MindWorks UK

TTP TECHNOLOGY TRANSFER PARTNERS

Novell

Measuring file system performance

Joe R. Doupnik

MindworksUK and Univ of Oxford

jrd@netlab1.oucs.ox.ac.uk

MindWorks IT Training & Event Specialists

JRD NETWORKING RESEARCH INSTITUTE

Common comments

The file system is slow (circumstances not stated)

Backups are glacial, exceed time window

Users with huge directories get poor performance

Copying large files takes forever

File system X is “better” than Y

Going about investigation

Clients have applications, which talk to the local o/s, which talks to a protocol stack, which speaks over the wire to the server stack, to filters (NCP), on to file systems (caches intervening) and then the SAN, not to forget LUM lookups in eDir

Pinpointing a file system as a slow component is not done well through many layers

We try to go to the file system host itself & test there

Common ways of investigating

Do simple file copying from place A to B

- **File caching makes results be suspect for many cases**

Exercise various kinds of file operations, such as create, read, write, rewrite, delete, list directories

Simulate many clients doing “normal” file work

Add protocol layers, such as NCP, versus direct

But wait, there is more…

File throughput numbers are one thing

What the server experiences is another:

CPU consumption, caching, disk i/o's per second

The system ought to provide headroom to handle peak load, congestion and other events

We need to look beneath the covers

Simple local writing using dd

Write null bytes from /dev/zero in 4KB i/o chunks

Two CPUs, 3GB memory, ESXi VMDK disks on same SATA drive

```
# cat ddtest
#!/bin/sh
rm ddtest.out
for FS in EXT3 XFS REISER NSS; do
echo "Starting $FS"
echo "Starting $FS" >> ddtest.out
time (dd if=/dev/zero of=/home/$FS/test bs=4096 count=2000000 >> ddtest.out; sync )\
         2>> ddtest.out
sleep 60
done
#
```

That's 8.2 GB worth of pure writing

dd writing test results

```
Starting EXT3
8192000000 bytes (8.2 GB) copied, 91.2973 seconds, 89.7 MB/s
real      1m50.239s
user      0m0.460s
sys       0m17.161s

Starting XFS
8192000000 bytes (8.2 GB) copied, 88.8803 seconds, 92.2 MB/s
real      1m40.796s
user      0m0.444s
sys       0m19.137s

Starting REISER
8192000000 bytes (8.2 GB) copied, 99.4317 seconds, 82.4 MB/s
real      1m58.680s
user      0m0.328s
sys       0m23.029s

Starting NSS
8192000000 bytes (8.2 GB) copied, 136.9 seconds, 59.8 MB/s
real      2m17.566s
user      0m0.396s
sys       0m20.221s
```

Without salvage

Vmstat during dd writing test

```
# vmstat 5
procs ---5                                                  --- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  4    148   9792  13644 1892268    0    0    47  6996  174 1970  7 21 57 14  0
 0  4    148   9288  13992 1887808    0    0     1 81613  410 2870  1 12 22 65  0
 0  5    148  10788  14432 1886340    0    0    38 75495  407 3193  1 14  6 79  0
 0  5    148  13040  14760 1889096    0    0   110 74970  412 3064  5 13  0 82  0
 0  5    148   9384  15092 1886708    0    0    14 78352  409 3316  4 14  0 82  0
 0  9    148   9568  14948 1885824    0    0   129 72020  408 3078  1 12  0 87  0
 1  5    148  11004  14644 1888184    0    0   130 65163  426 2973  2 11  0 88  0
 1  6    148   7828  14232 1891680    0    0     5 79974  411 3018  1 12  0 87  0
 1  7    148   7620  13692 1891192    0    0    74 71867  411 3118  3 13  0 84  0
 0  7    148  10436  13580 1888220    0    0     1 59913  441 2795  1 11  9 79  0
```

Waiting (wa) on disk rotation/head motion dominates results

id = %idle wa = %wait (in driver for disk)

Simple copy, caching effects

```
-rw-r--r-- 1 root root 707782656 Jun 16 18:32 oes2sp3-x86_64-Beta1.iso
# ln -s oes2sp3-x86_64-Beta1.iso oes.iso
# time cp oes.iso /home/EXT3

real    0m23.266s
user    0m0.064s
sys     0m3.452s
# time cp oes.iso /home/EXT3

real    0m16.271s
user    0m0.076s
sys     0m2.996s
# time cp oes.iso /home/EXT3

real    0m8.168s
user    0m0.020s
sys     0m1.528s
# time cp oes.iso /home/EXT3

real    0m9.809s
user    0m0.024s
sys     0m1.552s
# rm /home/EXT3/oes.iso
# time cp oes.iso /home/EXT3

real    0m7.002s
user    0m0.036s
sys     0m1.092s
#
```

File not touched prior to this cp step

EXT3 7-23 sec

“Practice makes perfect”

Caching over several passes is effective

But this is only half of the operation, as it turns out

More simple cp, vs file system

XFS

```
# time cp oes.iso /home/XFS

real    0m7.221s
user    0m0.012s
sys     0m1.152s
# time cp oes.iso /home/XFS

real    0m7.715s
user    0m0.012s
sys     0m1.252s
# time cp oes.iso /home/XFS

real    0m7.182s
user    0m0.052s
sys     0m1.140s
# rm /home/XFS/oes.iso
# time cp oes.iso /home/XFS

real    0m7.984s
user    0m0.052s
sys     0m1.144s
```

7-8 sec

REISER

```
# time cp oes.iso /home/REISER

real    0m9.307s
user    0m0.032s
sys     0m1.540s
# time cp oes.iso /home/REISER

real    0m19.056s
user    0m0.012s
sys     0m2.304s
# time cp oes.iso /home/REISER

real    0m8.726s
user    0m0.012s
sys     0m1.896s
# rm /home/REISER/oes.iso
# time cp oes.iso /home/REISER

real    0m15.615s
user    0m0.016s
sys     0m2.028s
```

9-19 sec

NSS nosal

```
# time cp oes.iso /home/NSS

real    0m14.823s
user    0m0.032s
sys     0m2.560s
# time cp oes.iso /home/NSS

real    0m14.572s
user    0m0.028s
sys     0m2.660s
# time cp oes.iso /home/NSS

real    0m14.081s
user    0m0.020s
sys     0m2.740s
# rm /home/NSS/oes.iso
# time cp oes.iso /home/NSS

real    0m17.589s
user    0m0.028s
sys     0m2.352s
```

14-17 sec

NSS salvage

```
# time cp oes.iso /home/NSS

real    0m14.931s
user    0m0.036s
sys     0m2.376s
# time cp oes.iso /home/NSS

real    0m13.738s
user    0m0.048s
sys     0m2.680s
# time cp oes.iso /home/NSS

real    0m14.406s
user    0m0.028s
sys     0m2.652s
# rm /home/NSS/oes.iso
# time cp oes.iso /home/NSS

real    0m18.537s
user    0m0.032s
sys     0m2.492s
```

14-18 sec

NSS has significant write-behind dumping after command has finished.
NSS kernel workers queue requests and go on and on, a heavy load

Reiser times vary by factor of two, when repeating this test many times

System usage (“sys” above) varies, heaviest on NSS

cp, including flush to disk (sync)

EXT3

```
# time (cp oes.iso /home/EXT3; sync)

real    0m20.712s
user    0m0.024s
sys     0m2.556s
# time (cp oes.iso /home/EXT3; sync)

real    0m20.995s
user    0m0.032s
sys     0m2.328s
# time (cp oes.iso /home/EXT3; sync)

real    0m21.696s
user    0m0.028s
sys     0m2.392s
# rm /home/EXT3/oes.iso
# time (cp oes.iso /home/EXT3; sync)

real    0m18.330s
user    0m0.052s
sys     0m1.836s
```

18-21 sec

XFS

```
# time (cp oes.iso /home/XFS; sync)

real    0m17.334s
user    0m0.036s
sys     0m2.128s
# time (cp oes.iso /home/XFS; sync)

real    0m17.211s
user    0m0.016s
sys     0m2.196s
# time (cp oes.iso /home/XFS; sync)

real    0m17.230s
user    0m0.048s
sys     0m2.276s
# rm /home/XFS/oes.iso
# time (cp oes.iso /home/XFS; sync)

real    0m16.963s
user    0m0.024s
sys     0m2.256s
```

17 sec

REISER

```
# time (cp oes.iso /home/REISER; sync)

real    0m21.563s
user    0m0.028s
sys     0m2.224s
# time (cp oes.iso /home/REISER; sync)

real    0m20.933s
user    0m0.028s
sys     0m2.584s
# time (cp oes.iso /home/REISER; sync)

real    0m21.459s
user    0m0.020s
sys     0m2.820s
# rm /home/REISER/oes.iso
# time (cp oes.iso /home/REISER; sync)

real    0m20.641s
user    0m0.020s
sys     0m1.952s
```

20-21 sec

This “other half” is to account for time to actually write data to disk, which some file systems (EXT3, REISER) are reluctant to do promptly

cp, including flush to disk

NSS without salvage

```
# time (cp oes.iso /home/NSS; sync)

real    0m21.366s
user    0m0.016s
sys     0m2.924s
# time (cp oes.iso /home/NSS; sync)

real    0m19.812s
user    0m0.020s
sys     0m2.888s
# time (cp oes.iso /home/NSS; sync)

real    0m19.349s
user    0m0.032s
sys     0m3.060s
# rm /home/NSS/oes.iso
# time (cp oes.iso /home/NSS; sync)

real    0m19.266s
user    0m0.028s
sys     0m2.836s
```

19-21 sec

NSS with salvage

```
# time (cp oes.iso /home/NSS; sync)

real    0m20.022s
user    0m0.040s
sys     0m3.128s
# time (cp oes.iso /home/NSS; sync)

real    0m20.281s
user    0m0.024s
sys     0m3.052s
# time (cp oes.iso /home/NSS; sync)

real    0m19.813s
user    0m0.024s
sys     0m2.884s
# rm /home/NSS/oes.iso
# time (cp oes.iso /home/NSS; sync)

real    0m25.087s
user    0m0.028s
sys     0m2.792s
```

20-25 sec

Account for time to actually write data to disk

Salvage has a visible effect even in this simple case

cp test results

Simple copying over and over is heavily influenced by retaining copies in cache memory

Flushing cache to disk shows all the file systems to behave similarly, with

- **XFS being a little quicker than others**
- **NSS using most CPU resources**

Simple tests, too many numbers

Venerable test tool *iozone* is wonderful about generating lots of values

It focuses on doing many fundamental i/o operations, with main interest on repeating for

a) various file sizes

b) within that various sizes of i/o requests

c) all very nicely in sequential order

This is unlike everyday usage

***Iozone* is not a good predictor for our purposes**

***Bonnie*++ is better for us but still synthetic**

Iozone 3.347 “all” test. Yikes!

KB	reclen	write	rewrite	read	reread	random read	random write	bkwd read	record rewrite	stride read	fwrite	frewrite	fread	freread
64	4	120493	205728	1124074	1336792	760859	320021	720041	598110	1183548	180788	187742	913649	1101022
64	8	300328	214096	1421755	1452528	1330167	272577	727850	571375	3541098	566551	582535	3057153	4018152
64	16	342040	233461	1599680	1609270	1421755	304761	704914	359444	1226821	250919	287156	1336792	1363961
64	32	426897	231050	1562436	1599680	1484662	254971	987600	359444	1255511	336889	342040	1143223	1392258
64	64	457452	241440	831569	818885	916769	233461	1017549	238015	1210227	242531	249985	1279447	1357066
128	4	234871	210884	1346196	1030846	1085013	200040	826202	780556	1085013	205791	181325	1185654	1217930
128	8	330015	157264	3124872	3657016	2985839	544601	1939522	1939522	4135958	540217	584313	2784517	4267461
128	16	871818	247656	1663139	1684006	1579934	342216	1085013	805138	1684006	265015	271722	1487977	1391557
128	32	499970	260641	1663139	1684006	1561553	336005	1082825	621518	1663139	305421	78919	1455701	1543594
128	64	547377	258632	1451764	1618028	1504659	274782	1254941	378663	4267461	1032829	1243315	1022989	1076312
128	128	522346	263973	1436229	1622919	1504659	250545	715059	241094	776042	272411	256531	1406136	1539168
256	4	212419	108889	1334162	1384034	1108314	165069	944525	1044693	1195962	234605	213179	1224606	1292410
256	8	359977	240813	1570244	1675612	1325925	106576	1019886	1286217	1447470	259872	248103	1422540	1543163
256	16	454627	261199	1718521	1766587	1354356	176811	1080434	1117543	1942349	267179	193195	1319408	1354356
256	32	473263	103385	1591186	1718521	1649865	100870	1112909	551540	1907837	121673	56990	1543163	1652404
256	64	595594	276181	1447470	1652404	1534342	112577	1099237	713067	1168627	314161	316476	1437779	1561112
256	128	579209	287265	1422540	1591186	1543163	287650	1286217	434044	1108314	379708	465466	1398455	1514860
256	256	630572	248275	944525	812329	1600674	137574	1325925	268381	1158540	267512	120756	1340826	1543163
524288	64	1021211	33549	3034057	2492848	2977318	3858	2265829	6504334	2983624	32074	30761	2187395	2750911
524288	128	1422264	21043	3013009	3136144	3041396	5694	2537632	6807619	3047495	20051	21640	2958771	2989339
524288	256	1369814	20239	2991710	2766206	2895948	8850	2808875	5181477	2925467	27899	20216	2610554	2910255
524288	512	1392257	29623	2821377	2894381	2813216	13278	2259626	3461465	2738438	20044	20202	2717054	2855396
524288	1024	1310824	32620	2694017	2665974	2773953	23165	2511522	3735047	2735168	27350	29332	2677863	2761385
524288	2048	1167185	19024	2073899	1962954	2038333	16807	2162163	2938029	2157360	23795	22889	2267221	2248897
524288	4096	976425	25004	1731671	1773652	1630188	25094	1782303	1322876	1719393	20978	27205	1771524	1790636
524288	8192	934757	22581	1639911	1762234	1784960	24065	1510332	1658079	1796404	25765	23646	1707300	1787007
524288	16384	972966	33765	1761946	1672252	1755250	25923	1630755	1429621	1752979	31575	36339	1626814	1742648

Results are in KB/sec. From the results (~6.8GB/sec etc) we see very heavy influence of file caching, until the very end. All sequential.

This example happens to be for NSS without salvage, as a VMDK.

Exercise some file operations

Program bonnie++ v1.03a

Command: bonnie++ -d /home/$FS -s 6G -n 16:1000000:45 -u root
for FS = EXT3, XFS, REISER, NSS (no salvage)

Two CPUs, 3GB mem, ESXi, one SATA, mount options noatime,nodiratime

One 6GB file, create and manipulate contents:

	------Sequential Output------						--Sequential Input-				--Random-	
	-Per Chr-		--Block--		-Rewrite-		-Per Chr-		--Block-		--Seeks--	
	K/sec	%CP	K/sec	%CP	K/sec	%CP	K/sec	%CP	K/sec	%CP	/sec	%CP
EXT3	53210	91	74931	14	33991	8	51972	85	79574	9	237.4	1
XFS	56291	93	90658	15	29401	7	50491	85	79219	10	290.6	1
REISER	56071	99	79943	16	27900	7	48987	84	72470	12	218.3	1
NSS	30564	57	48543	8	36004	6	44439	74	79923	10	250.1	0

File metadata, 16K files variously sized between 45B & 1MB:

	------Sequential Create------						--------Random Create--------						Test Suite
	-Create--		--Read---		-Delete--		-Create--		--Read---		-Delete--		Completion
	/sec	%CP	/sec	%CP	/sec	%CP	/sec	%CP	/sec	%CP	/sec	%CP	
EXT3	82	11	107	6	256	3	99	13	56	4	236	8	26m 16s
XFS	92	8	127	8	2033	21	103	9	65	4	454	6	23m 13s
REISER	103	11	59	5	7419	79	100	10	56	5	4935	59	26m 9s
NSS	122	11	151	10	8384	35	106	9	65	4	1002	5	24m 12s

Note random access tests in this suite

Summary of the results

Manipulating a single very large file:

- **NSS writes more slowly than the POSIX systems**
- **NSS reads about as quickly as POSIX**
- **POSIX systems are roughly alike**

Manipulating file metadata:

- **NSS does well**
- **XFS is next, REISER follows**
- **EXT3 does least well**

Write commits, barriers

"The metadata benchmarks are also important if you are creating a new filesystem on top of a RAID device. Journaling filesystems can use write barriers to protect their journaled metadata. If you are using a hardware card to provide RAID functionality, these barriers might force the **entire cache** on the RAID card and all disks assembled into the RAID on that card to be synced, which can lead to extremely poor performance.

As a real-world example, an Adaptec 31205 12-port card with a RAID-6 and XFS using barriers can support less than 100 file creates per second in tests I recently performed. Explicitly disabling barriers in XFS when mounting the same filesystem gives closer to 6,000 file creates per second. Though I'm not advocating disabling barriers in XFS, in this particular hardware configuration it could be done without data loss risk."

http://www.linux.com/archive/articles/139742

NSS and barriers (XEN based)

"A value of 0 (no barriers) is the best setting to use when the virtual disks assigned to the guest server's virtual machine are based on physical SCSI, Fibre Channel, or iSCSI disks (or partitions on those physical disk types) on the host server.

In this configuration, disk I/O is handled so that data is not exposed to corruption in the event of power failure or host crash, so the XenBlk Barriers are not needed.

If the write barriers are set to zero, disk I/O performance is noticeably improved."

http://www.intl.novell.com/documentation/oes2/stor_nss_lx_nw/?page=/documentation/oes2/stor_nss_lx_nw/data/bbm6yhg.html

XFS barriers, vs RAID

"Q. Should barriers be enabled with storage which has a persistent write cache?

Many hardware RAID have a persistent write cache which preserves it across power failure, interface resets, system crashes, etc. Using write barriers in this instance is not recommended and will in fact lower performance.

Therefore, it is recommended to turn off the barrier support and mount the filesystem with "nobarrier". But take care about the hard disk write cache, which should be off."

http://xfs.org/index.php/XFS_FAQ#Write_barrier_support.

XFS, barrier on local disks

Bonnie++ testing on my own gear

- **locally attached disks**
- **no RAID**
- **SCSI and ESXi VMDK**

shows no change of performance for XFS with and without barriers

Competition: netbench / dbench

General usage tests come from *netbench*/*dbench*

***N-B* has hundreds++ real clients chat with a server doing office style work, measures write throughput**

The network is often the bottleneck

Tis "a little awkward" to setup on our sites

***Dbench* simulates those clients within the server, using the same file system requests recorded from *netbench* tests (*dbench* distribution has this script), eliminating the network**

Dbench, two flavors

***Dbench* v 3.02 performs the i/o requests as quickly as can be accepted. This is a massive stream of requests where the file system, caching, and o/s scheduler are pushed hard. We use it in this work.**

***Dbench* v4 is a different beast and repeats requests at the same rate as observed with netbench tests over a 100Mbps wire. Most results are plain 18MB/sec, and the question is revised to be what about server load. We do not use it here.**

Dbench client ops script, a snippet

```
Deltree "\clients\client1" NT_STATUS_OK
Mkdir "\clients" NT_STATUS_OK
NTCreateX "\clients\client1" 0x1 0x2 16385 NT_STATUS_OK
Close 16385 NT_STATUS_OK
NTCreateX "\clients\client1\mixfile" 0x40 0x1 9935 NT_STATUS_OBJECT_NAME_NOT_FOUND
QUERY_PATH_INFORMATION "\clients\client1\~dmtmp" 1004 NT_STATUS_OBJECT_NAME_NOT_FOUND
FIND_FIRST "\clients\client1\FILLER.*" 260 1366 0 NT_STATUS_NO_SUCH_FILE
NTCreateX "\clients\client1\~dmtmp" 0x1 0x2 9937 NT_STATUS_OK
Close 9937 NT_STATUS_OK
NTCreateX "\clients\client1\filler.000" 0x40 0x2 9938 NT_STATUS_OK
QUERY_FS_INFORMATION 1 NT_STATUS_OK
WriteX 9938 65534 1 1 NT_STATUS_OK
QUERY_FILE_INFORMATION 9938 258 NT_STATUS_OK
WriteX 9938 0 65536 65536 NT_STATUS_OK
WriteX 9938 65536 65536 65536 NT_STATUS_OK
WriteX 9938 131072 65536 65536 NT_STATUS_OK
WriteX 9938 196608 65536 65536 NT_STATUS_OK
WriteX 9938 262144 65536 65536 NT_STATUS_OK
WriteX 9938 327680 65536 65536 NT_STATUS_OK
WriteX 9938 393216 65536 65536 NT_STATUS_OK
WriteX 9938 458752 65536 65536 NT_STATUS_OK
WriteX 9938 524288 65536 65536 NT_STATUS_OK
WriteX 9938 589824 65536 65536 NT_STATUS_OK
WriteX 9938 655360 65536 65536 NT_STATUS_OK
WriteX 9938 720896 65536 65536 NT_STATUS_OK
WriteX 9938 786432 65536 65536 NT_STATUS_OK
WriteX 9938 851968 65536 65536 NT_STATUS_OK
WriteX 9938 917504 65536 65536 NT_STATUS_OK
WriteX 9938 983040 65536 65536 NT_STATUS_OK
Close 9938 NT_STATUS_OK
NTCreateX "\clients\client1\filler.001" 0x40 0x2 9939 NT_STATUS_O
WriteX 9939 65534 1 1 NT_STATUS_OK
QUERY_FILE_INFORMATION 9939 258 NT_STATUS_OK
WriteX 9939 0 65536 65536 NT_STATUS_OK
WriteX 9939 65536 65536 65536 NT_STATUS_OK
WriteX 9939 131072 65536 65536 NT_STATUS_OK
WriteX 9939 196608 65536 65536 NT_STATUS_OK
WriteX 9939 262144 65536 65536 NT_STATUS_OK
WriteX 9939 327680 65536 65536 NT_STATUS_OK
WriteX 9939 393216 65536 65536 NT_STATUS_OK
WriteX 9939 458752 65536 65536 NT_STATUS_OK
WriteX 9939 524288 65536 65536 NT_STATUS_OK
WriteX 9939 589824 65536 65536 NT_STATUS_OK
```

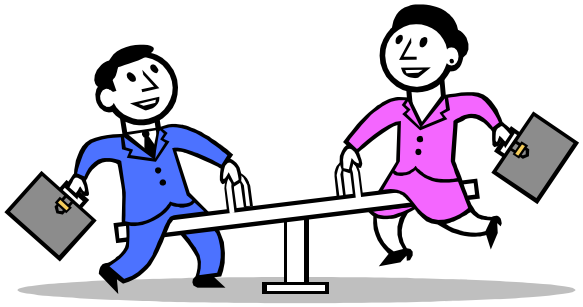
458K such operations in the script, which repeats to fill the test time

Each command was recorded from a Windows client doing MS office style set of operations

Dbench v3 ignores the time of arrival information, v4 enforces it

We use dbench v3 as a stress test

Competition: CPU work vs disk access



dbench **is forcing i/o requests as quickly as possible, so that the system is always busy. If it is not doing one thing then it is doing another.**

The workload is mixed, with multiple independent clients, closer to real life than *bonnie* et al

Dbench, 4 clients, ESXi, 1 CPU

Write throughput, CPU resources, io ops, effort
Wall clock time is ~7min, one CPU for service

File system	MB/s	user	system	io ops	ops/CPUsec
EXT3	503	1m53s	5m 4s	10M	24059
XFS	446	1m46s	4m47s	8.9M	22649
REISER	455	1m45s	5m12s	9.1M	21645
NSS nosalvage	279	1m18s	4m45s	5.6M	15473
NSS salvage	88	25s	1m39s	1.8M	14770

Commands (360 second duration):

```
dbench -t 360 -D /home/NSS* 4 >> filename
sync
sleep 60
```

* For each file system

A file write may or not result in a disk write

Dbench, 4 clients, ESXi, 2 CPU

Write throughput, CPU resources, io ops, effort
Wall clock time is ~7min, two CPUs for service

File system	MB/s	user	system	io ops	ops/CPUsec
EXT3	821	3m 6s	9m33s	16.3M	26714
XFS	698	3m 2s	8m45s	13.9M	19729
REISER	741	2m58s	10m51s	14.8M	17802
NSS nosalvage	310	1m26s	6m24s	6.2M	13097
NSS salvage	70	20s	1m39s	1.4M	11977

Commands (360 second duration):

dbench -t 360 -D /home/NSS* 4 >> filename

sync

sleep 60

* For each file system

A file write may or not result in a disk write

Comments

Compare the previous two slides

Adding a CPU

- **increases write rates, but does not double**
- **work done (i/o ops) increases, but does not double**
- **i/o ops per CPU cycle goes down, less efficient**
- **NSS salvage goes more slowly**

Contention between CPUs is apparent

Dbench, NSS salvage effects

```
ESXi VM, OES2 SP3 Beta 1 (64 bits), 4GB memory, 1 cpu 2.6GHz, locally
attached SATA drive (using vmdk)

dbench version 3.04 - Copyright Andrew Tridgell 1999-2004
Commands:       dbench -D /home/NSS 4 >> saltest.txt
                sync
                sleep 10
repeated five times

Salvage is inactive and volume has been purged:

Throughput 149.781 MB/sec 4 procs
Throughput 138.852 MB/sec 4 procs
Throughput 146.174 MB/sec 4 procs
Throughput 142.026 MB/sec 4 procs
Throughput 142.477 MB/sec 4 procs

Salvage is active and volume has been purged:

Throughput 59.557  MB/sec 4 procs
Throughput 28.6634 MB/sec 4 procs
Throughput 23.6824 MB/sec 4 procs
Throughput 21.5026 MB/sec 4 procs
Throughput 23.3666 MB/sec 4 procs
```

Salvage work extracts a toll! More as files accumulate

Inferences thus far

The reported write rates are far in excess of the physical disk drives, thus a lot of writes are to cache memory

Adding CPUs can help with such memory work, but efficiencies drop and later performance may drop

EXT3, XFS, REISER have roughly similar work output

NSS salvage really slows down things

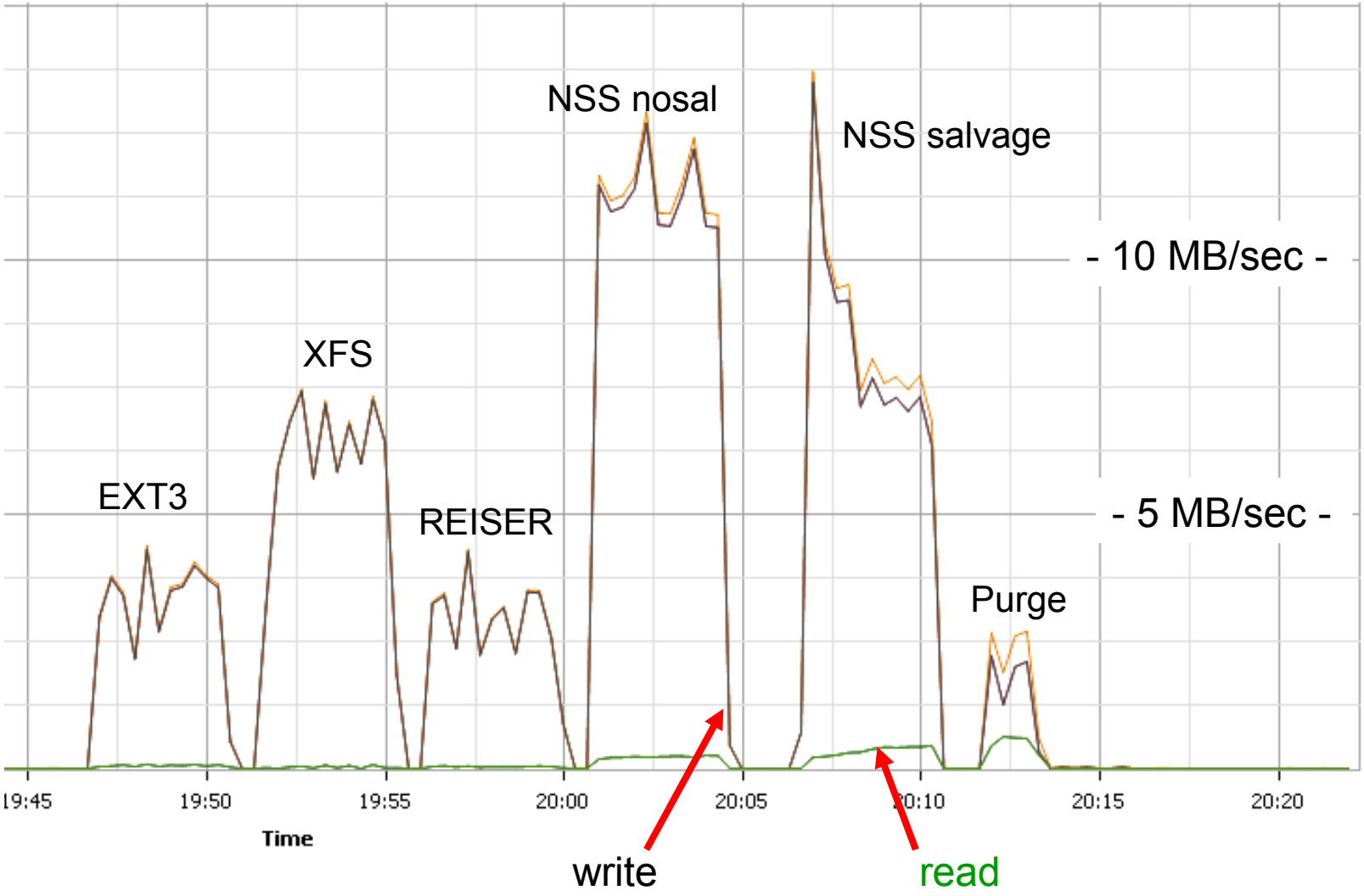
What the o/s thinks about this

We look beneath the covers to see how the o/s reacts to these dbench tests

In particular, we are curious about the disk channel traffic, comparing it with the application rates

I use a cheap method: ask ESXi to report statistics

ESXi view: dbench disk r/w rates



Note NSS salvage workload effect

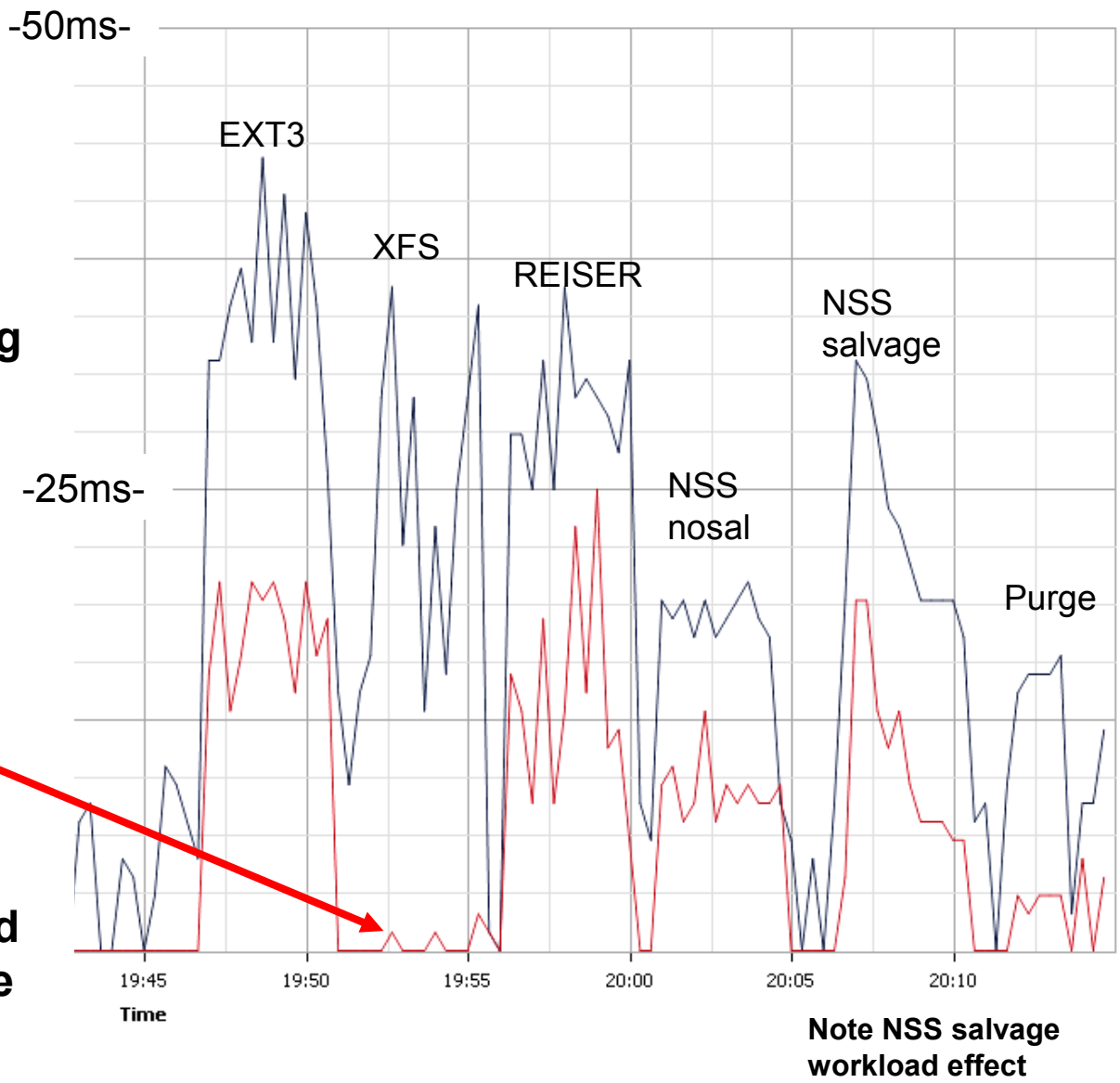
Nearly all app reads are from cache

ESXi view: datastore latency

Big means waiting on disk drive

Data proximity to reduce head motion delays

Grey: read
Red: write



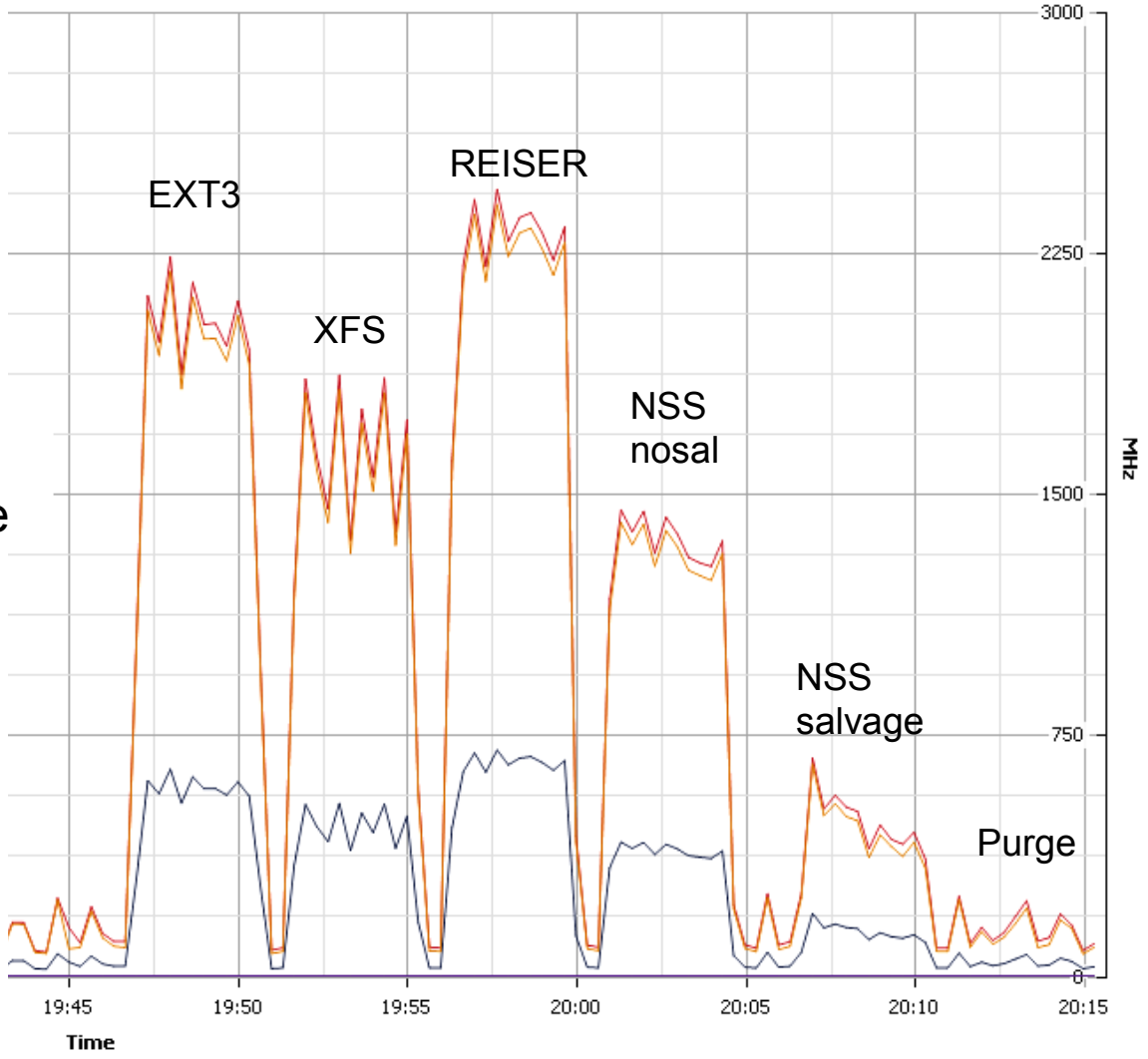
Note NSS salvage workload effect

ESXi view: CPU consumption

Red: % (100 at top)

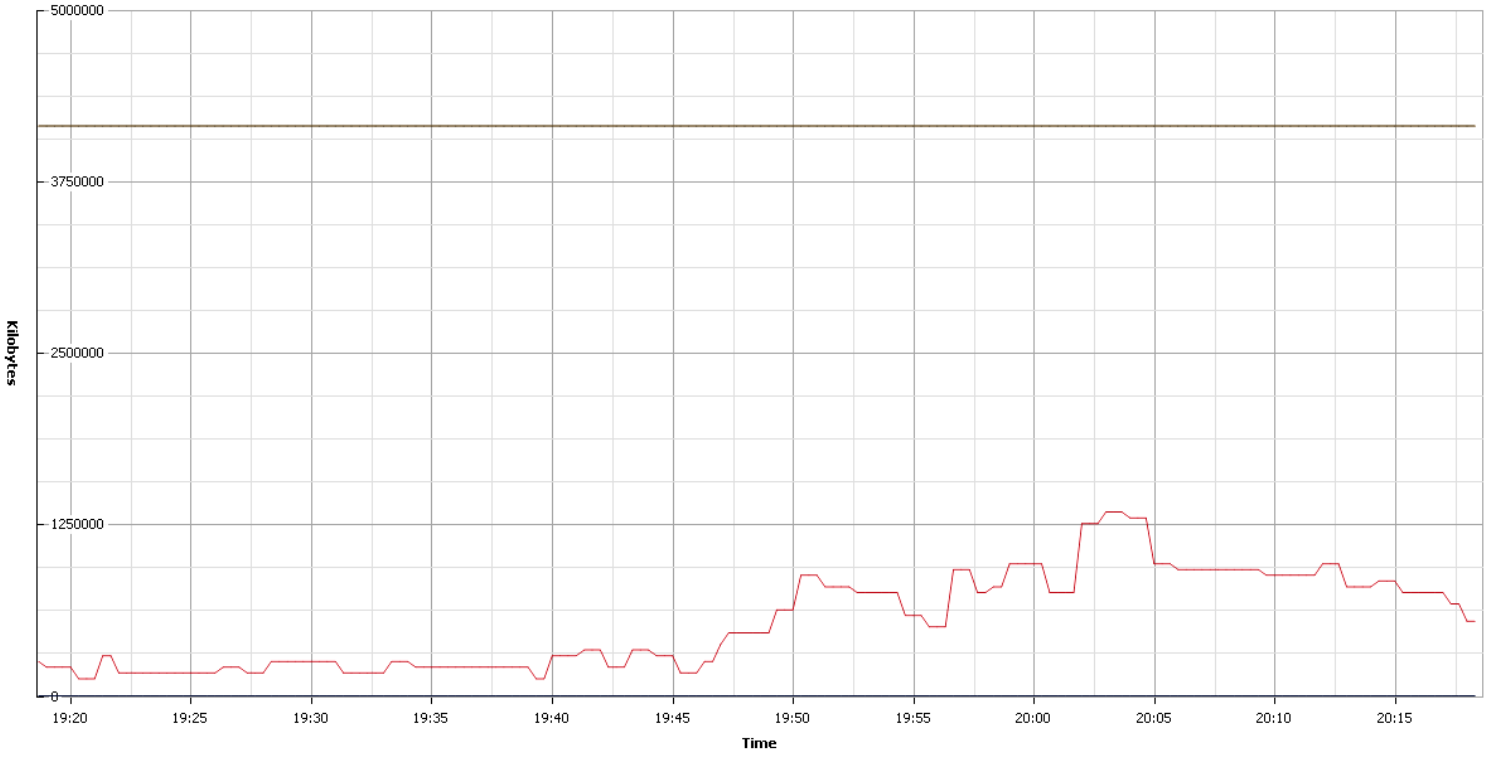
Blue: MHz, right scale

If the CPU is not busy then the system is likely waiting on disk



ESXi view: memory usage

Nothing much here, please move along



Performance Chart Legend

Key	Object	Measurement	Rollup	Units	Latest	Maximum	Minimum	Average
	OES2SP3	Balloon	Average	Kilobytes	0	0	0	0
	OES2SP3	Active	Average	Kilobytes	545256	1342176	125828	540830.2
	OES2SP3	Granted	Average	Kilobytes	4159488	4159488	4159488	4159488
	OES2SP3	Consumed	Average	Kilobytes	4159488	4159488	4159488	4159488

More inferences

EXT3 and REISER do least i/o to disk, thus we suspect they retain data in memory for long periods, a worry. They run CPU bound.

XFS and NSS do much more disk i/o, hopefully writing to disk as they go along. They are throttled by waiting on the disk drive, NSS especially.

XFS does good regionalized i/o to minimize head motion (as seen in the latency plots)

More inferences

What we don't know is what these guys write as they go along. It should be journal entries plus changed data.

But the low rates for EXT3 and REISER raise questions about how well they commit to disk in a timely fashion, such as a file is changed, closed, reopened soon afterward, changed again, etc, and eventually the last image is written to disk. We simply don't know, but I think they defer.

Extended attributes (xattr)

POSIX and NSS support reading and writing of xattr *name=value* attribute pairs

NSS requires two startup lines to expose NSS attribs through the POSIX xattr interface

/ListXattrNWMetadata

/CtimeisMetadataModTime

Put them in /etc/opt/novell/nss/nssstart.cfg

Xattr is used heavily by Samba/CIFS, I am told

Dbench, 4 clients, ESXi, xattr, 1 CPU

Write throughput, CPU resources, io ops, effort
Wall clock time is ~10min, one CPU for service

File system	MB/s	user	system	io ops	ops/CPUsec
EXT3	338	2m13s	6m48s	11.3M	20695
XFS	36	45s	2m15s	1.2M	7500
REISER	204	1m34s	9m14s	6.6M	10185
NSS nosalvage	47	28s	3m13s	1.6M	7240
NSS salvage	16	11s	1m32s	0.58M	5670

Commands (600 second duration):

```
dbench -x -D /home/NSS* 4 >> filename
sync
sleep10
```

* For each file system

Dbench, 4 clients, ESXi, xattr, 2 CPU

Write throughput, CPU resources, io ops, effort
Wall clock time is ~10min, two CPUs for service

File system	MB/s	user	system	io ops	ops/CPUsec
EXT3	535	2m 8s	9m55s	10.7M	14862
XFS	62	38s	2m24s	1.2M	6845
REISER	224	1m10s	9m18s	4.5M	7185
NSS nosalvage	66	24s	2m33s	1.4M	7738
NSS salvage	25	9s	1m 4s	0.51M	7018

Commands (600 second duration):

dbench -x -D /home/NSS* 4 >> filename
sync
sleep10

* For each file system

Comments

Compare the previous two slides

Discover that

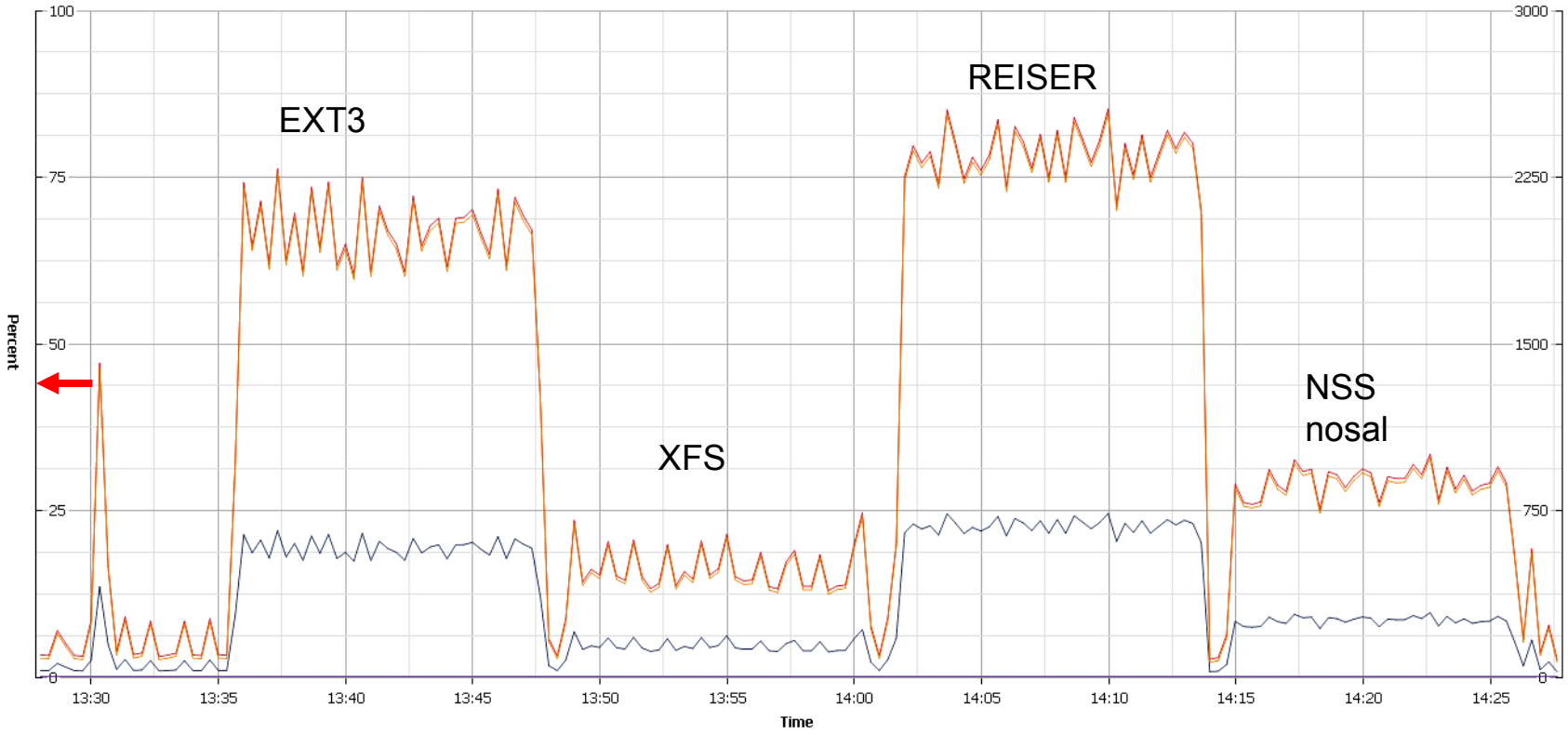
- **write rate goes up with CPU count**
- **i/o ops (work done) goes down with CPU count!**
- **i/o ops per CPU second goes down with CPU count**

Contention between CPUs is costing us,

- **xattr disk work makes this more obvious and has a serious performance impact**

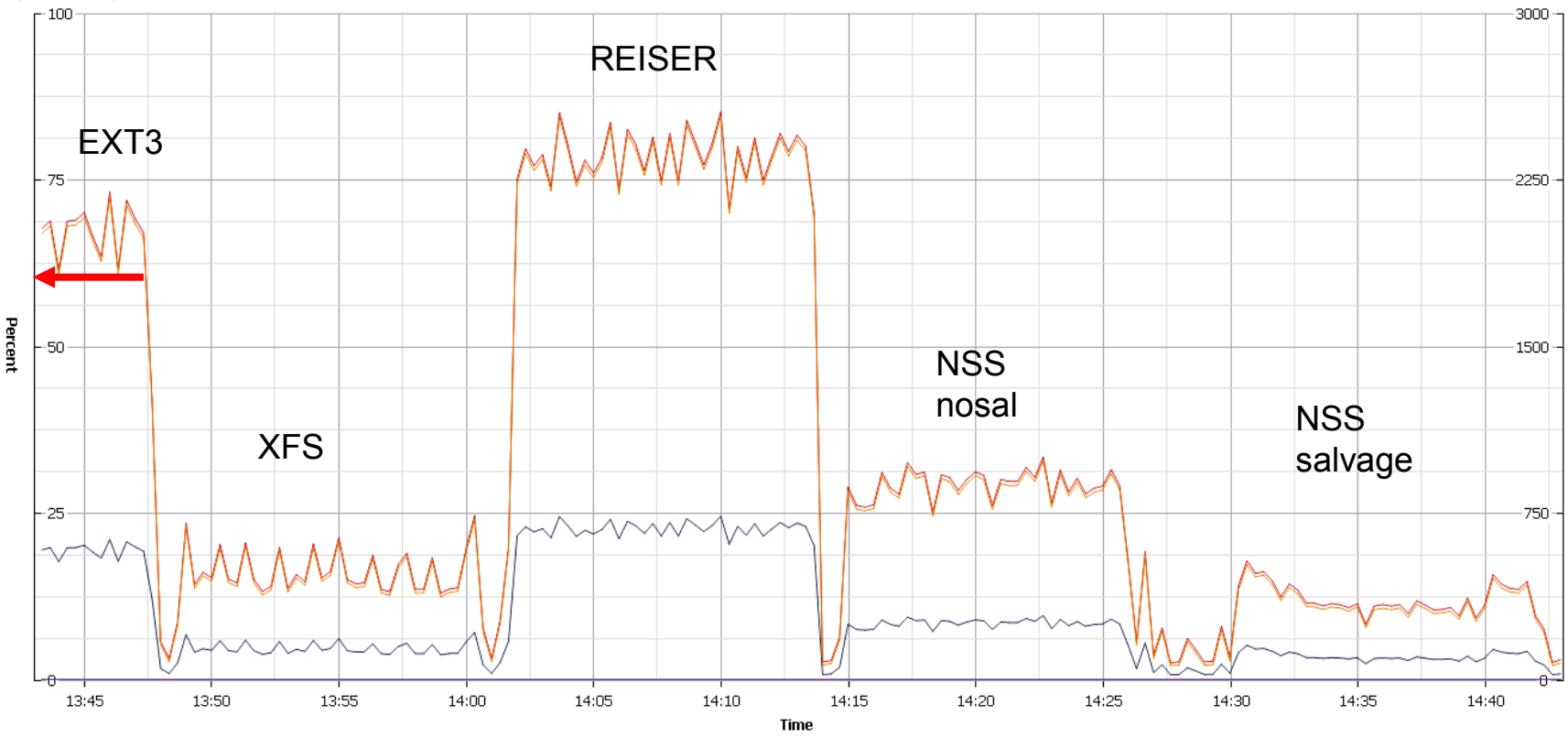
“Too many chiefs” (or “chefs”) effect

ESXi view: CPU usage with xattr



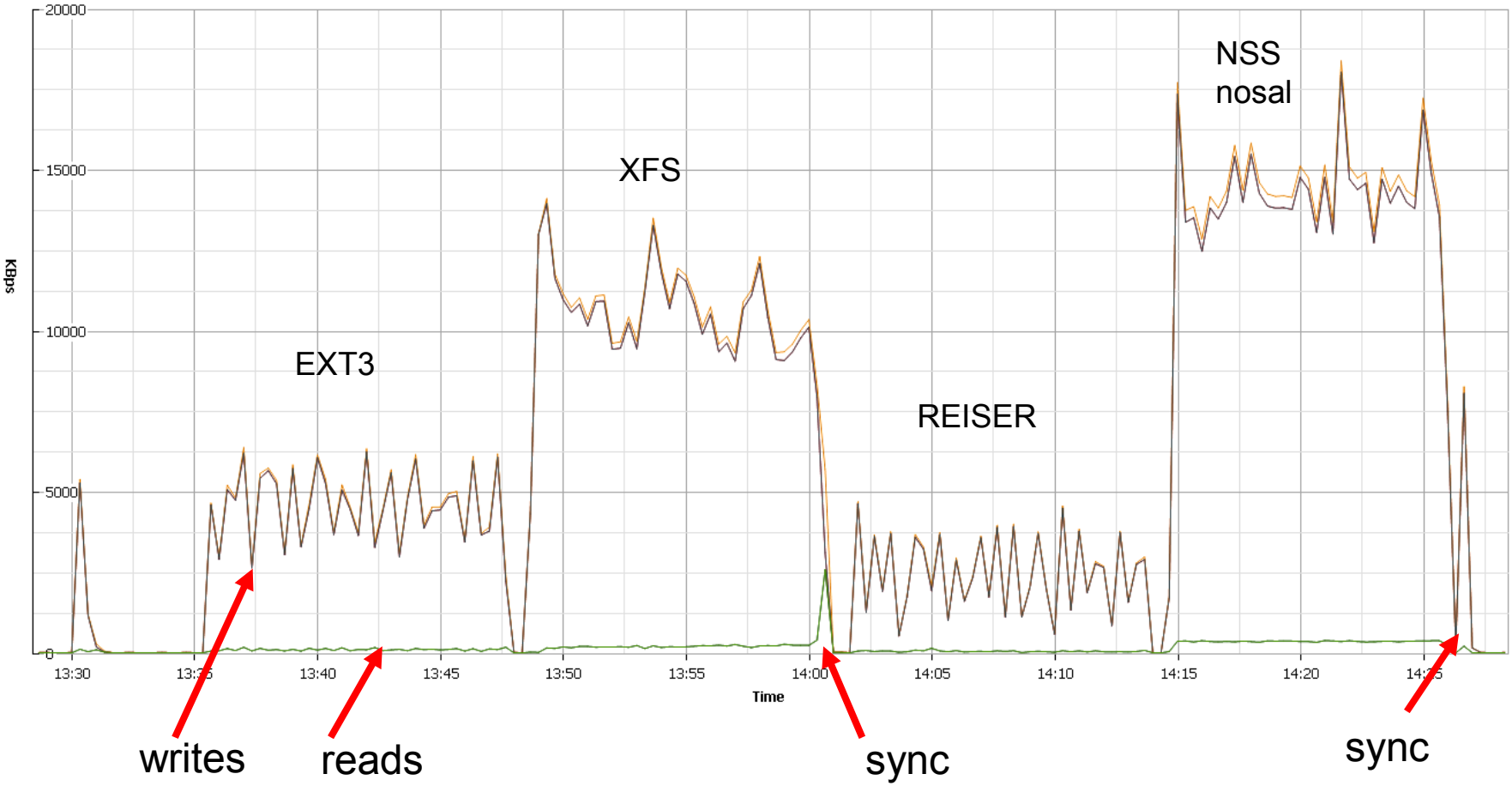
Continued on next slide

ESXi view: CPU usage with xattr



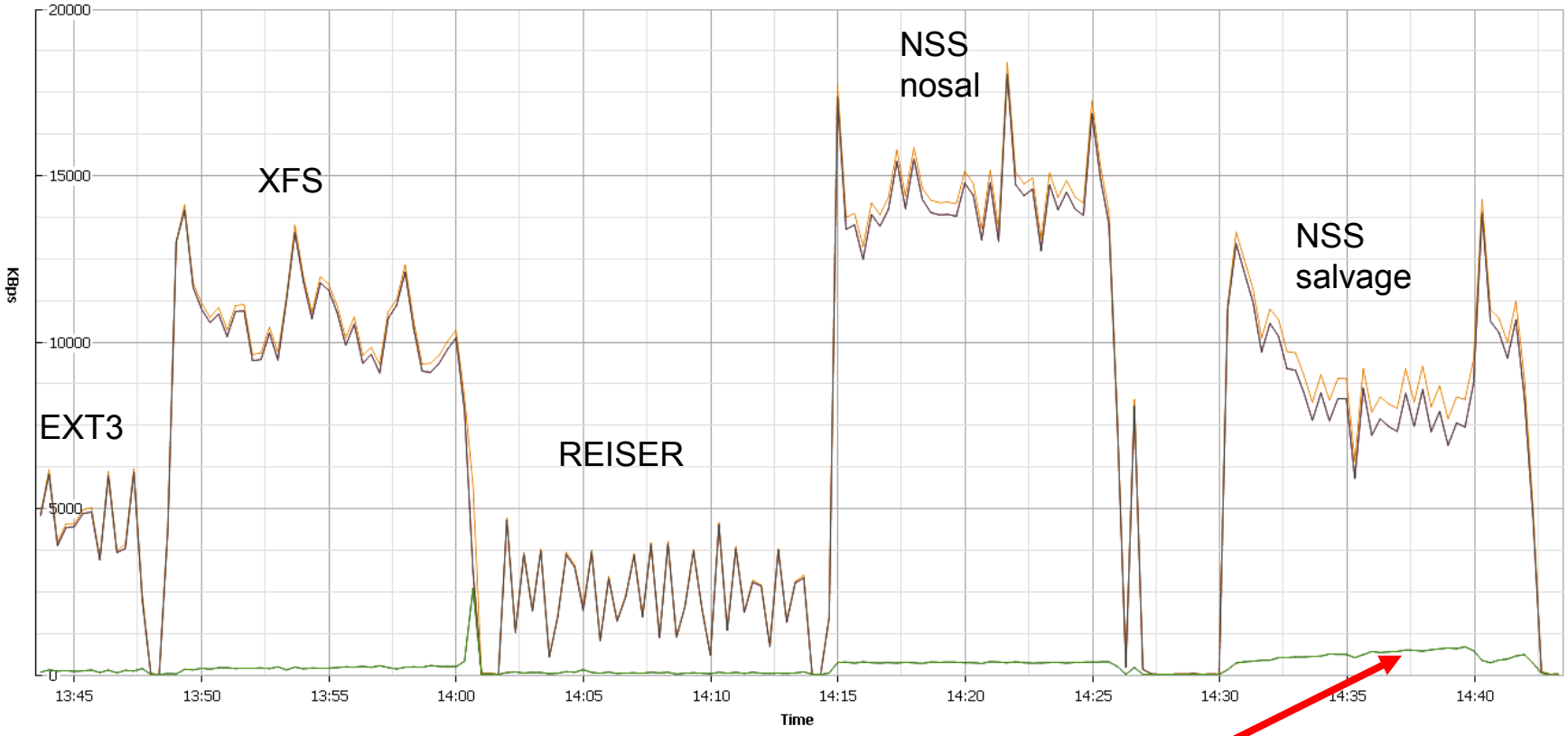
If the app is not going to or waiting on disk, then it is using CPU cycles tinkering with cache memory and file system code etc

ESXi view: disk r/w with xattr



Continued on next slide

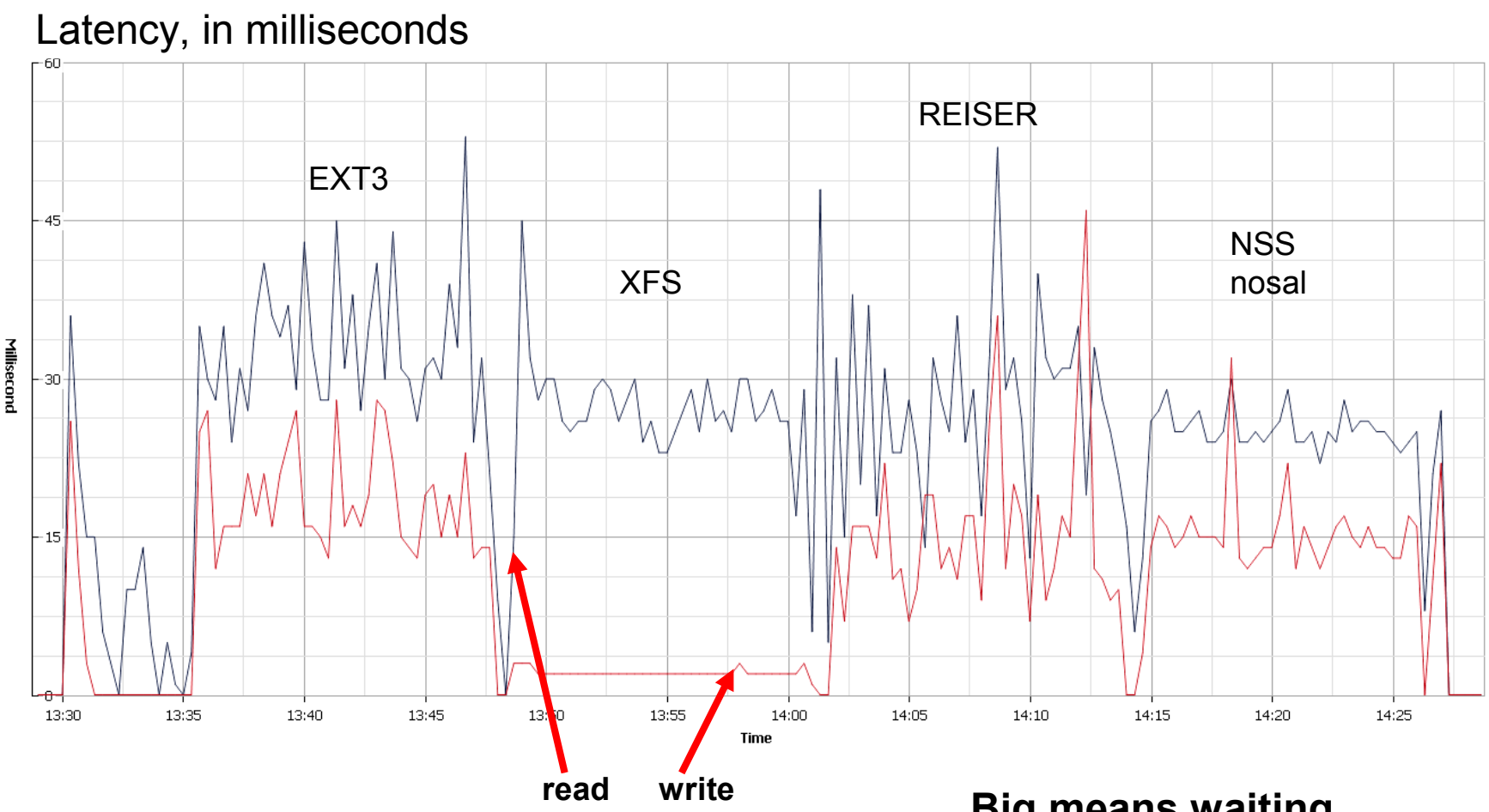
ESXi view: disk r/w with xattr, cont'd



See salvage accumulate

More disk activity with xattr work

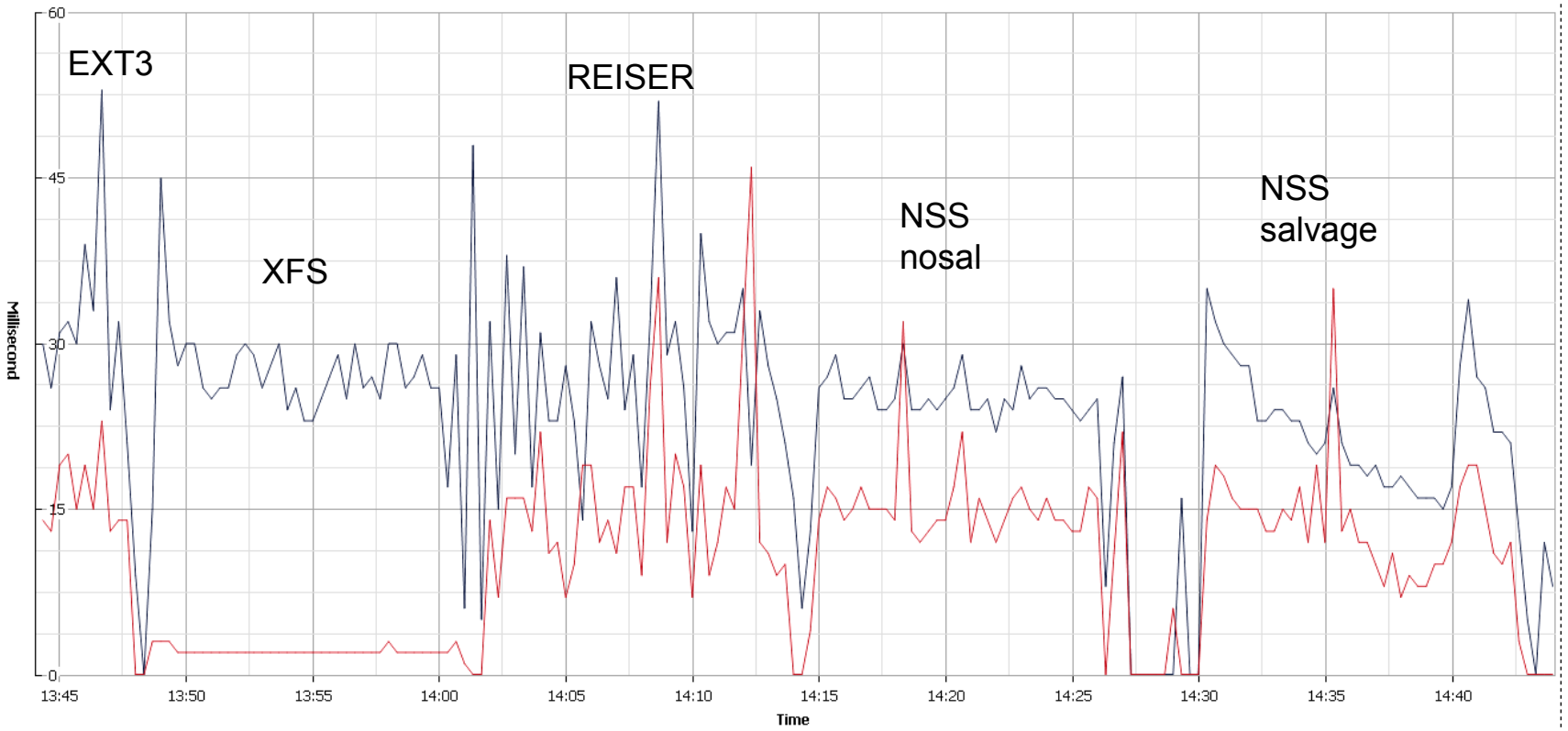
ESXi view: datastore latency with xattr



EXT3 and REISER jump around a lot

Big means waiting on disk drive

ESXi view: datastore latency with xattr, cont'd



Continued from previous slide

About same disk scattering with and without salvage

Comments

EXT3 and REISER do relatively little writing to disk, running mostly from memory cache. This is a serious concern for data loss and recovery from outages. REISER tends to use a lot of CPU.

XFS uses relatively least CPU resource, moderate system loading. Those actual disk writes are encouraging, data really does get to disk promptly (at a throughput cost).

[I feel most comfortable with XFS behaviour]

Comments

- NSS does a lot of disk writing (good), uses CPU very heavily with salvage, decent throughput without salvage
- NSS kernel workers (up to 70) are overdone by queueing a lot and then fighting for CPU time
- Salvage causes major throughput drop (much extra disk work), and NSS becomes slower with more files in the salvage bin
- Benefits are often worth the price, but keep salvage under control

Easy to remember tag lines

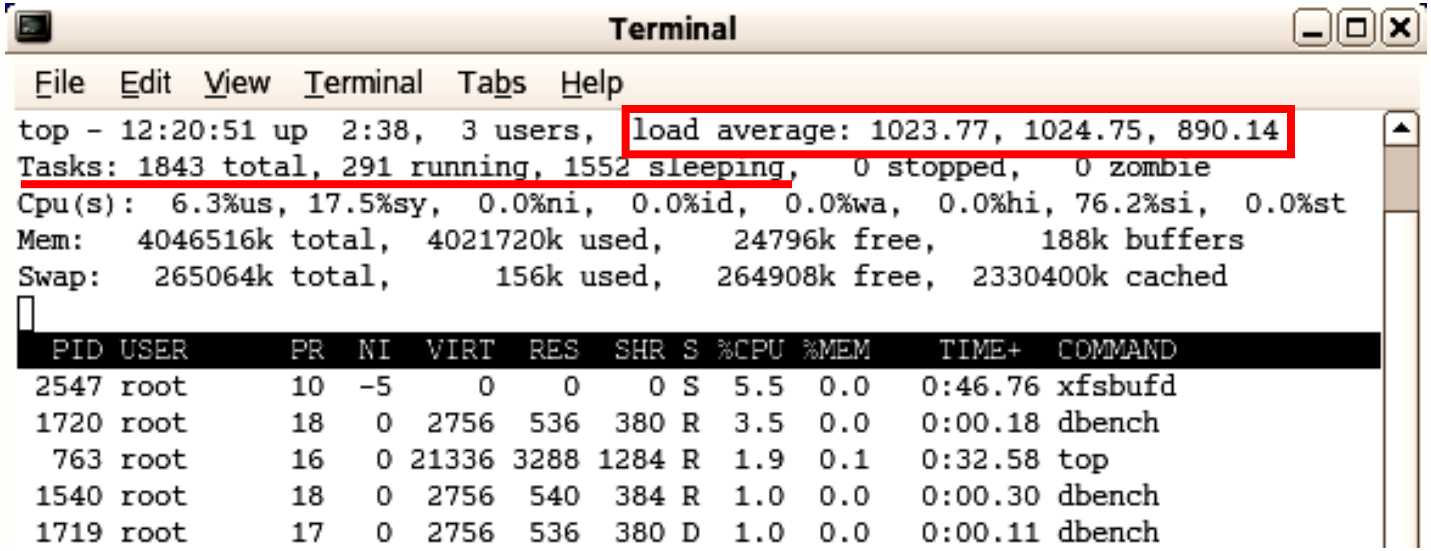
EXT3 and REISER are racing cars: go fast and hope for the best (stay in memory if at all possible)

XFS and NSS are Volvos: fasten safety belt, drive carefully (judicious use of memory and disk)

NSS salvage is a Volvo pulling a trailer full of spare parts

Corner case: gross overload

1000 dbench clients! It is rare to see load average of 1000+



1 CPU, 4GB memory, one virtual disk (local SATA), ESXi

Nearing a live-lock where system effort going round to each client for a bit of work is costing far more than the work

The system is happy, though prohibitively sluggish

NCP over NSS and POSIX

Generally speaking the NCP layer costs significant throughput reduction

Some editions of Novell Client for Linux have big problems, though Client32 does well

NCP performance can be improved significantly. I did the design for this, and we may see it sooner rather than later

MindWorks UK

MindWorks Inc. Ltd
210 Burnley Road
Weir
Bacup
OL13 8QE UK

Telephone: +44 (0) 170 687 1900

Fax: +44 (0) 170 687 8203

Web: www.mindworksuk.com

Email: training@mindworksuk.com