CERN Site Report

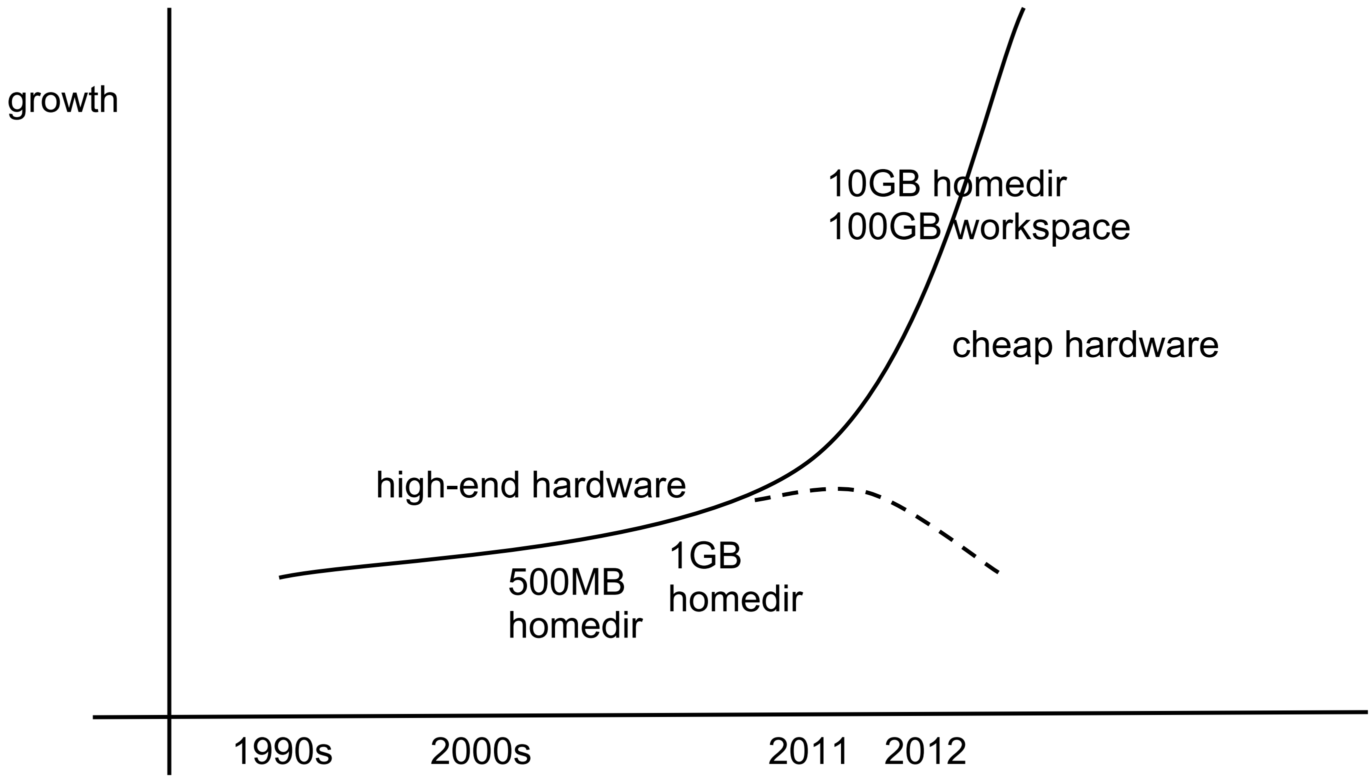
Jakub Moscicki (CERN IT)
Dan van der Ster (CERN IT)

European AFS and Kerberos Conference 2012
16-18 October 2012

Outline

- Service Evolution
  - 2012 and general perspective
  - storage architecture revision
- Feedback to community
- Performance Tuning
- Summary

AFS service growth

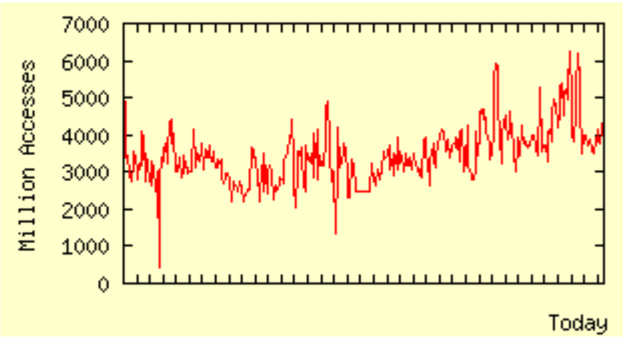


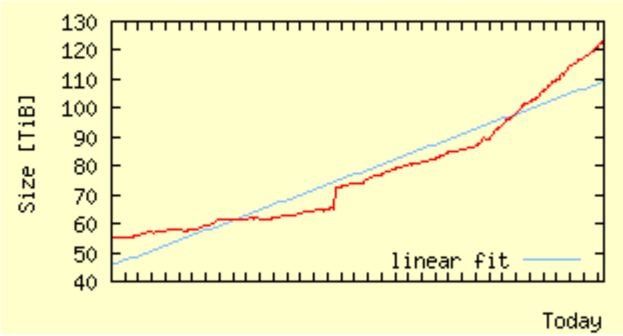
CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

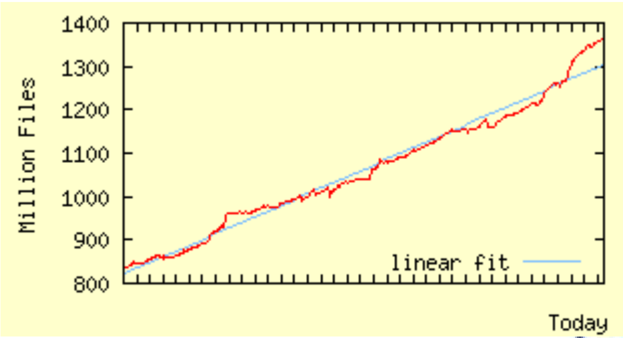
2012 Numbers

- OpenAFS 1.4.14+CERN patches
- AFS Usage:
  - ~29k users
    - up to 10GB home dir
    - ~6000 active last week
  - ~3k user *workspaces*
    - up to 100GB quota
  - ~200 *projects*
    - delegated administration
    - ~15k volumes

- Service scale:
  - ~73k volumes
  - 1.36 billion files
  - 131TB on disk
  - 341TB allocated quota
  - 59 servers
  - ~16k active clients including ~5k from outside CERN

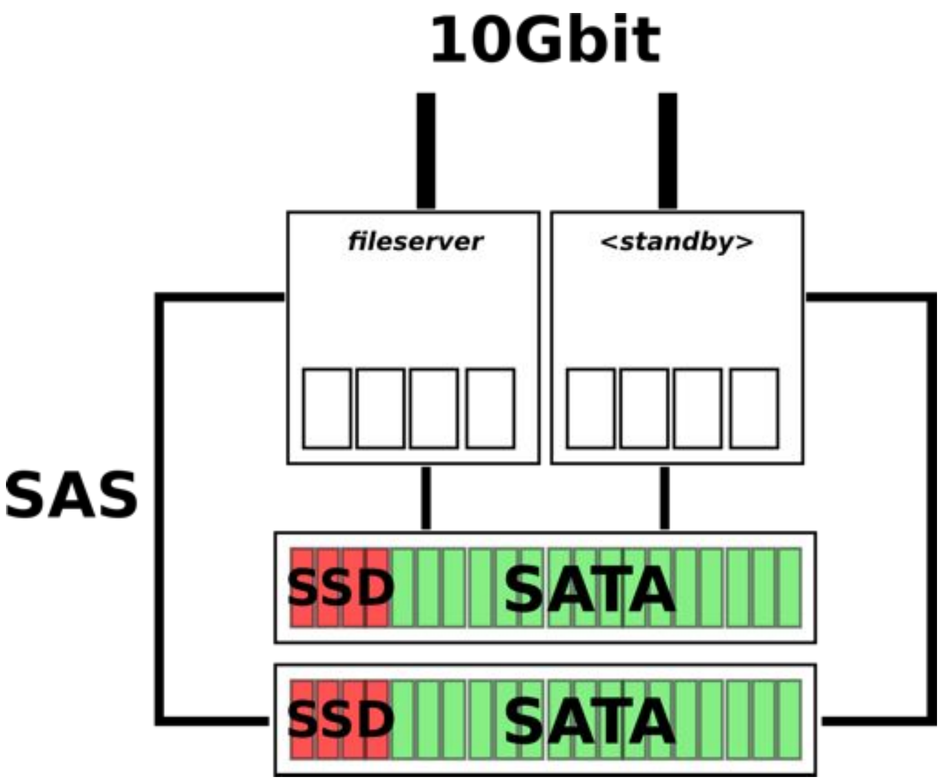






CERN AFS Hardware

- large storage units



- 2 headnodes
  - active
  - standby
- SATA disks for bulk storage (32TB) within SAS enclosure
- SSD cache
- 2x16x2TB SATA (RAID1)
- 2x4x250GB SSD (6%)
- 10Gb ethernet
- 4 Cores / 32GB RAM

Flashcache

- **flashcache**
  - used via device mapper
  - cache hit rates varies (from 40 to 90%)
  - needed to patch the code to handle shaky SSDs (next slide)
- **write-through** setup
  - high cost of warming up
  - volume dump: 4-5 x performance loss if filling the cache (worst-case scenario testing against disk streaming performance (150MB/s))
  - factor 2-3 for "real volumes"
- fileserver vs volserver
  - volserver operations (e.g. backup)
    - do not need caching
    - daily backups may actually invalidate the "useful" fileserver cache (no evidence though)
  - **blacklisting of volserver**
    - allows us to speed up backup considerably

SSD experience

Assorted problems with SSDs

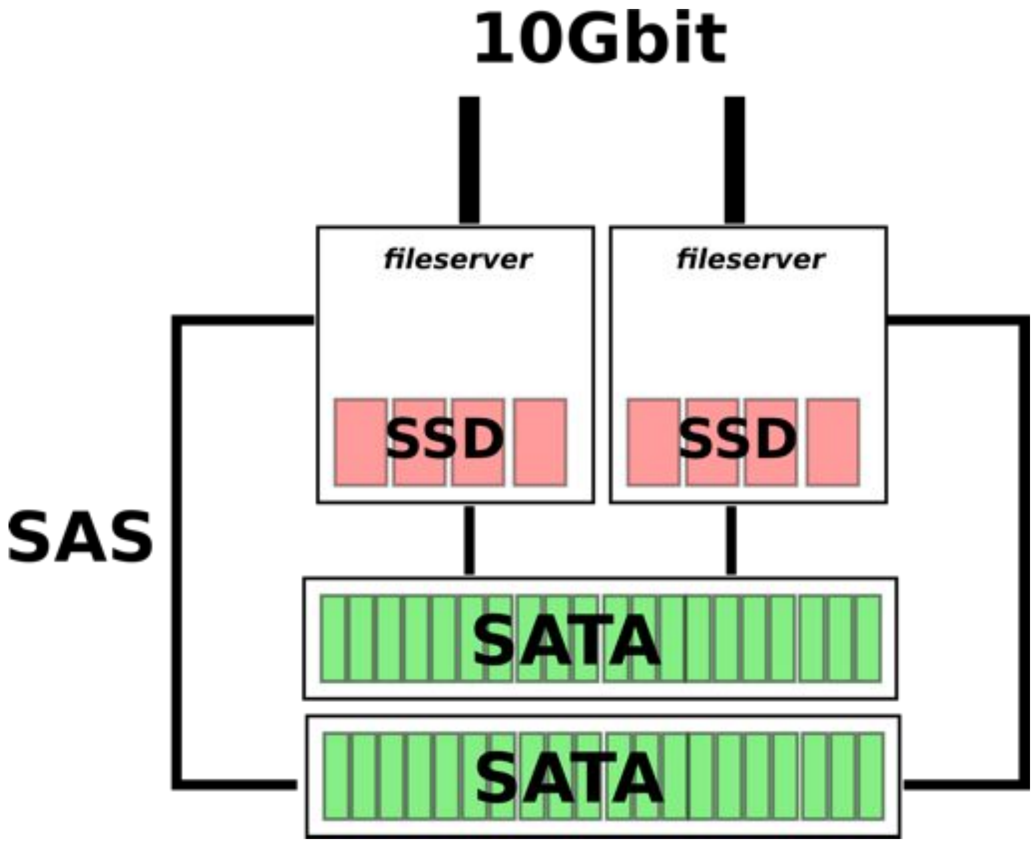
- devices dropping out for few seconds
- ...or disappearing completely from one or both nodes
- SAS->SATA converters giving problems?
- firmware upgrade on the drives improves the situation

Flashcache patched

- correctly bypass entire disk if drops out
- reattach devices on the fly
- extra protection to avoid multiple mounts

We eventually moved SSD to internal bays

Revised architecture

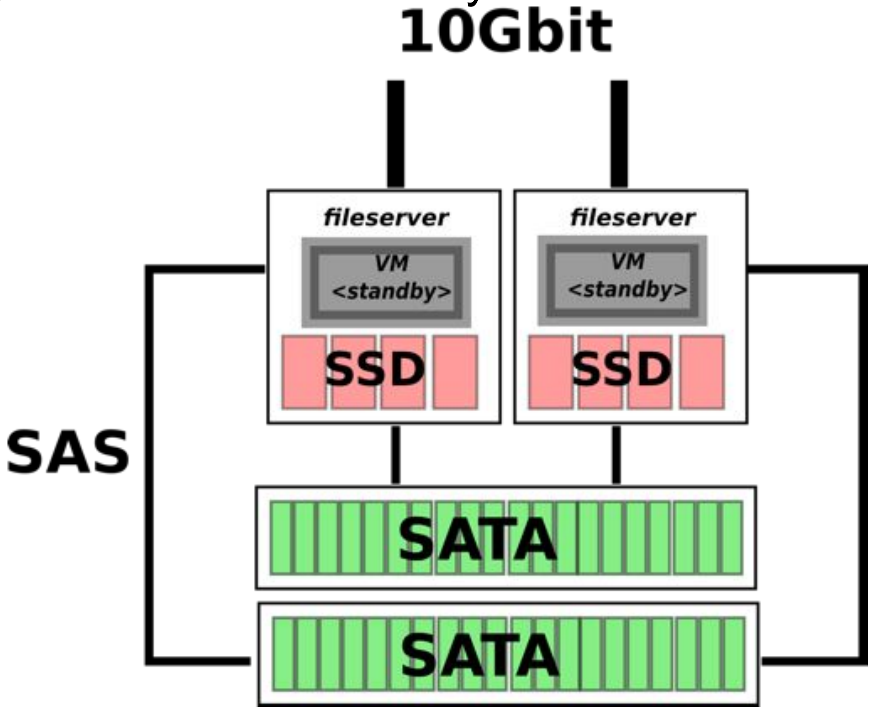


- not all SSDs visible from both servers
- but more stability
- related change:
  - active/active configuration (no standby)

Server failover with VMs

Failover using virtual machines

- suggested at EAKC2011 by Andrew Deason



KVM + libvirt /virtmanager

flip scripts: identity switch instantaneous swapping /usr/afs/local/sysid

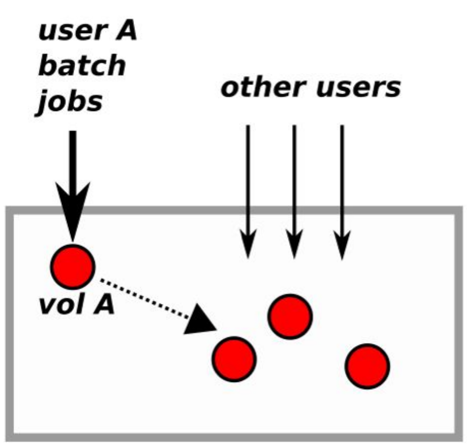
(in practice takes around 2 minutes + time to start a VM)

DSS Access latency (1)

- Introduction (details follow in part 2)
- Users don't complain about throughput
  - but they do about access time on the interactive prompt
    - "ls of death"
- Access latency
  - measured for all partitions and reported in monitoring console
  - mixed use-patterns
    - interactive access
    - batch farm (several thousand nodes, 35K cores)
  - incidents frequent
  - becomes more severe with new server architecture
    - more sharing, less isolation
    - more space, more/less IOPS(?)

Access latency (2)

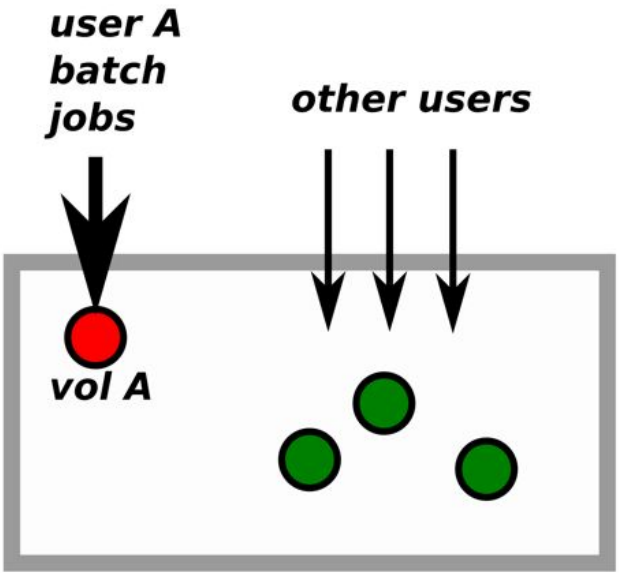
- Decomposed into to 2 (independent) problems
  - thread shortage
    - CERN patch for call rescheduling
    - two thresholds: n1 (idle server), n2 (busy server)
  - "rx-limit"
    - symptoms
      - threads available but idle
      - looks like lock contention in oprofile dumps
    - reproducible via synthetic tests
    - fixed by the network settings



one user / one volume impacts
all others on the same server

Tuning goals

- Desired result
  - much shorter access times in general for all
  - user hammering a volume will not affect others
  - but he may still slow down himself
- Service Classes
  - home directories (interactive access)
  - workspaces (batch access)



volscan

- useful and used
  - so far mainly for troubleshooting
- likely replacement for our monthly reporting tool
  - afsmounts
  - monthly cell snapshots (list of all mounts, files, volumes etc.)
- thanks to Michael Meffie (and EAKC 2011)
  - -ignore-magic option
    - ...so we did... ;-)

AFS with cloud storage

Reusing (bits of) volume magic

- to store association to the backend storage

Initial experiments and tests

- store files in external (cloud) storage
  - **libs3**
- use local storage of the file server node as cache
  - up to 32 file transfer in parallel in the background
  - file recall automatic

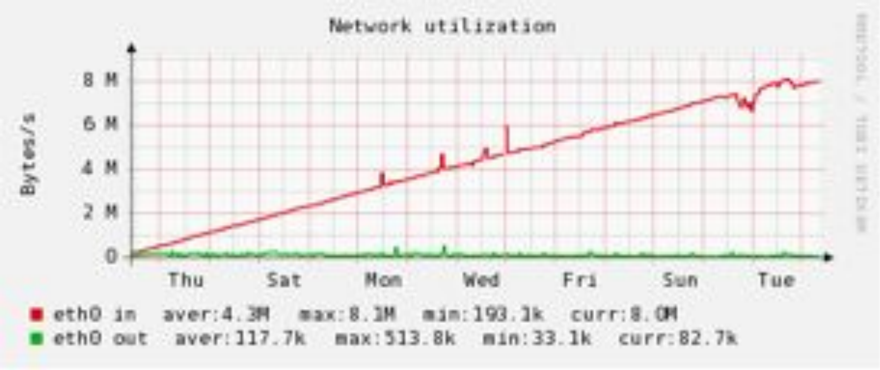
First conclusions

- it may be possible to keep rich AFS semantics (ACLs, ownership, consistency model)
- ... and streamline the backend storage

1.6 clients incident

1.6.0 clients deployed on remote sites in the LHC Grid

- client keep-alive packets (for NAT port mapping)
- bug: no proper connection cleanup (?)
- linearly increasing packet rate
- a remote 1,500 machine cluster generating 1MHz packet rate after few days
- impacted CERN's firewall and AFS fileservers
- 1.6.1-pre* patch improved the situation but not solved it
- After ~2 weeks → 300-400 pings/s per client
- 1.6.1 seems OK again



Clients

- Linux
  - SLC
- Windows 7
  - OpenAFS 1.7.15 via MSI installer
  - general feedback
    - too hard to install
    - recommended setup does not work (Heimdal crashes)
      - MIT kerberos instead (KfW 3.2.2)
    - still some issues (double entry KfW in start menu)
  - reported on openafs-info
    - https://lists.openafs.org/pipermail/openafs-info/2012-July/038326.html
    - Heimdal 1.5.1
      NIM 2.0.102.907
      OpenAFS for Windows 1.7.15
      Windows 7 Enterprise SP1, 64 bit
- Mac
  - installation provided by openafs
  - incompatibility with macports/kerberos

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Part II:
RX / fileserver Performance & Tuning

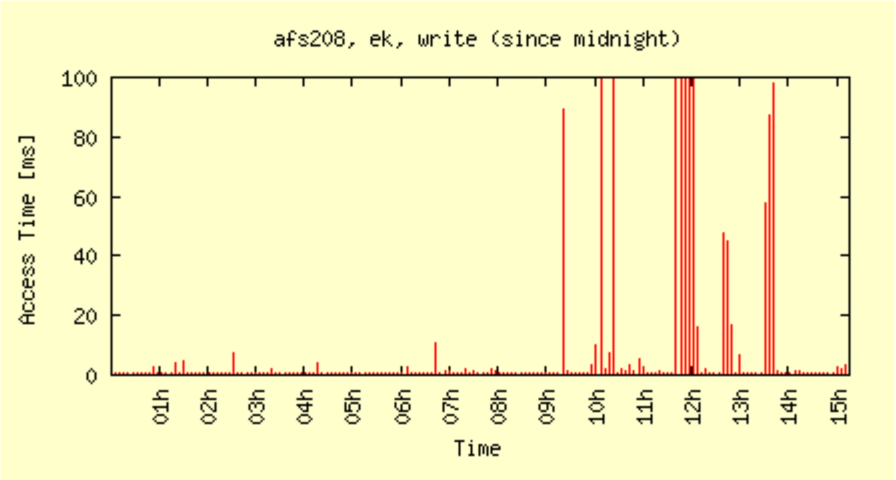
Introduction

- Kuba introduced the access latency issues we observe at CERN.
- During the past 1-2 months we've focused our efforts to find the root cause and *hopefully* a fix.
- What follows is an expansion of the mail sent to openafs-info last week:
  - https://lists.openafs.org/pipermail/openafs-info/2012-October/038822.html

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Problem Symptoms

- High latency incidents. What do we observe?
  - at least one user is hammering the fileserver (with 100 or more batch jobs)
  - 64kB write latency on *any* volume on the affected server goes from ~10ms as usual to more than 10-20 seconds
  - network throughput is "flat" for the duration of the incident, but well below the historical peak throughput
    - sometimes flat at ~50MBps or up to ~150MBps
    - server has a 10Gbps network card, 250MBps observed in past
  - CPU usage is also flat at ~120% (corresponding to 1 processor + a bit)
  - iostat shows little or no disk activity
  - no shortage of threads (more than 100 idle threads)



Debugging fileserver

- Our first efforts to debug this issue involved profiling and tracing the fileserver:
  - oprofile and stack traces (gcore, gdb) during incidents
- Various observations:
  - worker thread shortage: all worker threads are busy and requests are queued;
  - lock contention: idle worker threads waiting on internal locks which are not solicited by the listener thread = most time spent in pthread library;
  - listener thread becoming CPU-bound and using 100% CPU;
  - listener thread disappearing sometimes with the hot-thread feature enabled.
- Hints toward two largely independent root causes.

Thread Shortage Issue

- Thread shortage has been a known-problem in the past and the fileserver was patched at CERN to address it:
  - 240 server threads
  - call throttling (via rescheduling) to implement fair-share of the worker threads (i.e. prevent one client from consuming all workers)
- Rescheduling algorithm currently used in production:
  - up to n2=1/4th of server threads per volume under normal load
  - up to n1=1/8th of server threads per volume if there are other calls waiting for a thread.
  - calls beyond n2 or n1 are assigned to a rescheduling thread, of which there are ~10.

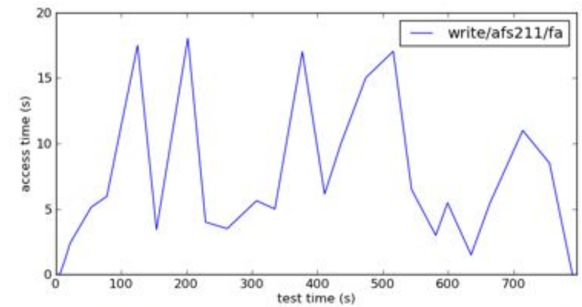
- Despite the rescheduling patch, we still observe latency issues.
- In other words, we regularly see very large latencies when there are many idle worker threads.
- So there must be another issue!

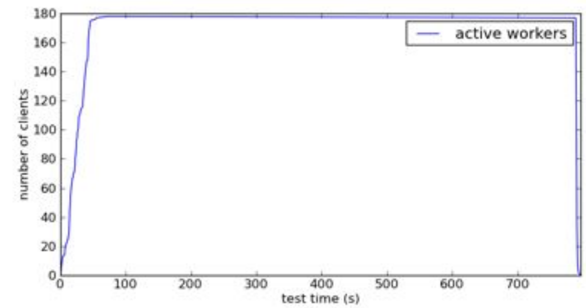
Isolating the 2nd Issue

- Next, our efforts focused on replicating high latency with minimal client connections.
- Set up a test server with to run synthetic stress tests against.
  - Fast new hardware with 10Gbps network (confirmed with iperf)
- Multiple test volumes:
  - 1GB file with random data
- Testing clients:
  - Each client tries to cp the 1GB file from AFS into /dev/null
- Hammer test:
  - Run N clients via our LSF batch system to scale from 10-1000 clients.

Stress Testing fileserver

- We were able to reproduce high latency *without* thread shortage or call rescheduling.
- Generally, 30 clients are able to increase the latency dramatically.
- With 180 clients (6 volumes, 30 clients each) the access time can be up to 15-20s.





High access latency (write) reproduced in the test environment for 6 x 30 test jobs on lxbatch. OpenAFS 1.4.14 +CERN. In practice for the end-users working interactively the service is unavailable.

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Issue #2?

- Profiling and tracing the `fileserver` during the stress tests confirmed the earlier observations of production servers.
- All indications pointed at an RX limitation, related to locking or thread scheduling
- So we decided to attempt further isolating the issue with `rxperf`.

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Latency testing with rxperf

CERN IT Department

servers

```
./1.4-cern/th_rxperf server -p 12314 -V -j -H -S 256 &
./1.6-stable/th_rxperf server -p 12316 -V -j -H -S 256 &
./1.6-master/th_rxperf server -p 12317 -V -j -H -S 256 &
```

readv, no jumbo frames, hotthreads, 256 server processes

hammer clients

```
for i in {1..250}
do
    args="-p 12317 -T 1 -s afs200 -V -j -H"
    ./th_rxperf client -c recv -b 20000000 -t 1 $args &
done
```

Note:
Client threads (`-t`) are handled on the server as up to 4 calls within the same connection.

So "`-t 250`" != 250 separate rxperf clients!!

latency probes

```
for port in 12314 12316 12317
do
    ./th_rxperf client -c recv -b 64000 -T 1 -t 1 -s afs200 -p $port -j -V -H
    ./th_rxperf client -c send -b 64000 -T 1 -t 1 -s afs200 -p $port -j -V -H
    ./th_rxperf client -c rpc  -S 64000 -R 64000 -T 1 -t 1 -s afs200 -p $port
done
```

Note: In the results that follow we show the master branch client numbers. 1.4.14 & 1.6-stable clients are ~identical.

rxperf baseline

CERN IT Department

- Baseline latency on an unloaded server:

```
client 1.6-master server 12314
RECV: threads    1, times         1, bytes        64000:          2 msec    [229.02 Mbps]
SEND: threads    1, times         1, bytes        64000:          1 msec    [267.40 Mbps]
RPC: threads     1, times         1, write bytes  64000, read bytes 64000:         2 msec    [3038.36 Mbps]

client 1.6-master server 12316
RECV: threads    1, times         1, bytes        64000:          1 msec    [324.22 Mbps]
SEND: threads    1, times         1, bytes        64000:          2 msec    [227.42 Mbps]
RPC: threads     1, times         1, write bytes  64000, read bytes 64000:         2 msec    [3442.34 Mbps]

client 1.6-master server 12317
RECV: threads    1, times         1, bytes        64000:          1 msec    [303.09 Mbps]
SEND: threads    1, times         1, bytes        64000:          1 msec    [387.52 Mbps]
RPC: threads     1, times         1, write bytes  64000, read bytes 64000:         2 msec    [3131.12 Mbps]
```

fileserver vs. rxperf

- With rxperf we found that with >=5 clients we could increase the latency dramatically:

```
client 1.6-stable server 12316
RECV: threads   1, times        1, bytes        1048576:             32 msec    [247.22 Mbps]
SEND: threads   1, times        1, bytes        1048576:           3307 msec    [2.42 Mbps]
```

- But this contradicted the fileserver stress tests:
  - Fileserver could handle up to 30 clients
- We compared our fileserver and rxperf and found the only difference to be **UDP buffer size** (fileserver: 2MB, rxperf: 64kB)
- *Indeed, increasing the rxperf server's UDP buffer size to 2MB raised the client limit to 30.*

The UDP Buffer

- Checking `netstat -s` (or `/proc/net/snmp`) we confirmed a large fraction of UDP inErrors during the stress testing
  - inErrors / inDatagrams > 10%
- Thus the root cause of the high latency had a strong suspect:
  - significant loss of ack or data packets was slowing down the read/write access time.
  - write latency suffers more than read, since writes fill more of the servers in buffer
- We confirmed very large inError counts on almost all of our fileservers.

rxperf results (2MB buffer)

- With a 2MB buffer, the access time sharply increases when #clients > 30.
  - ~4s latency with 30 clients; much longer with more connections.
- 250 clients makes the latency == duration of the hammer clients.

```
client 1.6-master server 12314
...
(I'm too impatient)
```

rxperf results (16MB buffer)

- After testing various sizes, we've concluded on 16MB
- Latency with 250 hammer clients:

```
client 1.6-master server 12314
RECV: threads   1, times        1, bytes        64000:      1088 msec   [0.45 Mbps]
SEND: threads   1, times        1, bytes        64000:       896 msec   [0.54 Mbps]
RPC: threads    1, times        1, write bytes  64000, read bytes 64000:      1714 msec   [4.66 Mbps]

client 1.6-master server 12316
RECV: threads   1, times        1, bytes        64000:       336 msec   [1.45 Mbps]
SEND: threads   1, times        1, bytes        64000:       248 msec   [1.96 Mbps]
RPC: threads    1, times        1, write bytes  64000, read bytes 64000:       453 msec   [17.66 Mbps]

client 1.6-master server 12317
RECV: threads   1, times        1, bytes        64000:       303 msec   [1.61 Mbps]
SEND: threads   1, times        1, bytes        64000:       262 msec   [1.86 Mbps]
RPC: threads    1, times        1, write bytes  64000, read bytes 64000:       475 msec   [16.84 Mbps]
```

- ~10% variance in these latencies.
- All server versions become responsive despite the high client load.
- (1.6-stable and master offer a further speedup)

Note: Here we show master branch client numbers. 1.4.14 & 1.6-stable clients are almost identical.

rxperf results (throughput)

- Our primary benchmark is latency, but we can also report on the observed throughput:

```
1.4-cern server:
250*RECV: threads   1, times      1, bytes     20000000:        65393 msec[2.33 Mbps]
5000000000B / 65.393s ~= 0.612 Gbps

1.6-stable server:
250*RECV: threads   1, times      1, bytes     20000000:        20455 msec[7.46 Mbps]
5000000000B / 20.455s ~= 1.96 Gbps

1.6-master server:
250* RECV: threads  1, times      1, bytes     20000000:        19397 msec[7.87 Mbps]
5000000000B / 19.397s ~= 2.06 Gbps
```

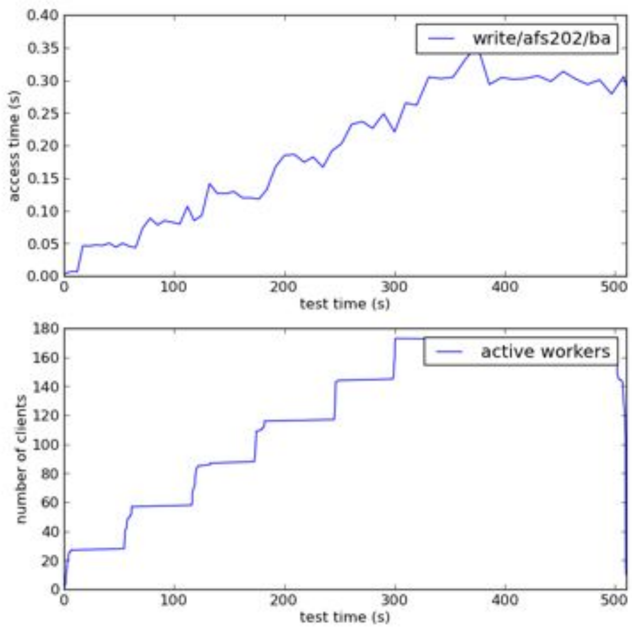
- Lower throughput for 1.4 was not expected; we often see 2Gbps on our production 1.4 fileservers.

DSS

fileserver w/ 16MB buffer

CERN IT Department

- Very large buffer was also confirmed to fix the latency in the fileserver stress test
- *Recall: 2MB, 180 clients -> 10-20s latency*
- With 16MB: 180 clients -> ~300ms latency



Stress test with UDP buffer size set to 16MB. In practice for the end-users working interactively the service is slightly slowed down.

Our UDP "fix" for RX

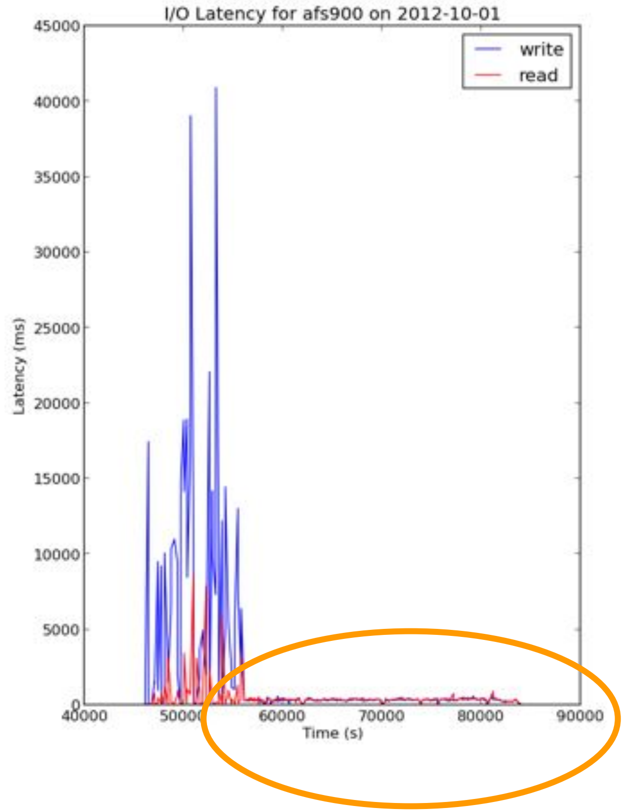
- The results have convinced us to start *slowly* rolling the following into production:

```
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.core.rmem_default=65536
sysctl -w net.core.wmem_default=65536
sysctl -w net.ipv4.tcp_rmem=4096 87380 16777216
sysctl -w net.ipv4.tcp_wmem=4096 65536 16777216
sysctl -w net.ipv4.tcp_mem=16777216 16777216 16777216
sysctl -w net.ipv4.udp_mem=16777216 16777216 16777216
sysctl -w net.ipv4.udp_rmem_min=65536
sysctl -w net.ipv4.udp_wmem_min=65536
sysctl -w net.ipv4.route.flush=1


fileserver -L -b 4096 -vc 16384 -s 32768 -l 8192 -p 256 -udpsize \
    16777216 -implicit rl -cb 524288
volserver -p 16 -udpsize 16777216
```

Very Large Buffer in Production

- We first implemented the change during an incident on one of our "jail" VM servers

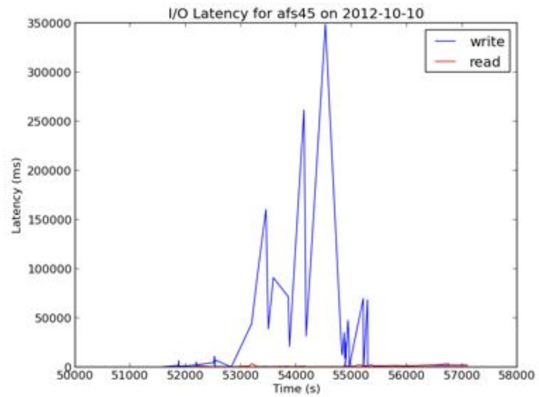


Decreased the access time from 40s to ~300ms.

CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Another example in production

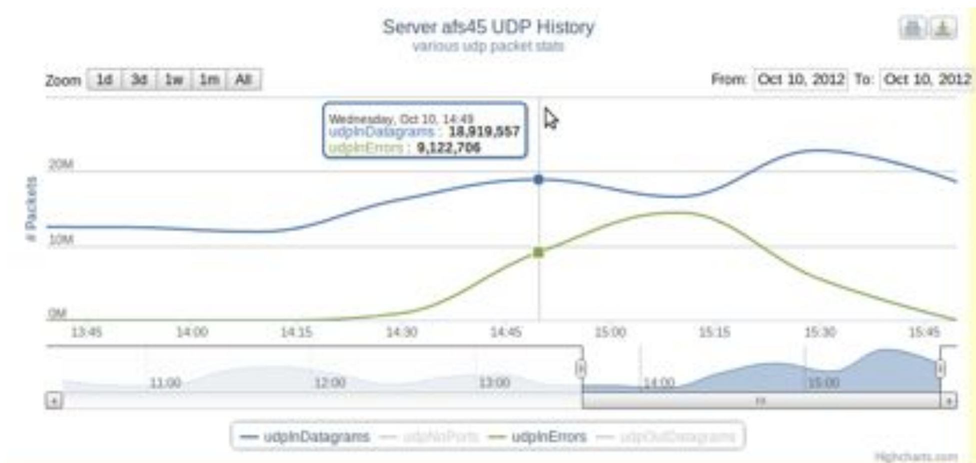
- Late last week we had a 2nd opportunity to deploy the change:



Two users hammering with batch jobs

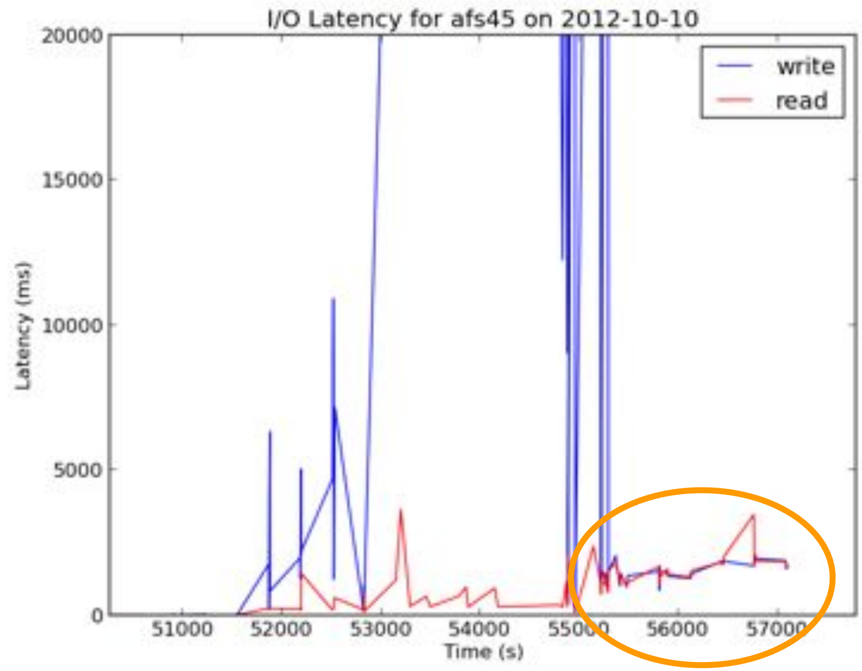
Up to 350s access latency observed

UDP inErrors/inDatagrams = 50% !!



CERN IT Department
CH-1211 Geneva 23
Switzerland
www.cern.ch/it

Another example in production (cont'd)



- After applying the 16MB buffers, latency dropped **from ~350s to 1s**.
- Tested the interactivity of the server (ls, touch, rm): *quite usable*.

Part II Summary

- We observe two separate issues which can send the access latency to ~infinity:
- Thread shortage
  - mitigated by our call rescheduling patch
- UDP buffer overflow / packet loss
  - eliminated with a 16MB buffer
- RX appears to be limited by the speed of the Listener, which is ~uniprocessor
  - Peek rxperf is 2Gbps

- 1.6.x gives 2-3x lower latency than 1.4.14

Overall summary

Aggressive service growth

- revised architecture
  - cheaper hardware = better scaling
- new issues/limits discovered
  - hopefully also have solutions

OpenAFS community

- appreciated and helpful
- major releases need more testing?

OpenAFS future and evolution at CERN

- CERN currently heavily depends on AFS
- we would like to (and have to) continue like this for (at least) some time (read: as long as possible)
- different groups at CERN also look at different solutions, possibly alternatives (NFS v4.1,...)