Embedded Filesystems
(Direct Client Access to Vice Partitions)

Hartmut Reuter
reuter@rzg.mpg.de

Hartmut Reuter, RZG, Germany
Felix Frank, DESY Zeuthen, Germany
Andrei Maslennikov, CASPUR, Italy

Overview

- Introduction: Deficiencies of AFS
- Embedded Filesystems
  - History, 1st implementation 2004
  - Current implementation together with AFS/OSD
  - Benchmark results of Andrei Maslennikov at FZ Karlsruhe
  - Benchmark results of Felix Frank at DESY Zeuthen
- Update to AFS/OSD
  - Policies implemented by Felix Frank
  - Benchmarks to file creation, striping and mirroring of files in OSDs
- Status of cell ipp-garching.mpg.de

AFS Deficiencies

- There are new sites starting with OpenAFS, but also old sites abandoning it. **Why?**
  - The last expert left the site. To the new guy it seems too complicated.
  - AFS is too slow. Therefore many sites have faster file-systems such as Lustre and GPFS in parallel.
    - Users don't like to maintain multiple copies of their files: which one is the correct version?
- For the 1st reason I don't have an answer. Someone else should prepare a GUI for easy installation, administration and problem analysis for AFS.
- For the 2nd reason I have two answers:
  - Add object storage to your cell and the total throughput especially to single hot spot volumes will increase visibly
  - Use Lustre or GPFS or … as embedded systems. So you can keep the user's view of a single filesystem and get the best performance out of it.

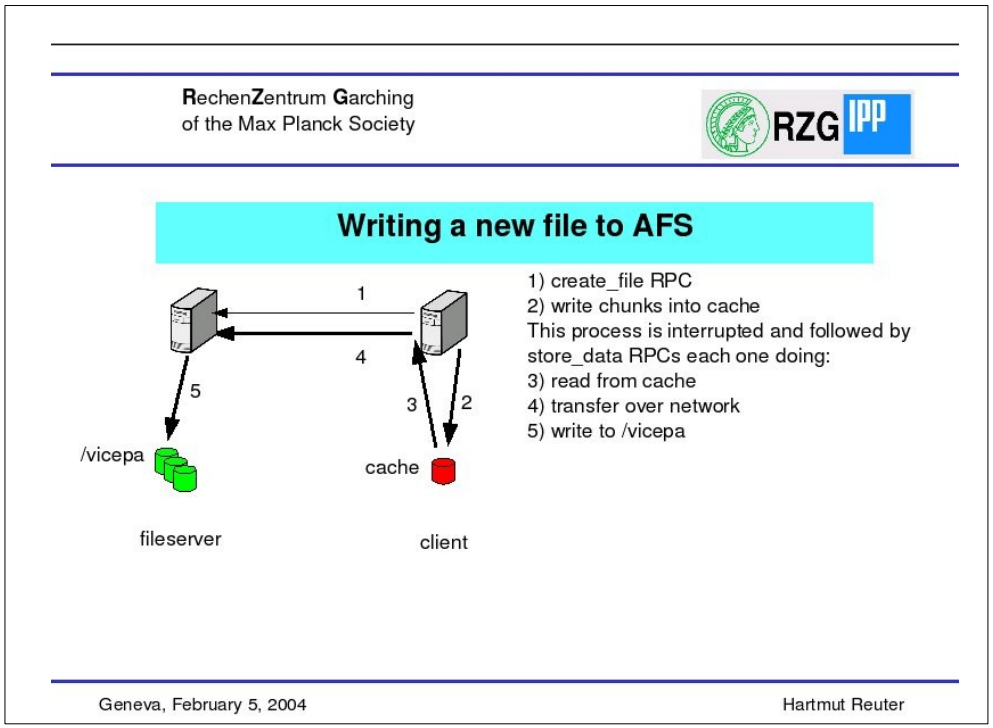
Embedded Filesystems

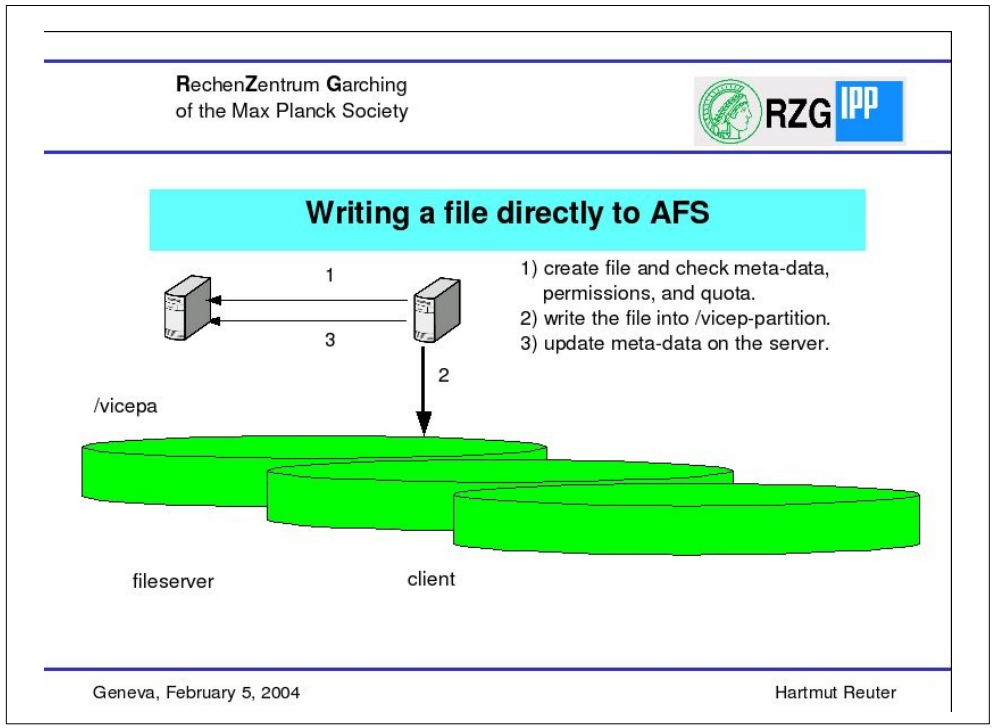
- Modern cluster file-systems such as Lustre or GPFS are much faster than AFS especially in combination with fast networks (Infiniband, 10GE)
- But they have other deficiencies:
  - Because of giant block size inefficient for small files
  - File creation and deletion is slow
  - No secure way to export into WAN or even to desktops
  - Accessible only from Linux (or AIX in case of GPFS).
- This is exactly complementary to AFS. Therefore combine both
  - Use these fast file-systems for rxosd (or fileserver) partitions
  - Export them to your trusted batch clusters and HPC environment
  - Allow the AFS clients in the batch cluster to read and write files directly

History

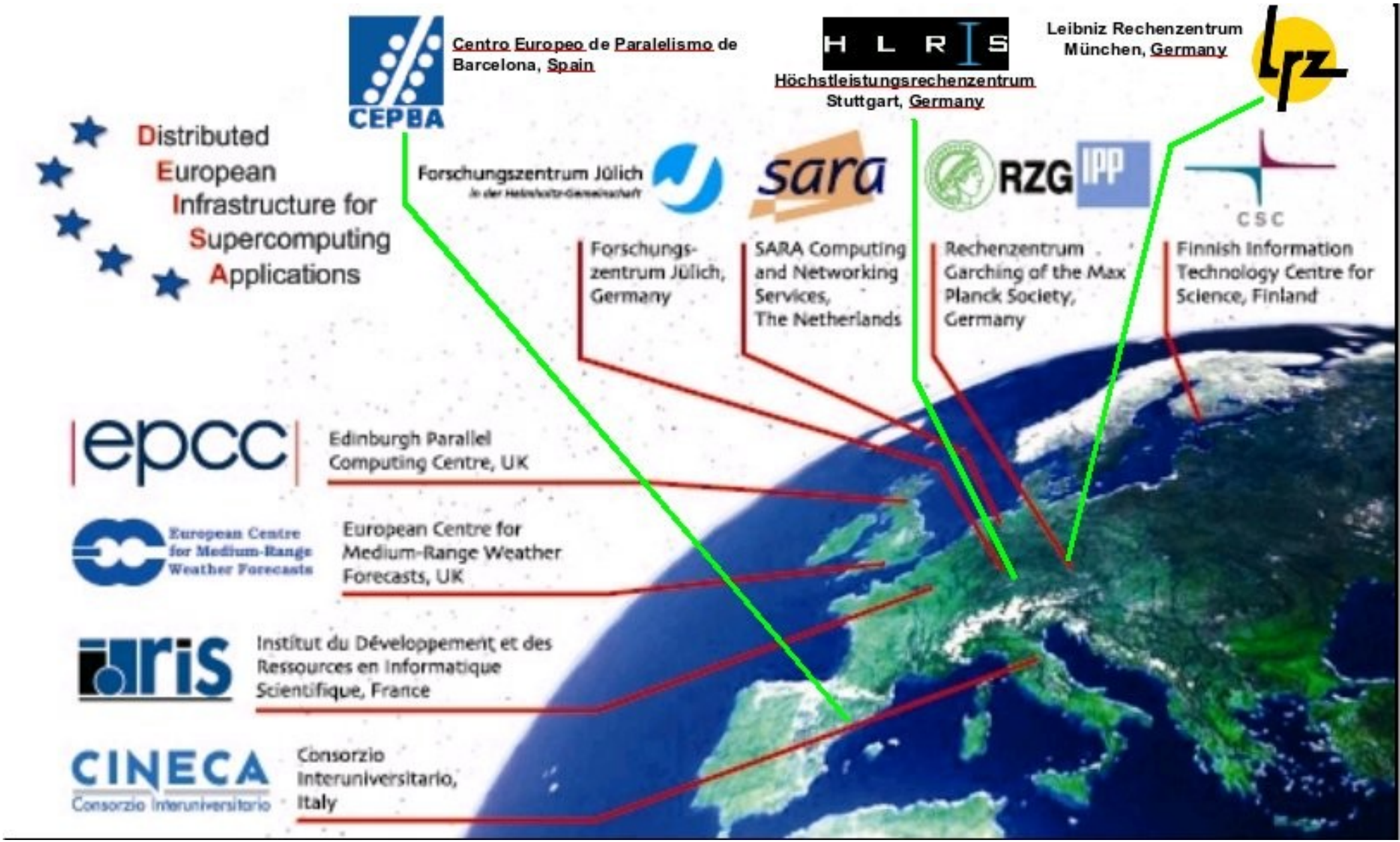
This idea is not completely new:

- Already in 2004 I gave a talk at CERN about using shared filesystems for AFS





History: DEISA in 2003



GPFS and AFS in DEISA

- 2003 - six years ago - the European supercomputing project DEISA installed a private 1 Gbit/s network between
  - FZJ, Jülich, Germany
  - RZG, Garching, Germany
  - Idris, Paris, France
  - Cineca, Bologna, Italy
- After some tuning GPFS was able to transfer data with 100 MB/s
  - AFS reached only 1 MB/s (small window size, high round trip time)
- Therefore it was worth to look closer at how AFS could make use of an underlying shared filesystem (in this case GPFS)
  - Put the AFS fileserver's vice partition into GPFS
  - Add a new RPC to ask the fileserver for the path and to check permissions
  - Let the AFS client read or write files directly from/to GPFS

AFS-client implementation

- The 1st implementation bypassed completely the cache being implemented in osi_vnodeops.c.
  - very fast for sequential read and write when large buffers were used, but
  - not everything worked (configure OpenAFS didn't, compile did).
  - different for each architecture / OS
- Tests 2003 with
  - GPFS at RZG
  - StoreNext at CASPUR, Rome
- Current implementation was written together with OpenAFS+OSD
  - uses the cache (preferably memory cache)
  - required restructuring of the cache manager code

Restructured Cache Manager

- The restructured AFS cache manager allows for multiple protocols

  0 == rx-fileserver (the classical AFS protocol)

  1 == vicep-access (embedded filesystems)

  2 == rxosd (object storage)

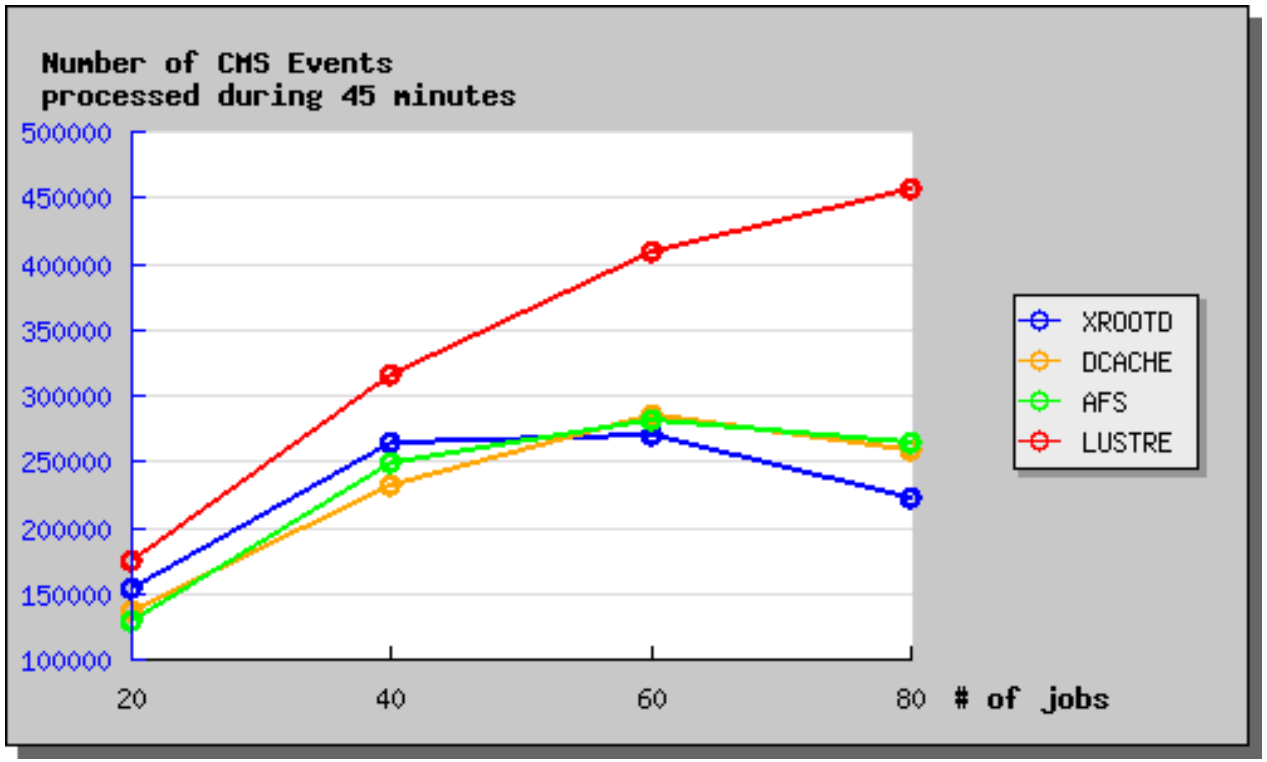
  - The „fs protocol“ command can enable or disable use of “vicep” or “rxosd“
  - The vcache entry contains a field “protocol” to specify for each file which protocol is to be used, default is 0.

Visible vicep partitions

- How can the cache manager know about visible vicep partitions?
  - Fileservers write their “sysid” file into the partition
  - Rxosds write an “osdid” file into the partition
  - „afsd -check-osds“ and „afsd -check-fspartitions“ inform the cache manager about visible vicep partitions belonging to rxosds or fileservers
    - The fileserver with the uuid found in the sysid file gets a flag
    - Volumes on such a fileserver also get a flag
- Plain AFS files:
  - If the volume is flagged the cache manager uses a special RPC to the fileserver to get the file's path information
  - If the file can be opened protocol for the file is set to 1 and all I/O is done directly
  - When the user closes the AFS file also the vicep-file gets closed
- OSD file:
  - If the file consists in a single object on a visible rxosd partition protocol is set to 1
  - Then similar procedure as for plain AFS file.

HEPIX-Tests (1)

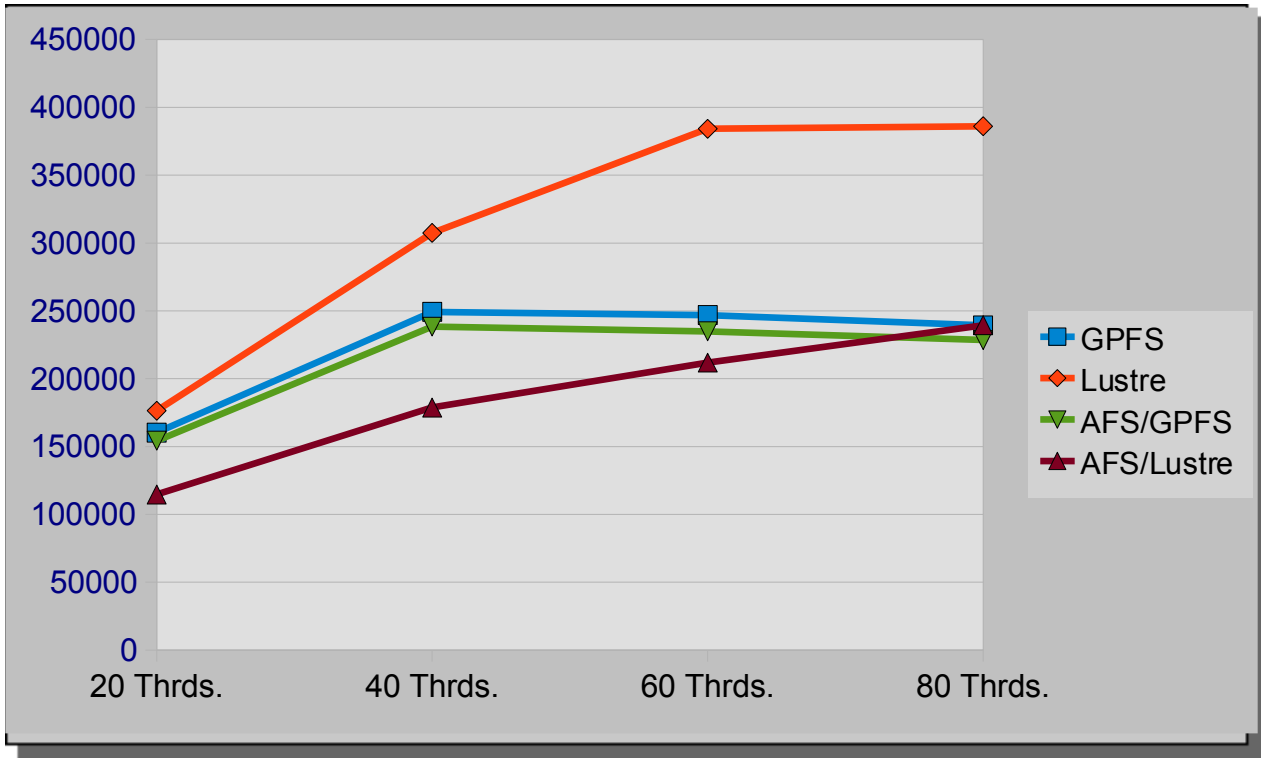
- The “HEPIX Storage Working Group“ developed last year a use case for distributed storage systems based on CERN's soft- and middleware stack for CMS
- In a 1st round in 2008 at FZK (Forschungszentrum Karlsruhe) the following systems Andrei Maslennikov compared: AFS, DCACHE, LUSTRE, and XROOTD.



Source: „HEPIX storage working group, - progress report, 2.2008-”, Andrei Maslennikov, Taipei October 20, 2008

HEPIX-Tests (2)

- In a 2nd round of tests in February/March 2009 again at FZK, but with a different server hardware the following systems were compared: LUSTRE, GPFS, embedded Lustre, embedded GPFS.



Source: „HEPIX storage working group, - progress report, 1.2009-", Andrei Maslennikov, Umeå, May 27, 2009

- Embedded GPFS is very near to native GPFS, imbedded Lustre is slow because of workaround (open and close for each chunk) in the AFS client for a problem with Lustre

Embedded Lustre at DESY Zeuthen

- While testing AFS with Lustre at DESY Zeuthen Felix Frank found a solution for the Lustre problem seen at Karlsruhe getting rid of the many open/close calls. Basically same client as in Karlsruhe used for GPFS
- Lustre (1.6.7 or 1.8) configuration:
  - OSS on Dell 1950, 2 2.33 Ghz CPUs 8 GB memory, SL 5.3
  - OST on RAID6 PERC6 controller (13+2) 1TB sata disks in Dell MD1000 enclosure connected with 2 Quadlane SAS cables.
  - No striping over OSTs, DDR Infiniband to clients
- Felix tested the performance with sequential write/read of large files:

```
write of 34359738368 bytes took 151.388 sec.
close took 0.134 sec.
Total data rate = 221449 Kbytes/sec. for write
```

```
read of 34359738368 bytes took 127.308 sec.
close took 0.000 sec.
Total data rate = 263564 Kbytes/sec. for read
```

2 reads embedded Lustre at DESY Zeuthen

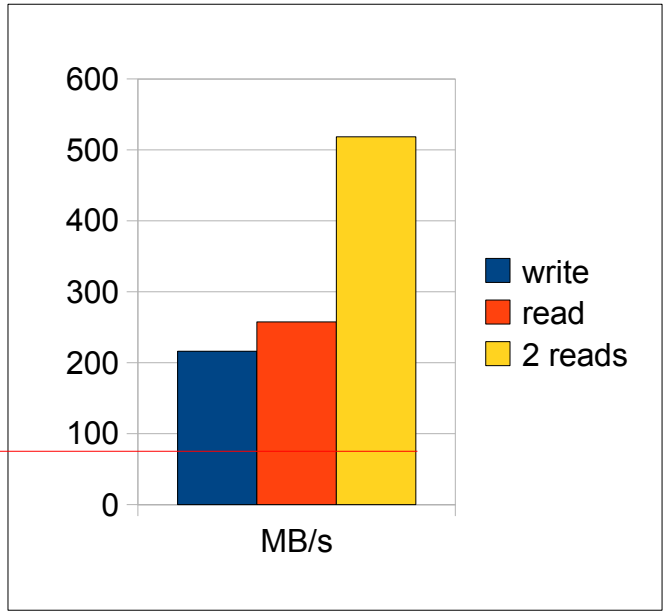
- Also reads on two clients at the same time are fast:

```
Total data rate = 270145 Kbytes/sec. for read
```

```
Total data rate = 260697 Kbytes/sec. for read
```

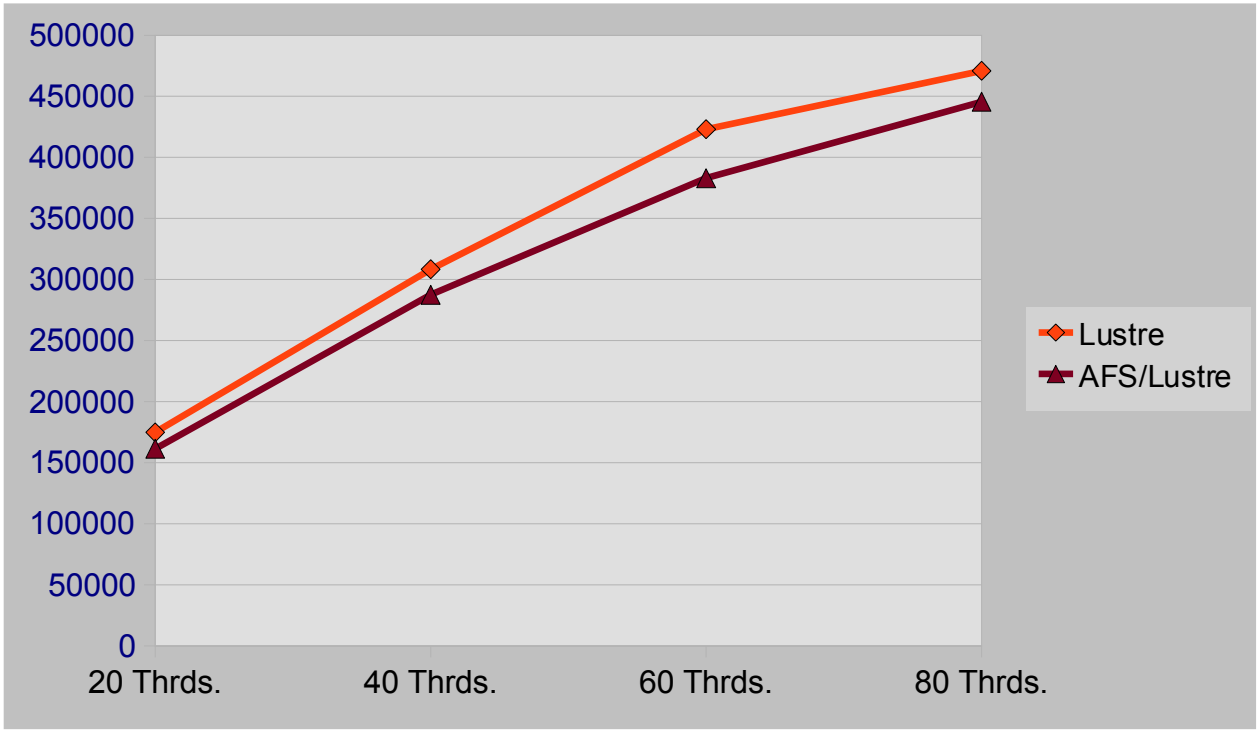
- The total throughput of ~ 518 MB/s shows that we are not at the limit of the server.
- The AFS-clients had 256 MB memory cache with 1 MB chunk size.
  - The performance depends on the chunk size:
  - With 64 KB chunks it goes down to 200 MB/s

Typical speed of normal AFS client



HEPIX-Tests (3)

- In a 3nd round of tests in yetserday again at FZK, but with the server hardware from March only Lustre and in AFS embedded Lustre were compared.



Source: SMS from Andrei Maslennikov, Umeå, , June, 3,2009

- Now with the modified client embedded Lustre comes close to Lustre native.

Short Update AFS/OSD Policies

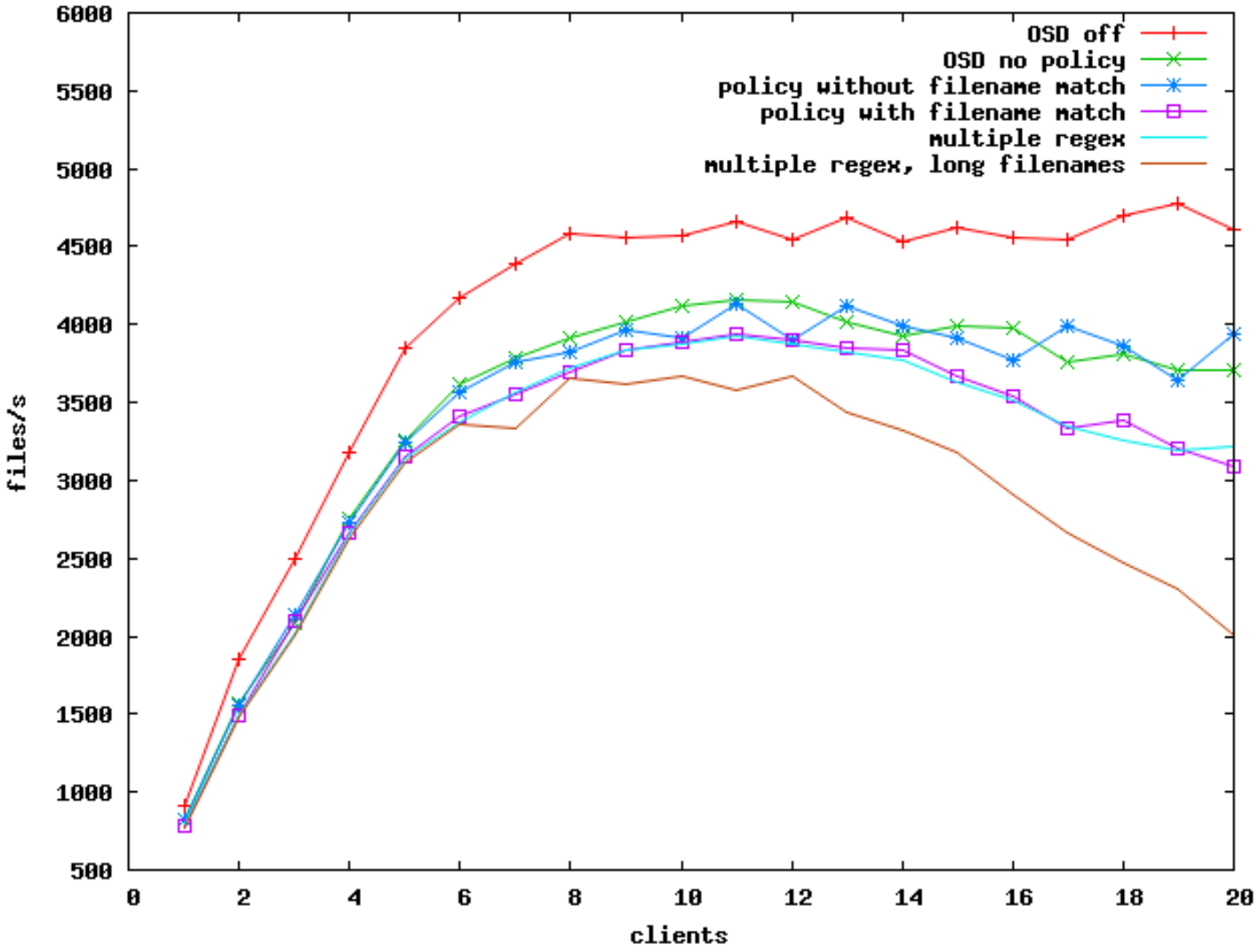
- Since the last workshop **Felix Frank** (DESY Zeuthen) implemented the **policies**. Policies are rules stored in the OSD-database which have a number and a name. A policiy number can be set for a whole volume or individual directories inside the volume.
- The fileserver gets the policies from the OSD-database and applies the rule when new files in the directory or volume are created or written for the 1st time.
- Policies basically can use filenames (suffixes or regular expressions) and file size to state whether a file
  - should be stored on the local disk or go into object storage
  - and if into object storage whether striped or mirrorered
  - and if striped with which stripe size.
- Simple example:

```
3 root

  ~'*.root' => location=osd, stop;
```

Means: all files with suffix „.root“ should be stored in object storage independent of their size. (Typical problem in the HEP-community because root does an fsync() after writing the tiny header or the file).

File creation rate



Evaluation of a policy has its price:

OSD off means normal AFS volume.

OSD no policy means files > 1 MB go into OSD

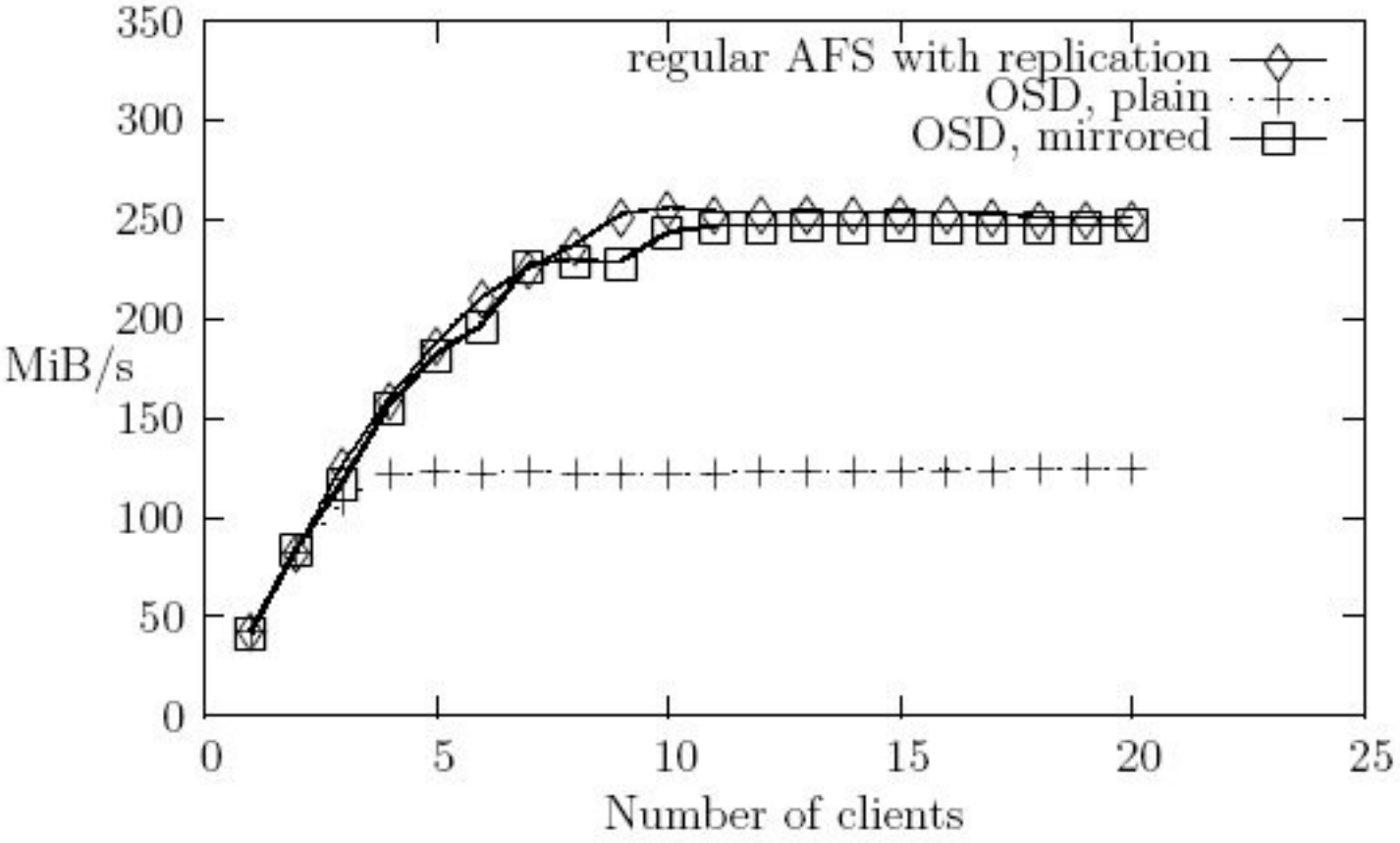
Most expensive is evaluation of regular expressions on file names.

Measurement and graphic by Felix Frank.

Read throughput of a single file, either replicated or in OSD plain or mirrored

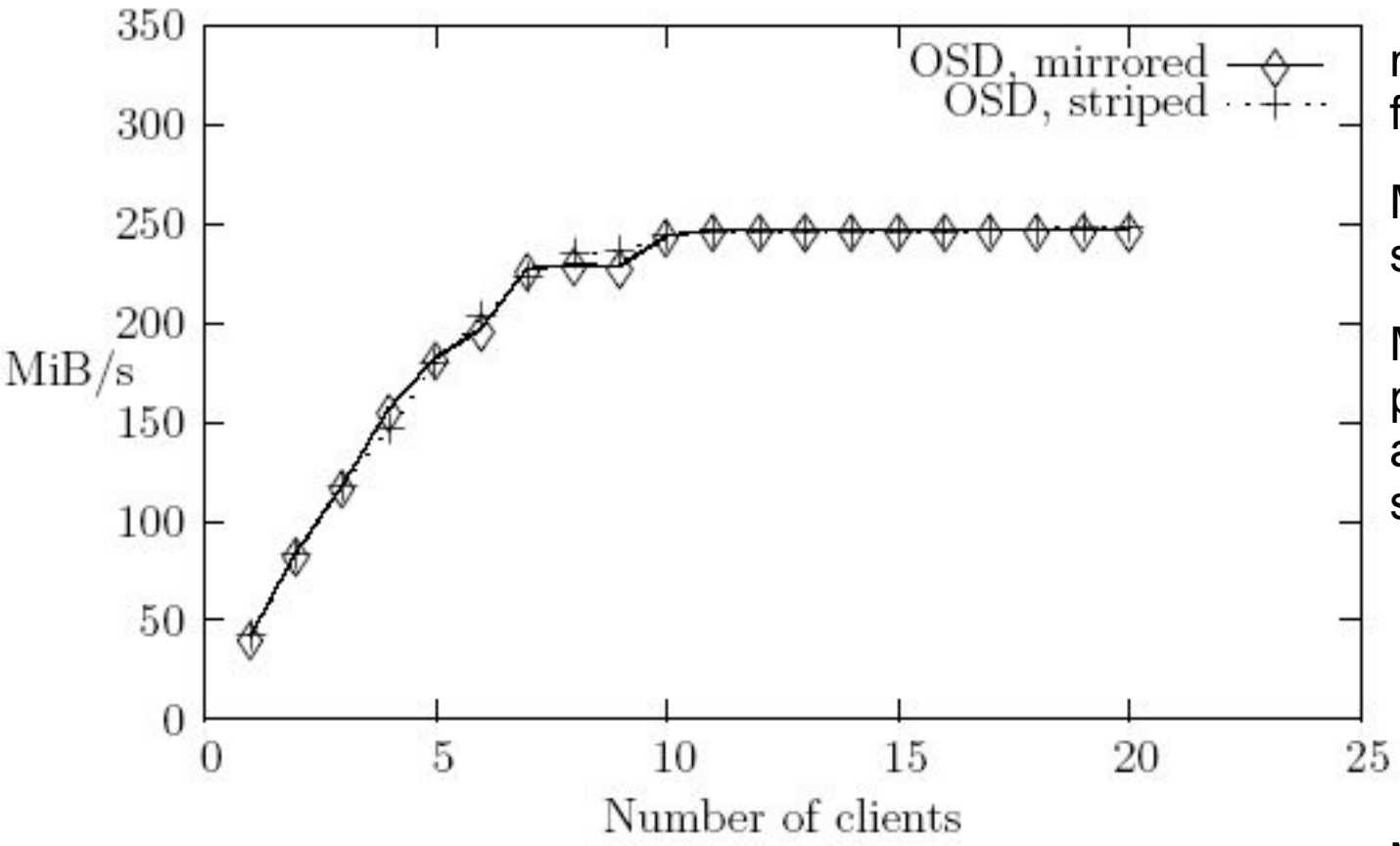
Tests with 2 server machines acting as fileservers and OSDs.

RO-replication and mirroring of OSD file are nearly equivalent for read, but mirrored OSD file can be overwritten.



Measurement and graphic by Felix Frank.

Reading a single file in OSD either mirrored or striped



Tests with 2 server machines acting as fileservers and OSDs.

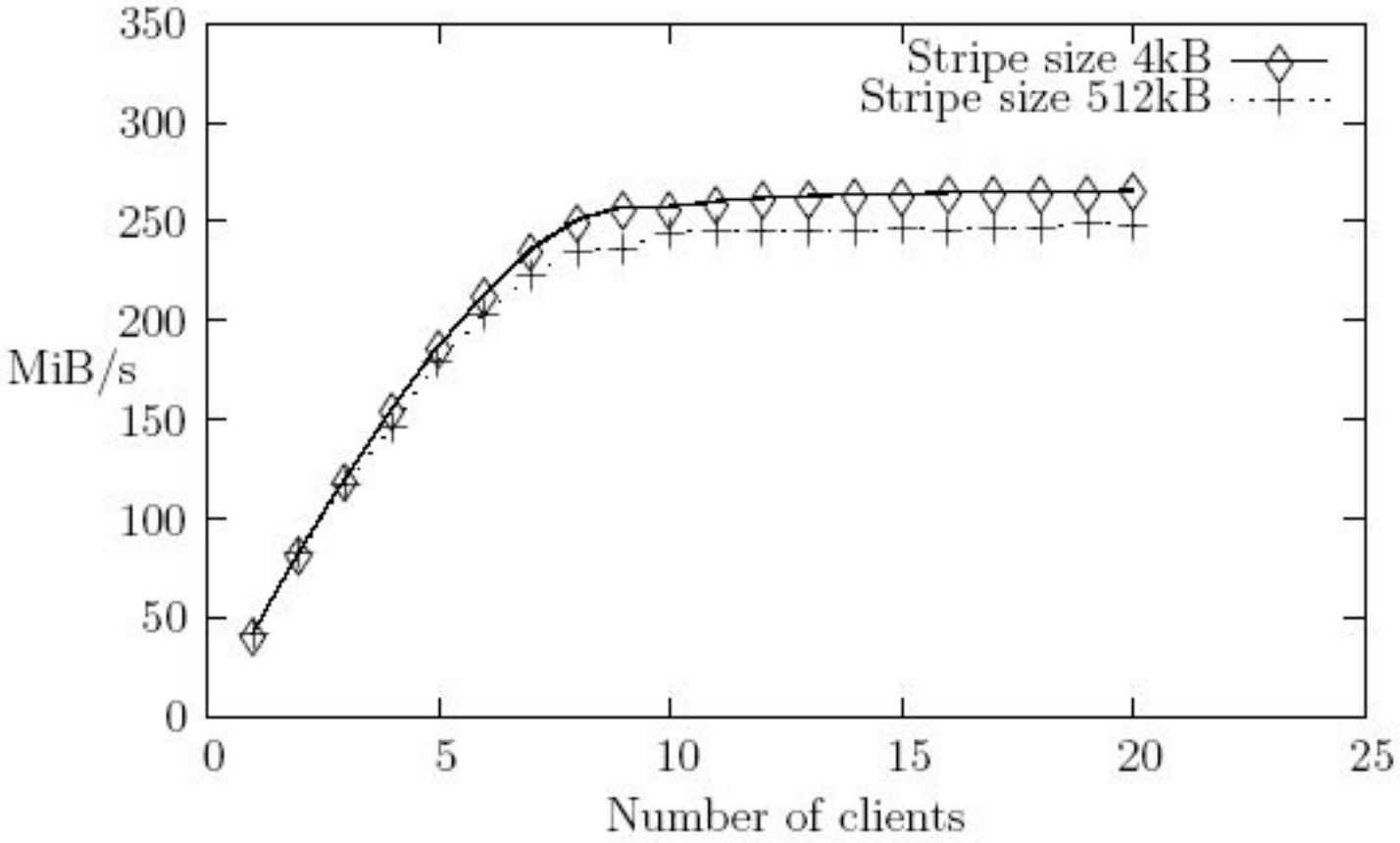
Mirroring and striping give same read throughput.

Mirroring has half write performance, but protects against loss of a disk system.

Measurement and graphic by Felix Frank.

Dependency on stripe size

Read throughput of a single striped file



For me it was surprising that read performance gets better with smaller stripe size!

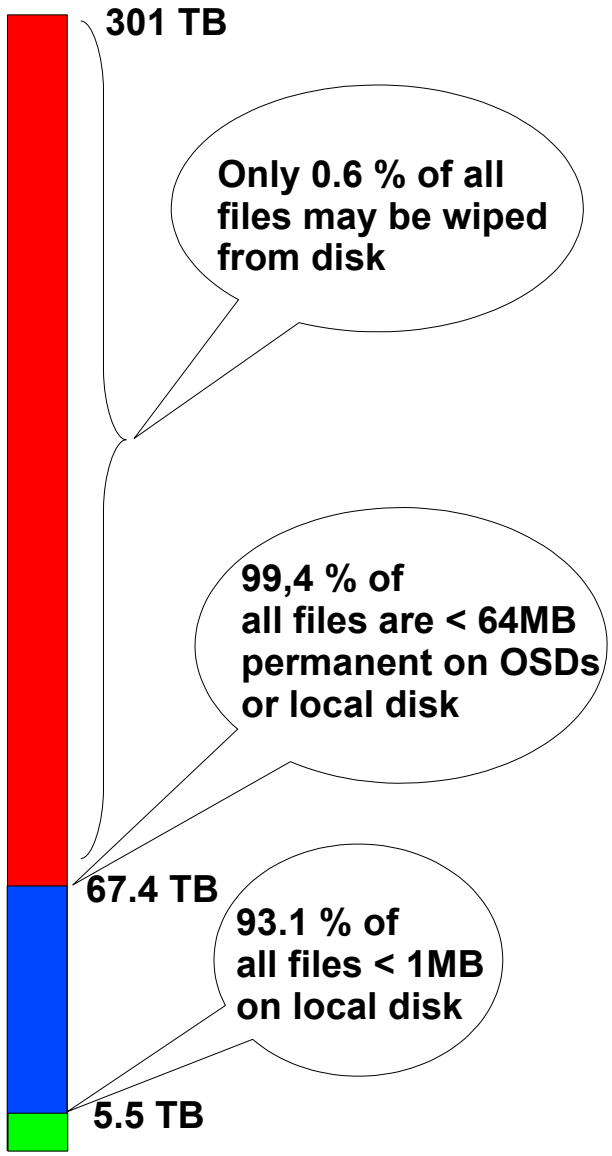
Measurement and graphic by Felix Frank.

AFS cell „ipp-garching.mpg.de“

40	fileservers with 195 TB disk space
21	non-archival OSDs with 100 TB disk space
2	archival OSD with HSM system TSM-HSM, will be replaced by HPSS by the end of the year
27000	volumes
8700	users
300	TB total data
4	TB data written per day
8	TB data read per day

File Size Histogram

File Size Range	Files	%	run %	Data	%	run %
0 B - 4 KB	64853397	50.31	50.31	80.867 GB	0.03	0.03
4 KB - 8 KB	10368948	8.04	58.35	56.730 GB	0.02	0.04
8 KB - 16 KB	9157574	7.10	65.46	97.675 GB	0.03	0.08
16 KB - 32 KB	10300431	7.99	73.45	215.601 GB	0.07	0.15
32 KB - 64 KB	7942739	6.16	79.61	363.820 GB	0.12	0.26
64 KB - 128 KB	6006376	4.66	84.27	523.980 GB	0.17	0.43
128 KB - 256 KB	4005120	3.11	87.37	709.525 GB	0.23	0.66
256 KB - 512 KB	4360813	3.38	90.76	1.484 TB	0.49	1.16
512 KB - 1 MB	3067697	2.38	**93.14**	1.980 TB	0.66	**1.82**
1 MB - 2 MB	2051828	1.59	94.73	2.851 TB	0.95	2.76
2 MB - 4 MB	2033889	1.58	96.31	5.361 TB	1.78	4.55
4 MB - 8 MB	1830856	1.42	97.73	9.617 TB	3.20	7.74
8 MB - 16 MB	1112483	0.86	98.59	12.125 TB	4.03	11.77
16 MB - 32 MB	598374	0.46	99.05	12.370 TB	4.11	15.88
32 MB - 64 MB	474283	0.37	**99.42**	19.651 TB	6.53	**22.41**
64 MB - 128 MB	306544	0.24	99.66	25.633 TB	8.52	30.93
128 MB - 256 MB	165577	0.13	99.79	28.363 TB	9.43	40.36
256 MB - 512 MB	130065	0.10	99.89	46.954 TB	15.60	55.96
512 MB - 1 GB	125479	0.10	99.99	84.709 TB	28.15	84.11
1 GB - 2 GB	13216	0.01	100.00	18.967 TB	6.30	90.41
2 GB - 4 GB	2264	0.00	100.00	5.896 TB	1.96	92.37
4 GB - 8 GB	873	0.00	100.00	5.244 TB	1.74	94.11
8 GB - 16 GB	599	0.00	100.00	5.652 TB	1.88	95.99
16 GB - 32 GB	174	0.00	100.00	3.784 TB	1.26	97.25
32 GB - 64 GB	120	0.00	100.00	5.306 TB	1.76	99.01
64 GB - 128 GB	21	0.00	100.00	1.725 TB	0.57	99.59
128 GB - 256 GB	4	0.00	100.00	611.258 GB	0.20	99.78
256 GB - 512 GB	2	0.00	100.00	667.311 GB	0.22	100.00
Totals:	128909746 Files			300.938 TB		



Data Transfer

```
> xfer
---snip---
fs afs17.rzg.mpg:      72.64 gb  rcvd      43.16 gb  sent per day (3.428044 days)
fs afs18.rzg.mpg:      76.10 mb  rcvd       2.92 tb  sent per day (3.427303 days)
---snip---
vos afs17.rzg.mp:     119,74 gb  rcvd       3.25 gb  sent per day (3.428044 days)
vos afs17.rzg.mp:       2.23 gb  rcvd      43.41 mb  sent per day (3.427303 days)
---snip---
osd           42:      48.23 gb  rcvd     210.64 gb  sent per day (3.427488 days)
osd           43:      78.61 gb  rcvd     224.65 gb  sent per day (3.427812 days)
Fileserver xfer:        1.15 tb  rcvd       4.42 tb  sent per day
Volserver xfer:       433.71 gb  rcvd     422.21 gb  sent per day
Rxosd xfer:             2.41 tb  rcvd       3.31 tb  sent per day
Total transfer:         3.99 tb  rcvd       8.15 tb  sent per day
```

- Fileserver transfer
  - Very active non-OSD volumes with lots of small files
- Volserver transfer
  - Nightly „vos release“ to obtain actual RO volumes (backup)
- Rxosd transfer
  - Direct read/write from clients, but
  - contains also creating of copies on archival OSDs

How to get AFS/OSD

- Right now AFS/OSD is still not in the official OpenAFS distribution and CVS
- AFS/OSD is maintained in the subversion server of DESY
  - You can view the source and patches at

```
https://svnsrv.desy.de/viewvc/openafs-osd/trunk/openafs
```

  - or check it out with

```
svn co https://svnsrv.desy.de/public/openafs-osd/trunk/openafs
```

  - To get full functionality configure with

```
--enable-namei-fileserver --enable-largefile-fileserver
--enable-object-storage --enable-vicep-access
```

Summary

- AFS/OSD is in full production at our site, but DESY Zeuthen and DESY Hamburg will follow this year.
- Embedded cluster filesystems (Lustre, GPFS) are not yet in production at any cell, but both DESYs will start to deploy them
  - They add high performance in batch clusters to AFS and
  - allow world wide access to data within these filesystems from any platform.
  - They liberate these filesystems from millions of small files better stored in AFS
  - They add HSM features to Lustre
- For future deployment of these features on more sites it is very important that this code comes into the main OpenAFS source tree.
  - If you don't configure your build with these features your AFS will be 100 % compatible to the stable release.
  - If you configure with object storage and vicep-access
    - your clients remain 100 % compatible with the stable release,
    - but your volumes have to be moved to the new servers

Questions or comments?

Thank you