Test comparativo soluzioni storage: hardware, filesystem e SRM

Giacinto Donvito

INFN-BARI

Outlook

- Why this test (requirements and goals)
- Description of the hw used
- Description of the client used
- Local test typology:
  - Serial Write, Serial Read, Random Read, Mixed Workload
- Final consideration
- CMS analysis job
- Test on Lustre
  - Tests@Bari
  - Tests@Torino
- Comparison between Lustre/StoRM and dCache
- Test on SSD:
  - For metadata handling and as ZFS cache
- Linux Software Raid experience
- Tier2 Scenario

LHC Tier2

- typical LHC Tier2 activity is:
  - 50% MonteCarlo Production
  - 50% Analysis
- typical LHC Tier2 size is:
  - ~600 CPU/core
  - ~2-300 TB
- typical LHC Tier2 I/O requirements:
  - Max 10MB/s per CPU/Core reading on the LAN (~5GB/s aggregate)
  - ~ 100MB/s writing aggregate

Goals of the test

- Testing hardware, software and storage system in order to achieve the required performance for a typical tier2 site
- we choose to test medium size storage boxes (~40TB) as those better fits the constraint for a tier2 site:
  - The cost is affordable
  - The number of box needed to achieve the aggregate bandwidth is not too high as it could become if smaller boxes were used
- Testing different storage system software in order to find the most efficient in dealing with the real CMS jobs
  - We choose to test dCache and Lustre as the first is in production since 5 years and the second is really promising
- We have kept into account also the electrical consumption and the rack unit required by each of available solutions
  - Trying to minimize those factor

Test storage

The results shown are referred to "local access" through underling file-system: nor network or application layer are involved

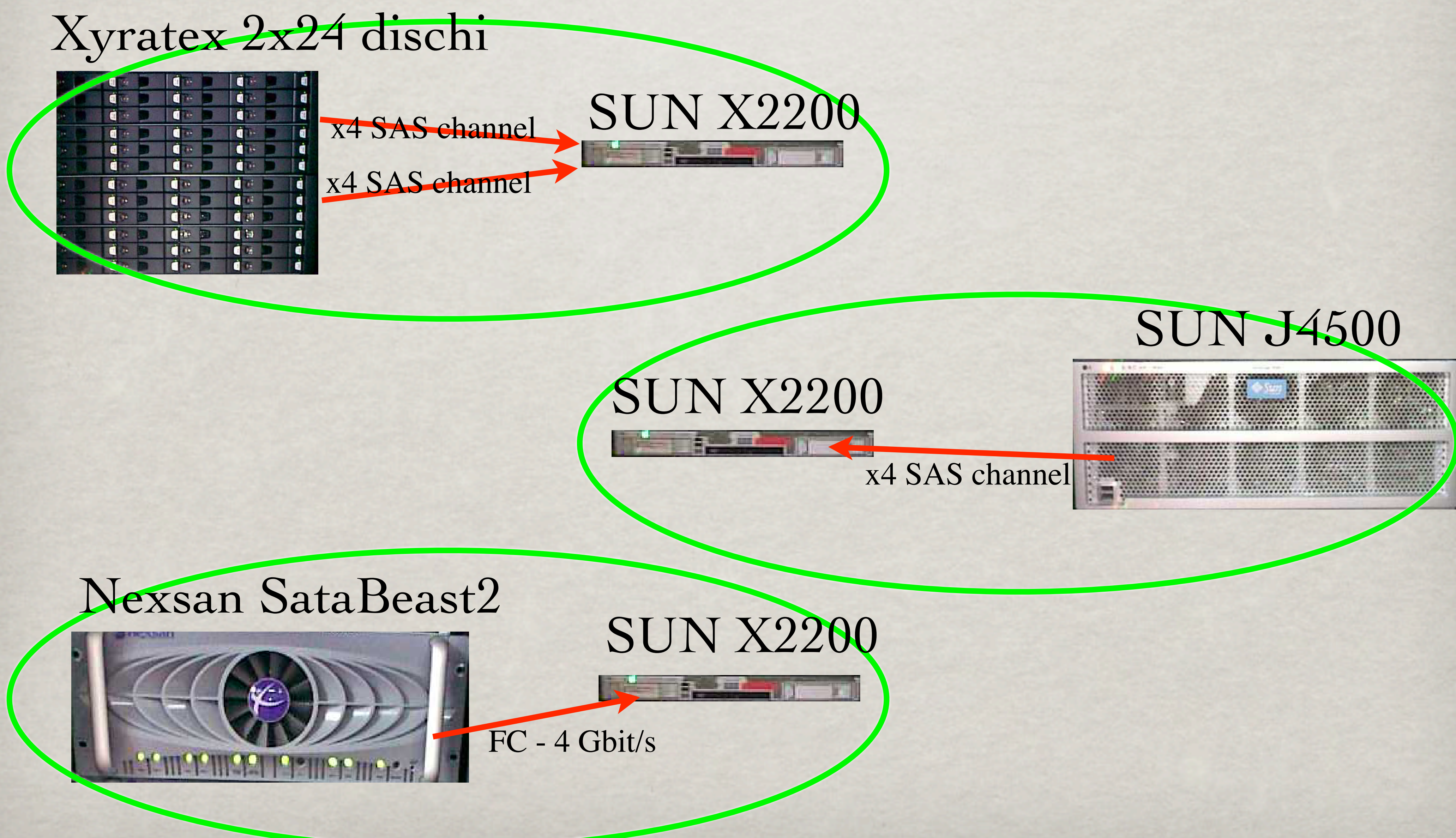
- several storage hardware has been tested, with few operating systems and file-systems
- the reference chunk size used is 256 Kbyte (as this is the default buffer for any dCache disk activity)
- the goal is to optimize the disk performance in order to be sure that the storage system software is not limited by the disk sub-system underneath
  - the disk sub-system in the case of dCache should provide the same performance of Lustre

Test storage

storage component under test

- **Hardware:**
  - Server:
    - SUN X2200: 4Core Opteron/16GB RAM/SUN SAS RAID/QLogic FC
  - Disk subsystem:
    - SUN J4500 48 dischi da 1TB (SAS attached)
    - Xyratex JBOD SAS 2x24Dischi
    - Nexsan SataBeast2 42 dischi 1TB 1x4Gbit FC
- **Software:**
  - Operating System:
    - OpenSolaris 2008.11
    - Scientific Linux 5.3
    - Debian Stable 5.0 (Lenny)
  - File-System:
    - XFS
    - ZFS
    - Lustre

Test storage
Hardware configuration



Test storage
Test description

- Test was performed with IOZone:
  - Open source software that provides high flexibility; it can test several aspects:
    - Metadata access, random access, pseudo random access, multi-threaded read/write, etc
  - It is standard so it could be used to compare with other tests and to evaluate also the application layer:
    - If there is too many differences between real application and the iozone test, this could mean that the application layer need to be optimized
  - It does not put so much load on the CPU and RAM sub-system so it is only limited from the I/O performance
  - All the tests are executed using 5GB files: this ensures that the RAM on the machine it is not able to keep all the data cached

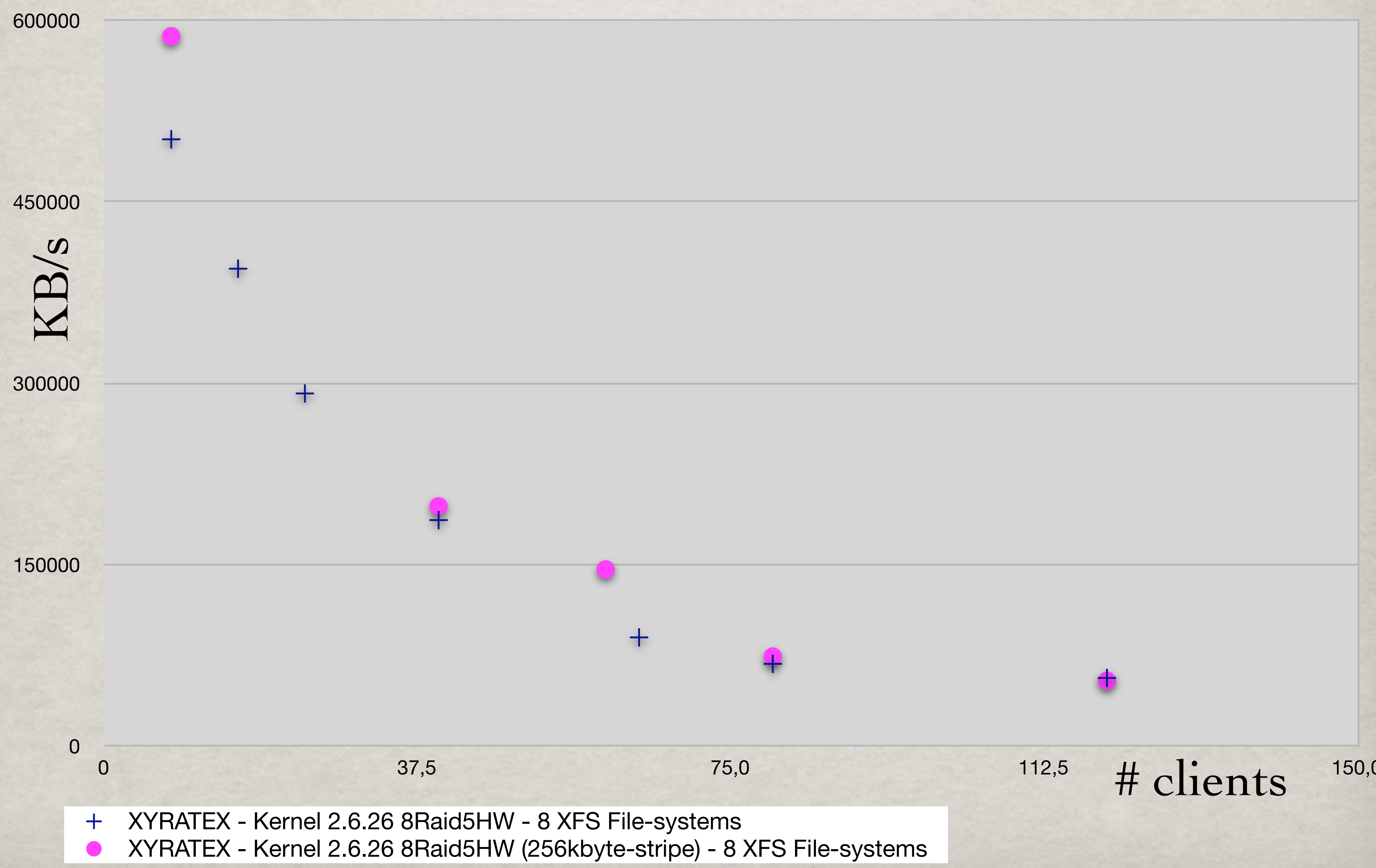
Test storage
serial write

- This test will measure the performance in writing files, when increasing the number of parallel processing and changing the HW or the file-system used
- Writing operations are sequential (from the beginning of the file to the end). The chunk size is always 256Kbyte
- This test involves also the metadata modification
    - especially with XFS and some hw it gives a significant decrease of performance
    - IOZone gives the possibility to measure the difference but as writing files without changing metadata is not useful in our environment (files are usually written one time and never modified) we do not show these results

Serial Write
SUN J4500

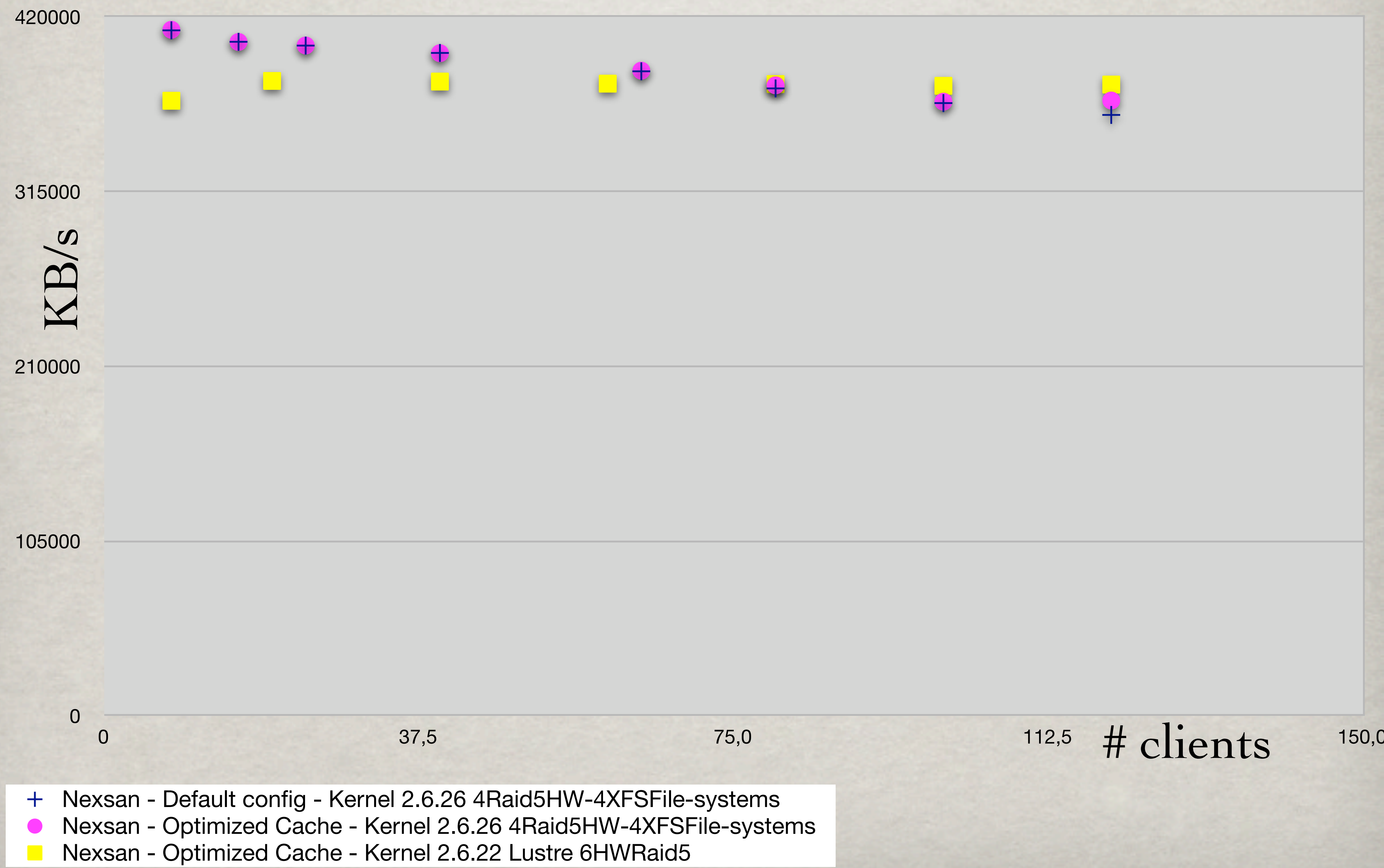
Serial Write

XYRATEX 2x24 Disk



Serial Write

Nexsan SataBeast2



Test storage – Results

serial write

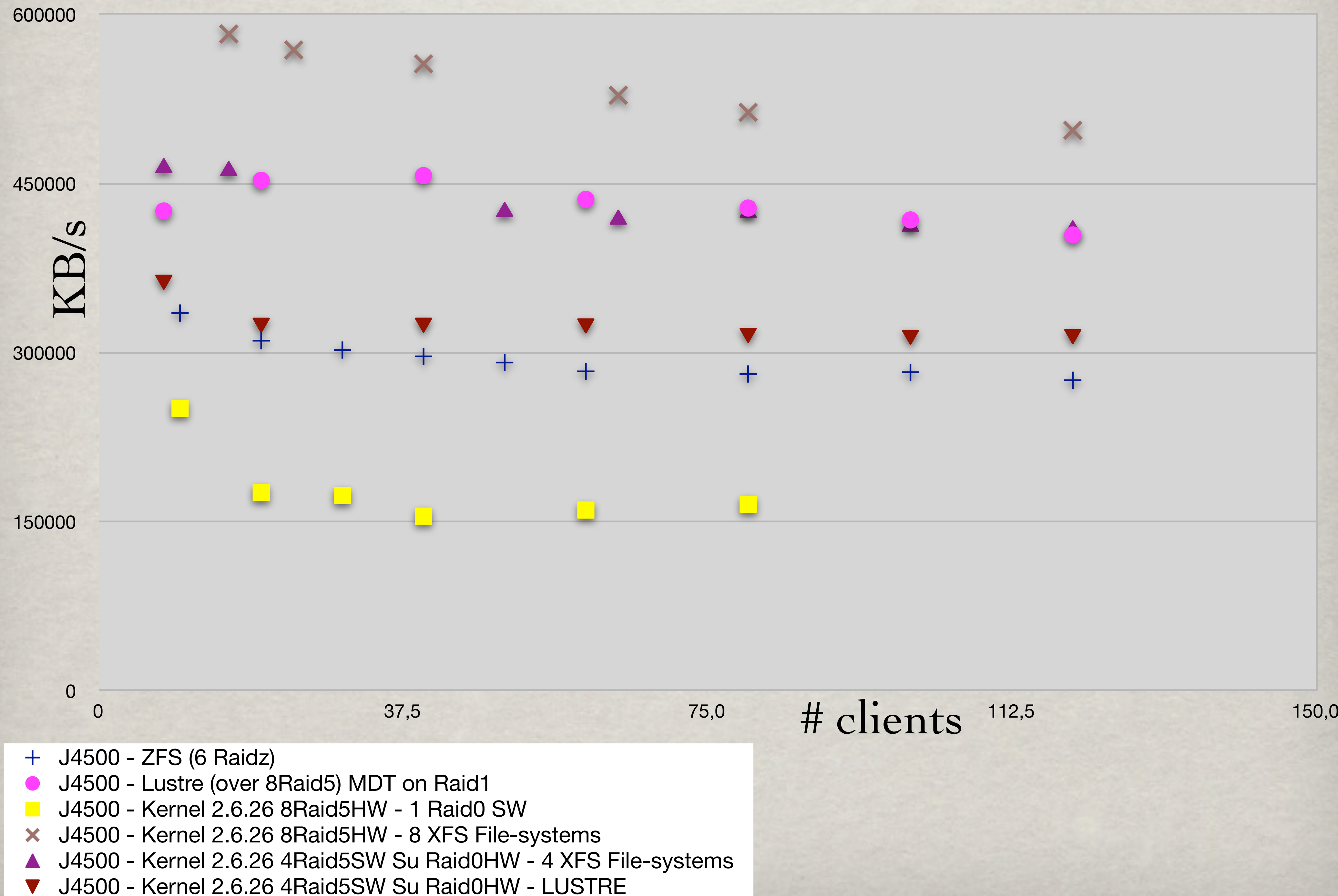
- The test shows that the usage of some raid card could have negative impact on the writing performance when the number of writing process increases
  - Usually the number of concurrent writing process is not so high for a typical tier2
  - it should be enough to serve about 10-20 writing process per disk server
- This test highlights that the nexsan raid or software raid (both linux and ZFS) are much more stable in dealing with increasing the number of concurrent writing process
  - It would be better to have separate device for containing metadata and journaling
  - the concurrency between writing data and metadata on the same device, is the main problem in this use case
- Lustre in this case greatly reduces this problem as the metadata are written on a dedicated device

Test storage
serial read

- The test measures the perfomance in sequential reading
  - showing the behavior when the number of concurrent processes increases
- all the access are executed using 256Kbyte of chunk size

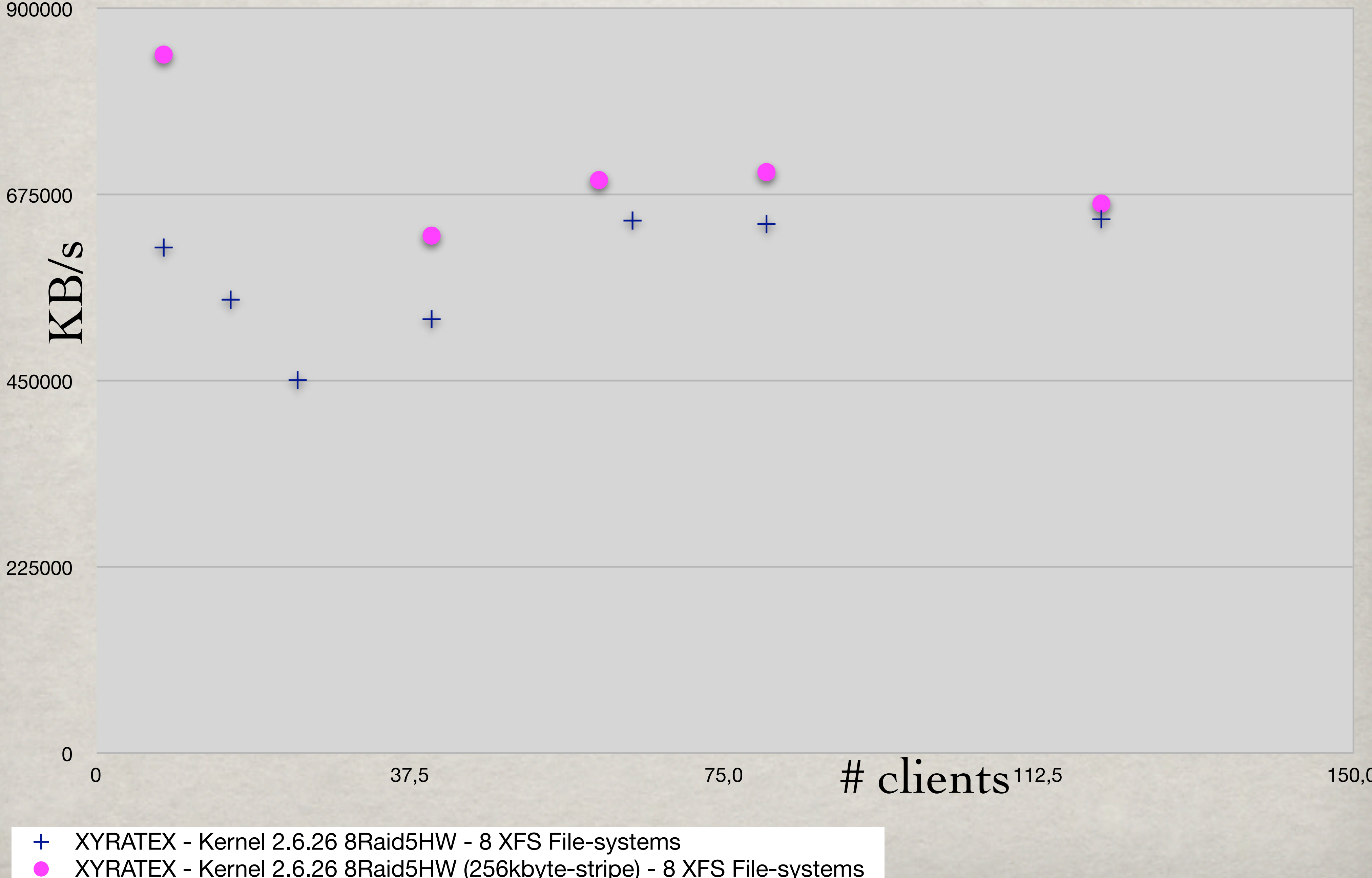
Serial Read

SUN J4500

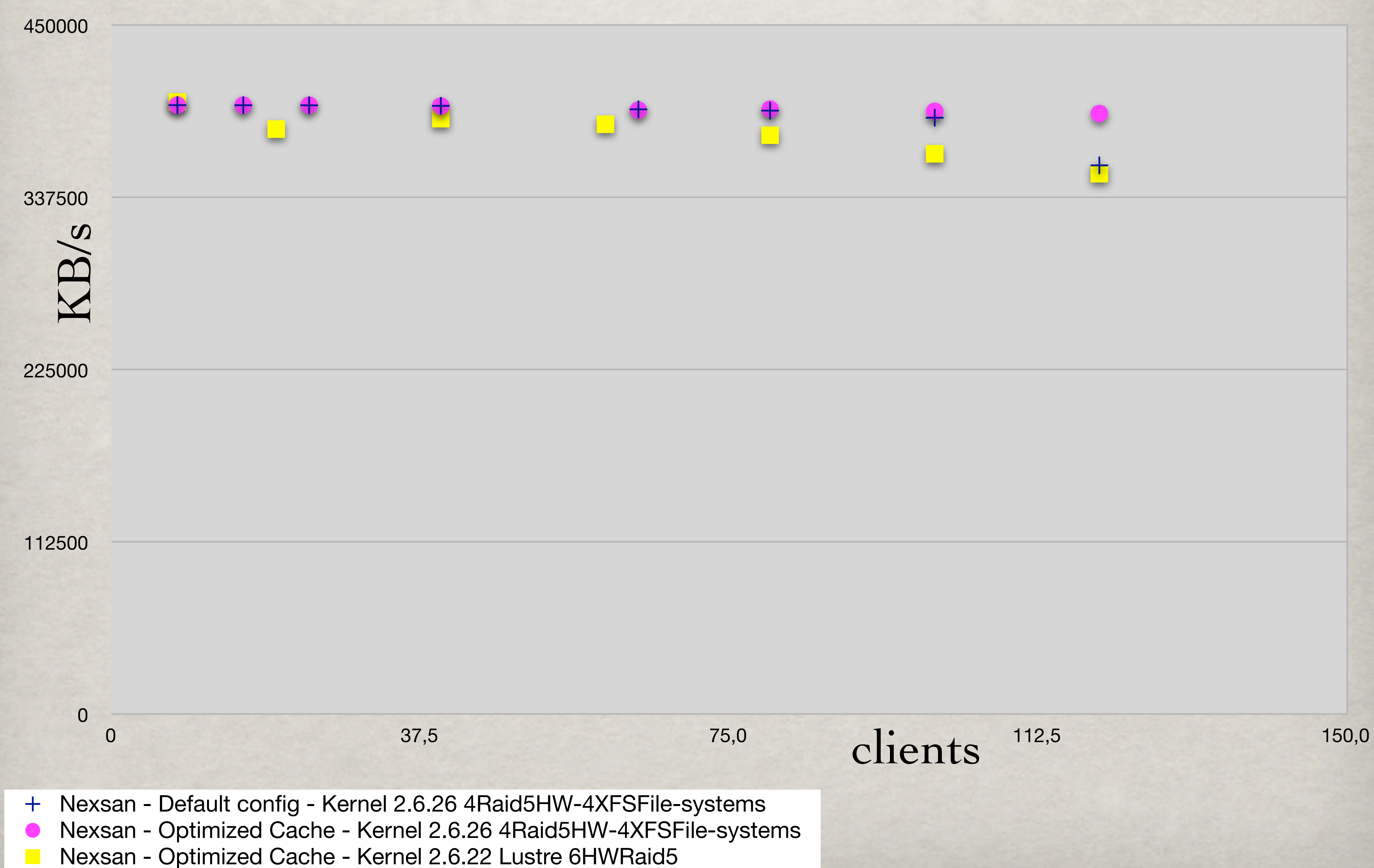


Serial Read

XYRATEX 2x24 Disk



Serial Read
Nexsan SataBeast2



Test storage – Results

serial read

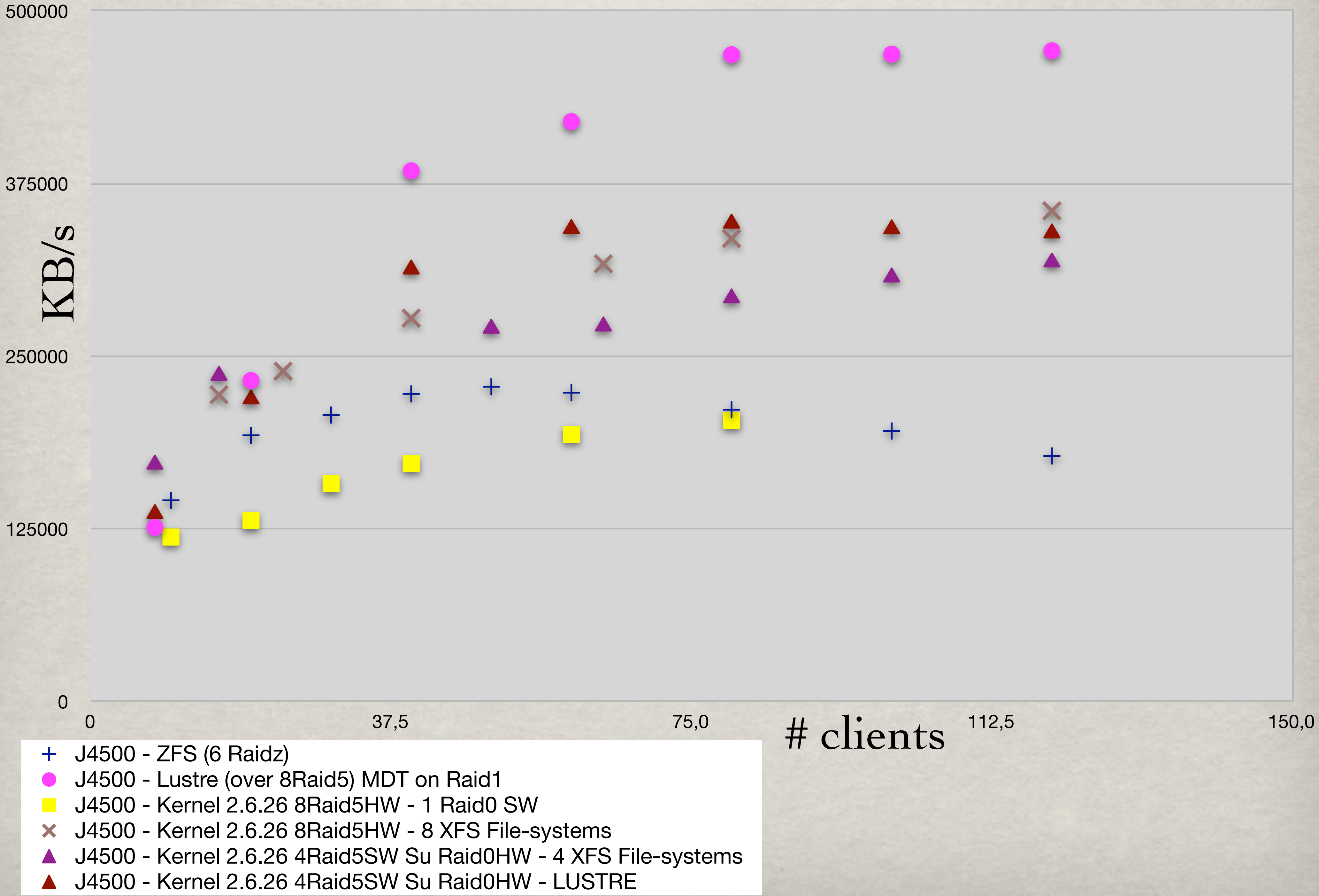
- The test shows that the Xyratex JBOD has a greater bandwidth (as result of implementing a 2x24 disks arrays)
- Also in this case the nexsan controller provides a good stability in performance when the number of processes increases
- Using Lustre in this case do not provide a big improvement

Test storage
Random read

- This test measures the performance in reading data using a random pattern access
  - showing the behavior when the number of concurrent processes increase
  - The chunk size also in this case is 256Kbyte

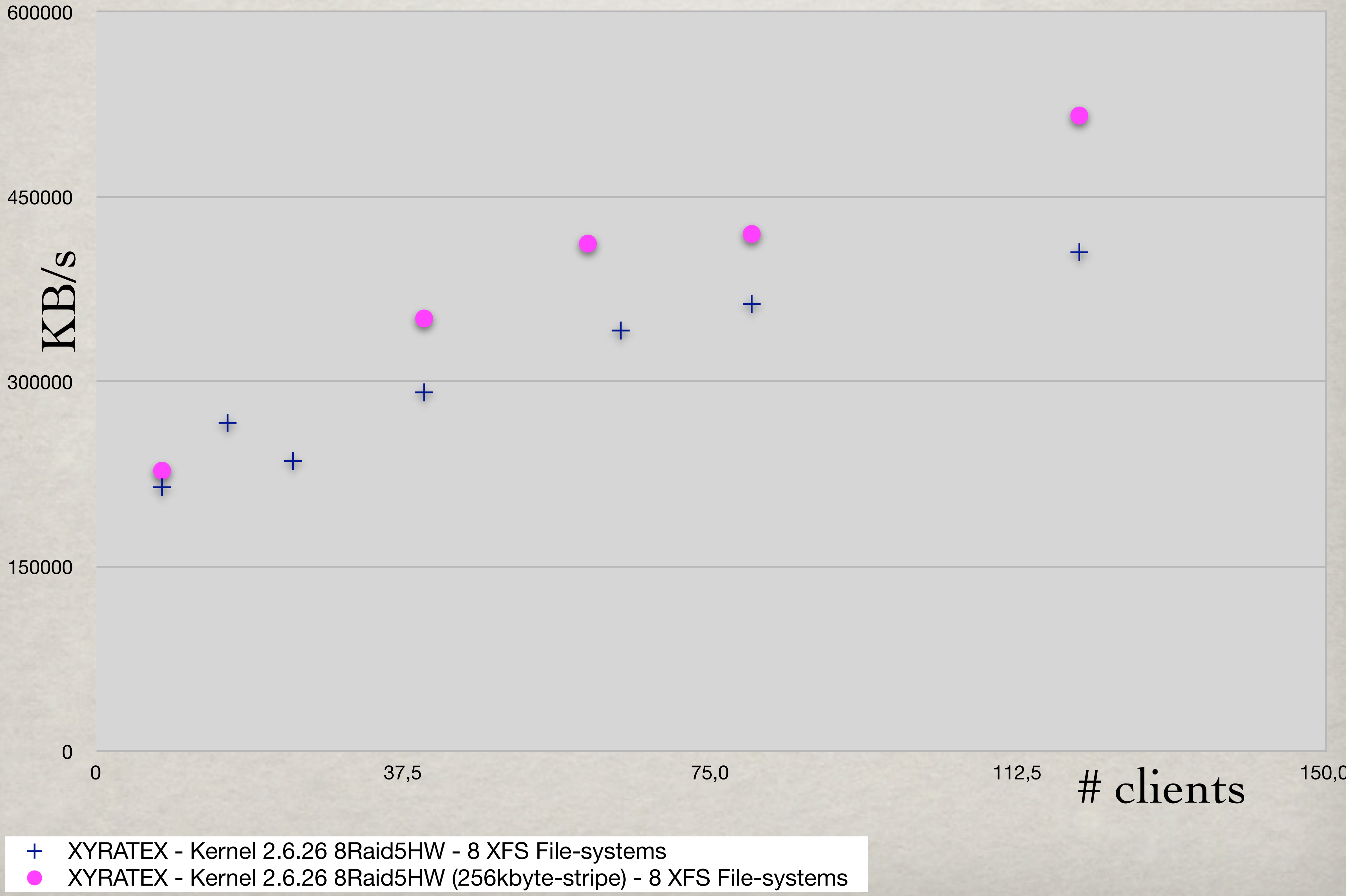
Random Read

SUN J4500



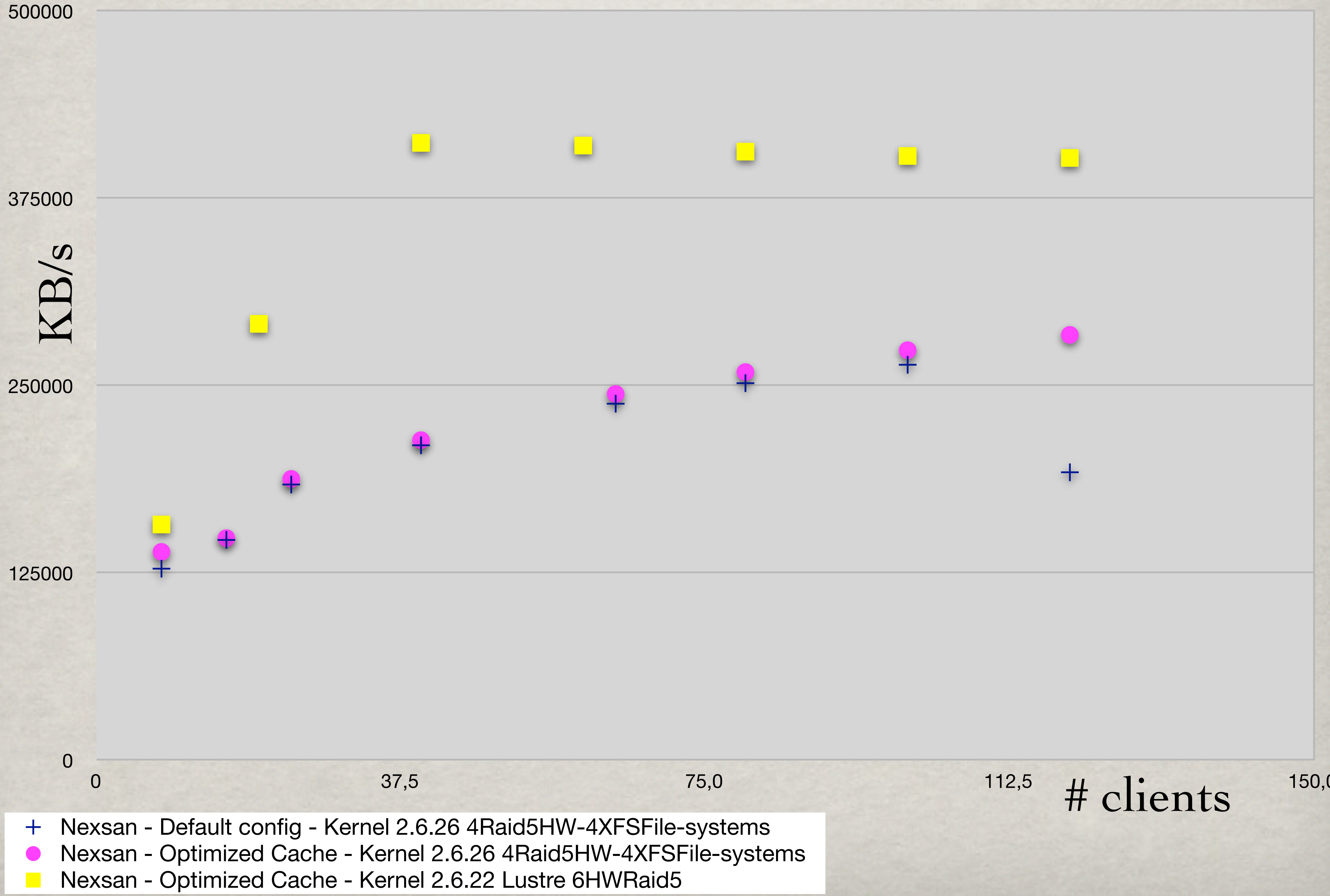
Random Read

XYRATEX 2x24 Disk



Random Read

Nexsan SataBeast2



Test storage – Results

random read

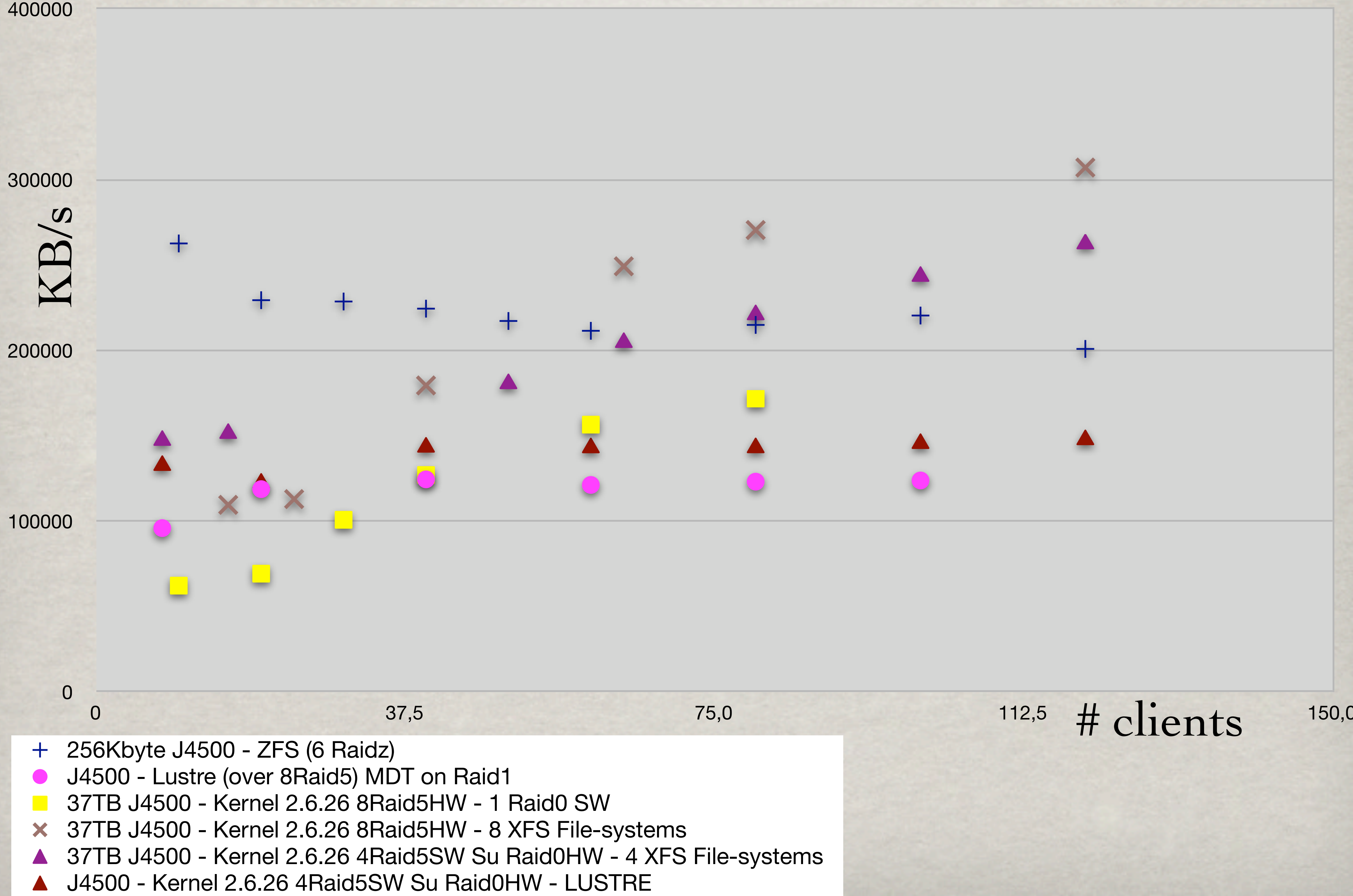
- This test shows that Lustre is able to increase the overall performance for a given hardware
- It is evident that increasing the stripe size at the hw raid level gives relevant improvement (look at the test with the xyratex jbod)
- In this test it is needed a fine tuning of the cache configuration for the nexsan controller in order to improve the performance
  - in this configuration it is possible to obtain similar performance with the other hardware that are configured with a larger number of devices

Test storage
Mixed workload

- This test measures the performance when there are both writing and reading access concurrent on the same server
  - the chunk size (for both reading and writing processes) is 256Kbyte
  - The processes are 50% writing and 50% reading
  - This is not the real case for a typical tier2 as the writing process have a minor role than reading process in the analysis activity

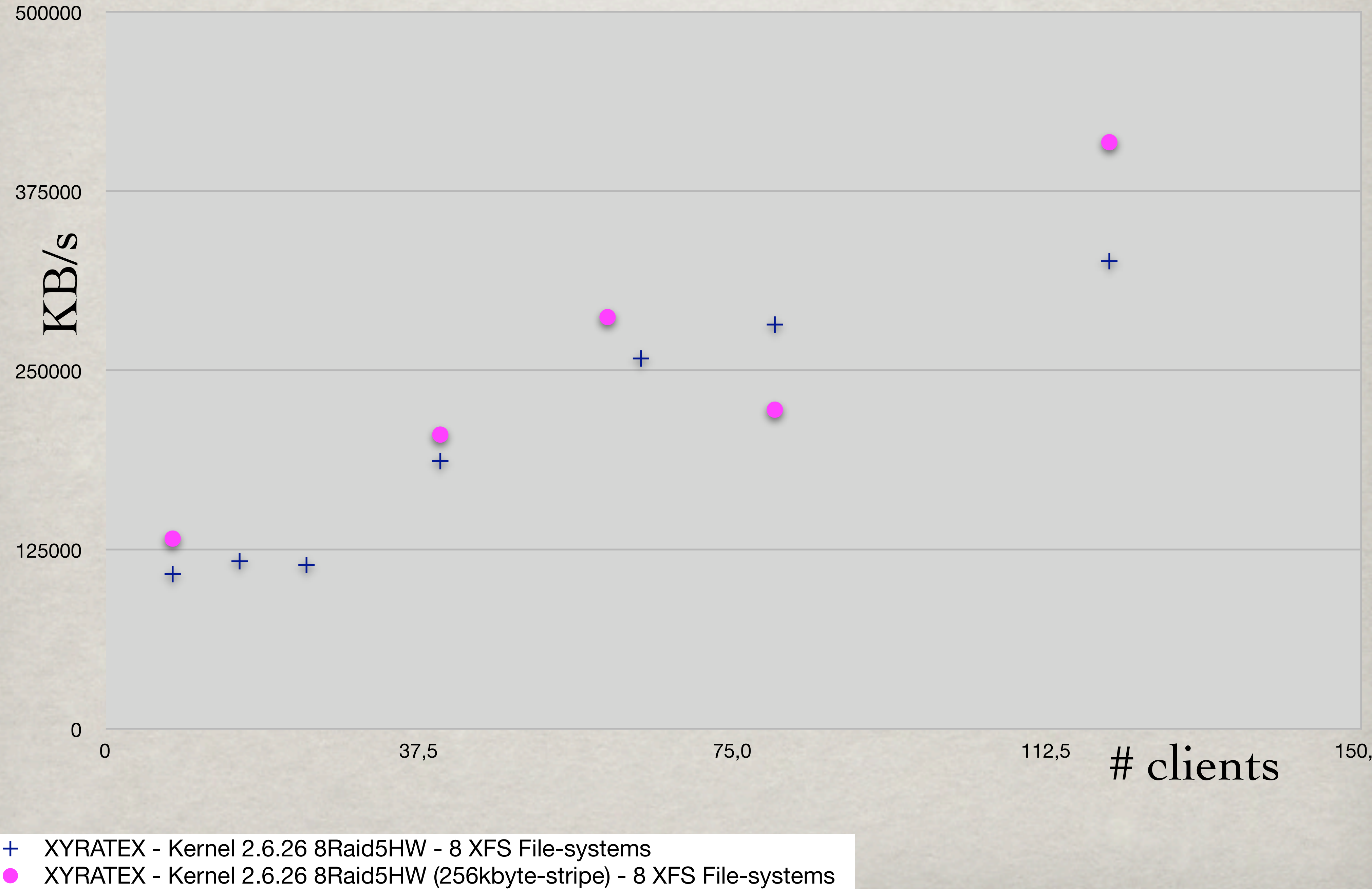
Mixed Workload

50% Read -- 50% Write -- SUN J4500



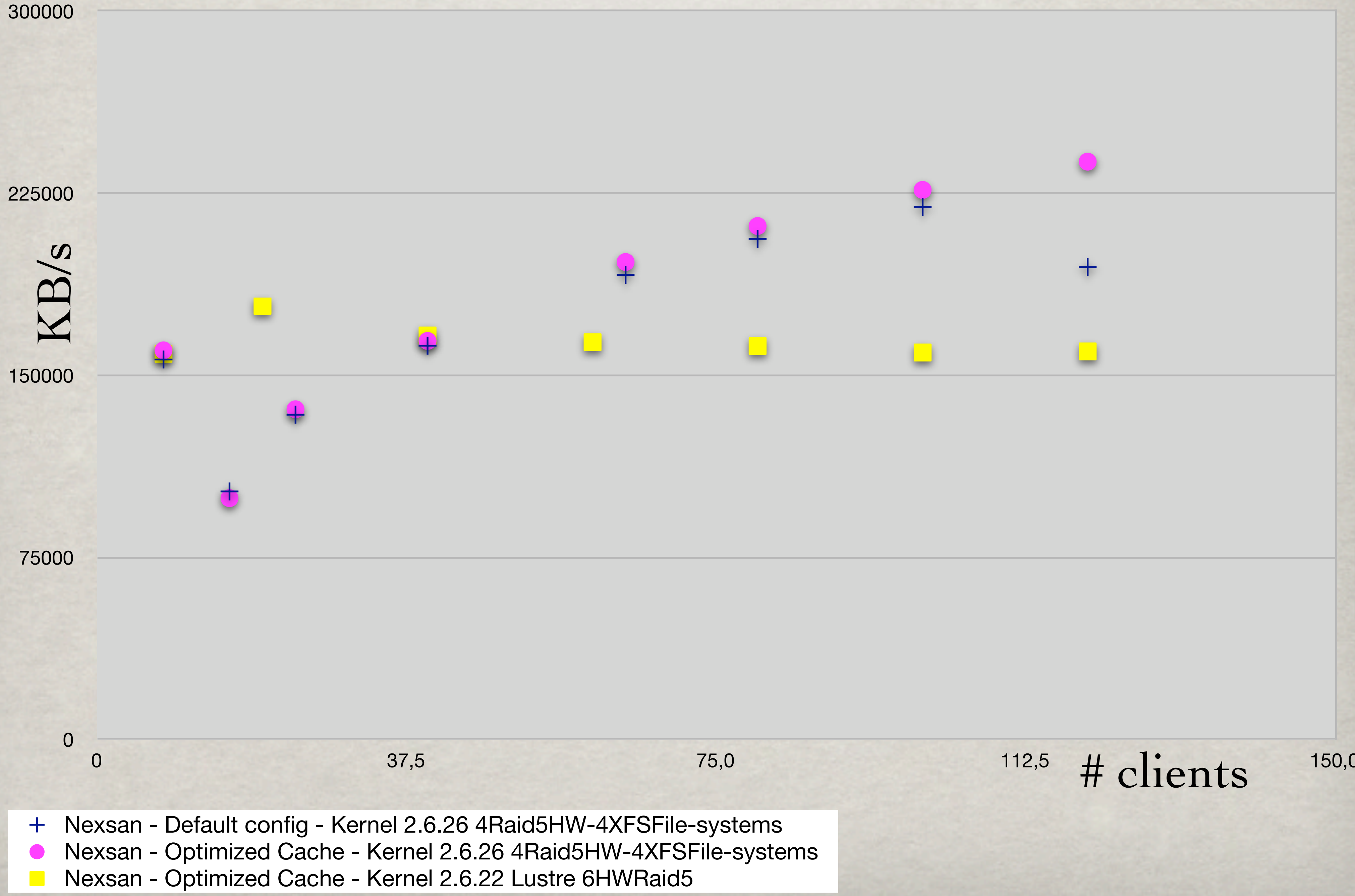
Mixed Workload

50% Read -- 50% Write -- XYRATEX 2x24 Disk



Mixed Workload

50% Read -- 50% Write -- Nexsan SataBeast2



Test storage – Results

Mixed Workload

- This test shows that the configuration with a larger number of devices (and file-system) deals better with concurrent writing and reading processes
- Lustre in this case seems non to overkill other solutions, in the next slides we will show a real use case of using lustre with a mixed access patterns

Final thoughts

HARDWARE

- **Nexsan:**
  - Really a good controller: it is able to sustain the load of a great number of concurrent writing processes -> good for a “import buffer” (for example in front of a tape infrastructure)
  - It could be interesting to have a higher number of raid devices in order to increase the random read performance (it would require a “waste” of usable disk space)
- **SUN J4500:**
  - The performances are good enough but it has the main advances is the good ratio between TB provided and Rack Unit and Watt required

Final thoughts

- **Xyratex:**
  - Great ratio between performance and price
  - It requires twice the space of a J4500
- **Software Raid:**
  - The performance achieved in the test shows that it reaches good level
  - Test executed on reliability in case of failure shows good behaviour (see next slides)

Final thoughts

SOFTWARE

- **XFS:**
  - a precise fine tuning was needed in order to improve the performance (specially for the writing)
    - blockdev --setbsz 4096 --setra 8192 /dev/sd$i ; mkfs.xfs -i size=2048 -b size=4096 -s size=4096 -f /dev/sd$i
    - it is still possible to improve the behaviour in writing (with a high number of parallel processes) using a small dedicated device for journaling

- **OS:**
  - In all the cases in which it was possible we preferred the latest Debian Stable 5.0 (Lenny) that proved to be greatly stable and performant
    - default kernel used: 2.6.26-x

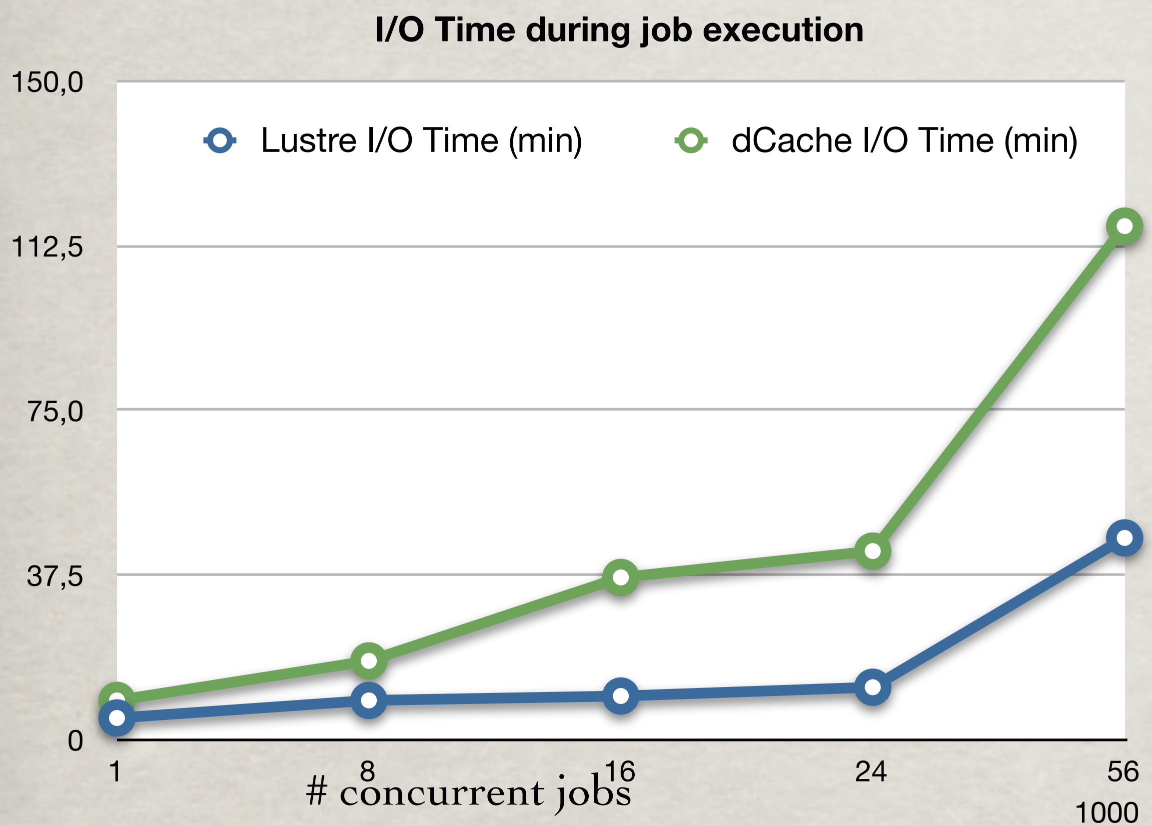
Final thoughts

- **Kernel:**
  - In all the cases in which Scientific Linux is required we have recompiled a vanilla kernel 2.6.22 compiling all the needed driver:
    - This usually gives us the possibility to solve some performance issues on some specific hw devices (SUN SAS Raid controller for example)
- **Lustre:**
  - Great scalability when the number of concurrent (random) reading processes increases
- **ZFS:**
  - It is very stable and show fairly good performance in all the cases even if it does not overkill the other in none of the tests

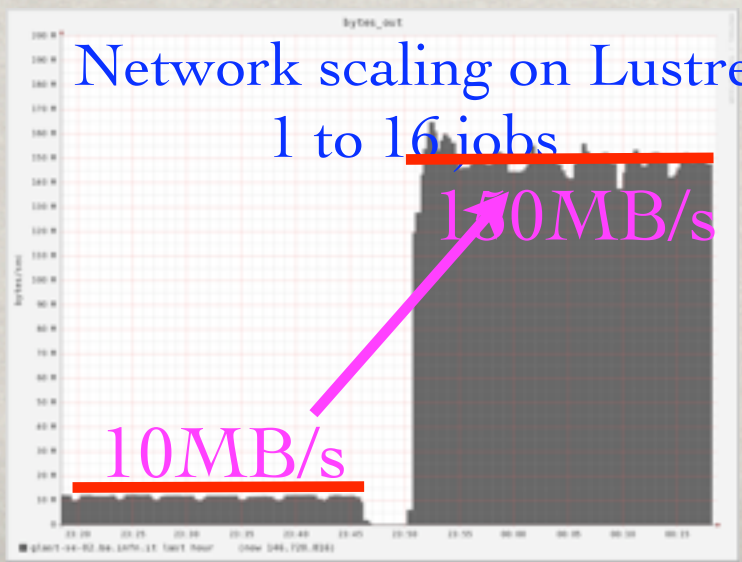
CMS analysis job

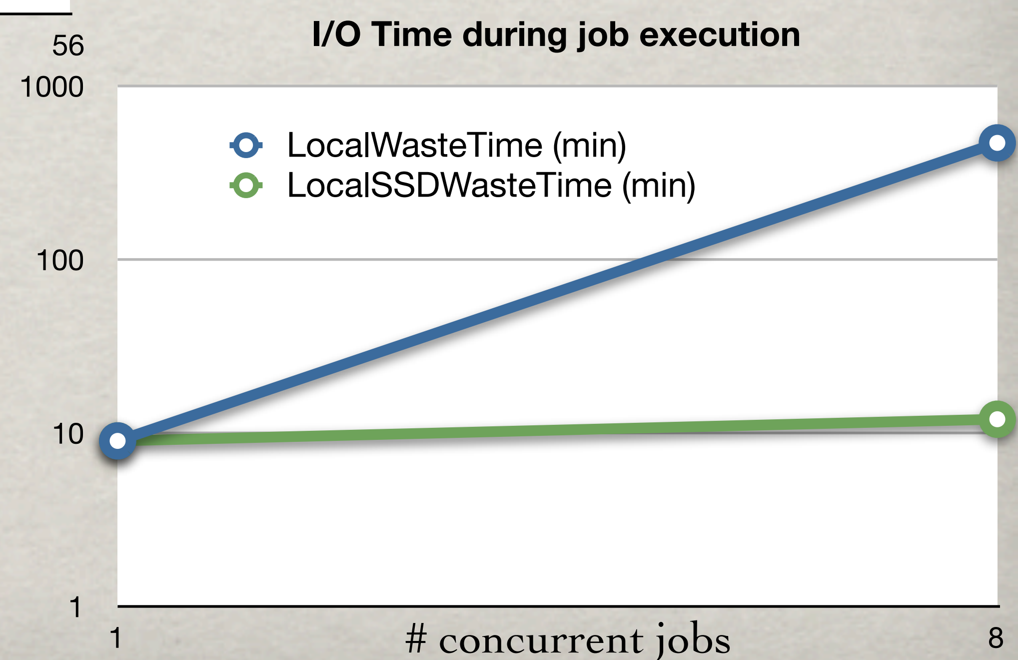
- The goal of the test is to measure the scalability of each configuration (meaning starting from the hw to the software used to manage the storage)
- this will help to understand how to build a storage infrastructure for a CMS Tier2 (this could be easily used for other LHC experiment too) in order to achieve the requested performance for serving the chaotic end-user analysis
- As test hw a nexsan configured with 6 Raid devices will be shown
- The first step is to optimize the running time of a single job, tuning all the parameters available (in dCache test, for example, it was of great help to change the read-ahead buffer).
  - we used this configuration in order to run the dCache test shown in the next slides, while no tuning was performed in the case of Lustre test
- When using dCache was not possible to use "Vector read-ahead" as the version of root used by CMS framework was still not able to use it.
  - test are planned to measure which will be the improvement in case of a "vector read-ahead" enabled root version
- The network is configured (bonding 4 different gigabit card) and tested in order to be sure that it is not a bottleneck

CMS analysis Job



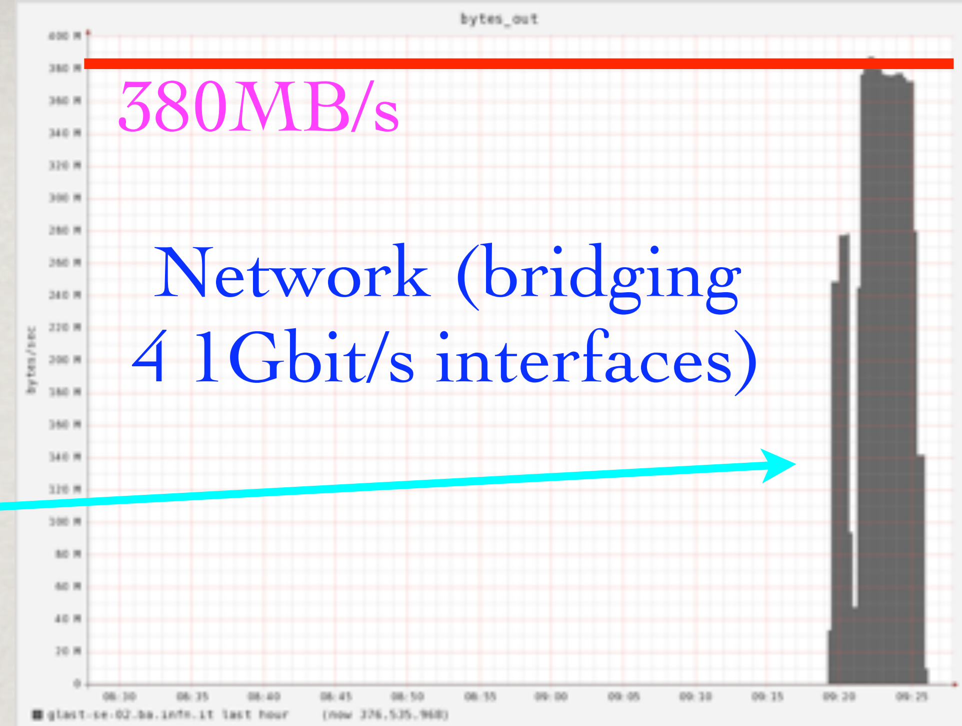
The plots shows the time “wasted” in making I/O during job execution using local disks (SATA or SSD) and remote storage (dCache or Lustre)

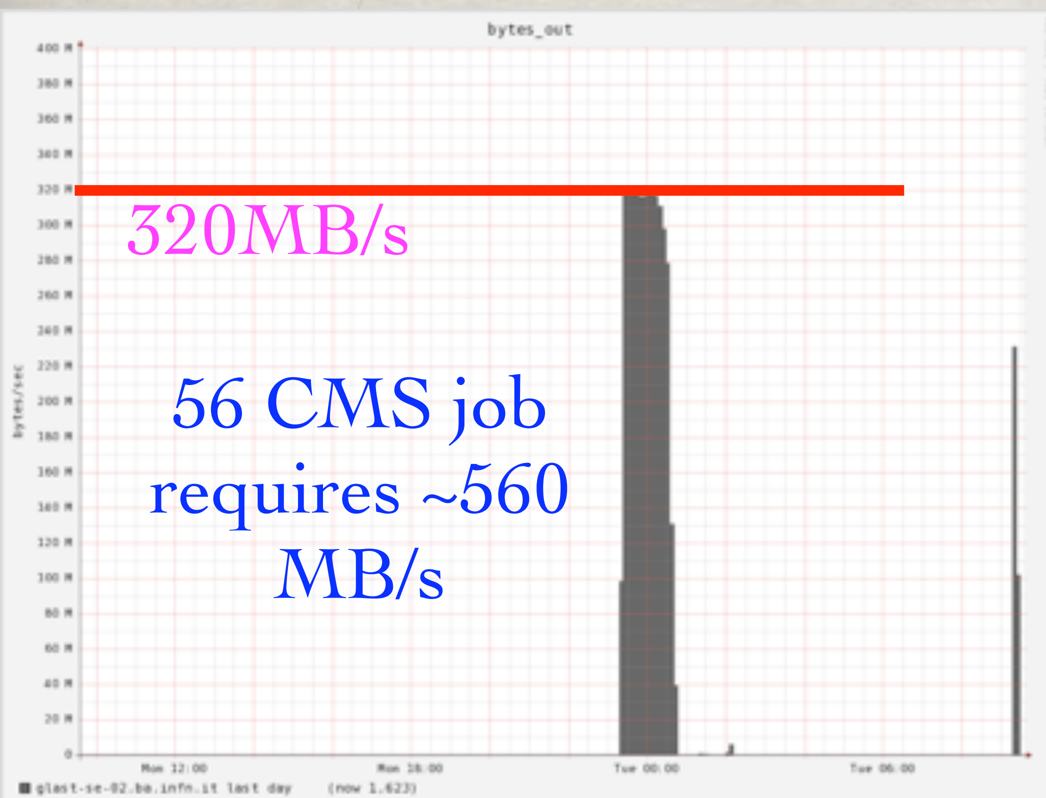


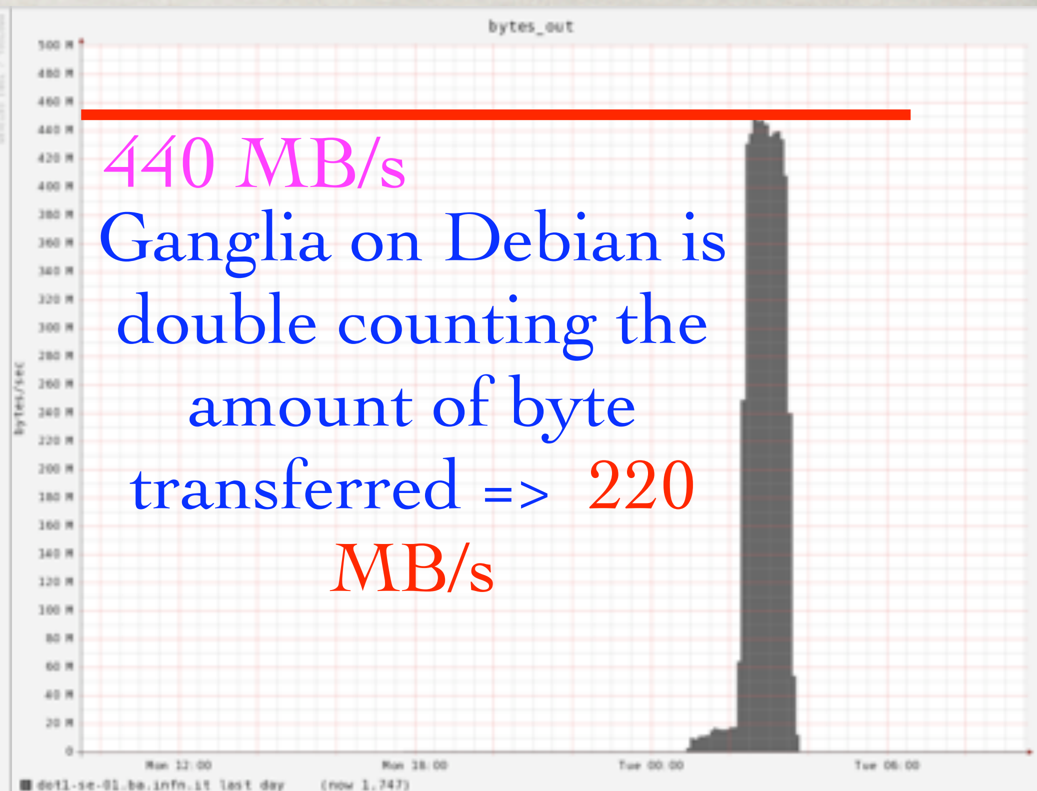


CMS analysis Job

The network link between server and clients has been tested with "iperf"





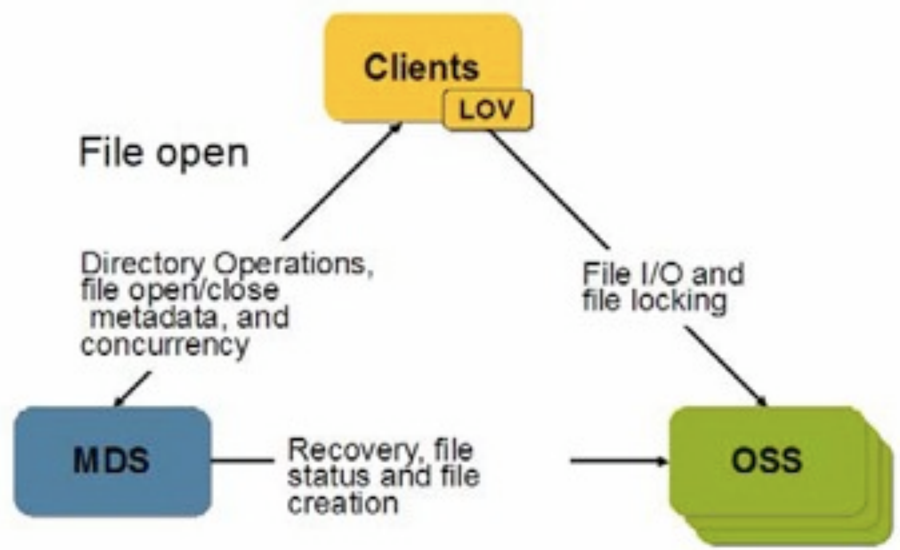


CMS analysis job results

- The result shows that:
  - The network is not the bottleneck with this hw
  - Using Lustre, the I/O achieved with the CMS jobs is about the 80% of the rate achieved locally in random read test with IOzone
    - It seems not so bad considering the network latency and the application layer
  - Lustre is exploiting, with a good efficiency, the ability to cache data locally at the client level
    - This behaviour could still be tuned by configuring the buffer size at the client level
  - dCache suffers of lack of performance also when 24 jobs are running concurrently
    - We could expect that "vector read-ahead" will improve this behaviour but test are still going on involving the developers
  - The test with the local access shows that a single SATA disk does not have the required performance for providing data to 8 cores machine, while once using SSD we can easily be limited by the WN CPU

Typical Lustre infrastructure

- Lustre file-system is a typical parallel file-system in which all the client are able to use standard posix call to access files
- The architecture is designed in order to have 3 different function that can be spitted among different host or joined in the same machine:
  - MDS: this service hosts the metadata information about each file and its location
    - There could be basically one active MDS per file-system
  - OSS: is the service that hosts the data
    - There could be up to 1000 OSS
  - Client: are the hosts that are able to read lustre file-system
    - There could be up to 20000 client in a cluster

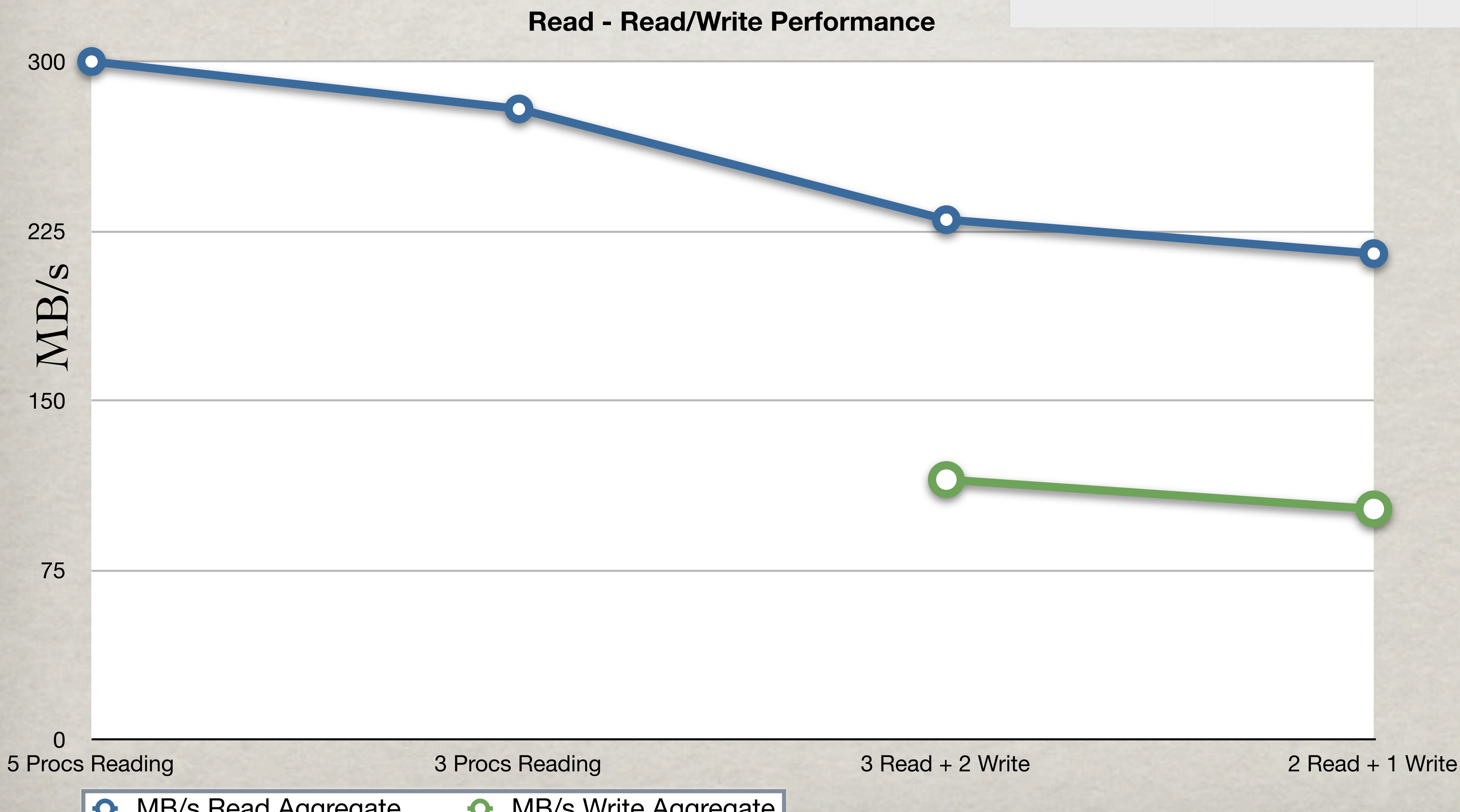


More tests on Lustre

- In order to better understand the behaviour in the mixed (reading and writing) pattern environment we tried to measure the performance in a "real life" usage through "dd":
  - we executed few test in which a given number of concurrent "dd" (5 or 3) are performed in "read-only" mode or "read-write" mode.

More tests on Lustre

Lustre	MB/s Read Aggregate	MB/s Write Aggregate
5 Procs Reading	300	
3 Procs Reading	279	
3 Read + 2 Write	230	115
2 Read + 1 Write	215	102

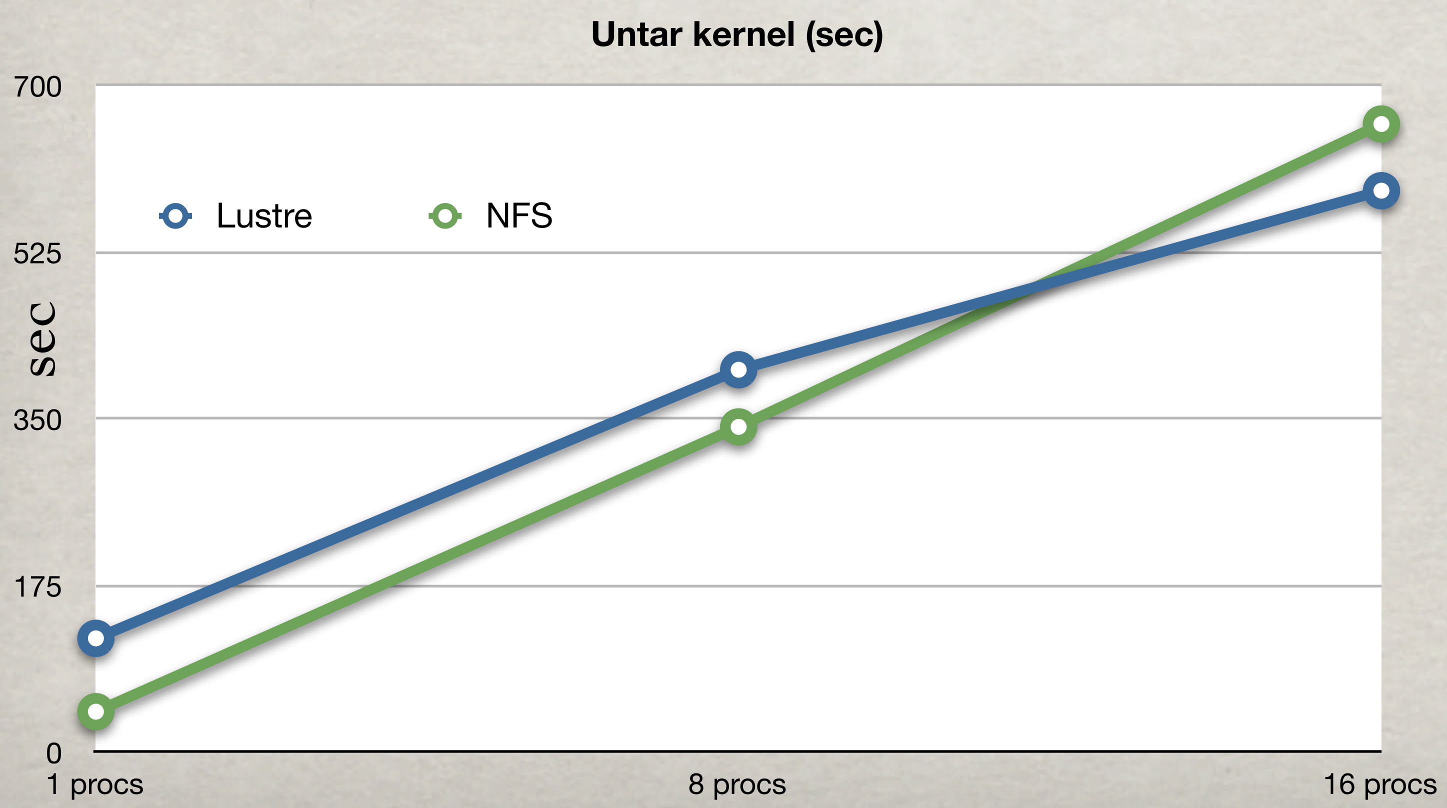


More tests on Lustre

- Calculating space occupancy:
  - "du" on 1M of lustre files:
    - 14 min
  - "du" on 300k of dCache (on PNFS) files:
    - 1 hour

More tests on Lustre

- The test measures the time needed to "untar" the kernel tar.bz2
  - it is a 54MB package resulting in ~30K files (368MB)
- The plot shows the behaviour while increasing the number of concurrent running processes

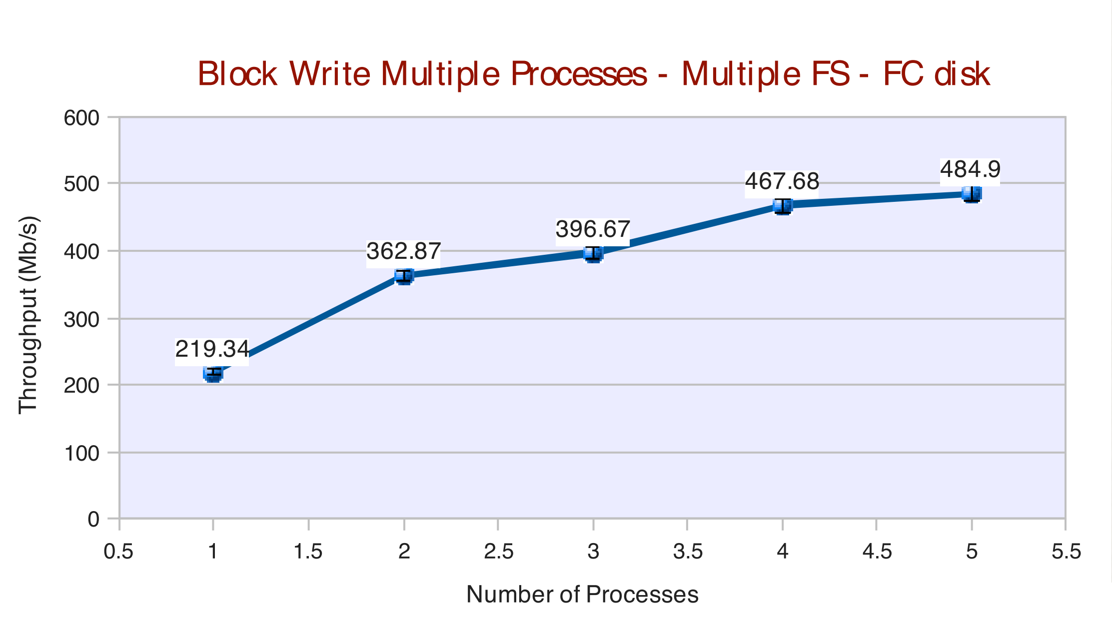


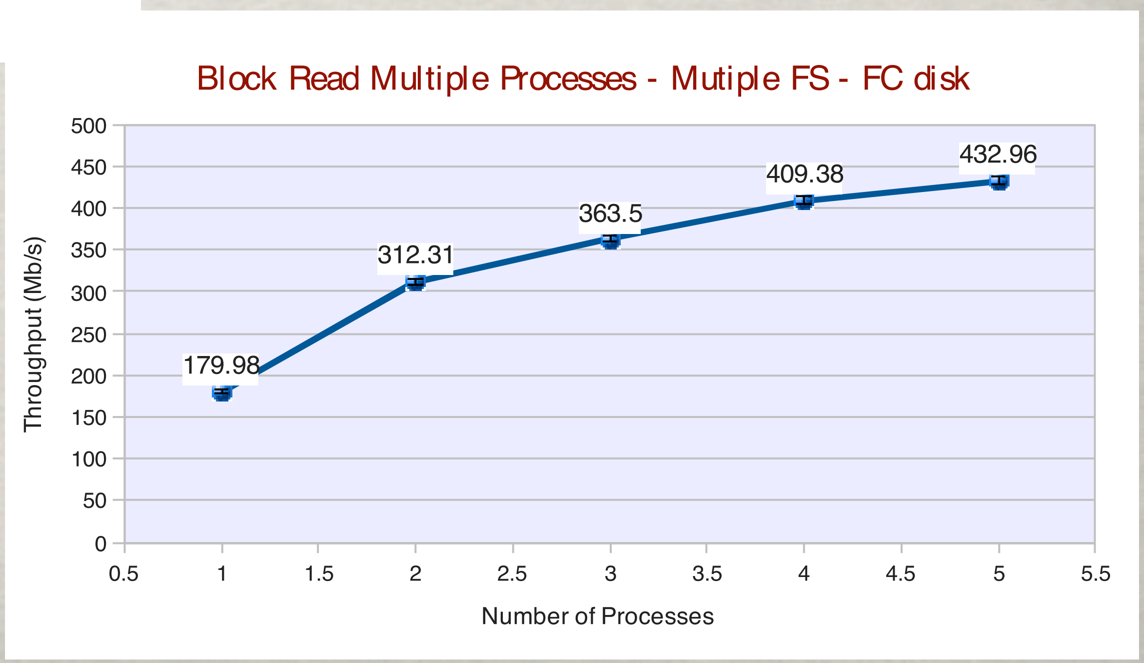
Lustre tests@INFN-Torino

- Servers:
  - IBM x3550 16GB RAM 2*E5420 (Lustre OST)
  - HP Proliant DL360G5 4GB RAM 2*E5130 (Lustre MDS)
    - CentOS 5.3 (64bit) [2.6.18-92.1.17.el5_lustre.1.6.7.1]
- Storage:
  - Sun Storage 6580 con 96 HDD 1TB SATA 7k
- Clients:
  - IBM Blade HS21 16GB RAM 2*E5420
    - CentOS 4.7 (32bit) [2.6.9-78.0.13.EL lustre 1.6.6]

Lustre tests@INFN-Torino

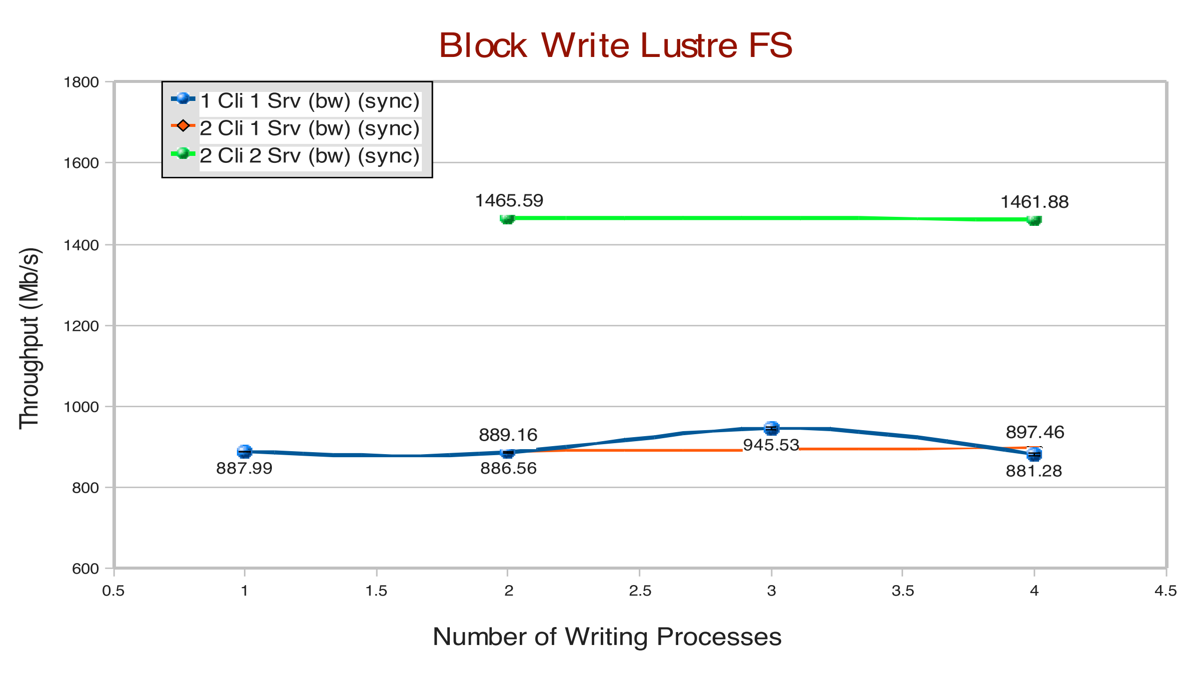
Preliminary Results (bonnie++)
Read/Write all file-system used

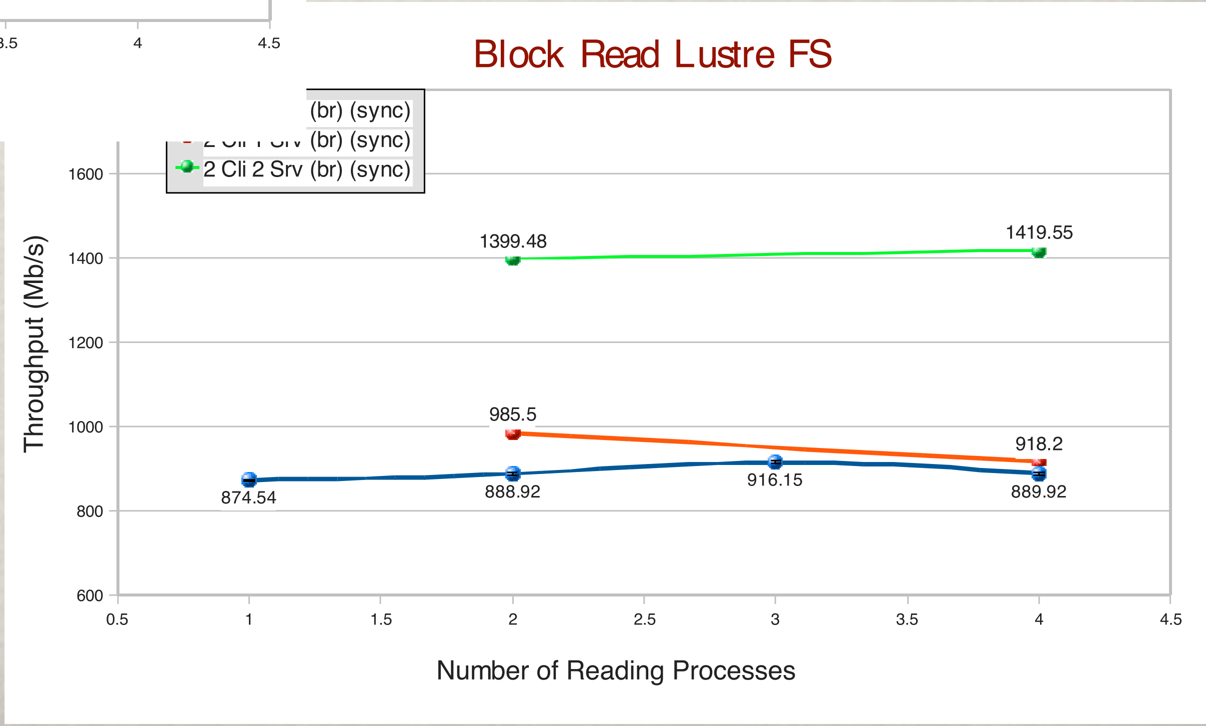




Lustre tests@INFN-Torino

Preliminary Results (bonnie++)
Read/Write Lustre





Testing Lustre 1.6.7

- All the operation are possible using few command line utilities and the /proc/ file-system
  - The interface is very “admin-friendly”
- It is quite easy to put an OST in read-only
- It is possible to make snapshot and back-up using standard linux tool and features like *LVM* and *rsync*
- it is possible to define easily how many stripes should be used to write each file and how big they will be (this could be configured at a file or directory level)
- Using SAN it is possible to serve the same OST with two servers and enable the automatic fail-over

Testing Lustre 1.6.7

- Very fast metadata handling
  - FNAL presented at CHEP09 “few thousand of ops”
  - we found that it is easy to create more than 1000 directories per second
- In case of an OST failure only the file (fully or partially) contained in that partition becomes unavailable
  - it is still possible to read partially the file in case it is split on few devices
- It is possible to have a “live copy” of each device (for example using DRDB and heartbeat)
  - it is feasible for both data and metadata
- It is possible to use kerberos authentication instead of the NFS-like one

Testing Lustre 1.6.7

- The client caches both data and metadata in kernel space
  - (temporarily) failure of a server are not disruptive in case of repetitive operation
  - The cache buffer on the client is shared: this is an advanced if several processes read the same file
  - The size of this buffer could be tuned (by /proc/ file-system)
- It is easy to understand which pool hosts each file
- The performance obtained by the application do not depend on the version of the library used (this could help when old experiment framework is still used)
- It is possible to tune the algorithm used in order to distribute the files over the pool, giving more or less importance to the space available on the OST itself

Testing Lustre 1.6.7

- It is possible to enable quotas per user or group
  - In the current version it is better to have OST smaller than 4TB
- Standard Posix ACLs are supported: it is possible to use standard unix tool to manage them
  - The ACLs should be enabled “system-wide” (on or off for the whole cluster)
- SRM layer is not built-in with the file-system
  - It is needed to install and manage srm/gridftp/xrootd software together with lustre layer
- It is needed to recompile the kernel in order to install lustre (also on client) or it is possible to use one of few kernels provided from the official web-site
  - Not all the kernel release are fully supported (<= 2.6.22)

News on Lustre 1.8.x

- **The latest version was released few days ago:**
  - **A lot of bug fix and few very interesting new features:**
  - OSS Read Cache:
    - It is now possible to cache read-only data on an OSS
    - It uses a regular Linux "pagecache" to store the data
    - OSS read cache improves Lustre performance when several clients access the same data set
  - OST Pools
    - The OST pools feature allows the administrator to name a group of OSTs for file striping purposes
    - an OST pool could be associated to a specific directory or file and automatically will be inherited by the files/directory created inside it

News on Lustre 1.8.x

- Adaptive Timeouts:
  - It is now possible to cache read-only data on an OS
  - Automatically adjusts RPC timeouts as network conditions and server load changes.
  - Reduces server recovery time, RPC timeouts, and disconnect/reconnect cycles.
- BUGFIX: The read performance will drop a lot if the application does stride read
  - We measured in our test this drop of performance, we will retry again with the new version in order to measure the increase of performance

Comparison StoRM/dCache

- Storm is available only on hw/sw architecture supported by glite (Storm 1.3 is not available on x86_64 machine)
- each of the dCache component could be installed on several different hw/sw infrastructure (already tested on SL3/4/5, Debian 4/5, OpenSolaris, SolarisOS)

- In Storm it is needed to change the default configuration in order to be able to write file on the underling file-system and read them back from SRM
- In dCache it is possible by default if all UID/GUID are correctly configured within the farm

- In Storm the quota management of the underling file-system could not be used as the file, written by SRM, are always owned by the same "admin" user.
- quota support in dCache is still not planned

Comparison StoRM/dCache

- SRM front-end in StoRM can be easily distributed and clusterized by using DNS balanced round robin
- At the moment SRM front-end in dCache could not be clusterized
- The speed of the SRM interface is pretty good => 0.1 s for listing a single file
- The same operation inside dCache tooks => 0.6 s
- The ACLs supported by the 1.4 will satisfy the SRM2.2 Addendum agreed with the experiments
- The same will happen with dCache 1.9.3 (to be released "soon")
- In StoRM the balancing between access doors (gridftp or other) is based on DNS round robin, or could be done "manually" changing a configuration file.
  - with StoRM 1.4 it could be possible but for the gridftp doors only
- In dCache each port could be balanced dynamically and the algorithm used could be easily tuned by the administrator

Comparison StoRM/dCache

- In StoRM the information provider publishes information with the granularity of file-system instead of directory so is not always possible to publish correct information about VO space usage and it is not properly dynamic
  - with StoRM 1.4 will be improved: it would be capable to understand directory and dynamically update the information
- In dCache the information system available on the recent release is easy to be used and dynamic
- Using StoRM it is not possible to answer to this question: "how much disk space is using that user, or that voms group?"
  - In lustre itself it is not easy to find out this information but this procedure could not be used at all (see what already discussed related to quotas)
- Using Chimera in dCache it is possible to find this information writing some tricky SQL query.
- In StoRM there is no accounting system
  - A new component will be added into 1.4 release that will add much more information in order to give the possibility to build some accounting system
- Using Chimera in dCache it will be possible to have much more information than in the past, but still there is some development work needed to have a complete system in place

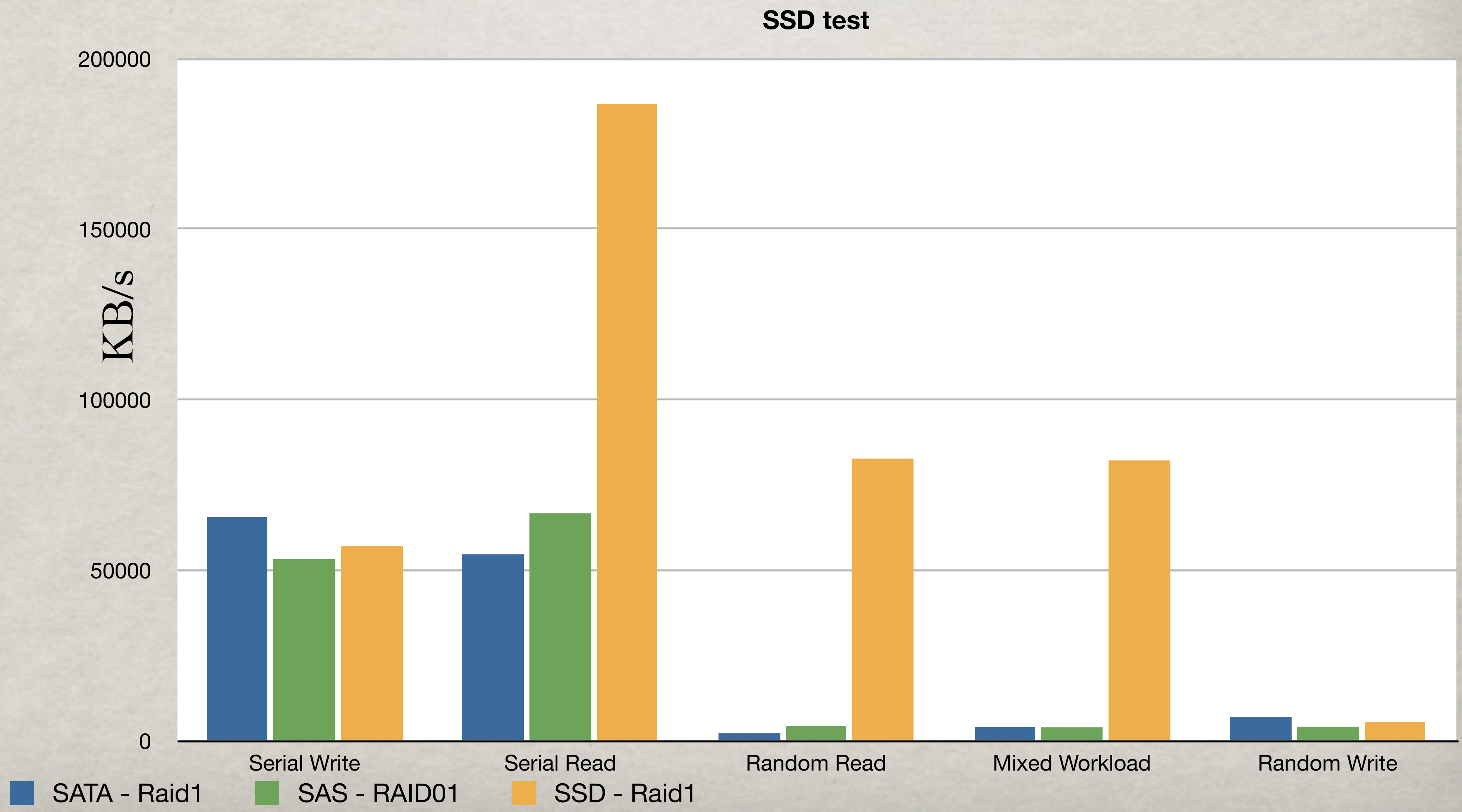
SSD Metadata I/O Simulation

- The goal of test is to emulate the access pattern on a device containing metadata for a parallel file-system (GPFS or Lustre)
  - For example in the case of Lustre I/O operation are made in chunk of 4Kbyte reading/writing on ext3 file-system
- During test the devices are formatted with the same configuration used by lustre
- The test is executed with 10 concurrent processes on the same devices
  - We measured the rate for: sequential read, sequential write, random read, random write, and mixed access as for the other test

SSD Metadata I/O Simulation

Disk Type	Serial Write	Serial Read	Random Read	Mixed Workload	Random Write
SATA - Raid1	65639	54693	2233	4043	6963
SAS - RAID01	53293	66710	4344	3917	4179
SSD - Raid1	57169	186707	82758	82177	5587

4Kbyte - Chunck Size	10 Procs

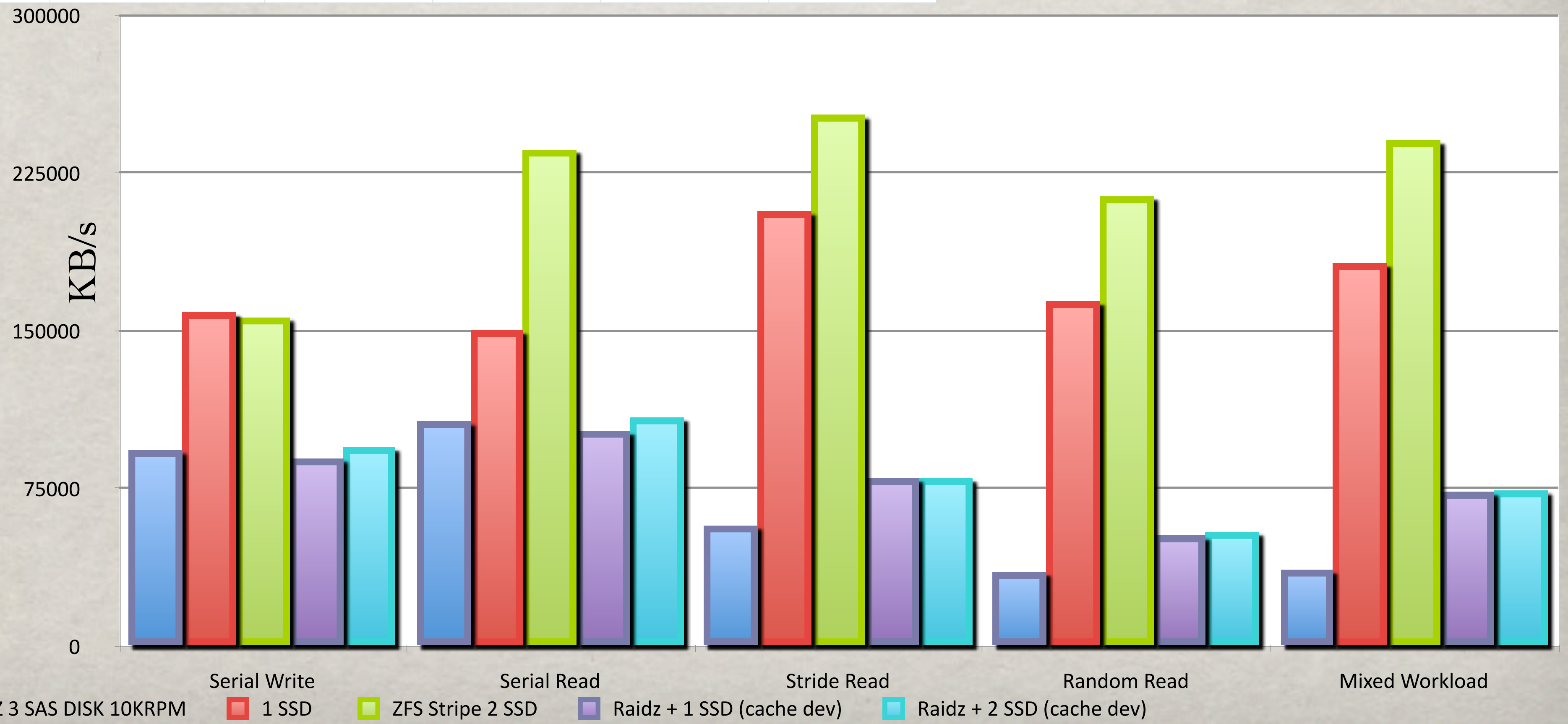


SSD as a ZFS cache device

- The goals of the test is to measure the improvement in the performance using an SSD device as a cache device using ZFS file-system (on OpenSolaris 2008.11)
- The test performed are related to a 100 concurrent processes with 256Kbyte of chunk size
- The test performed are: serial read and write, stride read (pseudo random read), random read and mixed workload
- The hardware configurations used were: 3 SAS drive (10Krpm) in a RaidZ configuration, 1 single SSD drive, 2 SSD drive using ZFS stripe, the RaidZ (3 SAS drive) with 1 SSD as device cache and the same RaidZ with 2 SSD drive as a cache.
- the SSD drive was the 32GB 2.5-inch enterprise distributed by SUN

SSD + ZFS

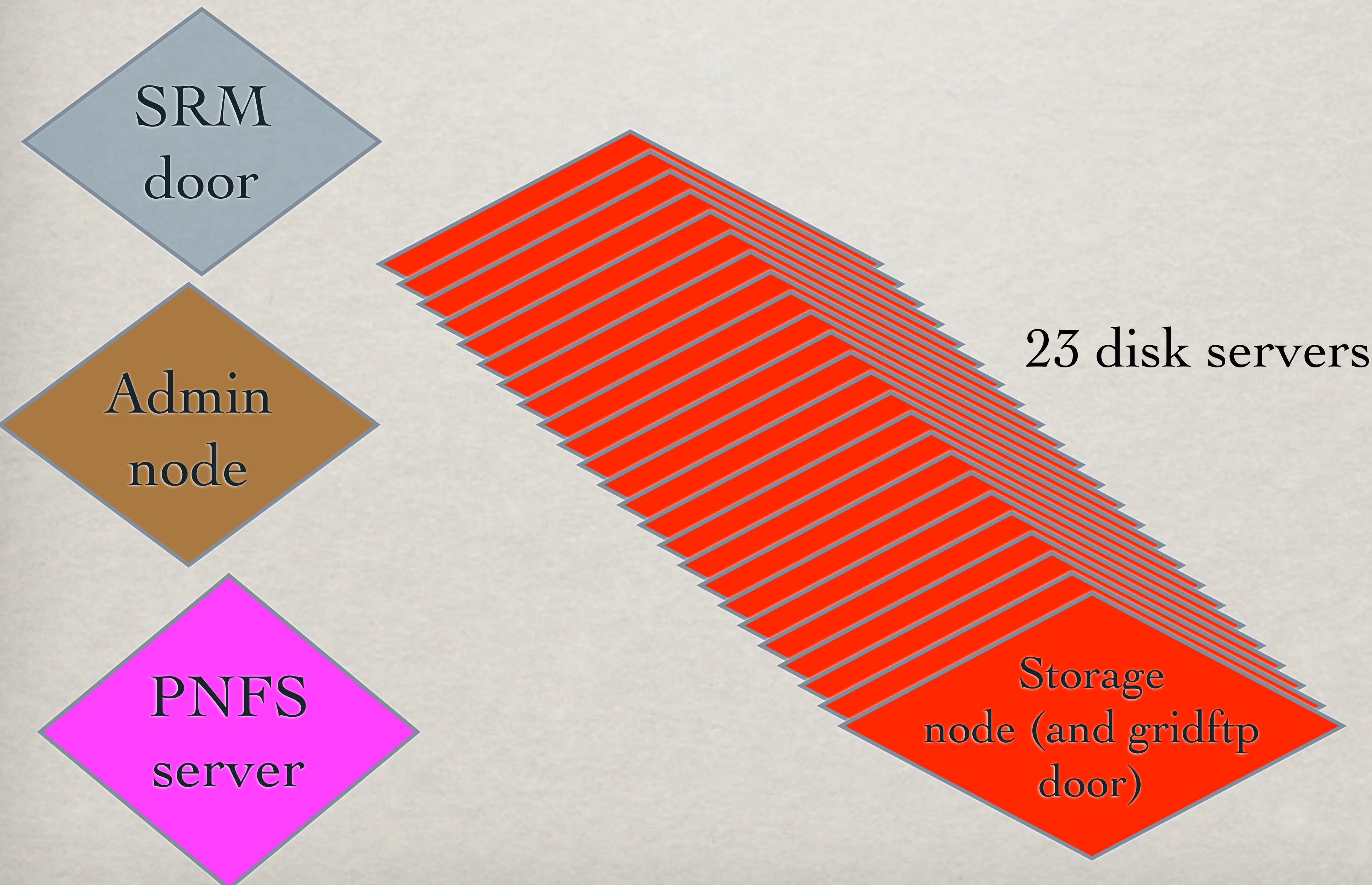
ZFS	100 procs				
	raidZ 3 SAS DISK 10KRPM	1 SSD	ZFS Stripe 2 SSD	Raidz + 1 SSD (cache dev)	Raidz + 2 SSD (cache dev)
Serial Write	93144	158748	156160	89159	94546
Serial Read	106891	150096	235975	102285	108654
Stride Read	57242	206830	252577	79735	79671
Random Read	35087	163984	213766	52551	54166
Mixed Workload	36254	182029	240419	73251	73936



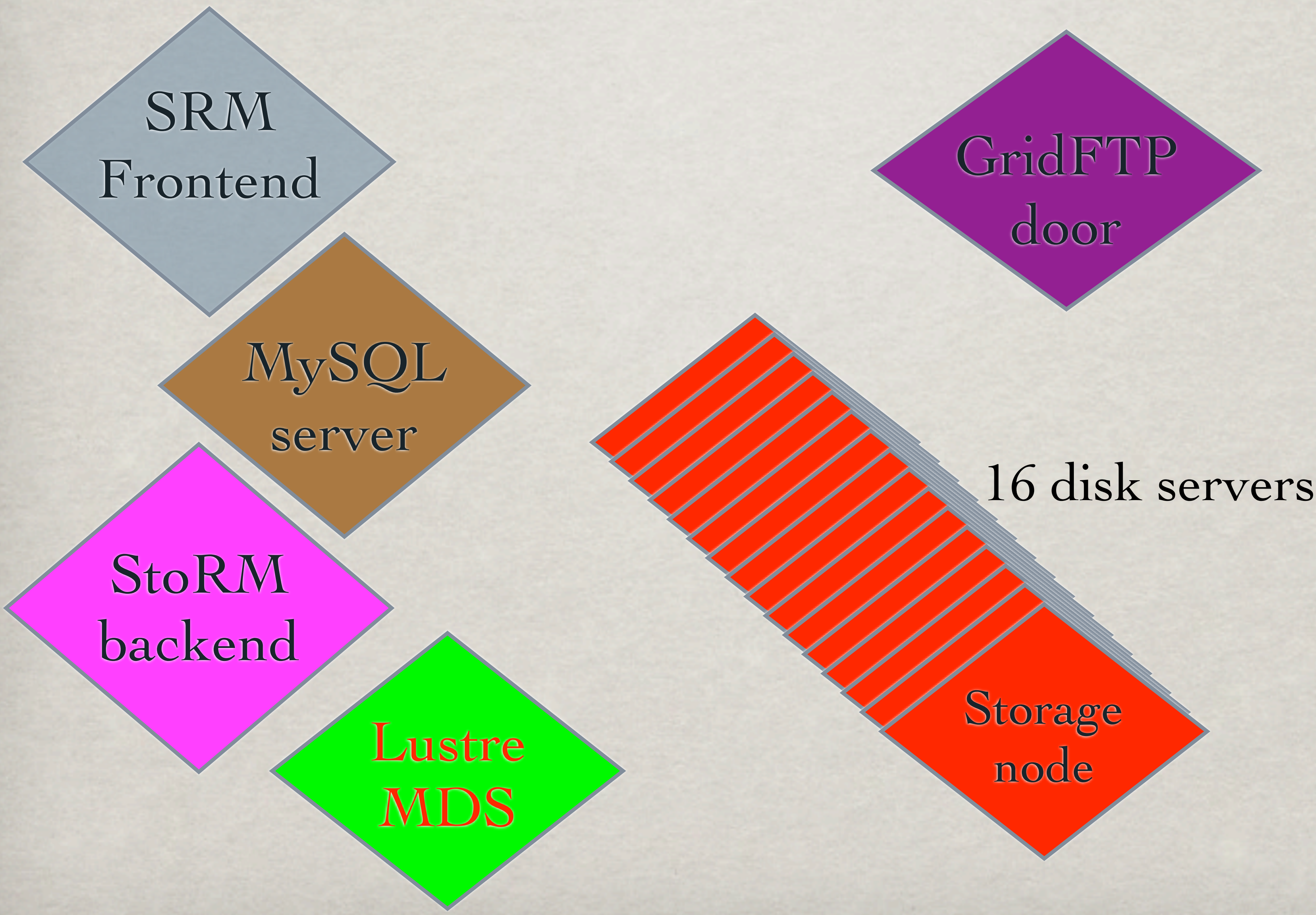
Linux Software Raid

- **Test performed:**
  - On two Raid5 with 8 disk each:
    - unplugging one disk during a I/O intensive operation cause a freeze of all the I/O for about 12 seconds
    - afterwards kernel highlight the missing device in the "messages" log
    - about 3 seconds later the I/O resumed with a degraded device
    - to start the rebuild of the raid it is enough a cli that declare the device as available to the raid
  - It is also possible to instantiate a service that monitor the sw raidsets and react to event with action or mails

Possible scenario dCache



Possible scenario Lustre



People involved

- Infrastructure and storage configuration, testing
  - Donvito Giacinto, Vincenzo Spinoso
- CMS job building
  - Alexis Pompili, Lucia Barbone

Back-up Slides

Hepix Tests

	20 threads	40 threads	60 threads	80 threads
AFS	43 MB/sec	93 MB/sec	95 MB/sec	91 MB/sec
LOC.XFS	144647 evs	183797 evs	232951 evs	166520 evs

	20 threads	40 threads	60 threads	80 threads
GPFS	166 MB/sec	262 MB/sec	259 MB/sec	269 MB/sec
NATIVE	1[illegible]0337 evs	249221 evs	246893 evs	239374 evs
XROOTD	[illegible]67 MB/sec	249 MB/sec	260 MB/sec	249 MB/sec
LOC.GPFS	147009 evs	226306 evs	251976 evs	243670 evs
AFS	57 MB/sec	110 MB/sec	97 MB/sec	127 MB/sec
LOC.GPFS	158864 evs	203023 evs	227967 evs	169664 evs
AFS	160 MB/sec	257 MB/sec	257 MB/sec	262 MB/sec
VICEGPFS	154342 evs	238358 evs	234893 evs	228481 evs

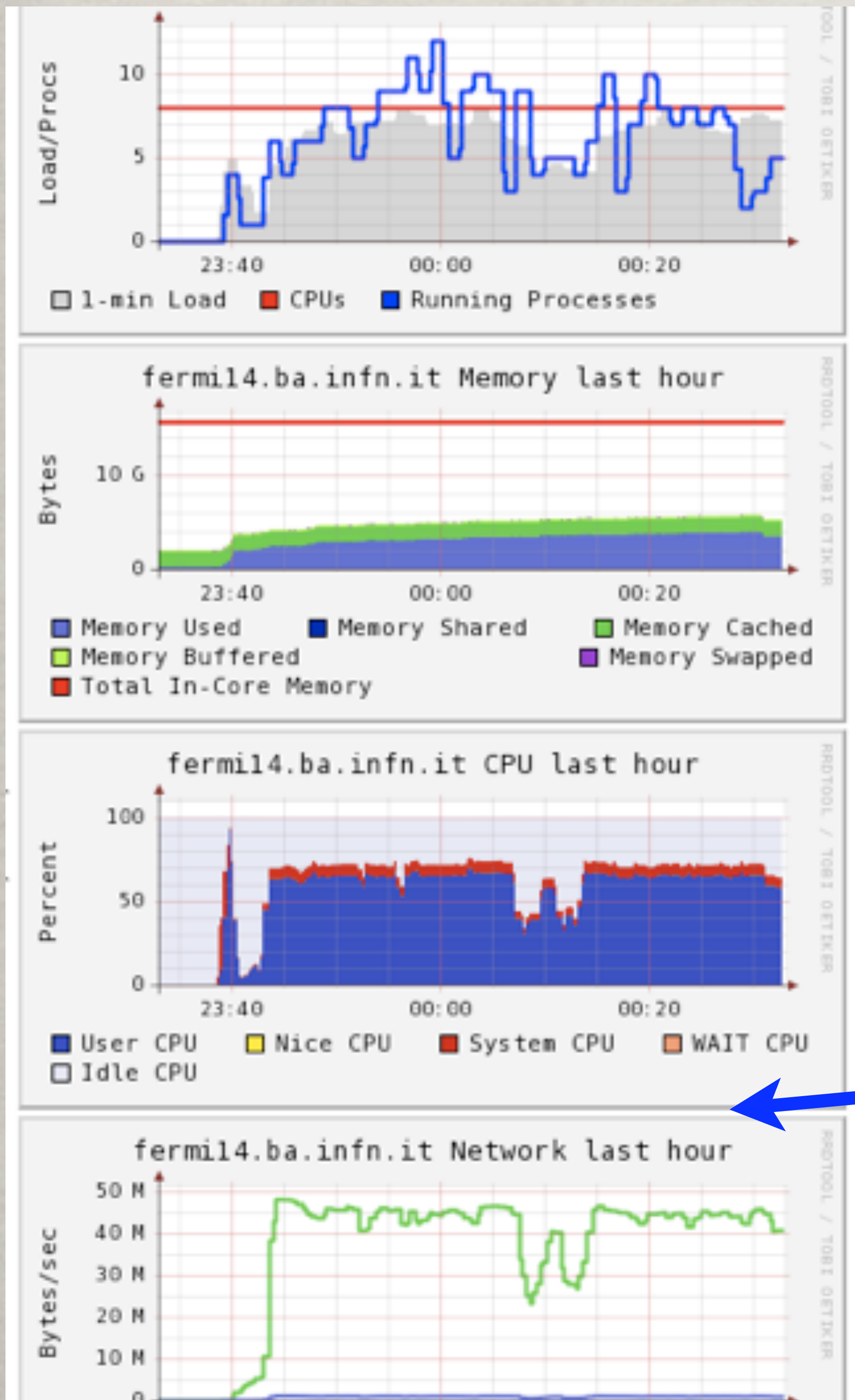
	20 threads	40 threads	60 threads	80 threads
LUSTRE	148 MB/sec	268 MB/sec	324 MB/sec	337 MB/sec
NATIVE	176413 evs	307474 evs	384154 evs	386029 evs
AFS	57 MB/sec	65 MB/sec	55 MB/sec	45 MB/sec
LOC.LUST	140016 evs	157830 evs	122681 evs	66630 evs
AFS	82 MB/sec	123 MB/sec	148 MB/sec	150 MB/sec
VICELUST	114710 evs	178764 evs	211824 evs	239416 evs

	20 threads	40 threads	60 threads	80 threads
XROOTD	151 MB/sec	261 MB/sec	264 MB/sec	221 MB/sec
	1[illegible]3860 evs	262922 evs	269424 evs	222834 evs
DCACHE	108 MB/sec	183 MB/sec	227 MB/sec	206 MB/sec
	13[illegible]818 evs	231622 evs	285539 evs	258600 evs
AFS	64 MB/sec	124 MB/sec	140 MB/sec	132 MB/sec
	128505 evs	248667 evs	280656 evs	263604 evs
LUSTRE	148 MB/sec	267 MB/sec	348 MB/sec	384 MB/sec
	17[illegible]202 evs	314563 evs	408825 evs	457108 evs

	20 threads	40 threads	60 threads	80 threads
XROOTD	104 MB/sec	125 MB/sec	131 MB/sec	140 MB/sec
OV. XFS	92292 evs	107912 evs	122863 evs	127910 evs
GPFS	147 MB/sec	186 MB/sec	156 MB/sec	162 MB/sec
	[illegible]45913 evs	161511 evs	140181 evs	140722 evs
AFS	53 MB/sec	70 MB/sec	71 MB/sec	98 MB/sec
OV. XFS	123906 evs	155653 evs	163381 evs	130062 evs
LUSTRE	178 MB/sec	256 MB/sec	227 MB/sec	201 MB/sec
	[illegible]75137 evs	272976 evs	271661 evs	276724 evs

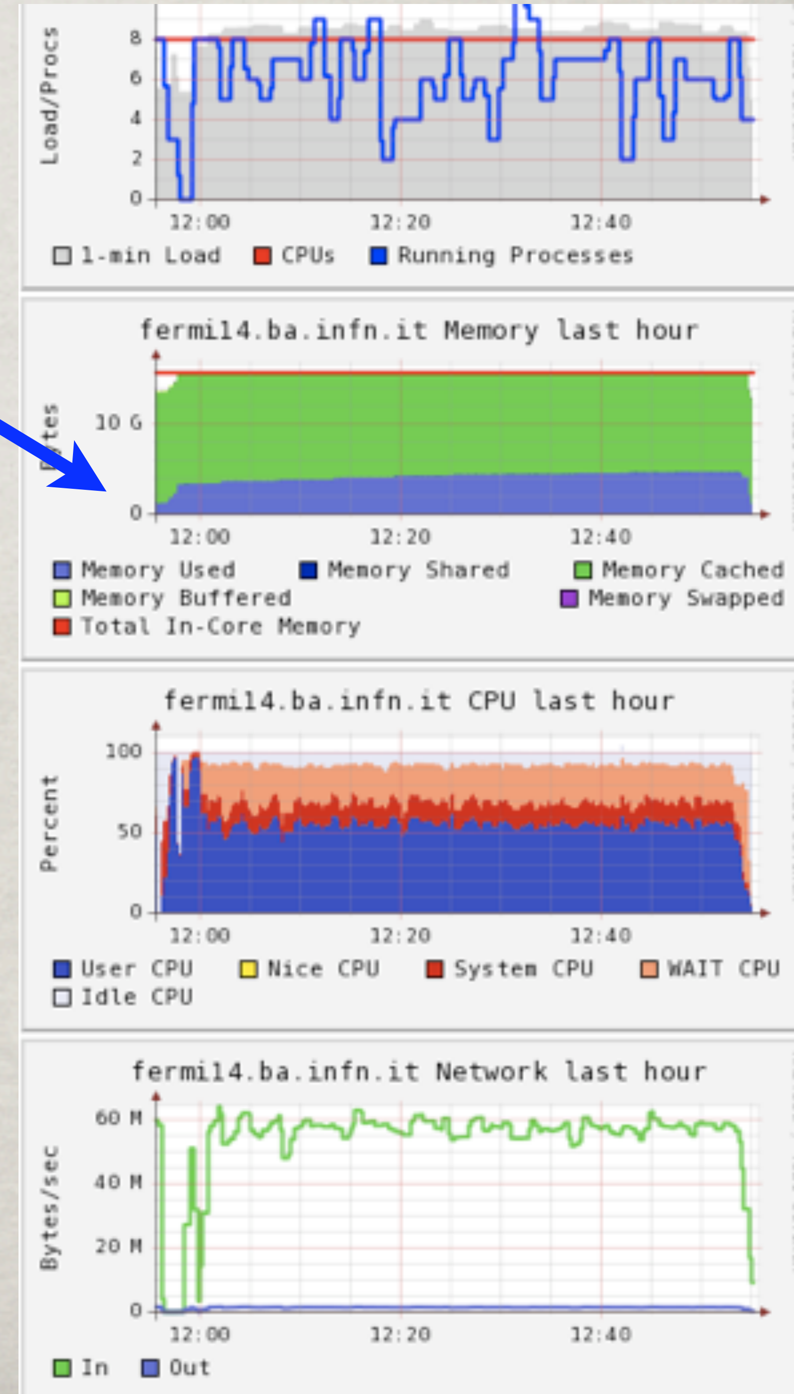
Lustre
Xrootd
GPFS
dCache

Client Load

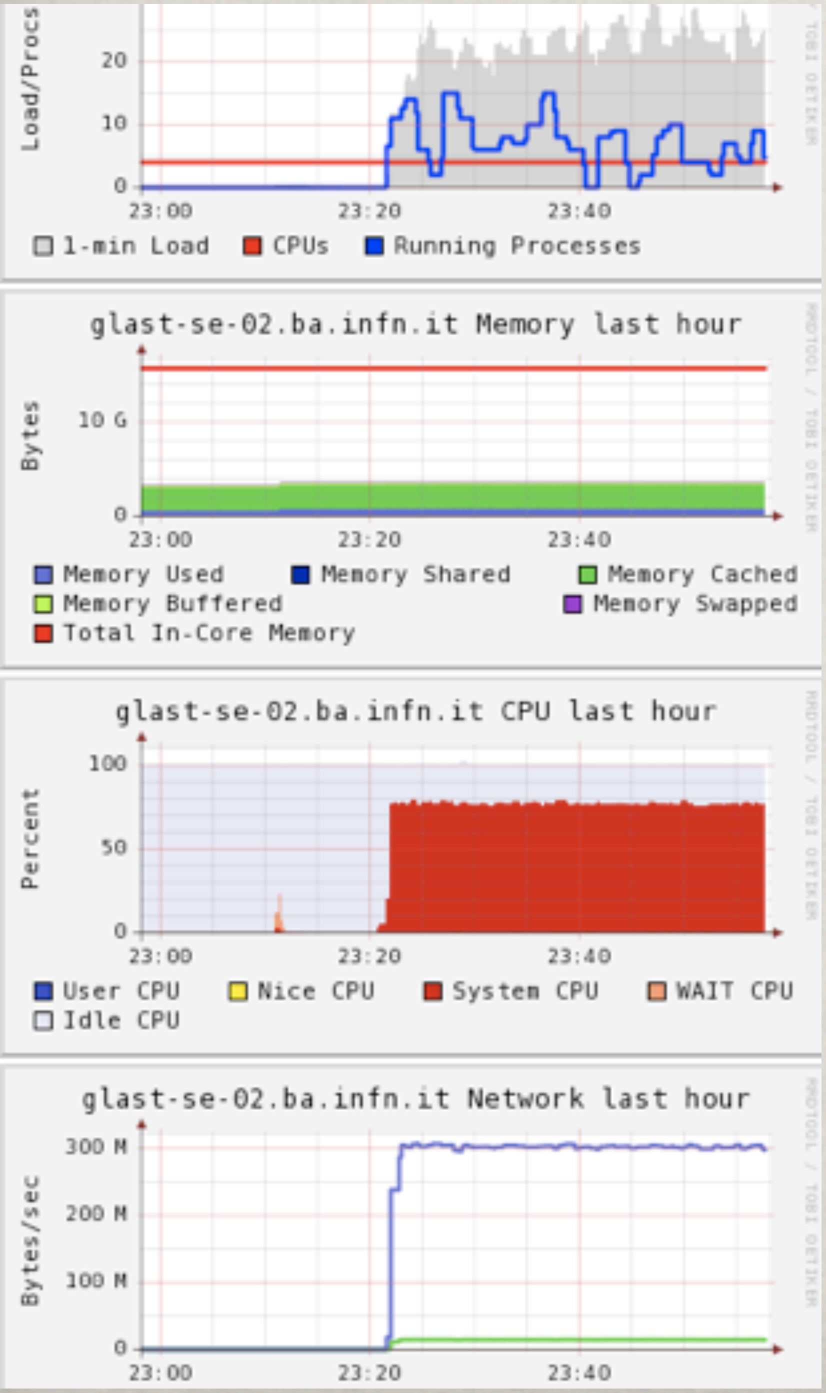


Lustre client (16 jobs running)

dCache client (16 jobs running)



Server Load



Lustre server

(56 jobs running)