Symantec VxFS 5.x releases df command expectations

March, 2013

Document version 1.1

The df –k command fudge factor

The df command tells lies, or to be more accurate VxFS tells lies when the df command asks for information. What we explain below is what information VxFS provides to the df –k command and why.

VxFS is able to dynamlically allocate its metadata on demand. All VxFS metadata is held in (hidden) files, therefore VxFS can simply grow these files whenever required by allocating new extents. This is an entirely different concept to conventional file system types. Conventional file systems pre-allocate their metadata, with the largest amount of space pre-allocated commonly being for ondisk inodes. Therefore after creating a new file system and mounting it for the first time the amount of free space available is reduced by the space that mkfs used for pre-allocated metadata, but as VxFS does not pre-allocate its inodes the amount of available free space will be higher for VxFS compared to conventional file system types.

This difference in design means VxFS can appear to have more free space available in a file system to hold file data, when in fact VxFS has yet to allocate space for inode metadata. To avoid confusion VxFS 'estimates' the number of inodes that might be required in a file system and reduces the count of free blocks reported by the df command accordingly.

Below we walk through an example to explain how VxFS estimates this information –

```
# /opt/VRTS/bin/mkfs /dev/vx/rdsk/testdg/vol3
    version 7 layout
    62914560 sectors, 31457280 blocks of size 1024, log size 65536 blocks
    largefiles supported
# mount -V vxfs /dev/vx/dsk/testdg/vol3 /mnt3
```

The following fsadm output shows that the total number of free blocks currently in the file system is 31382926 blocks:

```
# /opt/VRTS/bin/fsadm -E /mnt3 | head -4
  Extent Fragmentation Report
        Total    Average      Average     Total
        Files    File Blks    # Extents   Free Blks
            0           0           0     31382926
```

However the following output from the df -k command shows the total number of free blocks to currently be 29421501 blocks:

```
# /opt/VRTS/bin/df -k /mnt3
Filesystem             kbytes    used     avail    capacity  mounted on
/dev/vx/dsk/testdg/vol3 31457280 2035779  29421501  6%       /mnt3
```

The difference is:

31382926 (actual free block count) – 29421501 (df -k output) = 1961425 fs blocks

We therefore want to understand how VxFS estimated that this many blocks will be required for dymaically allocated inodes.

Using the fstyp command we can see the fs block size (bsize) and the total number of fs blocks in the file system (size) –

```
# /opt/VRTS/bin/fstyp -v /dev/vx/rdsk/testdg/vol3 | grep ^bsize
bsize  1024 size  31457280 dsize  31457280  ninode 0  nau 0
```

Using the fstyp command we can also calculate the size of an inode ondisk by greping for the 'inodes per block' count (inopb) –

```
# /opt/VRTS/bin/fstyp -v /dev/vx/rdsk/testdg-cpeaixs06/vol3 | grep ^inopb
inopb 4   inopau 0   ndiripau 0   iaddrlen 8    bshift 10
```

The inopb (inodes per block) is 4 and the fs block size is 1024bytes, which gives inode size as 1024/4 = 256byte inode size.

VxFS allocates inodes dynamically. As a general "rule of thumb" we choose to estimate that one quarter of the current number of free blocks will be needed for future dynamic allocations of inodes. VxFS therefore adjusts (reduces) the number of free blocks reported by the df command as follows:

```
1. Total number of free blocks           = 31382926

2. Total number of free inodes to be dynamically allocated
   as per the total number of free blocks to total number
   of free inodes ratio of 4:1           = 31382926 / 4
                                         = 7845731


3. Total number of free inodes currently in the
   file system                           = 28   (obtained from the IAS in fsdb)


4. Total number of free inodes = total number of inodes that can be allocated
   as per the current number of free blocks in the file system ( (2) above)
   minus the total number of free inodes already present in the file system
   ( (3) above )
                                         = 7845731 - 28
                                         = 7845703


5. Number of inodes per block is 4 from the above fstyp output (inopb)

6. Round down total number of free inodes on the boundary of 4 as
   4 inodes can fit in one FS blocks     = 7845703 & ~( 4 - 1)
                                         = 7845700

7. Number of blocks needed to accomodate total number of free inodes
   from (6) above                        = 7845700 / 4 = 1961425 FS blocks.


8. Reducing the number of blocks needed to accomodate free inodes (7) from the
total
   number of free inodes (1)             = 31382926 - 1961425
                                         = 29421501
```

Which matches with the count of available blocks (29421501) in the df –k command output above.

The df –i inode counters (when not using CFS)

By default mkfs of a VxFS file system will only create 32 inodes, numbered 0-31 inc.

However inode-0 and inode-1 are never utilized but they are marked as allocated, inode-2 is always the root inode and inode-3 is the lost+found directory

```
# mkfs -t vxfs /dev/vx/dsk/dg/vol
# mount -t vxfs /dev/vx/dsk/dg/vol /mnt
# cd /mnt
# touch file1

# ls -li /mnt
4 -rw-r--r--. 1 root root  0 Feb 26 04:55 file1
3 drwxr-xr-x. 2 root root 96 Feb 26 04:46 lost+found


# df -i /mnt
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/vx/dsk/dg/vol
                     7845180       5 7845176    1% /mnt
```

Above we currently have a correct inode used count (IUsed) of 5, for inodes 0/1/2/3/4

Inode-0 is never utilized, but marked as allocated
Inode-1 is also never utilized, but marked as allocated
Inode-2 is the root inode
Inode-3 is the lost+found directory
Inode-4 is the file we created above called "file1"

Therefore, the used inode count of 5, seen under "IUsed", is clearly correct.

However once again we see a fudge-factor count of 7845180 total inodes in the file system, under "Inodes", whereas in truth there are still only 32 inodes in this file system that were created at mkfs time.

Also for "IFree" we see a count of 7845176 inodes, which of course is another fudge-factor counter. Rounding has likely lead to the difference betwwen "Inodes" and "IFree" being 4 rather than 5.

In summary, the used inode count is correct. The total number of inodes and the number of free inodes are estimated using the number of free blocks and the number of inodes that genuinely exist.

The Inode Allocation Unit, and IAU delegations in CFS

VxFS manages its inodes ondisk by splitting them up into inode allocation units, IAU.

The number of inodes represented in an IAU depends upon the file system block size chosen at mkfs time.

```
Using a VxFS 1Kb file system block size each IAU will represent  8192 inodes.
Using a VxFS 2Kb file system block size each IAU will represent 16384 inodes.
Using a VxFS 4Kb file system block size each IAU will represent 32768 inodes.
Using a VxFS 8Kb file system block size each IAU will represent 65536 inodes.
```

Each IAU is also numbered, starting at zero, to identify the inode numbers it represents.

```
Using a 1Kb block size IAU-0 will represent inodes numbers        0 -  8191
Using a 1Kb block size IAU-1 will represent inodes numbers     8192 - 16383
Using a 1Kb block size IAU-2 will represent inodes numbers    16384 - 32768
etc..

Using a 8Kb block size IAU-0 will represent inodes numbers       0 -  65535
Using a 8Kb block size IAU-1 will represent inodes numbers   65536 - 131071
Using a 1Kb block size IAU-2 will represent inodes numbers  131072 - 196607
etc..
```

Each IAU has its own ‘summary information’ to keep count of the number of allocated inodes and number of free inodes available in the IAU.

In Cluster File System release 5.0 we introduced a concept of delegating metadata to each node in the cluster, this delegation of metadata allows CFS secondary nodes to update metadata on the local node without having to ask the CFS primary to do it – thus providing far greater node scalability. Logically, the delegation of metadata also protects us from two nodes updating the same metadata at the same time.

For ondisk inodes CFS will delegate an entire IAU to a node as required. For example if node1 has mounted the file system as a CFS Secondary and node1 wishes to create a new file, then the CFS primary node will delegate perhaps IAU-2 to node1, node1 can than allocate new inodes from IAU-2 – once all the inodes in IAU-2 are allocated node1 can request another IAU delegation from the CFS Primary.

Likewise, when deleting a file the IAU delegation for the IAU in which the file’s inode resides must be delegated to the node that is deleting the file. Thus IAU delgations are most commonly requested when allocating and freeing inodes.

Once an IAU delegation has been delegated to a node it can “timeout” if it is not forcibly revoked via a request from another node, timeouts occur after 180 seconds of inactivity on the IAU. A timeout will revoke the IAU delegation thus making its ‘summary information’ visible to the CFS primary.

However one IAU delegation will purposely not timeout on each node, thus one IAU will usually remain delegated to each node, this is for performance reasons.

It is also true that delegations for IAUs containing less than 64 free inodes will not timeout due to inactivity either, as it was thought better for the same node to allocate the last few remaining inodes in an IAU.

Most importantly, however, we have since made a fix so that IAU delegations will now always timeout if the IAU has no free inodes remaining at all. This simple and safe change will mean the df –i used inode count will now be far more accurate when using cluster mounted file systems because nearly all IAU delegations will now timeout after 180 seconds of inactivity.

In summary: IAU delegations will only be requested by a node when needed, IAU delegations can also timeout, however one IAU delegation on each node will not timeout such that each node will commonly be left with one IAU delegation each.

The df –i used inode counter (when using CFS)

How does it work using CFS?

- The current used inode count information provided by the 'df –i' command is always requested from (and collected by) the CFS primary.
- The 'df –i' command therefore displays the same information from any node in the cluster.

What information is collected by the CFS primary?

- Each IAU has its own 'summary information' to keep count of the number of allocated inodes and number of free inodes available in the IAU.
- A sum of all the IAU counts (visible to the CFS Primary) is made to produce a total count for the 'df –i' command display.

Why is the df –i used-inode-count often inaccurate when using CFS?

- As mentoned in the previous section in this document, an IAU must first be delegated to a node before it can be used.
- The 'df –i' command used inode count can be inaccurate because the CFS primary cannot see the correct IAU summary information for IAUs that are currently delegated to CFS secondary nodes.

What inaccuracy in the df –i used-inode-count should we expect using CFS?

- The 'df –i' command used inode count can be inaccurate because the CFS primary cannot see information for IAUs that are delegated to secondary nodes.
- The df-i used inode count inaccuracy will therefore depend upon how many IAUs are currently delegated to CFS secondary nodes.
- Once updates to an IAU have been inactive for 180 seconds its delegation will timeout, the CFS primary will then see an accurate inode used count for that IAU.
- As files are being created IAUs will be delegated one at a time and will timeout ~3 minutes after all the inodes in the IAU hace been allocated.
- As files are being removed the IAU holding each inode being freed will have to be delegated, but these delegations will also timeout ~3 minutes (180 seconds) after the IAU is last updated.
- Ulimately each IAU delegation will timeout after a period of ~3minutes of inactivity, however one IAU can remain delegated to each node.

- Therefore, the "generally expected" inaccuracy of the df –i used inode count after ~3mins of file create/remove inactivity will be the number of CFS secondaries multipled by the number of inodes represented by an IAU (which itself is dependant upon the file system block size). It is of course possible that an IAU with less than 64 free inodes remining (but more than zero free inodes) might exist and remain delegated to a CFS secondary node as well – when this situation occurs a second IAU can obviously remain delegated to a CFS secondary node and not timeout.

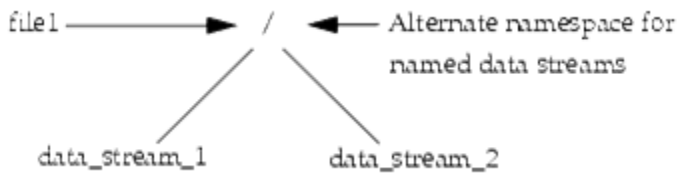
- Example of worst case scenario (but ignoring IAUs with <64>0 free inodes):
  4 node cluster, so 3 CFS secondaries using a 8Kb file system block size could have a maximum inaccuracy 196608 used inodes.
  Inaccuracy using an 8Kb vrs 1Kb file system block size comparison:
  3 CFS secondaries each with one IAU *8Kb fs block size * 8bits = 196608
  3 CFS secondaries each with one IAU *1Kb fs block size * 8bits = 24576

The df –i count, when using named data streams

The VxFS implementation for named data streams (alternate name spaces) also uses inodes, but the files cannot be seen in the regular name space.

Every file can have its own alternate namespace to store named data streams. The alternate namespace can be accessed through the named data stream APIs supported by VxFS. There are no VxFS-supplied administrative commands to use this feature. A VxFS API is provided for creating, reading, and writing the named data streams of a file.

Alternate Namespace



Example of creating a named stream on a file:

At mkfs time inode-0 and inode-1 are never utilized but they are marked as allocated, inode-2 is the root inode and inode-3 is the lost+found directory

```
# mkfs -t vxfs /dev/vx/dsk/dg/vol
# mount -t vxfs  /dev/vx/dsk/dg/vol /swapmnt
# cd /mnt
# touch file1
```

```
# ls -li /mnt
4 -rw-r--r--. 1 root root  0 Feb 26 04:55 file1
3 drwxr-xr-x. 2 root root 96 Feb 26 04:46 lost+found
```

```
# df -i /mnt
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/vx/dsk/dg/vol
                     7845180       5 7845176    1% /mnt
```

Above we currently have a correct inode used count of 5, for inodes 0/1/2/3/4

Inode-0 is never utilized, but marked as allocated
Inode-1 is also never utilized, but marked as allocated
Inode-2 is the root inode
Inode-3 is the lost+found directory
Inode-4 is the file we created above called “file1”

Set a named attribute using file1 (inode 4):

```
# attr -s attr1 -V "asdsaasd" file1
Attribute "attr1" set to a 8 byte value for file1:
asdsaasd
```

Above we created two new inodes, one for the named attribute directory, and another for the named attribute file.

So we now have an inode used count of 7.

```
# df -i /mnt
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/vx/dsk/dg/vol
                     7845184       7 7845177    1% /mnt
```

Using fsdb we can see inode 4 now has a pointer (via inattrino) to inode 5.
And
Inode-5 points back to inode 4 (via dotdot)
Inode-6 points back to inode 5 (via dotdot)

```
> 4i
inode structure at 0x00000a29.0000
type IFREG mode 100644  nlink 1  uid 0  gid 0  size 0
atime 1361883325 98317  (Tue Feb 26 04:55:25 2013 PDT)
mtime 1361883325 98317  (Tue Feb 26 04:55:25 2013 PDT)
ctime 1361883328 80615  (Tue Feb 26 04:55:28 2013 PDT)
aflags 0 orgtype 1 eopflags 0 eopdata 0
fixextsize/fsindex 0  rdev/reserve/dotdot/matchino 0
blocks 0  gen 2041052908  version 0 57   iattrino 0
dotdot 2  inattrino 5
de:  0 0 0 0 0 0 0 0 0 0
des: 0 0 0 0 0 0 0 0 0 0
ie:  0 0
ies: 0
>
> 5i
inode structure at 0x00000a29.0100
type IFATDIR mode 150755  nlink 2  uid 0  gid 0  size 96
atime 1361883328 80616  (Tue Feb 26 04:55:28 2013 PDT)
mtime 1361883328 80615  (Tue Feb 26 04:55:28 2013 PDT)
ctime 1361883328 80615  (Tue Feb 26 04:55:28 2013 PDT)
aflags 0 orgtype 2 eopflags 0 eopdata 0
fixextsize/fsindex 0  rdev/reserve/dotdot/matchino 0
blocks 0  gen 453887686  version 0 8   iattrino 0
dotdot 4  inattrino 0
>
```

```
> 6i
inode structure at 0x00000a29.0200
type IFREG mode 100644  nlink 1  uid 0  gid 0  size 8
atime 1361883328 80616  (Tue Feb 26 04:55:28 2013 PDT)
mtime 1361883328 89289  (Tue Feb 26 04:55:28 2013 PDT)
ctime 1361883328 89289  (Tue Feb 26 04:55:28 2013 PDT)
aflags 0 orgtype 1 eopflags 0 eopdata 0
fixextsize/fsindex 0  rdev/reserve/dotdot/matchino 0
blocks 1  gen 625322896  version 0 9  iattrino 0
dotdot 5  inattrino 0
de:  2655    0    0    0    0    0    0    0    0    0
des:    1    0    0    0    0    0    0    0    0    0
ie:     0    0
ies:    0
```

However, the two new named data stream inodes, inodes 5 and 6, cannot be seen in the namespace using cmd like ls –l

```
# ls -li /mnt
4 -rw-r--r--. 1 root root  0 Feb 26 04:55 file1
3 drwxr-xr-x. 2 root root 96 Feb 26 04:46 lost+found
```

An example of expected inode usage using named data streams:

We create about 300,000 files from a Windows client, using CIFS (Samba) share. The files are created and stored on FileStore using VxFS mounted on /vx/fs1.

Here is a comparison of the number of data files, and "df –i" used inode count output.

```
# pwd
/vx/fs1

# find . | wc -l
300037  <== number of files is ~300k

sfcstlx0304_01:/vx/fs1 # df -i
Filesystem            Inodes   IUsed   IFree   IUse% Mounted on
/dev/sda1           15564800  122178 15442622    1% /
udev                 2046965     578  2046387    1% /dev
/dev/sda5            13320192   19997 13300195    1% /opt
/dev/sda6            2441216    2613  2438603    1% /var
/dev/sda7            2441216     141  2441075    1% /tmp
tmpfs                2097152     118  2097034    1% /dev/vx
/dev/vx/dsk/sfsdg/_nlm_
                       31720    7658   24062   25% /var/lib/nfs/sm
/dev/vx/dsk/sfsdg/fs1
                     10240896 1200045 9040851   12% /vx/fs1   <=== inodes
```

Inodes used is 1200 thousand about 4 times the number of files of 300 thousand

Summary: This difference in the number of files and "df –i" output is expected and normal. And it does not negatively impact the inode monitoring functionality in FileStore. FileStore needs to monitor the inode usage, regardless of which format is being used, whether it is a regular data file inode, or windows NDS inode. Inodes are VxFS resources, and we want to capture their usage using the monitor, even though the 'df –i' used inode count does not agree with the number of files in the regular namespace in this case.