Please DO touch that dial

Intro to zSeries File System performance tuning

BY TJ MORRISSEY

An often-overlooked step during implementation of the zSeries File System (zFS) is performance monitoring and tuning for the installation's specific usage of zFS. To accomplish this, zFS provides a set of QUERY commands to monitor zFS performance and a set of tuning options to optimize the performance of zFS.

The default settings for the various tuning options are a good base starting point, and often work perfectly well. However, it's important to understand that because implementations and usage of file systems can vary greatly, you might be able to improve zFS performance significantly if you tune it for your installation's requirements.

File system structure

Prior to implementation, the first item to consider in your planning is the overall file system structure. Sometimes you might have very little or no flexibility in this decision because of a number of possible factors. However, in order to achieve the best performance from zFS, it is advisable, when possible, to try to spread your data out across the file system structure in a more lateral manner. To accomplish this, try to use multiple directories within specific file systems, and try to avoid situations where you have excessive amounts of files contained within a single directory. For example, using the tree in Figure 1, it would be counterproductive to have 300,000 files in *dir1* of the file system mounted at *data_fs1*. A better approach would be to spread these files out more equitably across multiple directories in *data_fs1*.

Garbage collection

It's also advisable to periodically perform garbage collection on your file systems and clean out any unnecessary files and directories that are actively in use. For example, some applications can create thousands of temporary files and not clean them up. Again, using the tree in Figure 1, suppose that application ABC wrote thousands of temporary files into *dir2*. Assuming that *dir2* is an active directory; this presence of a large amount of unnecessary files could affect zFS performance negatively. In this case, reducing the amount of unnecessary data within the file system might have a positive affect on performance.

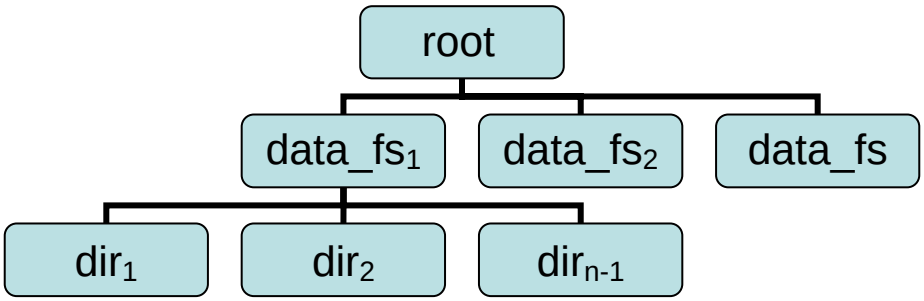

Figure 1. Directory tree

Number of file systems

Another item to consider is spreading your data out across multiple, smaller file systems, as compared to having fewer, excessively large file systems. For example, using the tree in Figure 1, assume you have a need for 50 GB of file system storage. In this case, it's advisable to have multiple file systems of a smaller size, say ten file systems of 5 GB each (represented by the *data_fs1 – data_fsn-1)*, as compared to having one large 50 GB file system (just *data_fs1* for example). This can have a positive effect on backup times, while avoiding the situation and the obvious problems that can arise from having 'all of your eggs in one basket'.

Performance and tuning basics

It's time to get to basic zFS performance monitoring and tuning. To start with, it's important to understand that you cannot modify all of the zFS performance tuning options dynamically. Therefore, as a general practice, it's important to establish a basis for the various settings initially in an environment where zFS can be restarted with no adverse impact. The most appropriate way to do this is in a test environment, where a series of workloads that approximate the proposed production workload can run against the target zFS file systems. Then, using the various zFS QUERY reports, you can evaluate the performance and make appropriate changes to get an acceptable starting point for the various configuration options that can affect zFS performance.

The recommended approach to begin analyzing zFS performance is to capture a snapshot of a few of the key reports during peak time. The zFS QUERY reports provide a large amount of information. Initially, it's advisable to limit your review to a couple of the key reports. For the purpose of this discussion, we will focus on a few specific sections of the KN, VM and LFS reports. To capture these reports for a defined time, enter a series of MODIFY commands to zFS. For example, suppose testing of production type workloads begin at 9 a.m. and run for three hours; using the F ZFS command, you can do the following:

1. Just prior to the 9 a.m. start of the test run, enter the following command to reset all the zFS performance counters: F ZFS,RESET,ALL
2. Start the test run

3. At 10 a.m. enter the following command to provide some of the key zFS performance statistics for the period from 9 a.m. to 10 a.m. and reset the counters for the next period: F ZFS,QUERY,KN, F ZFS,QUERY,VM, F ZFS,QUERY,LFS, and F ZFS,RESET,ALL.
4. At 11 a.m., enter the same four commands you entered at 10 a.m to capture the same statistics for the period from 10 a.m. to 11 a.m.
5. Repeat the same procedure again at 12 p.m. to get the statistics for the period from 11 a.m. to 12 p.m.

Now that you have three sets of reports, what do they all mean? Well, for purposes of this discussion we can simplify this to a few key entries in the KN, VM Cache, and LFS reports.

The KN report

First, the KN report. This report is perhaps the most basic measurement and all telling report of overall zFS performance. In simple terms, it is the total number of physical file systems (PFS) calls to zFS from z/OS UNIX System Services and the average response time in milliseconds for each specific set of PFS calls made to zFS. In general, it's desirable to see the average response time for most operations to be as low as possible. What constitutes low varies by installation, but a value in the single digit range is common (for example, something less than 10.000 on average).

There are a few key items to watch for in this report. Using the KN report in Figure 2 as an example:

```
IOEZ00438I Starting Query Command KN or XCF.
     zFS Kernel PFS Calls
     --------------------
Operation       Count           Avg Time
---------      ----------     ----------
zfs_opens         61            0.010
zfs_closes        59            0.004
zfs_reads         10            0.935
zfs_writes         6            0.023
zfs_ioctls         6            0.002
zfs_getattrs      91            0.005
zfs_setattrs       2            0.042
zfs_accesses      25            0.008
zfs_lookups       72          885.645
zfs_creates        2         6718.235
zfs_removes        1            0.000
zfs_links          0            0.000
zfs_renames        0            0.000
zfs_mkdirs         0            0.000
zfs_rmdirs         0            0.000
zfs_readdirs      57            0.168
zfs_symlinks       0            0.000
zfs_readlinks      0            0.000
zfs_fsyncs         0            0.000
zfs_truncs         0            0.000
zfs_lockctls       0            0.000
zfs_audits         0            0.000
zfs_inactives      3            0.005
zfs_recoveries     0            0.000
zfs_vgets          0            0.000
zfs_pfsctls        0            0.000
zfs_statfss       20            0.009
zfs_mounts         1          592.596
zfs_unmounts       0            0.000
zfs_vinacts        0            0.000
--------      ----------     ----------
*TOTALS*         416          187.058
```

Figure 2. KN report

In Figure 2, zfs_lookups and zfs_creates are on average quite elevated, as is the overall average response. There's also a single mount that is elevated, but this is common to see for mounts. Taking into consideration that lookups and creates would be common operations, while a mount would not be considered a common operation, this might warrant further investigation and tuning to try to improve the statistics for the lookup and create operations. Deficiency in other areas of zFS tuning, such as the various caches, can affect these results. Often, another performance report, such as LFS or VM, shows an area that's deficient in tuning.

Adjusting those areas that are deficient often leads to an improvement in the statistics for this specific report. Then, after making your adjustments, rerun your test workload again and capture another set of QUERY reports to confirm the results of your change.

VM cache report

Another report to look at is the VM caching report, which shows statistics on how the user file cache is performing. One key point to focus on initially in this report is the section on direct file reads.

```
File System       Reads  (Direct):
------------------
Reads Faulted       4      (Fault Ratio  0.658%)
Writes Faulted     22      (Fault Ratio  0.178%)
Read Waits          0      (Wait Ratio  0.00%)
Total Reads        26
```

Figure 3. Direct file reads

Of particular interest in this section is the fault ratio. Taking this fault ratio and subtracting it from 100, gives you the hit ratio. A good hit ratio is generally anything above 80%. In Figure 3, it's actually above 99%. However, this number varies by workload, and there are some exceptions. For example, the Lotus Domino Server might only show a hit ratio in the 60% to 70% range due to its access patterns and the amount of data, however, in general this is a good measurement of how the user file cache is tuned. You can usually improve this value by using the zfsadm config command to dynamically change the size of the *user_cache_size* parameter. *z/OS Distributed File Service zSeries File System Administration*, SC24-5989 contains additional information on the user file cache and how to make adjustments to improve the performance in this area.

LFS report

Last, but not least, is the LFS report. This report shows various pieces of information, all of which are discussed in more detail in *z/OS Distributed File Service zSeries File System Administration*, SC24-5989. Here, we'll just touch on three of the cache reports that are present in this section:

- vnode cache statistics
- directory cache statistics
- metadata cache statistics

Using the following sections of the LFS sample report as an example:

zFS Vnode Cache Statistics

Vnodes	Requests	Hits	Ratio	Allocates	Deletes
139462	8352	8090	96.863%	0	166

zFS Vnode structure size: 192 bytes
zFS extended vnodes: 65536, extension size 692 bytes (minimum)
Held zFS vnodes: 396 (high 139462) Open zFS vnodes:
239 (high 2218) Reusable: 136132

Metadata Caching Statistics

Buffers	(K bytes)	Requests	Hits	Ratio	Updates
32768	262144	12749	12292	96.4%	7632

Directory Cache Statistics

Dir Blocks	(K bytes)	Requests	Hits	Ratio	Deletes
32768	262144	30403390	30402952	99.998%	0

Figure 4. Sample sections from the LFS sample report

It's generally desirable to see the ratio (hit ratio) for each of these to be above 80%. If they are not, adjustments can generally be made to the *dir_cache_size*, *meta_cache_size* or *vnode_cache_size* to improve these values and overall performance. Use the zfsadm config command to tune the vnode_cache_size and meta_cache_size dynamically. Changing the dir_cache_size requires a restart of zFS. One item to note in the above is the hit ratio for the vnode cache. This can vary greatly and user operations within the file system can have an adverse affect. For example a user issuing an **ls** command in the OMVS shell will cause this value to be significantly degraded, giving a false indication that the hit ratio is bad, when in reality it may not be.

For more information

The above points are a good starting point, to get you on your way to tuning and monitoring zFS for your specific installations needs. There is a wealth of detailed information on this topic in Chapter 9, "Performance and debugging" in *z/OS Distributed File Service zSeries File System Administration*, SC24-5989. So go ahead and touch that dial!