k	38k	0	0	1396	2312
0	0	100	0	0	0	0	0	24k	37k	0	0	13[illegible]2	2254
0	0	99	0	0	0	0	104k	27k	37k	0	0	1388	2282
0	0	100	0	0	0	0	0	32k	41k	0	0	1377	2309
1	0	99	0	0	0	0	0	25k	46k	0	0	1369	2242
1	10	87	1	0	1	0	44[illegible]k	158k	21M	0	0	2937	2658
0	0	100	0	0	0	0	12k	33k	144k	0	0	1397	2333
0	0	99	0	0	0	0	368k	25k	86k	0	0	1448	2271
0	0	100	0	0	0	0	80k	29k	47k	0	0	1412	2338
0	0	99	0	0	0	0	0	31k	46k	0	0	1405	2336
1	0	99	0	0	0	0	32k	25k	50k	0	0	1382	2273
0	0	100	0	0	0	0	0	26k	36k	0	0	1362	2265
0	0	99	0	0	0	0	20k	31k	33k	0	0	1412	2364
0	0	100	0	0	0	0	80k	22k	26k	0	0	1362	2248
1	0	100	0	0	0	0	88k	25k	28k	0	0	1374	2289
1	1	99	0	0	0	0	32k	29k	46k	0	0	1416	2517
0	0	100	0	0	0	0	0	25k	100k	0	0	1383	2297
1	0	99	0	0	0	0	12k	24k	25k	0	0	1369	2266
0	0	99	0	0	0	0	148k	30k	41k	0	0	1414	2320
1	10	88	1	0	1	0	432k	150k	21M	0	0	2933	2657

Network spike
17-20MB/s

small or no network utilization

Network utilization

CMSSW 4_x

CMS I/O Optimization

Upcoming Enhancements

- Comparing 5.27 with ROOT trunk:
  - 5% performance increase in AOD unstreaming.
  - 15% performance increase in ROOT "Event" unstreaming.
  - Unstreaming uncompressed data goes at 326MB/s.
- "Real" Asynchronous prefetch (using threads and double-buffering)

Storage manager software

Optimizing storage access performance

- We need to measure the efficiency of each storage system and try to understand which and in which configuration could serve better the LHC analysis jobs

- System under test:
  - Lustre, HDFS, Xrootd, Glusterfs, ext3

Storage Systems under Test

CMSSW 3_10_1

- Server:
  - Lustre 2.0:
    - 3 RAID5 FS. Stripe-unit size: 128 KB. 5 Data disk each
  - Xrootd 3.0.0:
    - 13x1TB single disks. EXT3 FS
  - hadoop-0.20.2 (from http://newman.ultralight.org/)
    - 13x1TB single disks. EXT3 FS
- Clients:
  - SLC5.4 kernel 2.6.18-194.11.3
  - Fuse: fuse-libs-2.7.4-8
  - FUSE mount on the client (rdbuffer=32768)

Xrootd: performance consideration

- MTR3 CMS job looks like very random application:
  - Small read operation
  - quite random read seek operation
- We measure the CPU efficiency during the run (CPUTime/WallTime)
  - Used bandwidth is not a good metrics
- Surprisingly big RAID5 with Fiber Channel controller performs worst than simple single SATA disk for a single job
  - It was difficult to obtain >40% in cpu efficiency using raid5
  - While it was easy to got 90% with a single disk
- The problem seems to be correlated with IOPS and stripe size on the controller
  - The initial test point is 1MB of stripe size (on the RAID5)

Xrootd: performance consideration

- Reconfiguring the raid to 256kb of stripe size we easily got 80% of CPU efficiency for a single job
  - "cacheSize" value="20048576" ## "cacheHint" value="storage-only" ## "readHint" value="read-ahead-buffered"
  - looking to the used bandwidth: a single job is able to read at about 3MB/s constantly
- We tested: xfs, ext3, ext4
  - no mayor differences observed
- We tried to run up to 120 concurrent jobs against the same server:
  - 100MB/s of aggregated bandwidth at maximum
  - ~40% of CPU efficiency

Xrootd: performance consideration

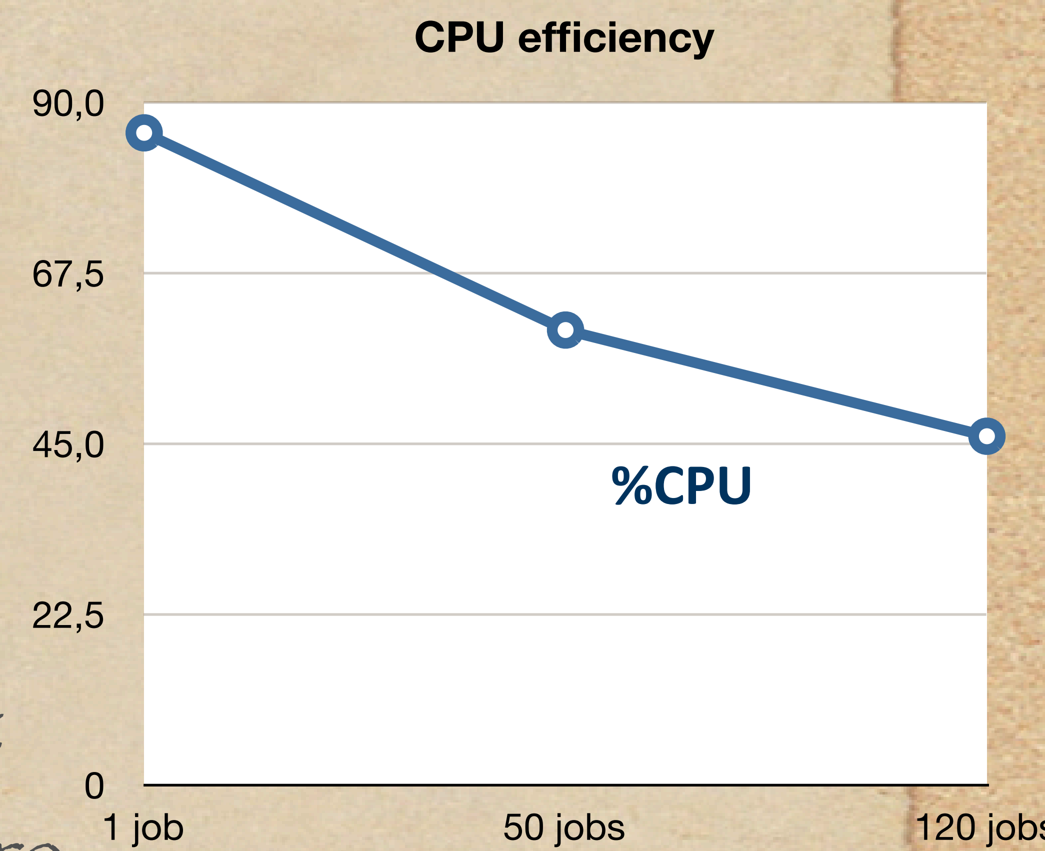
- It is clearly limited by disk IO
  - High I/O wait on the server
- The network is not a big issue here
- Changing the IO parameters in CMSSW do not add big improvements
- The raid controller under test do not support smallest stripe size
- This gives a measure of the scalability in "job-per-server" of the disk sub-system
  - maybe a single-disk configuration could give better performances
    - more test are still needed using "JBOD configuration"

Lustre: Performance

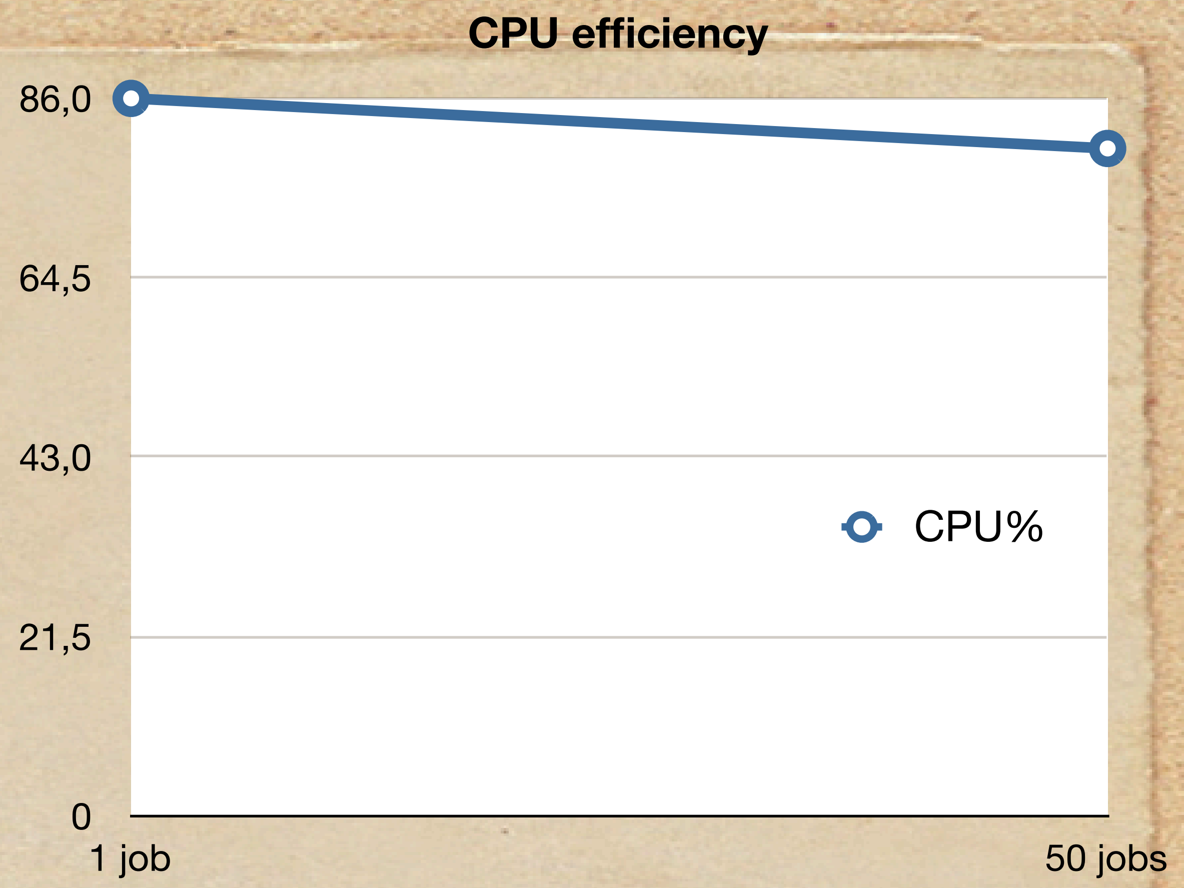
- Tuning a bit CMSSW parameters we easily got ~86% of CPU efficiency
  - "cacheSize" value="20048576" ## "cacheHint" value="lazy-download" ## "readHint" value="read-ahead-buffered"
  - Using a posix file-system the framework do not really download the files, but does only read-ahead-buffered
  - The configuration of the raid controller here do not affect to much the performance
- With this configuration a single job could read data with spikes of 50-60MB/s
  - there are, obviously, periods of time in which the job do not read data

Lustre: Performance

- In case of lustre, we observed that increasing the "cacheSize" could reduce the I/O on the disks
  - but this easily could become a bottleneck on the network
- For example running 120 jobs against a single disk server could require more than 250MB/s on the network
- If we reduce the "cacheSize" to 2MB this reduces the load on the network but increases the load on the disk subsystem

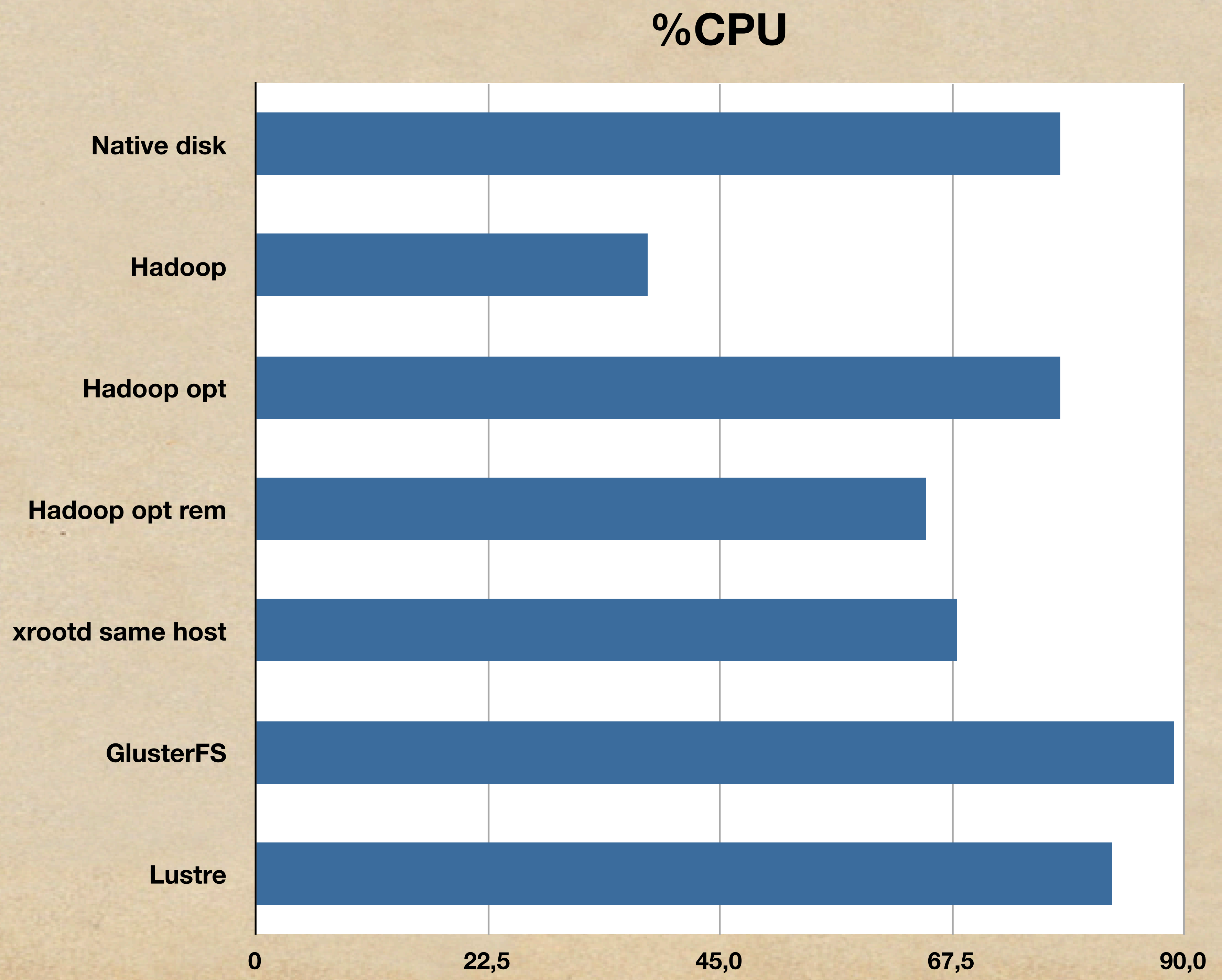


SSD test



- In order to be sure that we have a limitation on the storage sub-system we tested an SSD disk with an Xrootd server
  - a single MLC SSD (256GB) is able to provide data to 50 concurrent jobs without losing in CPU efficiency

Optimising the Single job



Optimising the Single job

- "Hadoop opt"=> rdbuffer=32768
- The CMSSW (cacheHint,readHint,cacheSize) tuning parameters are always used and tested until the best result is found
- "blockdev --setra" on each drive, was tuned in order to find the best solution
- It is possible to obtain the same performance with up to 4-5 concurrent job per single native disk
- Glusterfs: tuning iocache/read-ahead page-size in glusterfs.vol.sample
- Lustre: tuning read-ahead in: /proc/fs/lustre/llite/lustre-*/max_read_ahead_mb and /proc/fs/lustre/llite/lustre-*/max_read_ahead_per_file_mb

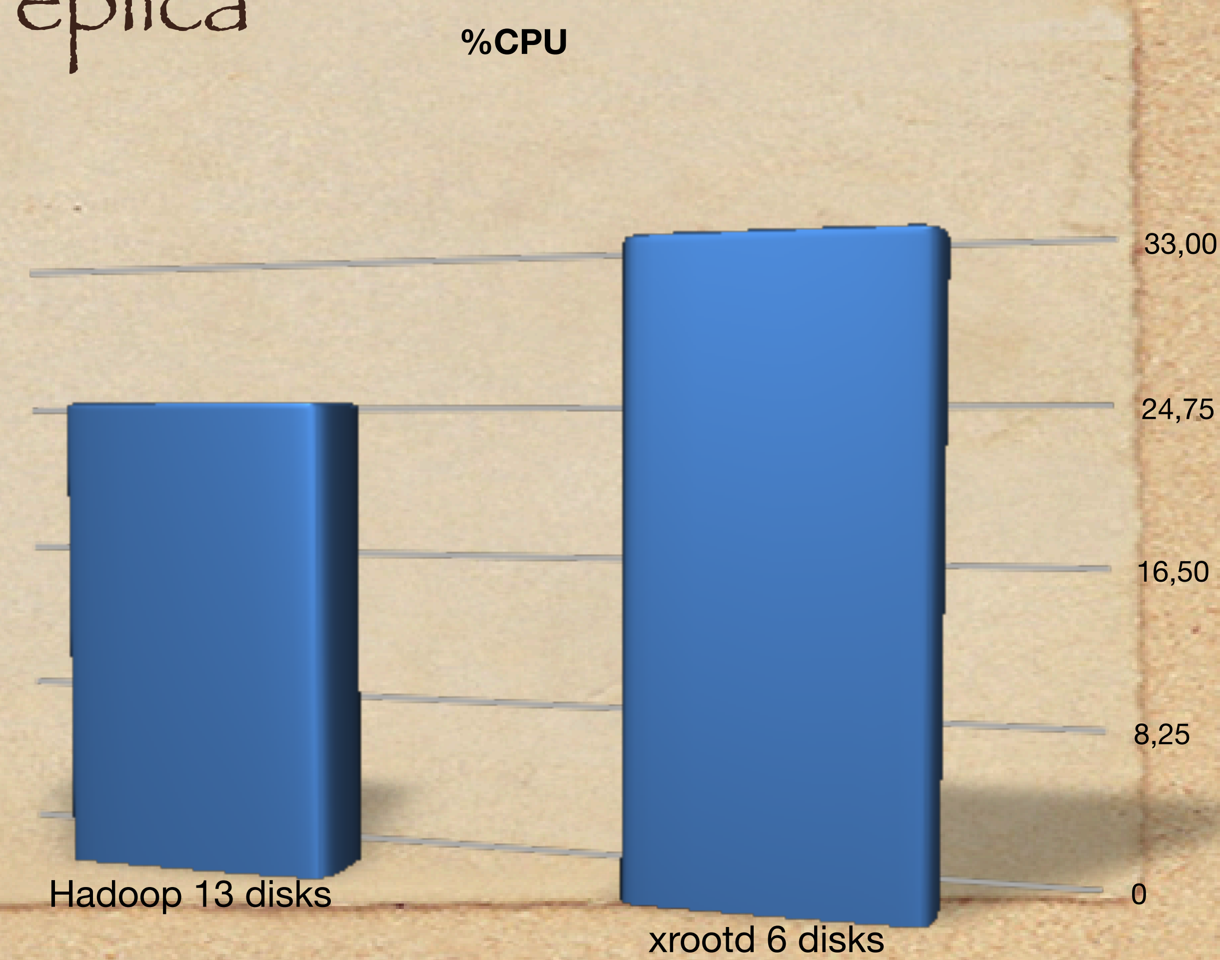
Performance Tests

- up to 116 concurrent jobs
- production farm used to run the jobs
- Each file on the server is used only by a single job
  - There is no "concurrency" on each file
- A single disk server:
  - 10Gbit/s network card
  - deep network testing to assure there are no network bottleneck
  - >400MB/s measured disk-to-network bandwidth

Performance test: hadoop vs xrootd

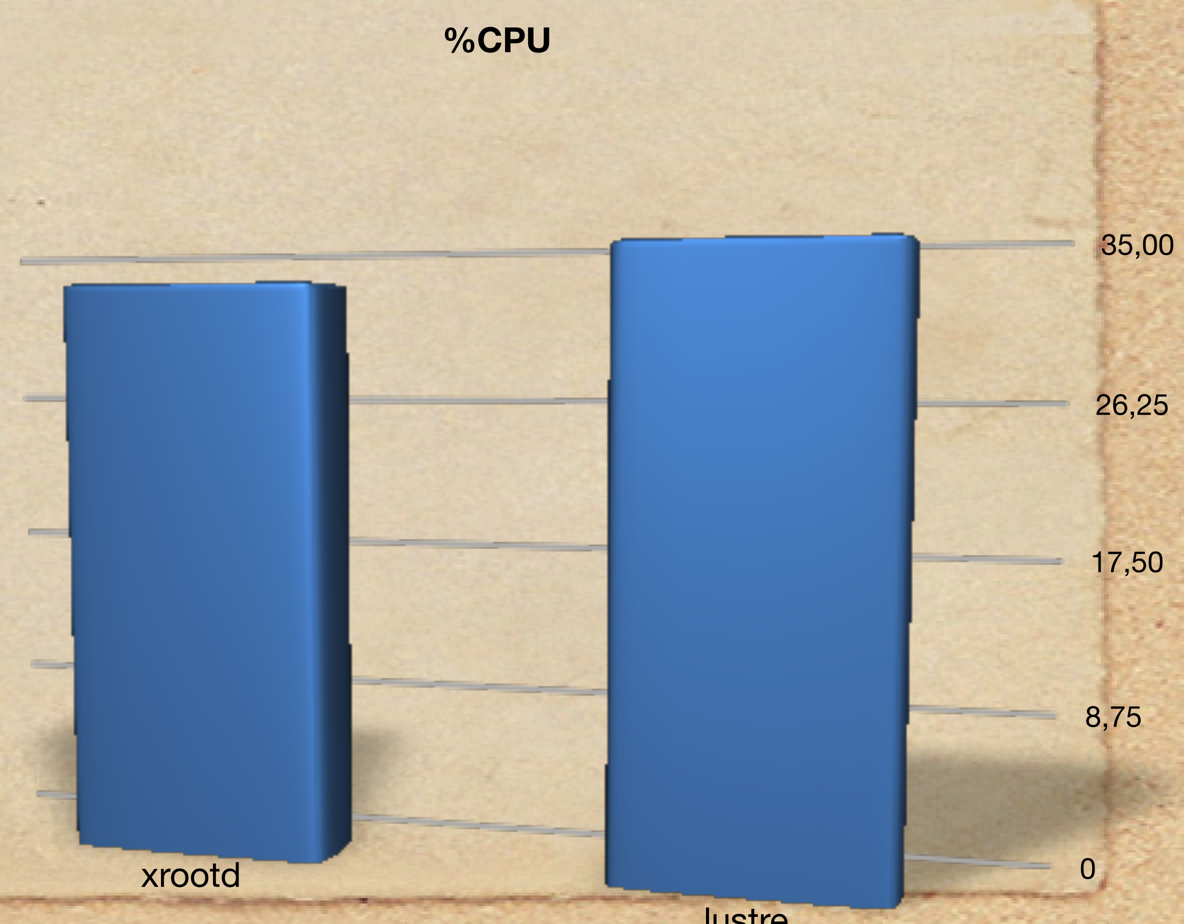
- Running 56 concurrent jobs
- Using 6 disks for xrootd
- Using 13 disks on hadoop installation
    - Reading data using "fuse optimized"
    - Single server: no "block replica"

- We have observed huge load on the server while running "hadoop test"
    - increasing the memory for java produced only small improvements

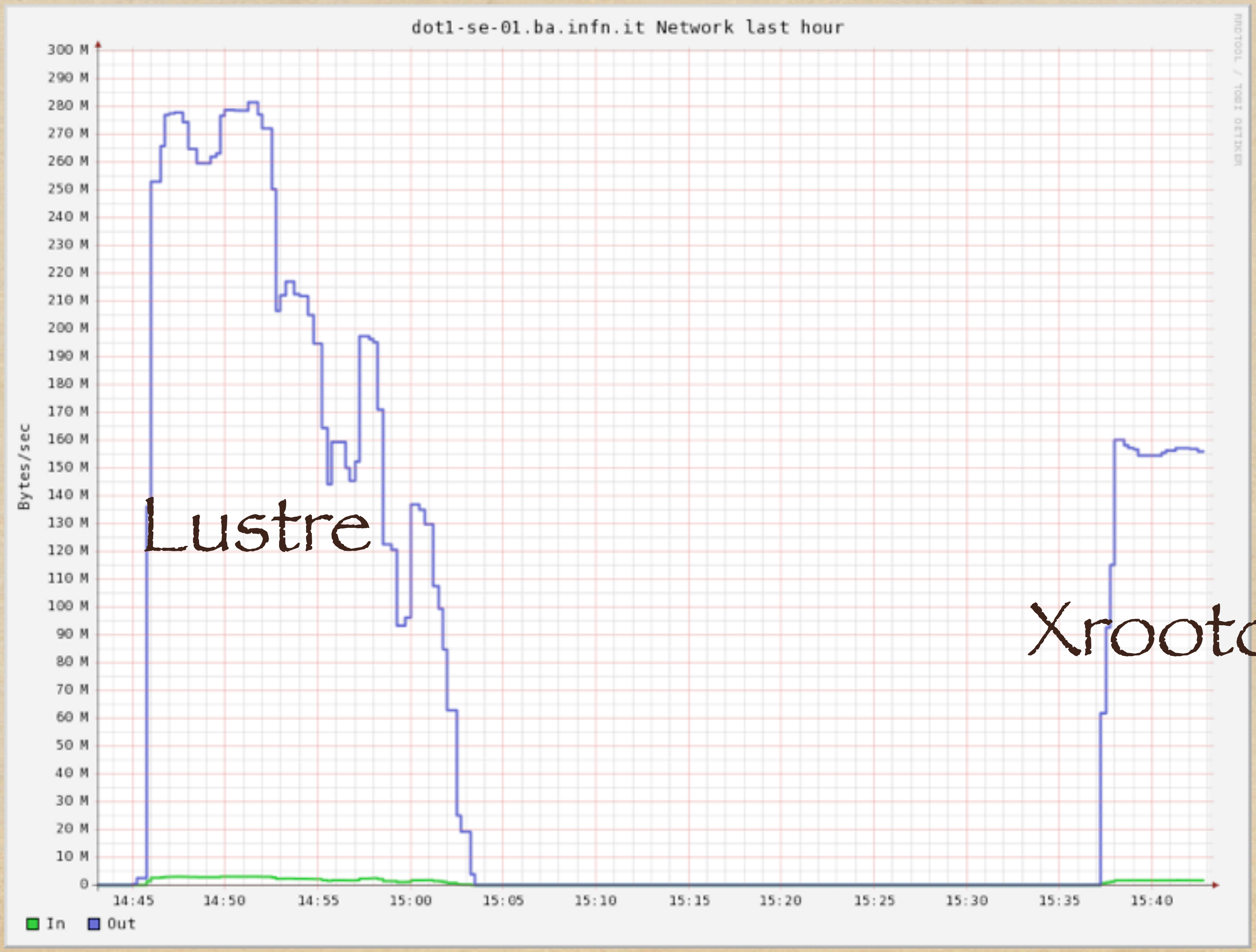


Performance Tests: lustre vs xrootd

- Running 116 concurrent jobs
- Reading ~1TB of data
- Always measuring the CPU efficiency
  - This is an interesting parameter both from user's point of view and from a site admin
- The network usage of the two solution is completely different (see next slide)
- In both cases the disk subsystem on the server is the bottleneck



Performance Tests: lustre vs xrootd

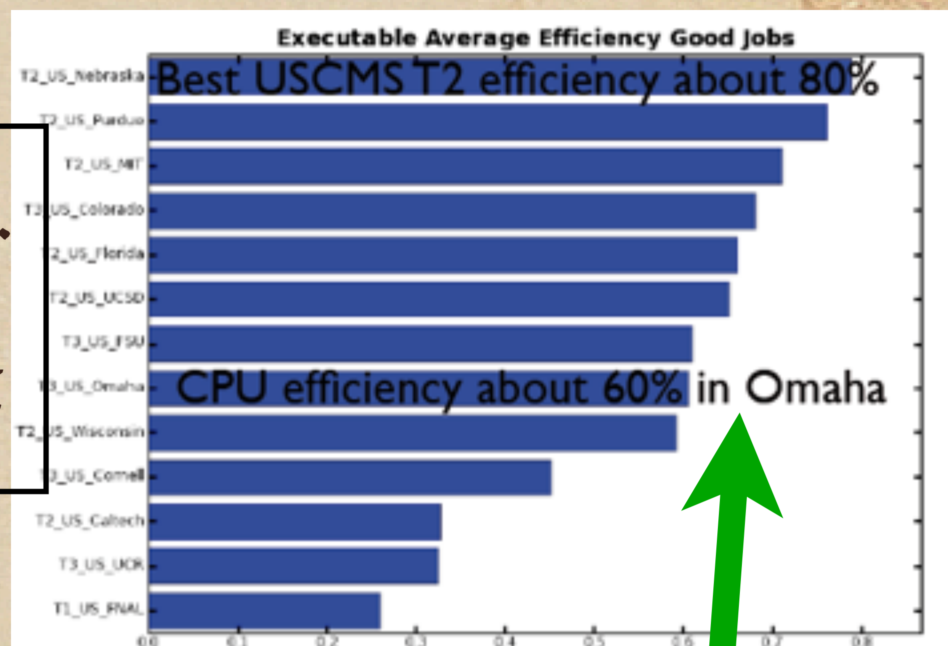


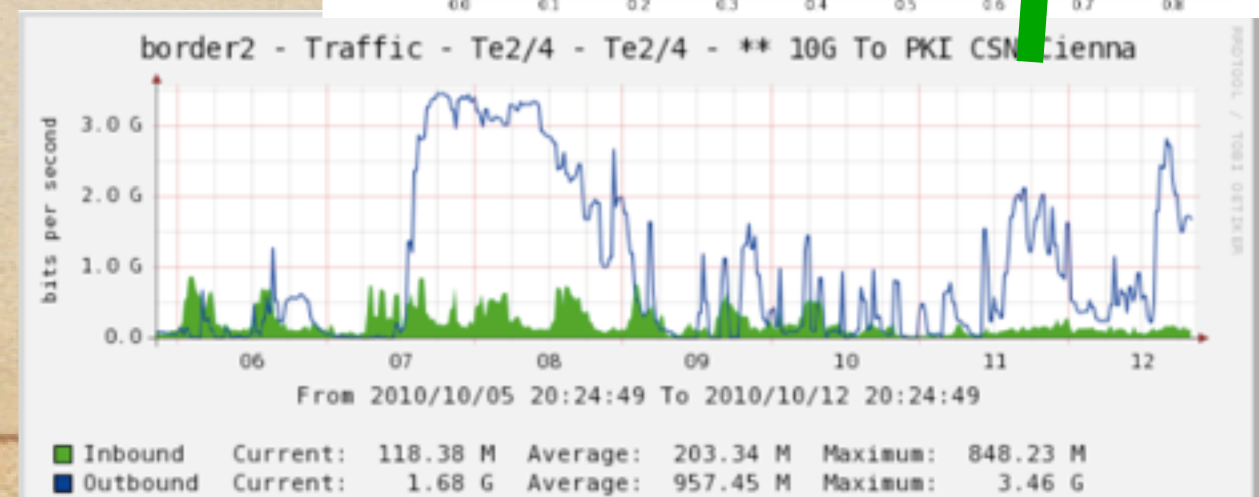
Reading Root data remotely

- The I/O optimization work on ROOT and CMSSW give us also the opportunity to explore the possibility to access data remotely

- Italian test:
  - CMSSW 4_1_3, I/O bound analysis
  - ping time = ~12ms
  - Job running in LNL data @Bari
  - CPU efficiency drop from 68% to 50% => ~30% of performance drop

U.S. Test





Infrastructure scaling

Client-server vs Peer-Network

- Client-server (i.e. Lustre):
  - Pro:
    - adding more server => ~linear scalability
    - good performance
    - fully posix compliance
  - Cons:
    - Failure of a server affect the operations
    - Need a good network design
    - Need powerful storage servers

Client-server vs Peer-Network

- Peer-Network (i.e. HDFS):
  - Pro:
    - failure os a server is not blocking
    - fits well with cheap hardware
    - the network is less critical and costly
  - Cons:
    - CPU efficiency and performance lower than parallel file systems
    - requires more rack space/power ...
    - not fully posix compliance

Client-server vs Peer-Network

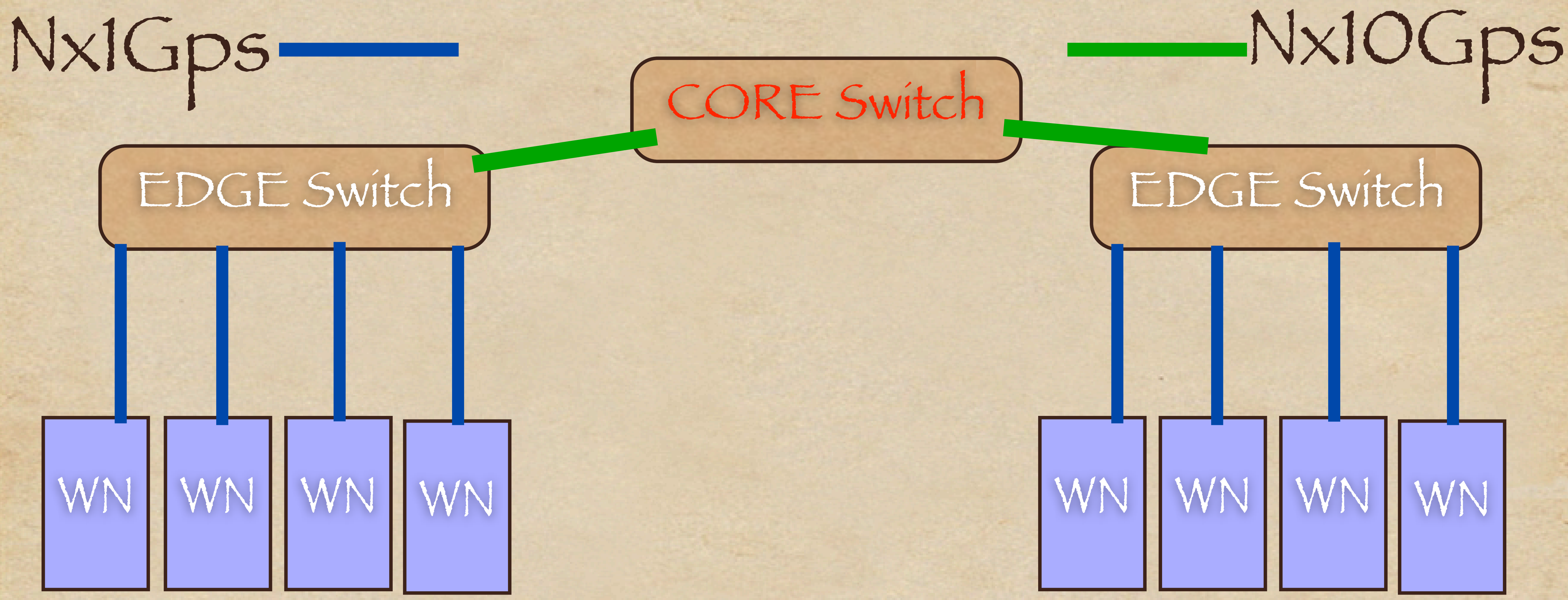
A critical view of the problems

- We always need to move data:
  - MapReduce algorithm also need to "move" from local disk
    - With a good network it could be faster
    - Nx1Gbit/s link could be needed on all the WN quite soon anyway
- Power consumption and space do also have a cost
- The real difference is about failures
- Using HDFS in HEP analysis, the CPU efficiency looks worst than Lustre/Xrootd

Worker Node network

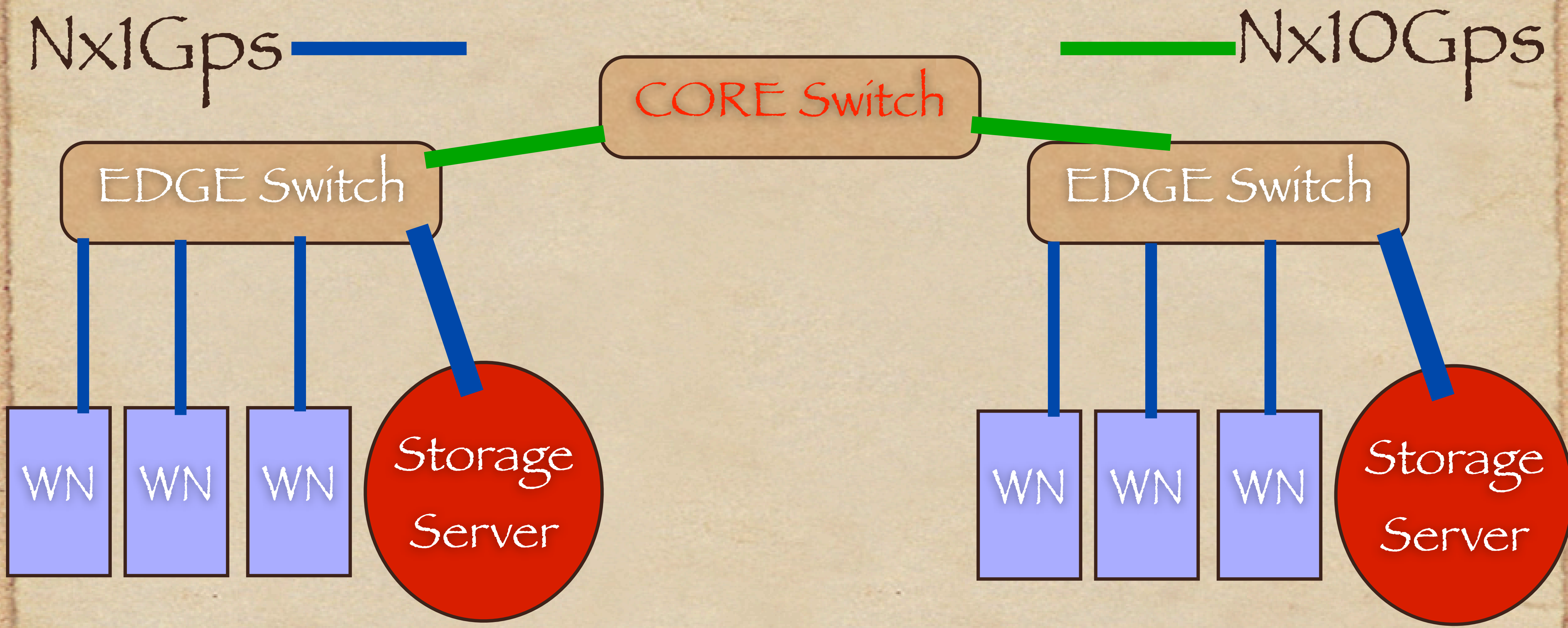
- 24 Cores right now => +50% next year
- 1 Gbps could be not enough (=> ~5MB/s per slot)
- 2 x 1Gbps could be enough for at least 1.5 years
  - in the next future 10Gbps on the WN should be a must
  - maybe multi(many) core aware application could help

Peer-Network



- In the HEP environment it is not so easy to exploit "affinity scheduling" algorithms so "rack awareness" does not help so much
  - ==> you need a good network anyway

Client-Server

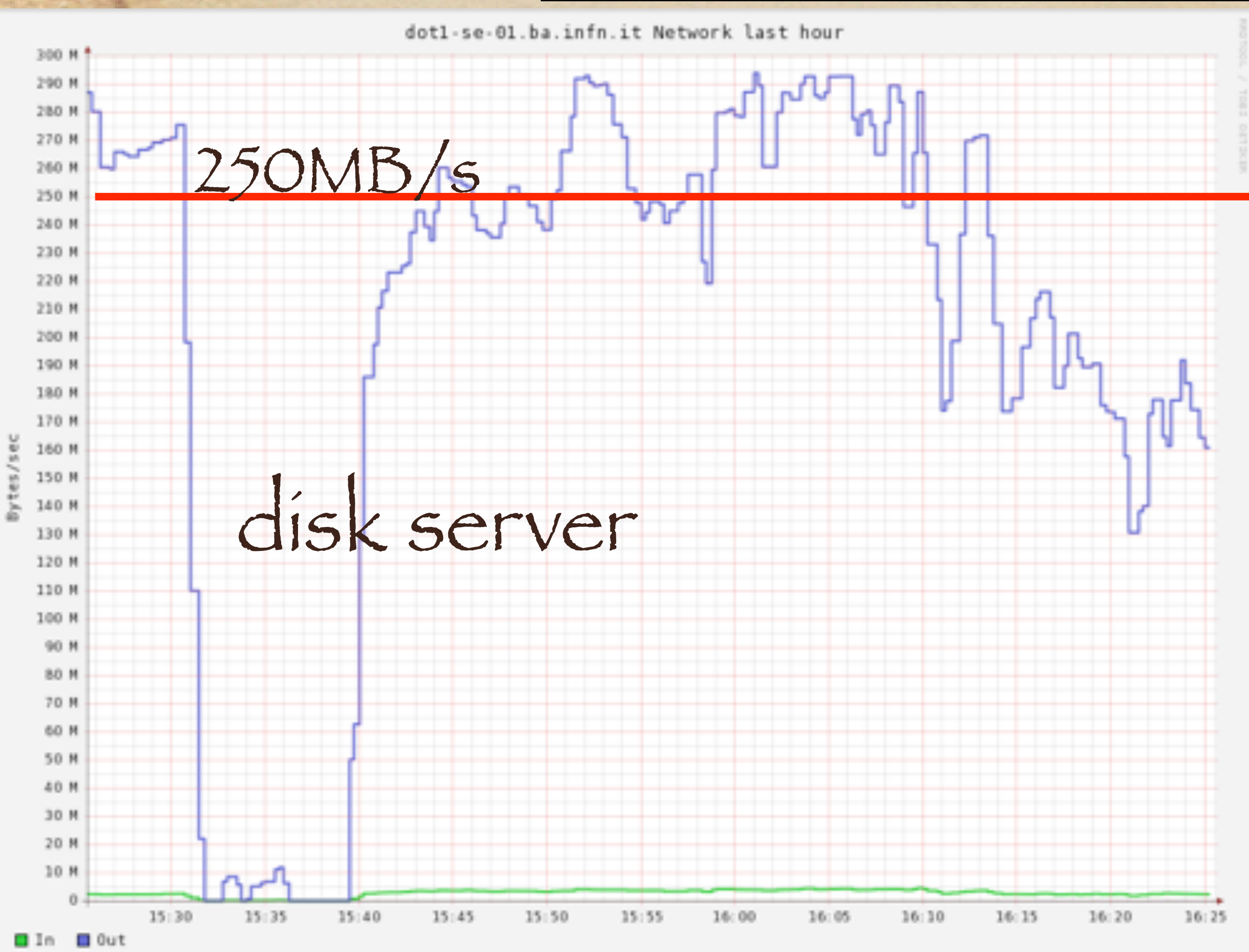


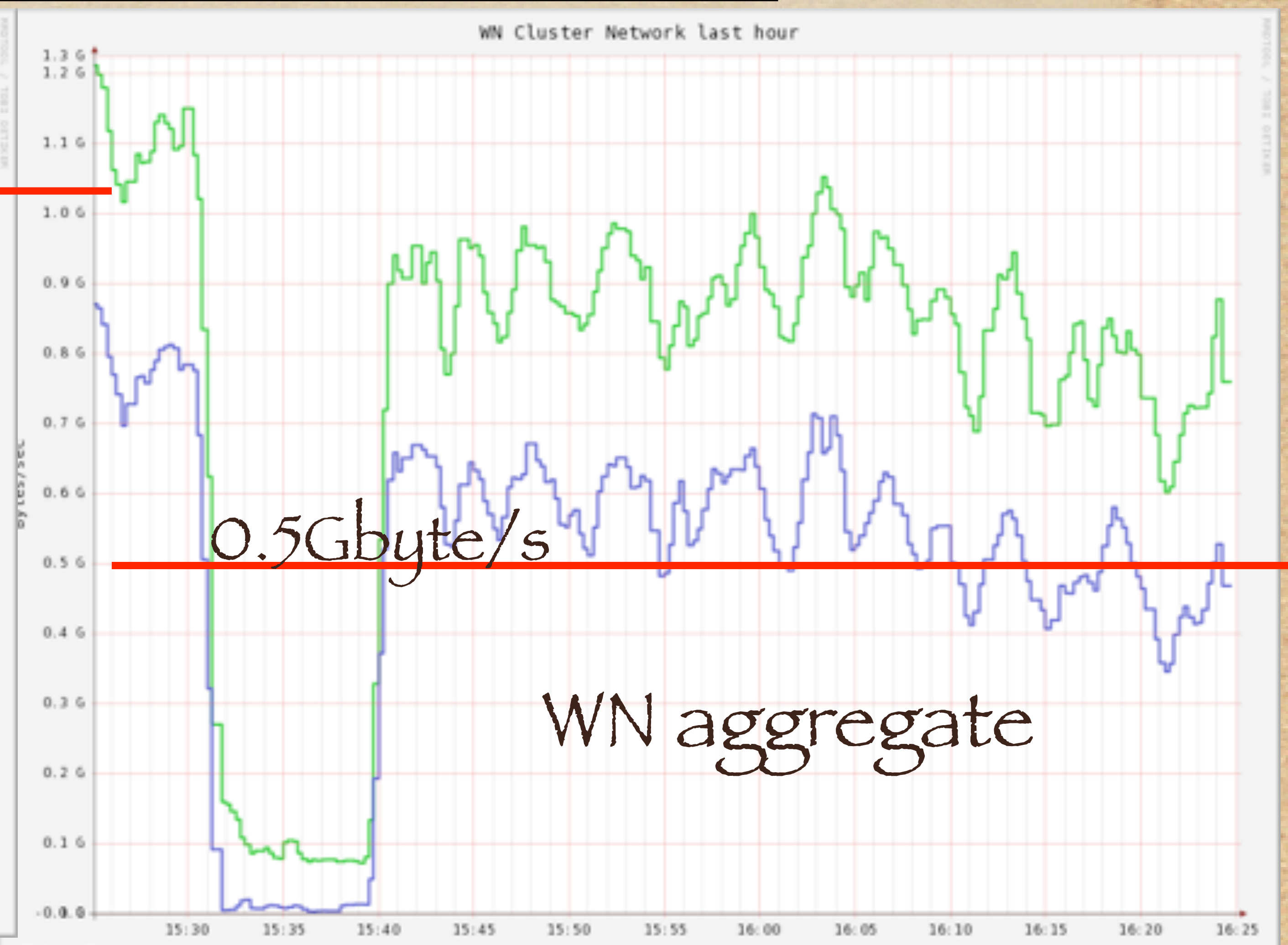
- With a smart design, the network cost in the two scenario could be basically the same

Hadoop large scale tests

- 160 concurrent ROOT jobs
- 2.5TB of input data completely analyzed in about 60min
- average ~20% of CPU efficiency

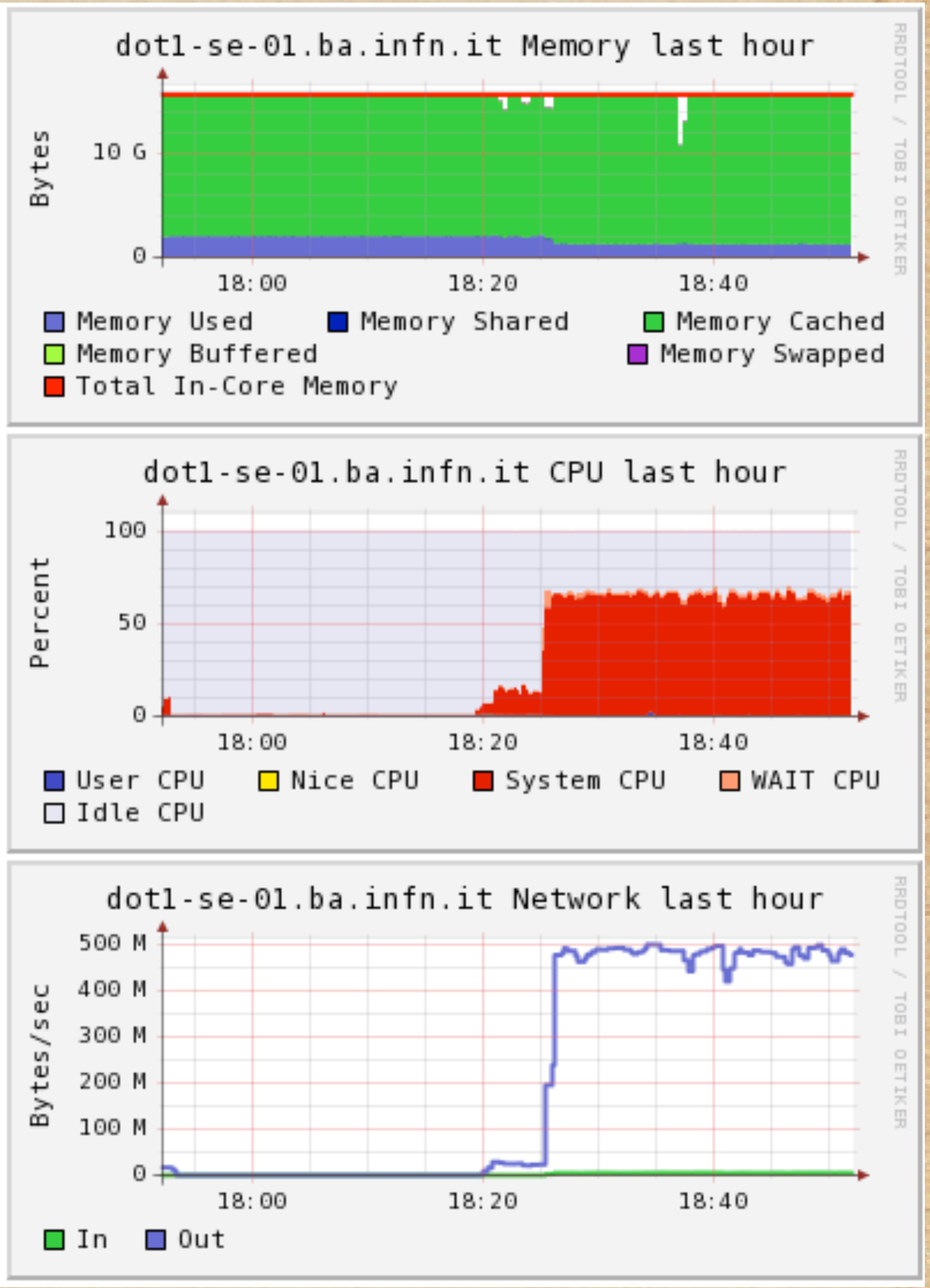
$\boxed{\text{=> ~800MB/s of effective ROOT I/O}}$





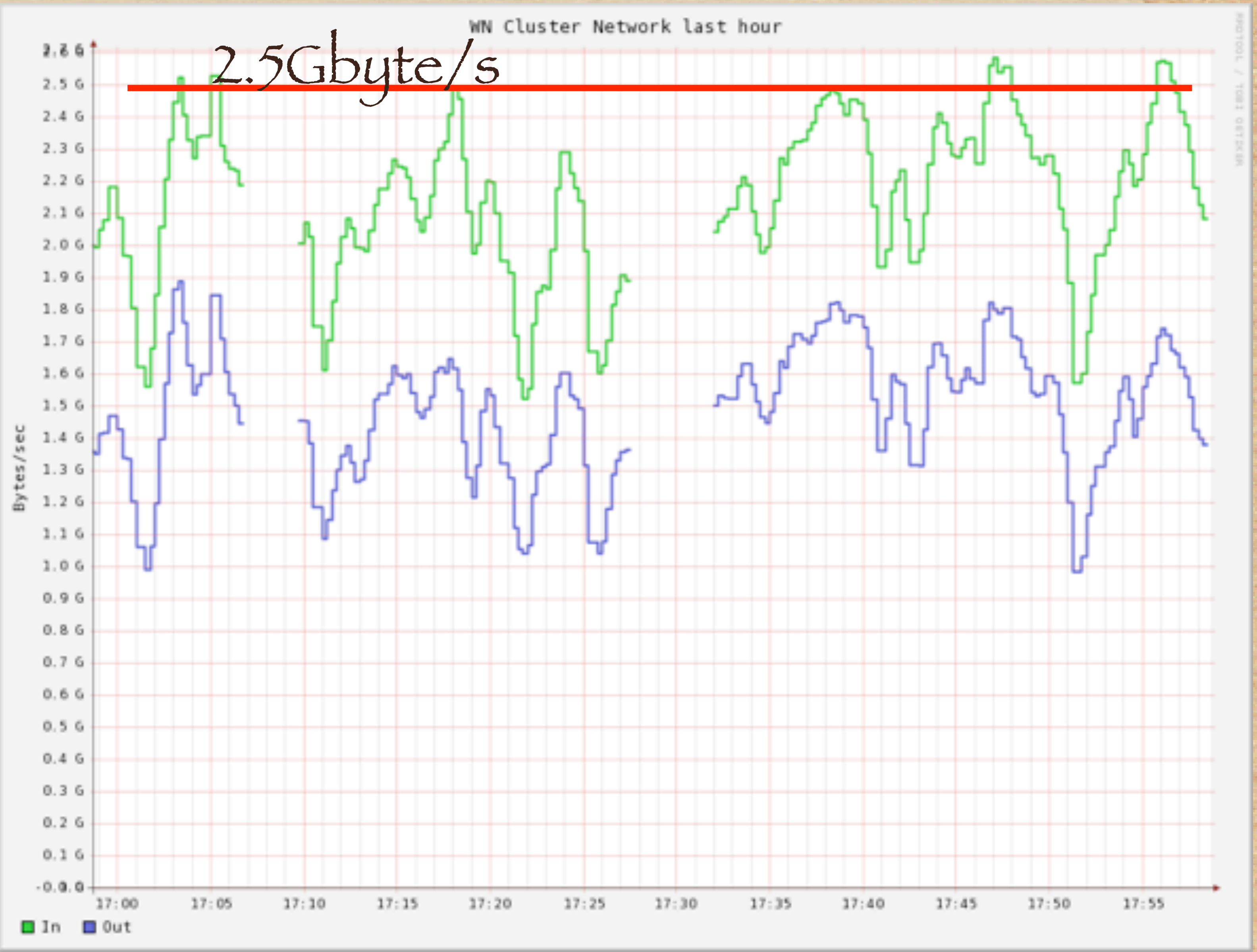
Optimized ROOT code

- Running "copy-event" CMSSW_4_X application could require up to 40MB/s flat on a single core
  - the plot show how a single LUSTRE server performs with 25 concurrent "copy-event" jobs
  - ~50% of CPU efficiency



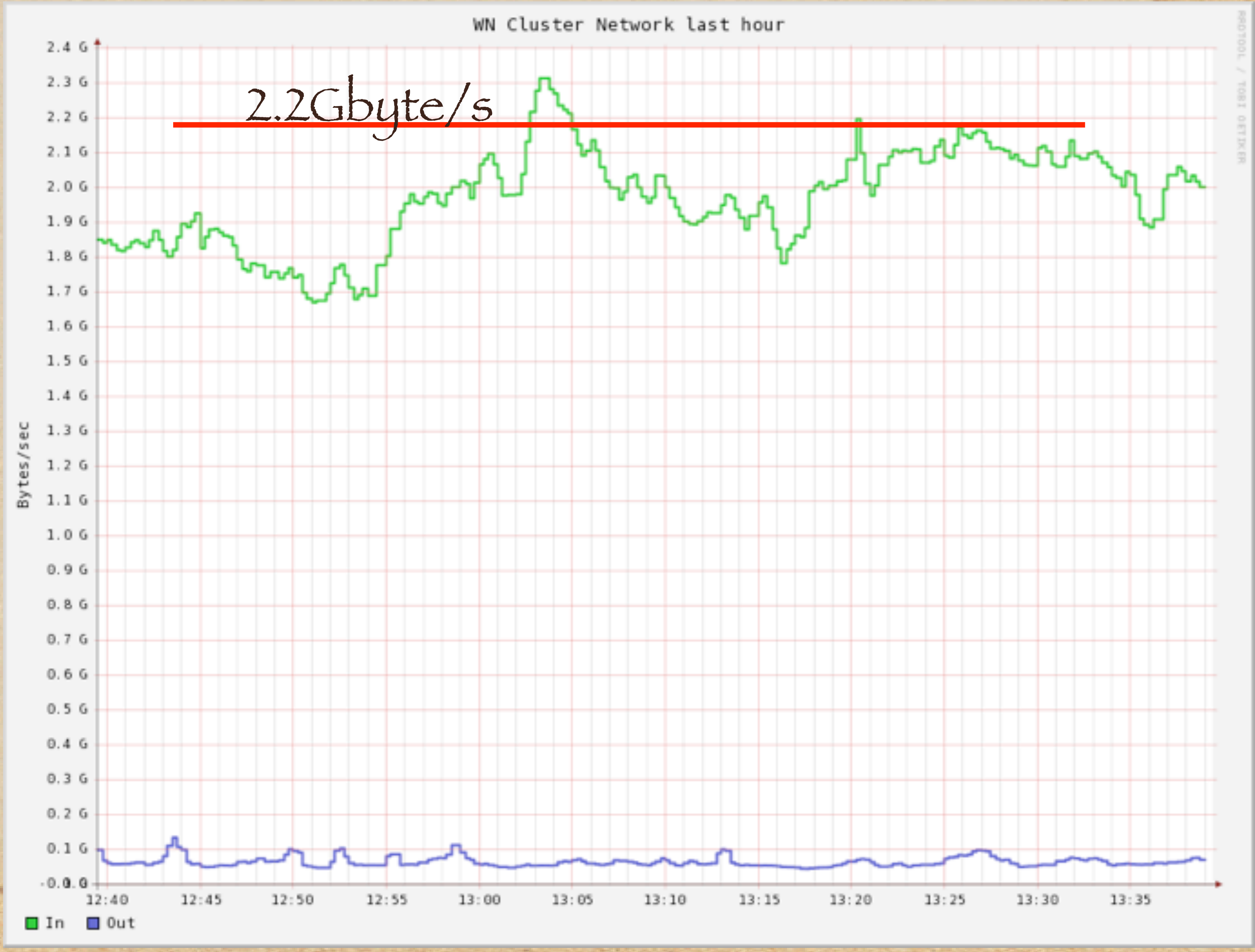
Hadoop "large" scale test

- Using hadoop on 130WN (1 disk per WN) +1 disk server
- Sequential (concurrent) read and write
- easy to go up to: 2.6GByte/s



Lustre “large” scale test

- Production environment
- 20 Server
- 130 WN
- Real analysis jobs



Hadoop failure test

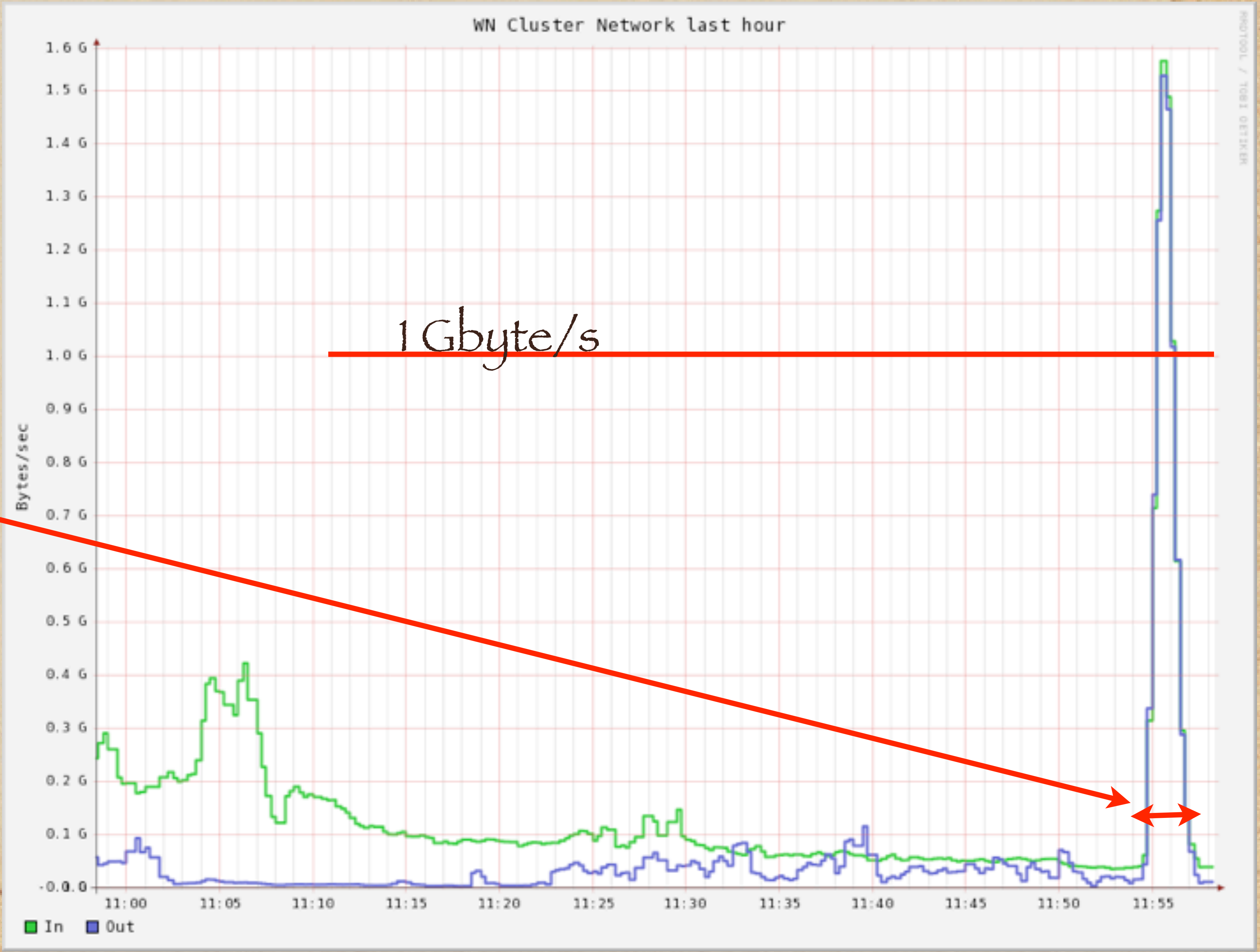
9TB node failure

Less then 1 hour to recover from a big host failure

Hadoop failure test

Single disk failure

Less than 5 minute to recovery from a single disk failure



Conclusions

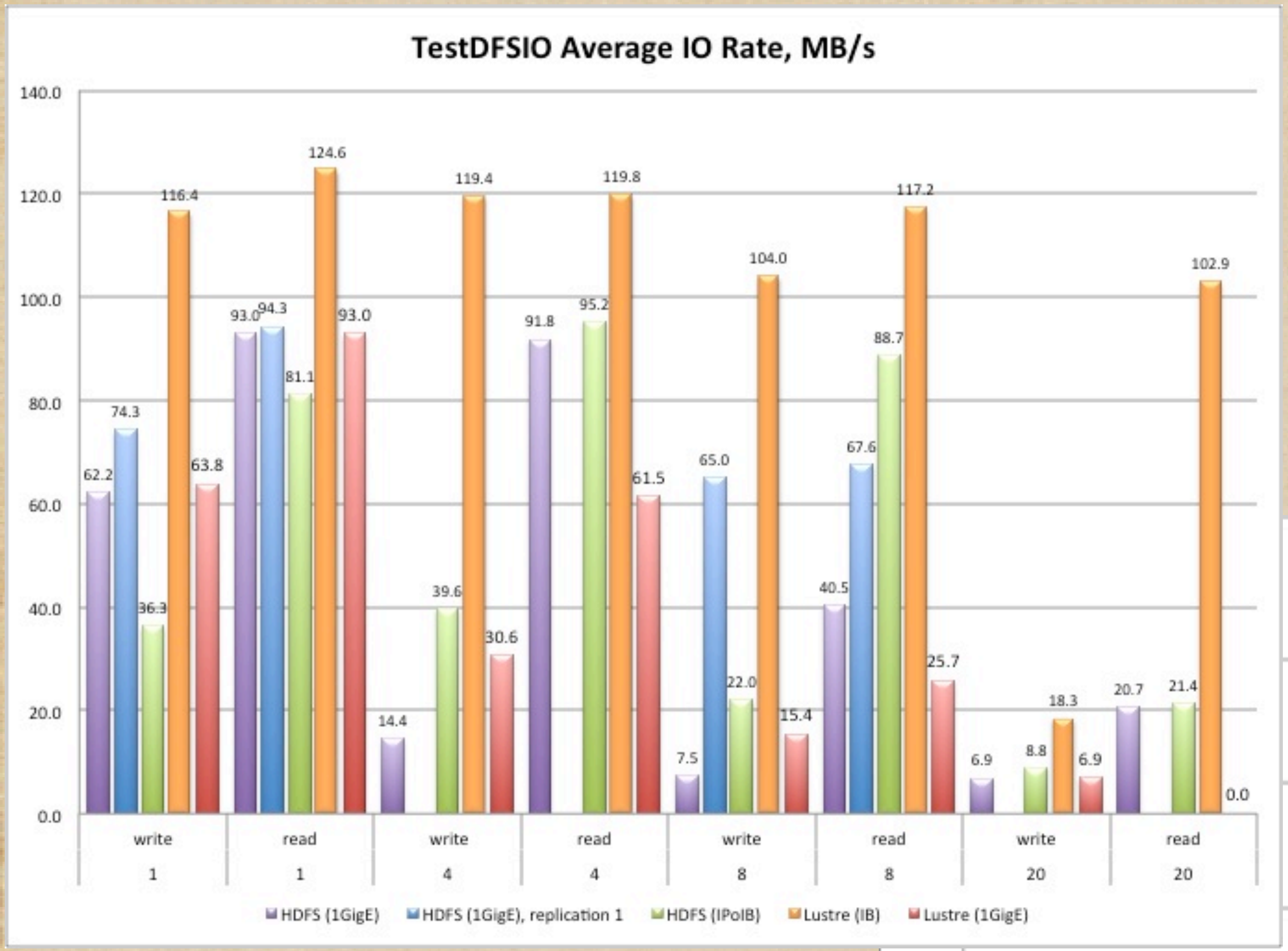
- Applications developers are working hardly to try to improve the overall performance
  - The access to data is less “random” now and it will be more efficient in the next releases...
  - but, it will surely become much more bandwidth demanding in the future
- CPU Technology is evolving putting new problems (many-cores)

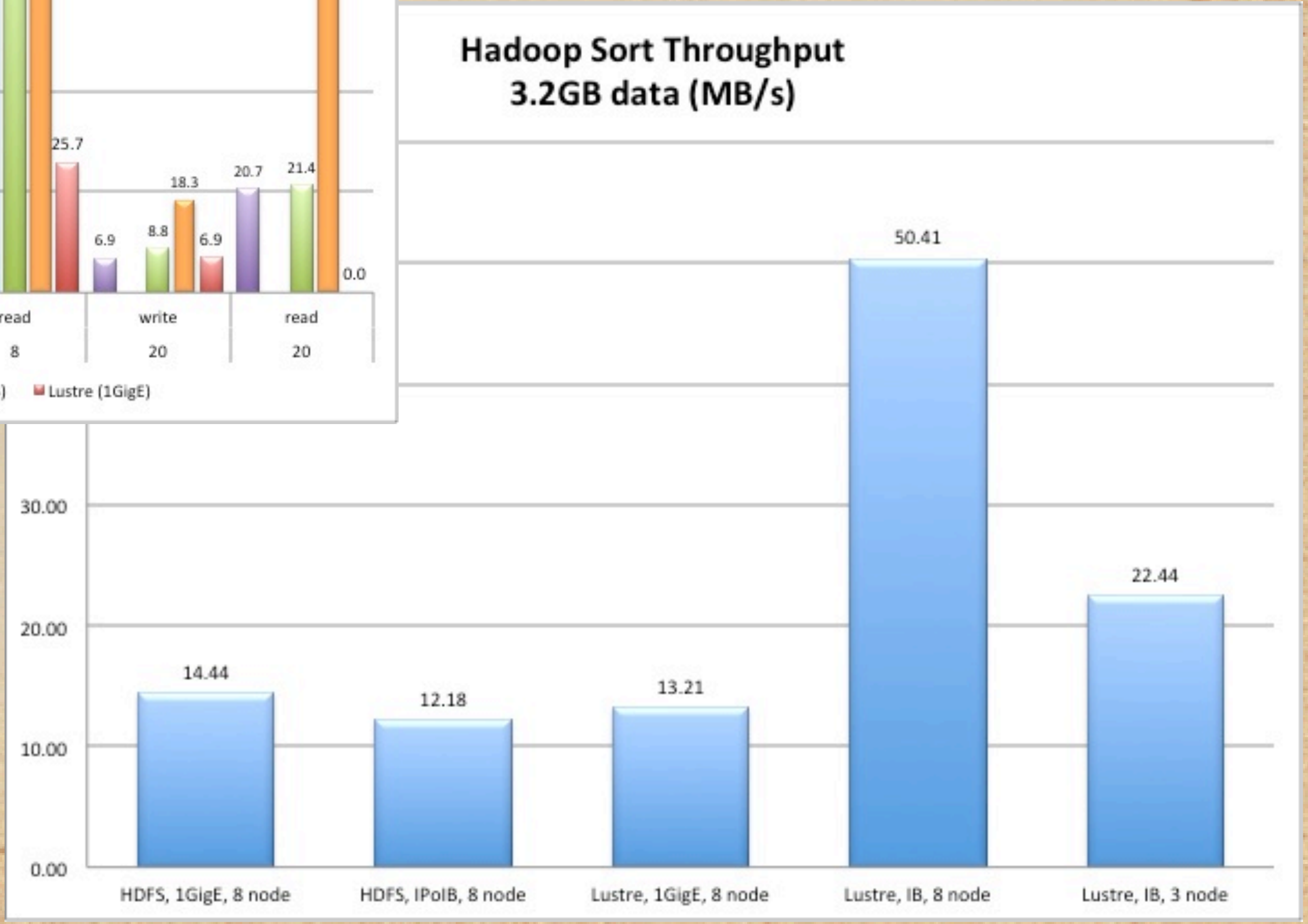
Conclusions

- Peer infrastructure are giving new possibilities
  - Still suffering on peak performance and
  - great benefit on failure resilience
- The local area network is evolving too:
  - the available bandwidth/€ is increasing rapidly
  - IB technology is becoming affordable compared with 10Gps ethernet technology

Back-up slides

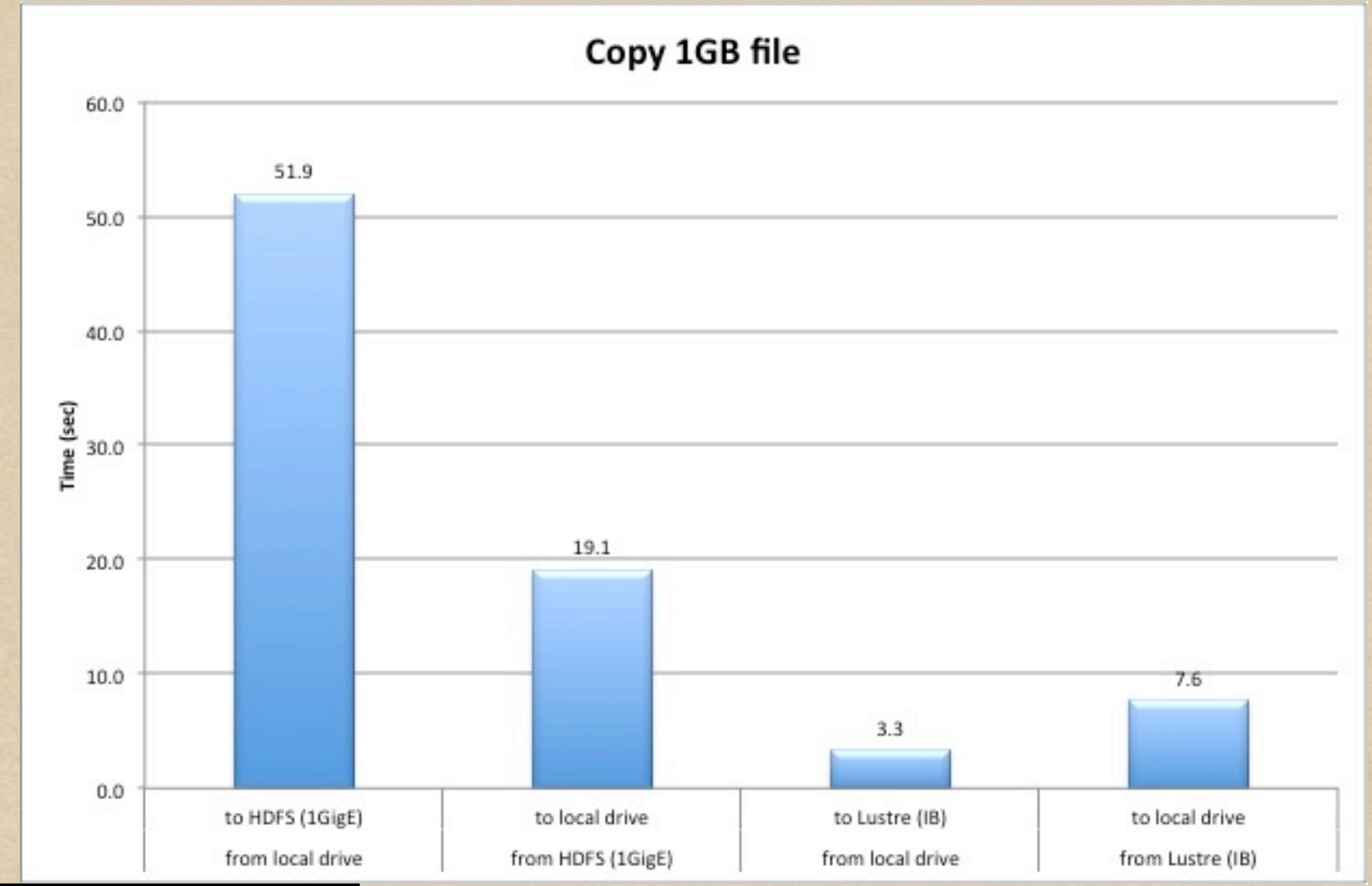
HDFS vs Lustre





Nathan Rutman, Xyratex
James B. Hofmann, Naval Research Laboratory

HDFS vs Lustre



Nathan Rutman, Xyratex
James B. Hofmann, Naval Research Laboratory

HDFS vs Lustre

Nathan Rutman, Xyratex
James B. Hofmann, Naval Research Laboratory

- Assume Lustre IB has 2x performance of HDFS 1GigE
  - 3x for our sort benchmark
- Top 500 LINPACK efficiency: 1GigE ~45-50%, 4xQDR ~90-95%

	Lustre / IB Cluster			HDFS / 1 GigE Cluster		
	Count	Price	Subtotal	Count	Price	Subtotal
Nodes	100	$7,500	$750,000	200	$7,500	$1,500,000
Switches	9	$6,500	$58,500	12	$4,000	$48,000
Cables	178	$100	$17,800	450	$10	$4,500
OSS	2	$52,000	$104,000	0	---	---
Storage	128TB	---	---	384TB	$100	$38,400
MDS	1	$34,000	$34,000	0	---	---
Racks	4	$8,000	$32,000	6	$8,000	$48,000
Total			**$996,300**			**$1,638,900**

Acknowledge

- testing on the storage technologies was supported by the MIUR (Italian Ministry for Education, University and Research) in the PRIN2008 project, under grant prot. 2008MHENNA 003.