Morgan Stanley

OpenAFS

On Solaris 11 x86

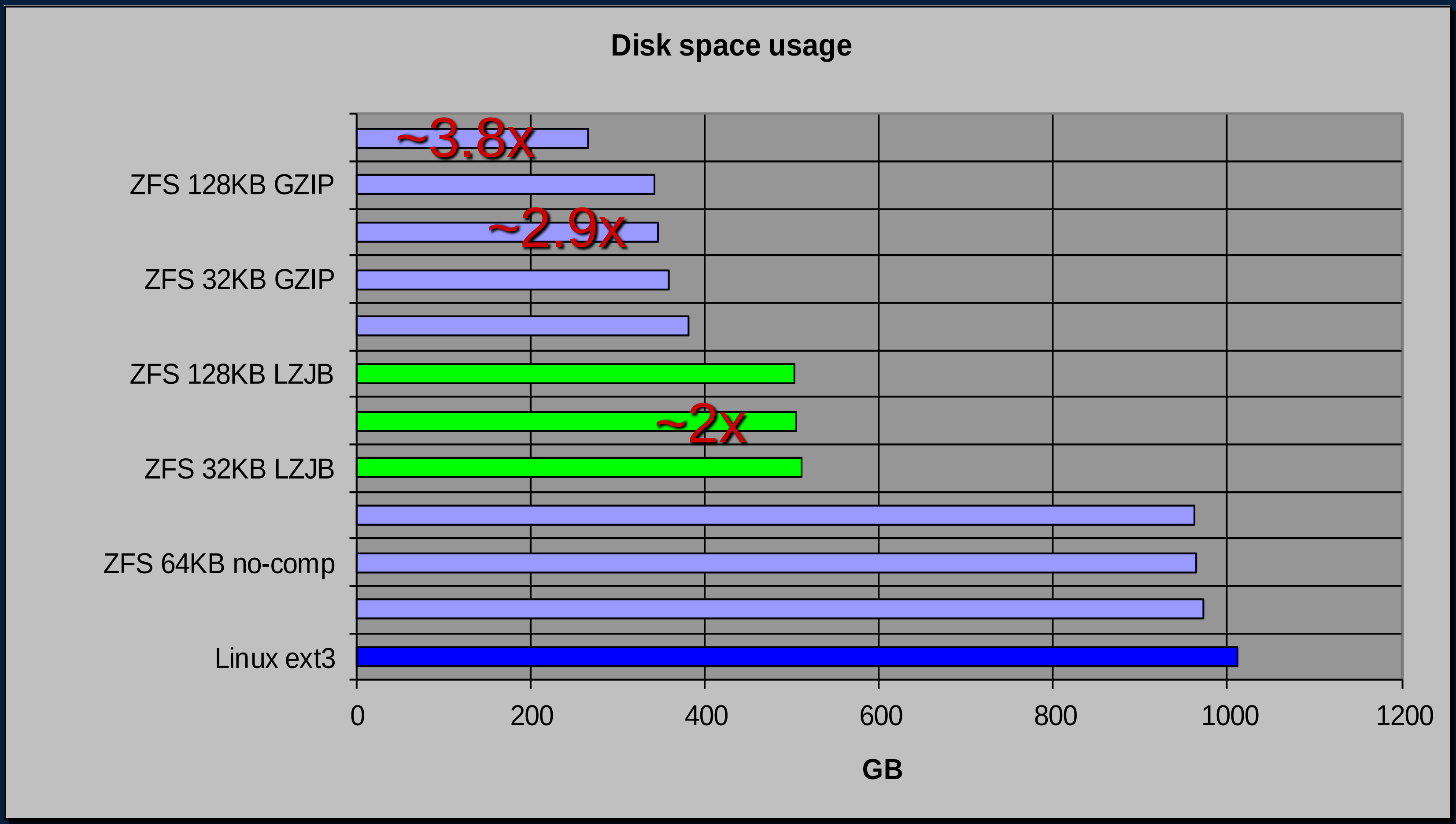
Robert Milkowski

Unix Engineering

Why Solaris?

- ZFS
  - Transparent and in-line data compression and deduplication
    - Big $$ savings
  - Transactional file system (no fsck)
  - End-to-end data and meta-data checksumming
  - Encryption
- DTrace
  - Online profiling and debugging of AFS
    - Many improvements to AFS performance and scalability
  - Safe to use in production

ZFS – Estimated Disk Space Savings

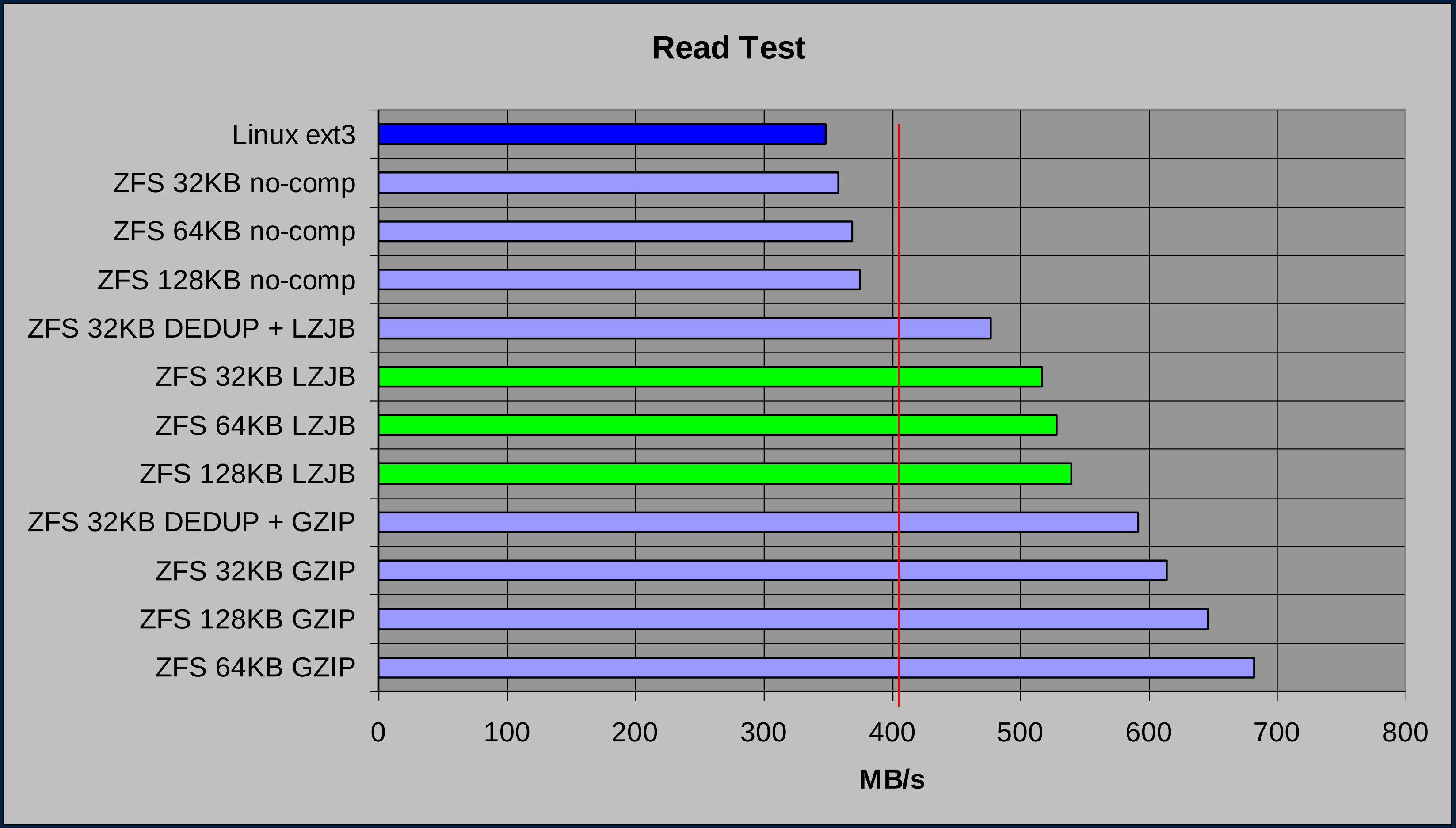


1TB sample of production data from AFS plant in 2010

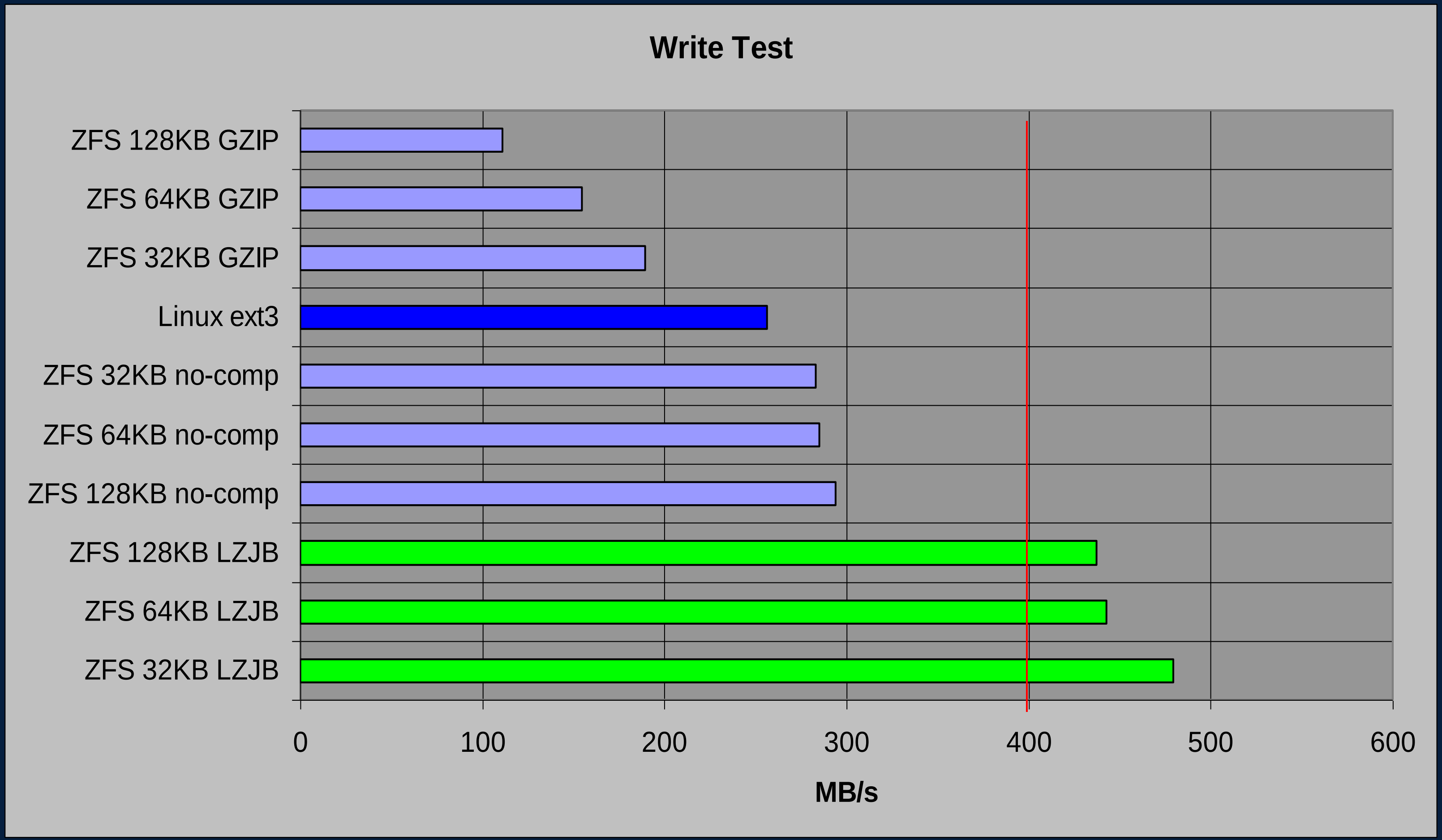
Currently, the overall average compression ratio for AFS on ZFS/gzip is over 3.2x

Morgan Stanley

Compression – Performance Impact



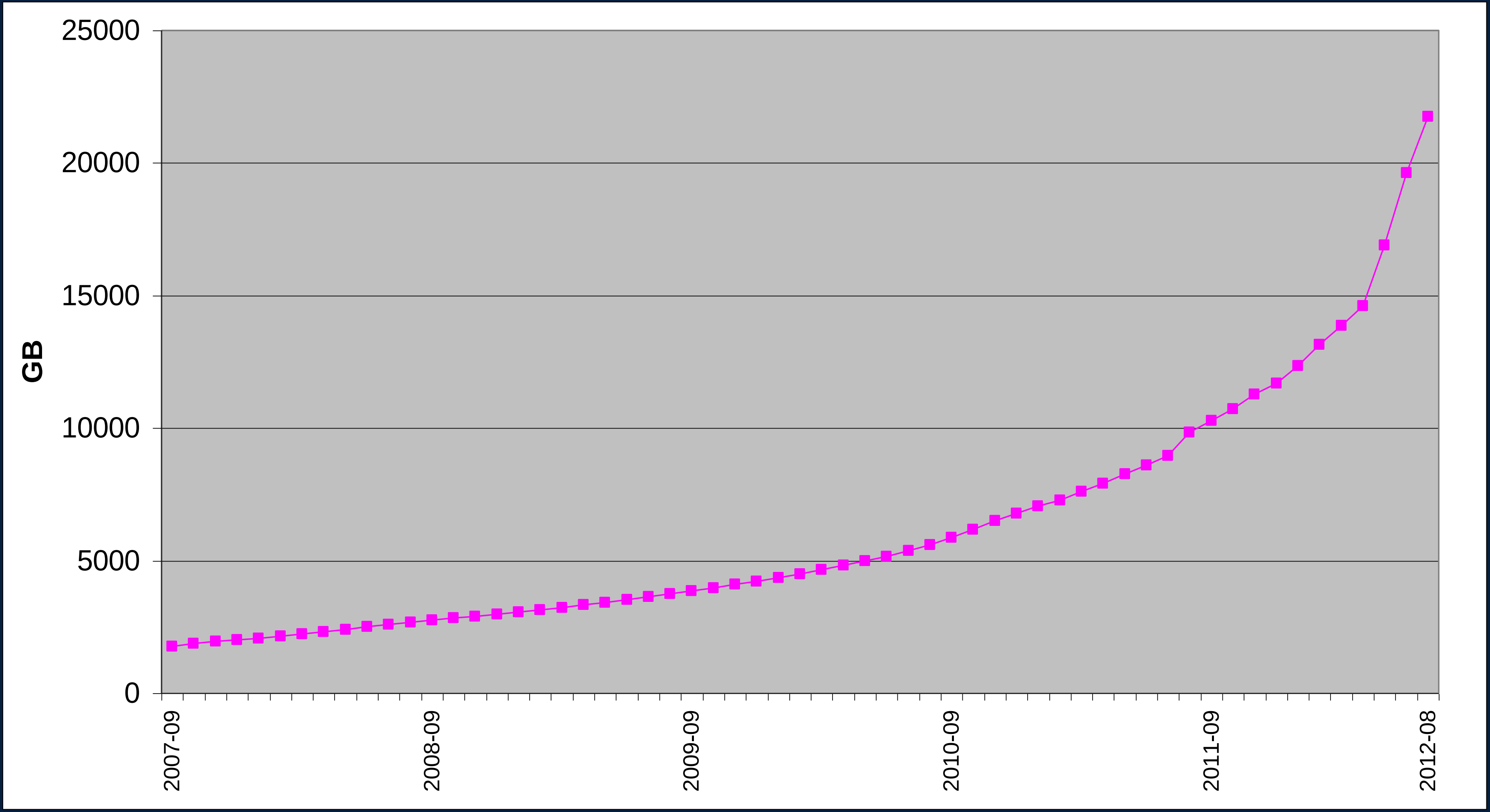
Compression – Performance Impact



Solaris – Cost Perspective

- Linux server
  - x86 hardware
  - Linux support (optional for some organizations)
  - Directly attached storage (10TB+ logical)
- Solaris server
  - The same x86 hardware as on Linux
  - 1,000$ per CPU socket per year for Solaris support (list price) on non-Oracle x86 server
  - Over 3x compression ratio on ZFS/GZIP
    - 3x fewer servers, disk arrays
    - 3x less rack space, power, cooling, maintenance ...

AFS Unique Disk Space Usage – last 5 years



Morgan Stanley

MS AFS High-Level Overview

- AFS RW Cells
  - Canonical data, not available in prod
- AFS RO Cells
  - Globally distributed
  - Data replicated from RW cells
  - In most cases each volume has 3 copies in each cell
  - ~80 RO cells world-wide, almost 600 file servers
- This means that a single AFS volume in a RW cell, when promoted to prod, is replicated ~240 times (80x3)
- Currently, there is over 3PB of storage presented to AFS

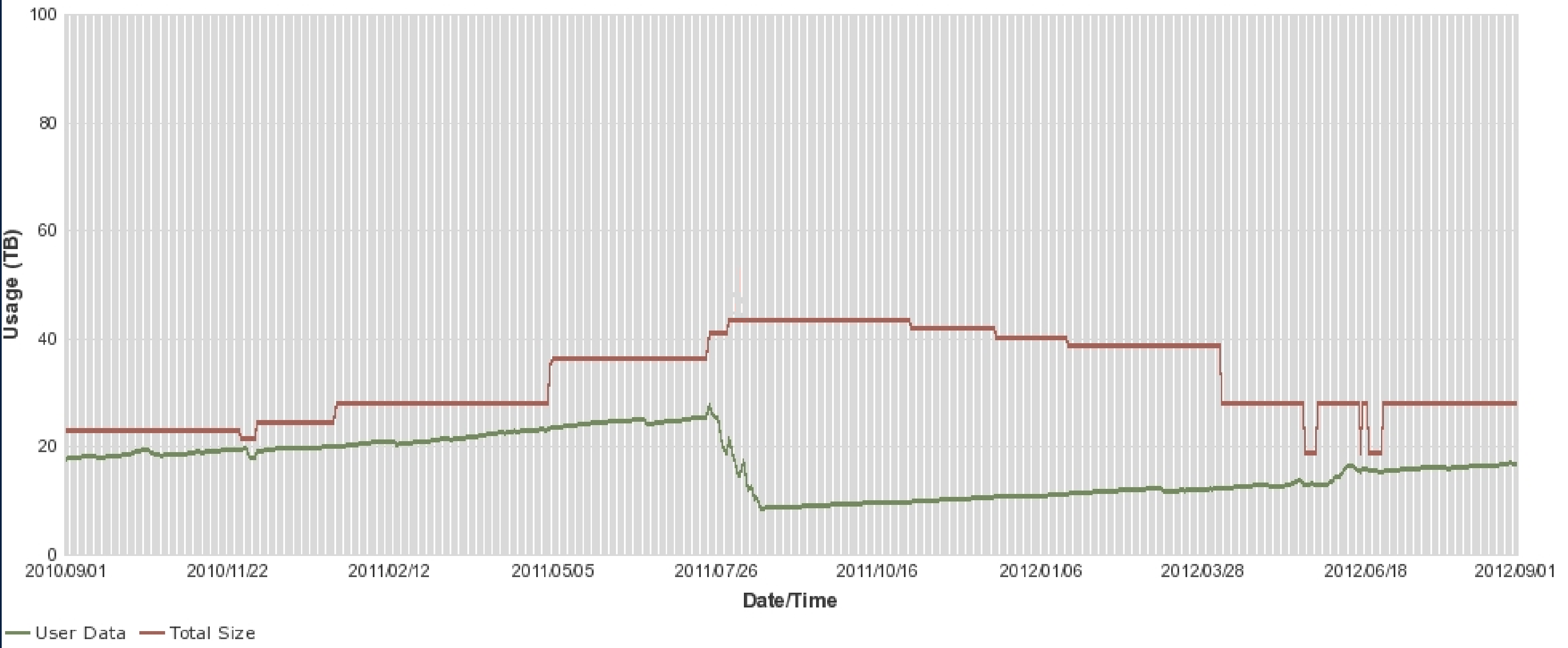
Typical AFS RO Cell

- Before
  - 5-15 x86 Linux servers, each with directly attached disk array, ~6-9RU per server
- Now
  - 4-8 x86 Solaris 11 servers, each with directly attached disk array, ~6-9RU per server
    - Significantly lower TCO
- Soon
  - 4-8 x86 Solaris 11 servers, internal disks only, 2RU
    - Lower TCA
    - Significantly lower TCO

Migration to ZFS

- Completely transparent migration to clients
  - Migrate all data away from a couple of servers in a cell
    - Rebuild them with Solaris 11 x86 with ZFS
      - Re-enable them and repeat with others
- Over 300 servers (+disk array) to decommission
  - Less rack space, power, cooling, maintenance... and yet more available disk space
- Fewer servers to buy due to increased capacity

q.ny cell migration to Solaris/ZFS



- Cell size reduced from 13 servers down to 3
- Disk space capacity expanded from ~44TB to ~90TB (logical)
- Rack space utilization went down from ~90U to 6U

Morgan Stanley

Solaris Tuning

- ZFS
  - Largest possible record size (128k on pre GA Solaris 11, 1MB on 11 GA and onwards)
  - Disable SCSI CACHE FLUSHES

    zfs:zfs_nocacheflush = 1
  - Increase DNLC size

    ncsize = 4000000
  - Disable access time updates on all vicep partitions
  - Multiple vicep partitions within a ZFS pool (AFS scalability)

Summary

- More than 3x disk space savings thanks to ZFS
  - Big $$ savings
- No performance regression compared to ext3
- No modifications required to AFS to take advantage of ZFS
- Several optimizations and bugs already fixed in AFS thanks to DTrace
- Better and easier monitoring and debugging of AFS
- Moving away from disk arrays in AFS RO cells

Why Internal Disks?

- Most expensive part of AFS is storage and rack space
- AFS on internal disks
  - 9U->2U
  - More local/branch AFS cells
  - How?
    - ZFS GZIP compression (3x)
    - 256GB RAM for cache (no SSD)
    - 24+ internal disk drives in 2U x86 server

HW Requirements

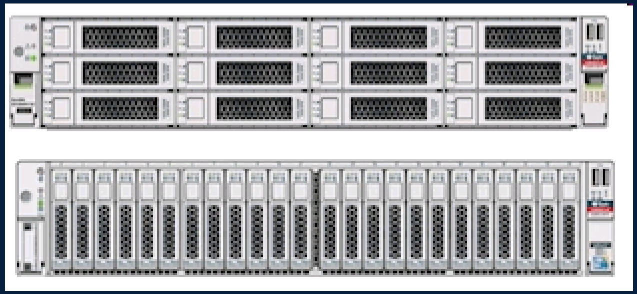
- RAID controller
  - Ideally pass-thru mode (JBOD)
  - RAID in ZFS (initially RAID-10)
  - No batteries (less FRUs)
  - Well tested driver
- 2U, 24+ hot-pluggable disks
  - Front disks for data, rear disks for OS
  - SAS disks, not SATA
- 2x CPU, 144GB+ of memory, 2x GbE (or 2x 10GbE)
- Redundant PSU, Fans, etc.

SW Requirements

- Disk replacement without having to log into OS
  - Physically remove a failed disk
  - Put a new disk in
  - Resynchronization should kick-in automatically
- Easy way to identify physical disks
  - Logical <-> physical disk mapping
  - Locate and Faulty LEDs
- RAID monitoring
- Monitoring of disk service times, soft and hard errors, etc.
  - Proactive and automatic hot-spare activation

Oracle/Sun X3-2L (x4270 M3)

- 2U
- 2x Intel Xeon E5-2600
- Up-to 512GB RAM (16x DIMM)
- 12x 3.5” disks + 2x 2.5” (rear)
- 24x 2.5” disks + 2x 2.5” (rear)
- 4x On-Board 10GbE
- 6x PCIe 3.0
- SAS/SATA JBOD mode



SSDs?

- ZIL (SLOG)
  - Not really necessary on RO servers
  - MS AFS releases >=1.4.11-3 do most writes as async
- L2ARC
  - Currently given 256GB RAM doesn’t seem necessary
  - Might be an option in the future
- Main storage on SSD
  - Too expensive for AFS RO
  - AFS RW?

Future Ideas

- ZFS Deduplication
- Additional compression algorithms
- More security features
  - Privileges
  - Zones
  - Signed binaries
- AFS RW on ZFS
- SSDs for data caching (ZFS L2ARC)
- SATA/Nearline disks (or SAS+SATA)

Morgan Stanley

Questions

DTrace

- Safe to use in production environments
- No modifications required to AFS
- No need for application restart
- 0 impact when not running
- Much easier and faster debugging and profiling of AFS
- OS/application wide profiling
  - What is generating I/O?
    - How does it correlate to source code?

DTrace – AFS Volume Removal

- OpenAFS 1.4.11 based tree
- 500k volumes in a single vicep partition
- Removing a single volume took ~15s

```
$ ptime vos remove -server haien15 -partition /vicepa -id test.76 -localauth
Volume 536874701 on partition /vicepa server haien15 deleted

real       14.197
user        0.002
sys         0.005
```

- It didn’t look like a CPU problem according to prstat(1M), although lots of system calls were being called

DTrace – AFS Volume Removal

- What system calls are being called during the volume removal?

```
haien15 $ dtrace -n syscall:::return'/pid==15496/{@[probefunc]=count();}'
dtrace: description 'syscall:::return' matched 233 probes
^C
[…]
  fxstat                                                           128
  getpid                                                          3960
  readv                                                           3960
  write                                                           3974
  llseek                                                          5317
  read                                                            6614
  fsat                                                            7822
  rmdir                                                           7822
  open64                                                          7924
  fcntl                                                           9148
  fstat64                                                         9149
  gtime                                                           9316
  getdents64                                                     15654
  close                                                          15745
  stat64                                                         17714
```

DTrace – AFS Volume Removal

- What are the return codes from all these rmdir()'s?

```
haien15 $ dtrace -n
            syscall::rmdir:return'/pid==15496/{@[probefunc,errno]=count();}'
dtrace: description 'syscall::rmdir:return' matched 1 probe
^C

  rmdir                                            2                 1
  rmdir                                            0                 4
  rmdir                                           17              7817
haien15 $

haien15 $ grep 17 /usr/include/sys/errno.h
#define EEXIST  17      /* File exists                          */
```

- Almost all rmdir()'s failed with EEXISTS

Morgan Stanley

DTrace – AFS Volume Removal

- Where are these rmdir()'s being called from?

```
$ dtrace -n syscall::rmdir:return'/pid==15496/{@[ustack()]=count();}'
^C
[...]
         libc.so.1`rmdir+0x7
         volserver_1.4.11-2`delTree+0x15f
         volserver_1.4.11-2`delTree+0x15f
         volserver_1.4.11-2`delTree+0x15f
         volserver_1.4.11-2`namei_RemoveDataDirectories+0x5a
         volserver_1.4.11-2`namei_dec+0x312
         volserver_1.4.11-2`PurgeHeader_r+0x87
         volserver_1.4.11-2`VPurgeVolume+0x72
         volserver_1.4.11-2`VolDeleteVolume+0x9a
         volserver_1.4.11-2`SAFSVolDeleteVolume+0x14
         volserver_1.4.11-2`_AFSVolDeleteVolume+0x2f
         volserver_1.4.11-2`AFSVolExecuteRequest+0x363
         volserver_1.4.11-2`rxi_ServerProc+0xdc
         volserver_1.4.11-2`rx_ServerProc+0xba
         volserver_1.4.11-2`server_entry+0x9
         libc.so.1`_thr_setup+0x4e
         libc.so.1`_lwp_start 1954
          1954
```

DTrace – AFS Volume Removal

- After some more dtrace’ing and looking at the code, this are the functions being called for a volume removal:

VolDeleteVolume() -> VPurgeVolume() -> PurgeHeader_r() -> IH_DEC/namei_dec()

- How long each function takes to run in seconds

```
$ dtrace -F -n pid15496::VolDeleteVolume:entry, \
               pid15496::VPurgeVolume:entry, \
               pid15496::PurgeHeader_r:entry, \
               pid15496::namei_dec:entry, \
               pid15496::namei_RemoveDataDirectories:entry \
                 '{t[probefunc]=timestamp; trace("in");}' \

         -n pid15496::VolDeleteVolume:return, \
            pid15496::VPurgeVolume:return, \
            pid15496::PurgeHeader_r:return, \
            pid15496::namei_dec:return, \
            pid15496::namei_RemoveDataDirectories:return \
            '/t[probefunc]/ \
             {trace((timestamp-t[probefunc])/1000000000); t[probefunc]=0;}'
```

DTrace – AFS Volume Removal

```
CPU FUNCTION
  0  -> VolDeleteVolume                         in
  0    -> VPurgeVolume                          in
  0      -> namei_dec                           in
  0      <- namei_dec                                         0
  0      -> PurgeHeader_r                       in
  0        -> namei_dec                         in
  0        <- namei_dec                                       0
...
  0      <- PurgeHeader_r                                     0
  0    <- VPurgeVolume                                        0
  0  <- VolDeleteVolume                                       0
  0  -> VolDeleteVolume                         in
  0    -> VPurgeVolume                          in
  0      -> namei_dec                           in
  0      <- namei_dec                                         0
  0      -> PurgeHeader_r                       in
  0        -> namei_dec                         in
  0        <- namei_dec                                       0
...
  0          -> namei_RemoveDataDirectories     in
  0          <- namei_RemoveDataDirectories                  12
  0        <- namei_dec                                      12
  0      <- PurgeHeader_r                                    12
  0    <- VPurgeVolume                                       12
  0  <- VolDeleteVolume                                      12
^C
```

Morgan Stanley

DTrace – AFS Volume Removal

- Lets print arguments (strings) passed to delTree()

```
$ dtrace -q -n pid15496::VolDeleteVolume:entry'{self->in=1;}' \n
            -n pid15496::delTree:entry \
  -n '/self->in/{self->in=0;trace(copyinstr(arg0));trace(copyinstr(arg1));}'

/vicepa/AFSIDat/+/+w++U/special/zzzzP+k1++0  +/+w++U/special/zzzzP+k1++0
```

- delTree() will try to remove all dirs under /vicepa/AFSIdat/+
  - But there are many other volumes there – directories full of files, so rmdir() fails on them
- After this was fixed - http://gerrit.openafs.org/2651
  - It takes <<1s to remove the volume (~15s before)
  - It only takes 5 rmdir()'s now (~8k before)

DTrace – Accessing Application Structures

```
[…]
typedef struct Volume {
    struct rx_queue q;          /* Volume hash chain pointers */
    VolumeId hashid;            /* Volume number -- for hash table lookup */
    void *header;               /* Cached disk data - FAKED TYPE */
    Device device;              /* Unix device for the volume */
    struct DiskPartition64
     *partition;                /* Information about the Unix partition */

}; /* it is not the entire structure! */


pid$1:a.out:FetchData_RXStyle:entry
{
  self->fetchdata = 1;
  this->volume = (struct Volume *)copyin(arg0, sizeof(struct Volume));
  this->partition = (struct DiskPartition64 *)copyin((uintptr_t) \
                   this->volume->partition, sizeof(struct DiskPartition64));
  self->volumeid = this->volume->hashid;
  self->partition_name = copyinstr((uintptr_t)this->partition->name);
}
[…]
```

Morgan Stanley

volume_top.d

```
Mountpoint                    VolID      Read[MB] Wrote[MB]
================ ============= ========= =========
/vicepa               542579958        100        10
/vicepa               536904476          0        24
/vicepb               536874428          0         0
                 ============= ========= =========
                                       100        34

   started: 2010 Nov  8 16:16:01
   current: 2010 Nov  8 16:25:46
```

rx_clients.d

```
CLIENT IP           CONN  CONN/s MKFILE RMFILE  MKDIR  RMDIR RENAME LOOKUP   LINK SYMLNK SSTORE DSTORE
=============== ====== ====== ====== ====== ====== ====== ====== ====== ====== ====== ====== ======
172.24.40.236     6009    133    234    702    234    234      0      0    234    235    235      0
172.24.3.188       178      3      0      1      0      0      0      0      0      0      3      0
172.24.41.86         2      0      0      0      0      0      0      0      0      0      0      0
10.172.170.236       2      0      0      0      0      0      0      0      0      0      0      0
                ====== ====== ====== ====== ====== ====== ====== ====== ====== ====== ====== ======
                  6191    137    234    703    234    234      0      0    234    235    238      0


   started: 2010 Nov  8 13:13:16
   current: 2010 Nov  8 13:14:01


        SSTORE = Store Status
        DSTORE = Store Data
```

vm_top.d

VM NAME	TOTAL IO READ	TOTAL IO WRITE	TOTAL MB READ	TOTAL MB WRITE	AVG [KB] READ	AVG [KB] WRITE	AVG [ms] READ	AVG [ms] WRITE	MAX [ms] READ	MAX [ms] WRITE
evm8223	27499	3260	757	31	28	9	0	0	9	43
evm8226	20767	3475	763	34	37	10	0	0	15	162
evm8228	27283	3431	737	34	27	10	0	0	24	40
evm8242	33927	3448	536	24	16	7	0	0	16	39
evm8244	27155	3371	744	33	28	10	0	0	110	31
evm8247	33743	3223	535	24	16	7	0	0	30	75
evm8252	33816	3133	541	31	16	10	0	0	27	67
evm8257	16787	3432	557	31	33	9	0	0	1	0
evm8258	27144	3352	742	33	28	10	0	0	26	57
evm8259	27017	3469	748	36	28	10	0	0	30	95
evm8263	33446	3076	532	23	16	7	0	0	15	37
evm8264	27155	3461	743	33	28	10	0	0	16	28
============	=======	======	=====	=====	====	=====	====	=====	====	=====
totals	**335739**	**40131**	**7939**	**373**	**24**	**9**	**0**	**0**	**110**	**162**

Morgan Stanley

Questions

DTrace – Attaching AFS Volumes

- OpenAFS 1.4.11 based tree
- 500k volumes in a single vicep partition
- Takes ~118s to pre-attached them
  - All metadata cached in memory, 100% dnlc hit, no physical i/o
- A single thread spends 99% on CPU (USR) during pre-attachment
- Another thread consumes 99% CPU as well (36% USR, 64% SYS)

```
haien15 $ prstat -Lm -p `pgrep fileserver`

   PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
  7434 root      36  64 0.0 0.0 0.0 0.0 0.0 0.0 .3M   1 .3M   0 fileserver_1/6
  7434 root      99 1.3 0.0 0.0 0.0 0.0 0.0 0.0   0   2 270   0 fileserver_1/8
  7434 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 fileserver_1/5
  7434 root     0.0 0.0 0.0 0.0 0.0 100 0.0 0.0   0   0   0   0 fileserver_1/4
[...]
```

DTrace – Attaching AFS Volumes, tid=6

```
haien15 $ dtrace -n profile-997'/execname=="fileserver_1.4.1" && tid==6/
                                {@[ustack()]=count();}'
                 -n tick-10s'{trunc(@,5);printa(@);exit(0);}'


[…]

libc.so.1`lwp_yield+0x7
  fileserver_1.4.11-2`FSYNC_sync+0x87
  libc.so.1`_thr_setup+0x4e
  libc.so.1`_lwp_start
  9432


vol/fssync.c:
  354 while (!VInit) {
  355 /* Let somebody else run until level > 0. That doesn't mean that
  356 * all volumes have been attached. */
  357 #ifdef AFS_PTHREAD_ENV
  358 pthread_yield();
  359 #else /* AFS_PTHREAD_ENV */
  360 LWP_DispatchProcess();
  361 #endif /* AFS_PTHREAD_ENV */
  362 }
```

DTrace – Attaching AFS Volumes, tid=6

- FSSYNC is the mechanism by which different processes communicate with fileserver
- There is a dedicated thread to handle all requests
- It "waits" for a fileserver to pre-attach all volumes by calling pthread_yield() in a loop
  - This saturates a single CPU/core
  - Might or might not impact start-up time depending on a number of CPUs and other threads requiring them, in this test case it doesn't contribute to the start-up time
- FIX: introduce a CV
  - CPU utilization by the thread drops down from 100% to 0%

DTrace – Attaching AFS Volumes, tid=8

- It must be the 2nd thread (tid=8) responsible for the long start up time

```
haien15 $ prstat -Lm -p `pgrep fileserver`

   PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
  7434 root      36  64 0.0 0.0 0.0 0.0 0.0 0.0 .3M   1 .3M   0 fileserver_1/6
  7434 root      99 1.3 0.0 0.0 0.0 0.0 0.0 0.0   0   2 270   0 fileserver_1/8
  7434 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 fileserver_1/5
  7434 root     0.0 0.0 0.0 0.0 0.0 100 0.0 0.0   0   0   0   0 fileserver_1/4
[...]
```

DTrace – Attaching AFS Volumes, tid=8

```
haien15 $ dtrace -n profile-997'/execname=="fileserver_1.4.1" && tid==8/
                                {@[ustack()]=count();}'
                 -n tick-10s   '{trunc(@,3);printa(@);exit(0);}'


[…]

              fileserver_1.4.11-2`VLookupVolume_r+0x83
                fileserver_1.4.11-2`VPreAttachVolumeById_r+0x3e
                fileserver_1.4.11-2`VPreAttachVolumeByName_r+0x1d
                fileserver_1.4.11-2`VPreAttachVolumeByName+0x29
                fileserver_1.4.11-2`VAttachVolumesByPartition+0x99
                fileserver_1.4.11-2`VInitVolumePackageThread+0x75
                libc.so.1`_thr_setup+0x4e
                libc.so.1`_lwp_start
               8360
```

Morgan Stanley

DTrace – Attaching AFS Volumes, tid=8

```
$ dtrace -F -n pid`pgrep fileserver`::VInitVolumePackageThread:entry'{self->in=1;}'
      -n pid`pgrep fileserver`::VInitVolumePackageThread:return'/self->in/{self->in=0;}'
      -n pid`pgrep fileserver`:::entry,pid`pgrep fileserver`:::return
         '/self->in/{trace(timestamp);}'

CPU FUNCTION
  6  -> VInitVolumePackageThread                    8565442540667
  6    -> VAttachVolumesByPartition                 8565442563362
  6      -> Log                                     8565442566083
  6        -> vFSLog                                8565442568606
  6          -> afs_vsnprintf                       8565442578362
  6          <- afs_vsnprintf                       8565442582386
  6        <- vFSLog                                8565442613943
  6      <- Log                                     8565442616100
  6      -> VPartitionPath                          8565442618290
  6      <- VPartitionPath                          8565442620495
  6      -> VPreAttachVolumeByName                  8565443271129
  6        -> VPreAttachVolumeByName_r              8565443273370
  6          -> VolumeNumber                        8565443276169
  6          <- VolumeNumber                        8565443278965
  6          -> VPreAttachVolumeById_r              8565443280429
  6            <- VPreAttachVolumeByVp_r            8565443331970
  6          <- VPreAttachVolumeById_r              8565443334190
  6        <- VPreAttachVolumeByName_r              8565443335936
  6      <- VPreAttachVolumeByName                  8565443337337
  6      -> VPreAttachVolumeByName                  8565443338636
             [... VPreAttachVolumeByName() is called many times here in a loop]

                        [ some output was removed ]
```

DTrace – Attaching AFS Volumes, tid=8

```
$ dtrace -n pid`pgrep fileserver`::VPreAttachVolumeByName:entry'{@=count();}' \
         -n tick-1s'{printa("%@d\n",@);clear(@);}' -q
  26929
  20184
  16938
  14724
  13268
  12193
  11340
  10569
  10088
  9569
  8489
  8541
  8461
  8199
  7941
  7680
  7480
  7251
  6994
  ^C
```

When traced from the very beginning to the end the number of volumes being pre-attached goes down from ~50k/s to ~3k/s

DTrace – Frequency Distributions

```
$ dtrace -n pid`pgrep fileserver`::VPreAttachVolumeByName:entry'
                {self->t=timestamp;}'
         -n pid`pgrep fileserver`::VPreAttachVolumeByName:return'/self->t/
                {@=quantize(timestamp-self->t);self->t=0;}'
         -n tick-20s'{printa(@);}'
[…]
  2  69837                        :tick-20s

           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |                                         83
            2048 |@@                                       21676
            4096 |@                                        17964
            8192 |@@                                       19349
           16384 |@@@                                      32472
           32768 |@@@@@                                    60554
           65536 |@@@@@@@@@                                116909
          131072 |@@@@@@@@@@@@@@@@@@@                      237832
          262144 |                                         4084
          524288 |                                         393
         1048576 |                                         0
```

DTrace – Attaching AFS Volumes, tid=8

```
haien15 $ ./ufunc-profile.d `pgrep fileserver`

[…]

  VPreAttachVolumeByName_r                                    4765974567
  VHashWait_r                                                 4939207708
  VPreAttachVolumeById_r                                      6212052319
  VPreAttachVolumeByVp_r                                      8716234188
  VLookupVolume_r                                            68637111519
  VAttachVolumesByPartition                                 118959474426
```

It took 118s to pre-attach all volumes. Out of the 118s fileserver spent 68s in VLookupVolume_r(), the next function is only 8s. By optimizing the VLookupVolume_r() we should get the best benefit. By looking at source code of the function it wasn't immediately obvious which part of it is responsible…

DTrace – Attaching AFS Volumes, tid=8

- Lets count each assembly instruction in the function during the pre-attach

```
$ dtrace -n pid`pgrep fileserver`::VLookupVolume_r:'{@[probename]=count();}'
         -n tick-5s'{printa(@);}'
[…]
  e                                                                108908
  entry                                                            108908
  91                                                             11459739
  7e                                                             11568134
  80                                                             11568134
  83                                                             11568134
  85                                                             11568134
  87                                                             11568134
  89                                                             11568134
  8b                                                             11568134
  77                                                             11568135
  78                                                             11568135
  7b                                                             11568135
```

DTrace – Attaching AFS Volumes, tid=8

- The corresponding disassembly and source code

```
VLookupVolume_r+0x77:           incl    %ecx
VLookupVolume_r+0x78:           movl    0x8(%esi),%eax
VLookupVolume_r+0x7b:           cmpl    -0x1c(%ebp),%eax
VLookupVolume_r+0x7e:           je      +0x15    <VLookupVolume_r+0x93>
VLookupVolume_r+0x80:           movl    0x4(%edx),%eax
VLookupVolume_r+0x83:           movl    %edx,%edi
VLookupVolume_r+0x85:           movl    %edx,%esi
VLookupVolume_r+0x87:           movl    %eax,%edx
VLookupVolume_r+0x89:           cmpl    %edi,%ebx
VLookupVolume_r+0x8b:           je      +0xa2    <VLookupVolume_r+0x12d>
VLookupVolume_r+0x91:           jmp     -0x1a    <VLookupVolume_r+0x77>
```

```
/* search the chain for this volume id */
for(queue_Scan(head, vp, np, Volume)) {
    looks++;
    if ((vp->hashid == volumeId)) {
        break;
    }
}
```

DTrace – Attaching AFS Volumes, tid=8

- Larger hash size should help
- Hash size can be tuned by -vhashsize option
  - Fileserver supports only values between <6-14>
  - It **silently** set it to 8 if outside of the range
  - We had it set to 16... (only in dev)
  - Fixed in upstream
    - Over 20x reduction in start up time

DTrace – Attaching AFS Volumes, Multiple Partitions

- Two AFS partitions
- 900k empty volumes (400k + 500k)
- How well AFS scales when restarted?
  - One thread per partition pre-attaches volumes
  - All data is cached in-memory, no physical i/o
- Each thread consumes 50-60% of CPU (USR) and spends about 40% of its time in user locking
  - But with a single partition the thread was able to utilize 100% CPU

```
haien15 $ prstat -Lm -p `pgrep fileserver`
[...]
 PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
7595 root      54 4.3 0.0 0.0 0.0  40 0.0 1.6 18K  17 37K   0 fileserver_1/8
7595 root      54 4.2 0.0 0.0 0.0  40 0.0 1.7 18K  23 37K   0 fileserver_1/7
7595 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   8   0   4   0 fileserver_1/6
[...]
```

DTrace – Attaching AFS Volumes, Locking

```
$ prstat -Lm -p `pgrep fileserver`

     PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
    7595 root      54 4.3 0.0 0.0 0.0  40 0.0 1.6 18K  17 37K   0 fileserver_1/8
    7595 root      54 4.2 0.0 0.0 0.0  40 0.0 1.7 18K  23 37K   0 fileserver_1/7
    7595 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   8   0   4   0 fileserver_1/6
  [...]
```

```
$ plockstat -vA -e 30 -p `pgrep fileserver`
  plockstat: tracing enabled for pid 7595
  Mutex block

   Count     nsec Lock                         Caller
  -------------------------------------------------------------------------------
  183494   139494 fileserver`vol_glock_mutex fileserver`VPreAttachVolumeByVp_r+0x125
    6283   128519 fileserver`vol_glock_mutex fileserver`VPreAttachVolumeByName+0x11
```

139494ns * 183494 = ~25s

30s for each thread, about 40% time in LCK is 60s *0.4 = 24s

- plockstat utility uses DTrace underneath
  - It has an option to print a dtrace program to execute

DTrace – Attaching AFS Volumes, Locking

```
vol/volume.c
        /* if we dropped the lock, reacquire the lock,
         * check for pre-attach races, and then add
         * the volume to the hash table */
        if (nvp) {
            VOL_LOCK;
            nvp = VLookupVolume_r(ec, vid, NULL);
```

- For each volume being pre-attached a global lock is required
- It gets worse if more partitions are involved
- FIX: pre-allocate structures and add volumes in batches

DTrace – Attaching AFS Volumes

- Fixes (all in upstream)
  - Introduce CV for FSYNC thread during initialization
  - Allow for larger hash sizes
    - Increase the default value
    - Fileserver warns about out of range value
  - Pre-attach volumes in batches rather than one at a time
- For 1.5mln volumes distributed across three vicep partitions
  - All data is cached in memory, no physical i/o
  - Before the above fixes it took **~10 minutes** to pre-attach
  - With the fixes it takes less **than 10s**
  - This is over 60x improvement (better yet for more volumes)