#SHAREorg

zFS Diagnosis I: Performance Monitoring and Tuning Guidelines

Scott Marcotte
IBM

February 7, 2013 8AM
Yosemite B
Session Number 12730

smarcott@us.ibm.com

IBM

Topics

Title	Slides
Fundamentals	3-5
Storage	6-8
User File Cache	9-12
Metadata/Backing Cache	13-16
DASD IO	17-19
Lock Contention	20-22
Additional Items	23
Sysplex Sharing	24-25
Object Caching	26-28
Sysplex Statistics	29-32
Going Forward	33
z/OS 11/12 Summary	34

IBM

Fundamentals I: Overview (Most of this presentation is for z/OS 13):

- **zFS cache defaults are small**
  - **Larger users of zFS should perform tuning for best performance**
- **zFS has F ZFS,QUERY commands which can be used to gauge performance**
  - **Also has F ZFS,RESET to reset statistics**
  - **Individual stats only 4 byte words – can wrap quickly**
    - **Useful mainly for analysis of peak usage, not long-term usage**
- **Cache sizes can be dynamically altered via zfsadm config**
- **F ZFS,QUERY,STORAGE – Shows how much memory zFS is using - IMPORTANT**
- **Ensure that zFS is not paging**

3

zFS caching defaults have been historically low. When zFS was introduced, HFS was the primary file system and zFS kept its defaults low since most file systems would be HFS and therefore system memory should be used for HFS caching. As more customers have moved away from HFS onto zFS, those defaults will not provide optimal performance. Currently, zFS tuning is a manual process where the user must use zFS commands to gauge performance and then possibly alter zFS cache sizes and then re-gauge performance. zFS provides an F ZFS,QUERY command which is used to show zFS performance. Since the internal counters are only four bytes large, they can wrap quickly (a few hours of peak usage for heavily loaded systems) and therefore, the query commands should be viewed as usable for monitoring perk performance but not necessarily useful for long-term monitoring (such as days, weeks, or months). zFS is storage constrained in its primary address space, and care must be taken when altering the caching defaults for certain caches.

IBM

Fundamentals II: Tuning zFS For All Environments

- **Tune zFS by specifying the following zFS startup parameters:**
  - **User_cache_size – Amount of memory used to cache the contents of user files.**
  - **Meta_cache_size/metaback_cache_size – Amount of memory used to cache disk blocks that contain metadata.**
    - Metadata is anything on disk that is not user file data such as directories, access control lists (ACLs), structures that track free file system space etc…
  - **Vnode_cache_size – Number of objects that are cached in memory.**
    - A file, directory or symbolic link, currently or recently of interest to applications is represented in memory by a vnode (also called evnode) and that will anchor additional structures required to process requests for the object.
    - zFS caches the most recently accessed objects by applications.
    - This parameter is more important to the sysplex environment.
- **Can also dynamically alter cache sizes via zfsadm config**

4

zFS allows startup parameters to be specified in a dataset that is specified on the IOEFSPRM DD statement in the zFS JCL procedure, or via parmlib search. All zFS options are specified as limits, which determines the maximum amount that zFS will ever use for that option.

IBM

Fundamentals III: F ZFS,QUERY,KNPFS – zFS summary

PFS Calls on Owner

Operation	Count	XCF req.	Avg Time
zfs_opens	2414314	0	0.004
zfs_closes	2413205	0	0.003
zfs_reads	1809051	0	0.083
zfs_writes	**732783**	**0**	**0.017**
zfs_ioctls	1453868	0	0.001
zfs_getattrs	3041548	0	0.002
zfs_setattrs	**10613**	**0**	**0.092**
zfs_accesses	38730578	0	0.002
zfs_lookups	5926262	0	0.041
zfs_creates	**9763**	**0**	**0.426**
zfs_removes	**10604**	**0**	**1.532**
zfs_links	**0**	**0**	**0.000**
zfs_renames	**3710**	**0**	**0.489**
zfs_mkdirs	**333**	**0**	**1.247**
zfs_rmdirs	**529**	**0**	**0.275**
zfs_readdirs	784790	0	0.550
zfs_symlinks	**380**	**0**	**0.380**

- This report shows all of the calls made to zFS since last statistics reset or since start of zFS
- **Boldface** are write operations

....

Operation	Count	XCF req.	Avg Time
zfs_readlinks	34178	0	0.086
zfs_fsyncs	**250**	**0**	**2.560**
zfs_truncs	**3931**	**0**	**0.012**
zfs_lockctls	0	0	0.000
zfs_audits	4970	0	0.046
zfs_inactives	2032174	0	0.001
zfs_recoveries	0	0	0.000
zfs_vgets	3854	0	0.009
zfs_pfsctls	68	0	0.088
zfs_statfss	42700	0	0.008
zfs_mounts	120	0	91.581
zfs_unmounts	2	0	215.575
zfs_vinacts	0	0	0.000
TOTALS	59464578	0	0.017

- The *TOTALS* line shows total calls to zFS and the average zFS response time in milliseconds
- Knowing the last reset time, or zFS startup time (from system log), can determine zFS call rates
- Read operation response time desired to be < 1 msec, hopefully significantly less.

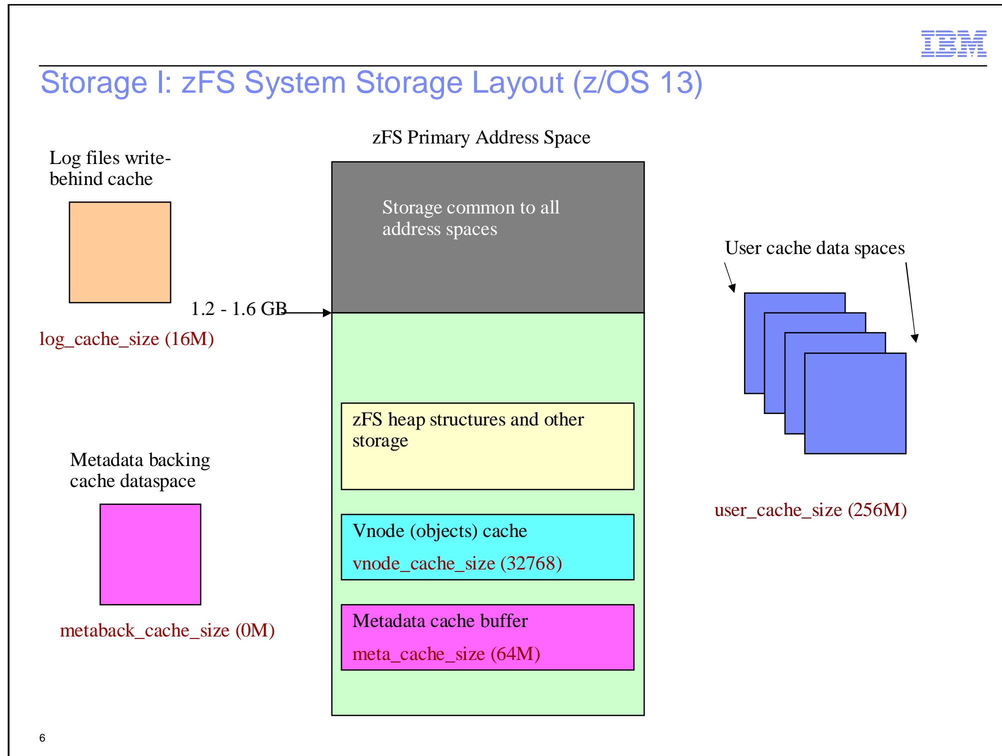
5

The F ZFS,QUER,KNPFS report will show the calls for file systems owned by the local system (for the single-system environment that is all file systems). It has another identical report showing the file systems that are owned by a remote system in the sysplex to break down locally owned, and zFS sysplex client performance.

If a user does not reset the zFS statistics via the F ZFS,RESET command, then the statistics shown will be since zFS startup. If the F ZFS,RESET command is used, then those statistics will be since the reset command affected that particular report (a user can reset the statistics for only one report, but it is recommended that they always issues F ZFS,RESET,ALL to reset all statistics to keep things clear). The system log and zFS job output (if using JES output) will show the time of the last statistics reset. The last statistics reset time along with the *totals* line can be used to determine the call frequency to zFS.

Determining the call frequency to zFS determines the importance of tuning zFS, the higher the rate of calls to zFS the more important its performance is to the system. In the example above, zFS has a light load and is actually performing well, tuning is not necessary based on these results.

The XCF Req. column is used to indicate when an owner system has to callback to clients to reclaim the object lock(s) of the object that are affected by the operation. It would always be zero for a single system or a system only using z/OS Unix sysplex sharing as in this example.



zFS is a 31 bit C program and therefore its structures reside in its primary address space with the exception of some dataspaces used to contain in-memory copies of disk data. zFS does put its internal trace table above the 2GB bar and will run in 64 bit mode when cutting a trace record but all other code runs in 31 bit mode. Depending on the installed programs and z/OS configuration, part of the address space is storage that is common to all address spaces and unavailable for zFS use; typically, this leaves zFS with approximately 1.2 to 1.6GB of storage for its use.

The log files write-behind cache is not something a user has to tune, its shown here for completeness of the picture. Every zFS file system has a log file that contains transaction records for recent disk updates and the log file is replayed at mount time if the system crashed to put the file system in a consistent state. This cache, contained in a dataspace contains a write-behind cache for the blocks of the log files of file systems. It is dynamically grown by zFS as more file systems are mounted, and does not require any customer tuning.

The user file cache contains in-memory copies of disk blocks that contain user file data. The most recently used pages by applications are kept in this cache, which is contained in one or more data spaces (with a maximum of 32 data spaces used). Only the contents of user files are stored in the user cache data spaces, the control structures that locate the data in the cache are kept in the zFS heap in the zFS primary address space. Although in theory the zFS user cache could be 64GB (32 2GB data spaces), the primary address space constraints of zFS limit the maximum to approx. 48GB, and then only if the vnode and metadata cache are kept small (at their defaults). The default size of the user file cache is 256M.

IBM

Storage II: Monitoring Primary Storage (F ZFS,QUERY,STORAGE)

- **Sample Output (example here shows that zFS storage dangerously high):**

```
IOEZ00438I  Starting Query Command STORAGE. 778
                  zFS Primary Address Space Storage Usage
                  ---------------------------------------

Total Storage Available to zFS: 1738539008 (1697792K) (1658M)
Non-critical Storage Limit: 1717567488 (1677312K) (1638M)
USS/External Storage Access Limit: 1675624448 (1636352K) (1598M)
Total Bytes Allocated (Stack+Heap+OS): 1669189632 (1630068K) (1591M)
Heap Bytes Allocated: 1587033610 (1549837K) (1513M)
Heap Pieces Allocated: 11445446
Heap Allocation Requests: 4
Heap Free Requests: 3
```

- Total storage available is amount zFS can use, after factoring common storage
- USS/External Storage Access Limit – Do not define caches so big that this is exceeded:
  - If exceeded, application requests to access un-cached objects fail with ENOMEM
- Total Bytes Allocated shows how much storage zFS is using:
  - Includes zFS heap storage and zFS runtime stacks for application calls
  - And any operating system storage allocated on behalf of zFS
- Try not to define caches so large that: Bytes Allocated + X MB > USS/External limit

7

Shown in the slide is the first portion of the output of the F ZFS,QUERY,STORAGE command. It shows the total available storage to zFS in the address space and how much zFS is using. The USS/External storage access limit is an important number since it's the limit that zFS will allow new storage to be obtained on behalf of applications access to objects that are not currently cached in the vnode cache. For example: if an application is attempting to open a file, and that file does not have a vnode yet in the zFS vnode cache, it will get an ENOMEM failure if zFS exceeds the USS/external access limit. zFS will still allocate storage if its past this limit, but only for critical internal structures to keep zFS running properly, not new accesses to new objects by applications.

The example shown in the slide is a case where the user is dangerously close to the limit, it is not recommended to run this close to the limit, and in this case, it might be recommended to reduce the size of one or more of the zFS caches. For most customers, you want to keep a nice buffer zone between the total bytes zFS has allocated and the USS/External limit, this buffer zone is somewhat workload dependent. As will be explained later, the buffer zone will depend on z/OS Unix and application behavior as shown by the vnode cache performance report.

IBM

Storage III: Monitoring zFS Storage continued…

Heap Usage By Component

Storage Usage By Component

Bytes Allocated	No. of Pieces	No. of Allocs	Frees	Component
….				
49176	84	0	0	Aggregate Management
108092	16	0	0	Filesystem Management
194574144	800172	0	0	Vnode Management
196775488	401617	0	0	Anode Management
351680	3082	0	0	Log File Management
150692144	287625	0	0	Metadata Cache
…..				
493877648	7964319	0	0	Cache Services
138924280	655378	0	0	User File Cache
…. .				

- F ZFS,QUERY,STORAGE also shows usage by zFS sub-component

Aggregate/Fileset management are mounted file system structures

Vnode/Anode Management is storage related to vnode cache.

Metadata cache storage is for metadata and backing cache

Cache Services is storage related to all the caches

User File Cache is storage related to user file cache.

8

This slide continues the sample output from the prior slide. The F ZFS,QUERY,STORAGE command also shows a breakdown of heap storage used by component. The slide here shows the significant portion of that report related to the tuning parameters customers would specify for zFS: vnode_cache_size, meta_cache_size/metaback_cache_size, and user_cache_size. All zFS caches are managed by a common component. That common component has storage that is used to link objects together in various sets (some examples of sets would be the set of vnodes that belong to a specific file system, or the set of dirty metadata cache buffers for a particular directory) and manage access of sets. This storage is therefore related to all caches. Reducing or increasing any zFS cache would affect the amount of storage shown in this field.

The example from this slide is for a sysplex where the vnode_cache_size is 400,000, the user cache size is 2GB, the metadata cache is 100M and the metaback_cache_size is 2GB.

IBM

User File Cache I: Background

- **Cache is comprised of one or more data spaces** - simply an array of 4K pages.
- **Smallest addressable unit is 4K page** - nicely matches VSAM dataset control interval size
- **Files need not have all of their pages in the cache**
- **Files further broken down into 64K segments**,
  - A file will have zero or more segments cached at one time.
  - Each segment itself is sparse – not all the pages in a segment need to be in memory
  - **The structure that represents a segment is in zFS primary storage**
    - Thus the user file cache primary address space storage is mainly segment storage and the anchors to the segments for each file.
- **Locking is done at the segment level**
- **Parallel reading and writing to the same file is allowed**
  - Contention would occur at segment level
  - Writing is partially serialized when extending file
- **Full read-ahead and write-behind supported**
  - Metadata updates performed on background tasks

9

The user file cache is simply a cache of in-memory copies of the 4K VSAM control intervals that contain user file data from zFS file systems. Files in the zFS are conceptually partitioned into 64K segments, each segment addressing sixteen 4K pages. A file need not have all of its segments cached, and a segment need not have all of its pages in the cache. zFS uses a structure in its primary address space to represent a file segment, and this structure and the anchors to the list of segments in memory for a file make up much of the user cache storage that occupies the zFS primary address space.

IBM

User File Cache II: Recommendations

- **Utimate goal: 100% hit ratio**
  - A hit means an attempt to find a portion of a user file finds the data is in the cache.
- **Cache hit ratios very workload dependent:**
  - A bunch of processes running shell scripts in OMVS accessing small files will likely achieve a near 100% hit ratio
  - A Domino server workload could at best achieve a 70% hit ratio
  - In practice, hit ratios will rarely or never be 100%
- **F ZFS,QUERY,VM – shows user file cache performance (next slide)**
- **Some Guidelines:**
  - If hit ratio is below 90% or the user cache request rate is very high:
    - Adjust cache size upward
    - Factor in zFS memory usage to make sure zFS not driven too low in primary storage – use **f zfs,query,storage** report to estimate primary space growth
    - Monitor performance again, if it helped then repeat these steps
    - If the increase did not help performance, then your workload might not benefit from a larger cache, might as well go back to prior size.
  - Use **zfsadm config –user_cache_size** to dynamically change cache size
    - Should be done off-peak - its expensive if it's a large delta from current size
  - Update zFS startup parameters (**user_cache_size**) so it starts with desired size in future

10

In an optimal world, one could cache all file system data in memory achieving a 100% hit ratio per cache access. Since systems do not have an unlimited amount of memory, the user has to decide how much memory they would like to assign to zFS for user file caching, trying to balance the needs of the rest of the system and the needs of applications that use zFS to achieve best application response time. The hit ratio that is achieved in the user file cache is often workload dependent. For example, with Domino server workloads, the amount of data accessed by Domino clients was so large it could not be contained in even a large user file cache, and the access pattern was often one that did not repeatedly access the same data. In those workloads, 70% would be considered a good ratio. For some shell scripts running in OMVS, if they are dealing with mainly smaller files they could often achieve a 100% hit ratio. If zFS has a high file read/write request rate (as shown by the query,vm report, which is described on the following slide), then tuning the zFS user file cache is certainly worthwhile. A reasonable goal would be to attempt to get over a 90% hit ratio, and even higher if there is a high request rate to zFS. Workloads are often variable, and there may be periods of time when the zFS request rate is high, and periods of time when it is low. It's the high request rate periods that will benefit greatly from a properly sized user file cache.

Therefore, if it appears that raising the cache is a worthwhile goal for the system's workloads, then the query,storage report should be used to determine the current amount of memory used by the user file cache and the total amount of storage being used by zFS. Then determine a safe amount to raise the cache. For example, if the user file cache is currently using X amount of storage in the zFS primary address space, and it was desired to raise the cache size by 50%, then one can estimate that zFS will use 1.5X storage in its primary address space for structures that manage the user file cache. If this would leave the total zFS storage in the address space below the desired limit then it would be safe to make that cache adjustment.

The zfsadm config –user_cache_size command can be used to dynamically alter the cache size while zFS is running. It will stop application activity and sync data and re-size the cache, so if the relative change is great (either upward or downward) then it could take some time, so it should be an off-peak operation. Once you settle on an optimal size you can add the line: "user_cache_size=XM" in the zFS startup parameters (IOEFSPRM DD or your parmlib search dataset members) to ensure zFS starts with the desired size in the future.

IBM

User File Cache III: F ZFS,QUERY,VM --- Cache Statistics

```
IOEZ00438I Starting Query Command VM. 367
           User File (VM) Caching System Statistics
           ---------------------------------------

External Requests:
-----------------
Reads       943879   Fsyncs          73    Schedules     4109
Writes      428723   Setattrs      3303    Unmaps        2436
Asy Reads   747874   Getattrs   1641816    Flushes          0

File System Reads:
-----------------
Reads Faulted        10088   (Fault Ratio   1.069%)
Writes Faulted          10   (Fault Ratio   0.002%)
Read Waits            8171   (Wait Ratio    0.866%)
Total Reads          18791

File System Writes:
------------------
Scheduled Writes     23868   Sync  Waits        328
Error Writes             0   Error Waits          0
Scheduled deletes     1330
Page Reclaim Writes      0   Reclaim Waits        0
Write Waits            102   (Wait Ratio   0.024%)
```

Reads and Writes are file read and write requests made to user file cache since the last time statistics were reset

Reads/Writes Faulted shows miss count and ratio:

hit ratio = 100 – fault ratio

(hit ratio @ 99% in this example)

High page reclaim write and wait rates, relative to request rate, show a cache that is too small for amount of data being written

11

Most of the zFS query command output shows statistics since the last reset or since startup if no reset commands were ever done. Statistics are reset via the F ZFS,RESET command. The user file cache report will show the external request rate and the fault ratio (miss rate). The hit ratio is simply 100 – fault ratio. In this example, the hit ratio is 99% which means the user file cache is performing very well for the workload and hence no tuning is needed.

An important concept regarding file caching is that not only is the cache used for containing data to avoid reads from disk, but it also is used for asynchronous write-behind of data. If the page reclaim writes and/or page reclaim waits is high relative to the request rate, that means a miss had found that the oldest data in the cache was actually dirty and had to be written, which adds additional wait time to the request as it has to wait for writing of the oldest data to make room in the cache for the data it wants to read. In this case, the working set of data being written is larger than the cache, and is a good indicator that performance may improve with an increase in the size of the user file cache.

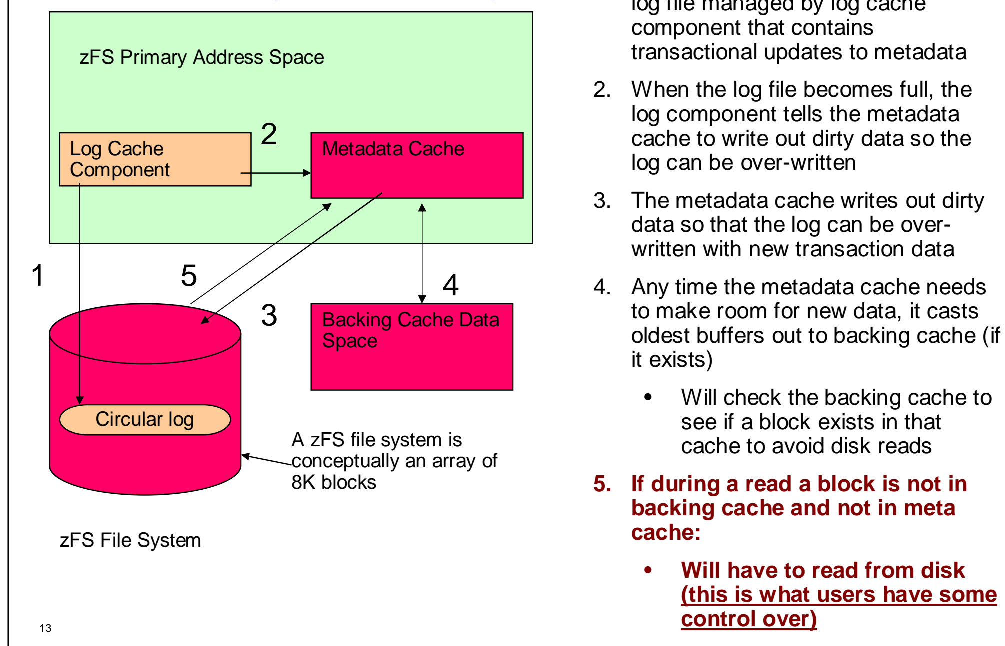
User File Cache IV: F ZFS,QUERY,VM continued…

```
Page Management (Segment Size = 64K) ) (Page Size = 4K)
--------------------------------------------------
Total Pages        65536    Free         65451
Segments           16384
Steal Invocations   2405    Waits for Reclaim      0

Number of dataspaces used:    4  Pages per dataspace:     16384

Dataspace  Allocated     Free
Name       Segments      Pages
--------   ---------    ---------
ZFSUCD00        2        16352
ZFSUCD01        0        16384
ZFSUCD02        3        16363
ZFSUCD03        2        16352
```

Shows cache size and how many pages free (unused) and data space breakdown

- **Waits for Reclaim indicate tasks waiting to reclaim oldest pages for a miss**
- **A high value (relative to request rate) suggests a possible need to increase user file cache.**

- **In the simple example shown here, taken late at night on a small production system, the default user cache size of 256M is fine and does not need tuning.**



A zFS file system conceptually blocks two 4K VSAM control intervals into one 8K block. The metadata cache and backing cache contains in-memory copies of any 8K block that does NOT contain the contents of a user file.

Updates to metadata in zFS are handled by transactional updates. Transactions are started and the changes made to the metadata are written to 8K pages in the log cache component, and as those pages are filled they are written out to the circular log file in the file system. When that circular log is becoming full, the log cache component signals to the metadata cache that the log is becoming full and to write out dirty blocks for that file system, once that is complete the log file can be overwritten again with new transactional data. The administrator has little control over how the metadata is written to disk, they can only control the size of the caches. It is important to note that if the metadata cache is too small that caches misses will be more frequent and if the oldest buffers in the cache are dirty then that would force a write of that data to make room for the new disk block. Journaling file systems like zFS have a rule that any log file pages containing update records that describe changes to a disk block must be written and on disk before that metadata disk block is written. This means that if the oldest buffers are dirty then it might force more disk-writes and IO waits for log file pages. Future slides will show how to detect this condition.

What the administrator can control is how much data to cache, the more blocks that are cached the less likely a disk read is required and the less likely the oldest buffers have to be cast out of the cache to make room for a new disk block. The backing cache is simply a read-only cache that has copies of file system metadata blocks, no writing to disk is performed from this cache. Basically it's a cache between the metadata cache and the disk.

IBM

Metadata/Backing Cache II: Recommendations

- **Goal is to achieve very high hit ratio of metadata cache**
  – Should be > 90% hit ratio, Preferably closer to 100%
- **Use of backing cache can help certain workloads** that access large amounts of metadata (directory searches for example)
  – Backing cache hit ratios, because it's a 2nd level cache are much lower than metadata cache, but:
    • Any hit is an eliminated disk IO and
    • Some locks are held over metadata cache accesses for control structures in a file system, so it can also reduce lock contention if IO is avoided
- **F ZFS,QUERY,LFS** – shows metadata and backing cache statistics (with other information)
- **Some Guidelines for metadata cache:**
  – If hit ratio is below 98%:
    • Adjust cache size upward – note that meta cache comes directly from zFS primary
    • Factor in zFS memory usage to make sure zFS not driven too low in primary storage – use **f zfs,query,storage** report to estimate primary space growth
    • Metadata and backing cache control structure storage is = Cache size / 64.
- **Some Guidelines for backing cache:**
  – Attempt to define or increase backing cache
  – Is the hit ratio significant enough to make a difference? If so then repeat the procedure until an optimal size reached.
    • Alternatively could work your way down from the maximum you could assign to it (2GB).
- Use **zfsadm config –meta_cache_size/-metaback_cache_size** to dynamically change cache size
  – **NOTE: Its not allowed to create a backing cache if it did not exist at zFS startup (z/OS 13)**
- Update zFS startup parameters (**meta_cache_size** & **metaback_cache_size**) so it starts with desired sizes in the future

14

The way in which zFS accesses the metadata cache will typically yield a high hit ratio, even for a smaller cache. One should certainly strive to achieve a 90% or higher hit ratio, better if its closer to 100% during peak usage. Metadata updates lock higher level structures before accessing the metadata and backing caches, so avoiding disk reads will also reduce lock contention wait times too. Its also important to note that since the backing cache is a 2nd level cache it is likely going to have a much lower hit ratio than the metadata cache. Even a hit ratio of 30-40% can yield performance improvements if zFS is being heavily used. The backing cache value is questionable if the hit ratio is very low, say 5-10% as the storage investment may not be worth it if that storage could help other components of the system.

The F ZFS,QUERY,LFS, as shown on the following slides shows detailed information related to the metadata backing cache.

In terms of tuning the caches, the backing cache is likely easier since it is external to the zFS primary address space. The easiest method is to start the cache as large as it can be based on available real memory, possibly even using its maximum size if available. Measure performance and determine the hit ratio of the cache. If the hit ratio looks reasonably promising, at least 30-40% or more, try reducing storage to see if you get almost the same value using less memory. This way you make the best possible choice, you do not use more memory than you need. The alternative is to start small and work larger looking for better hit ratios until they do not seem to improve. Either method will get their in the end. It is important to note that for z/OS 13 zFS, you cannot dynamically create a backing cache via zfsadm config: you can alter its size if it already exists but it can only be created at startup time.

The metadata cache is a bit trickier since it is stored in the zFS primary address space. The f zfs,query,storage report should be used to first determine how much primary storage zFS has left and determine a safe amount to increase based on primary address space usage. There will be a slight amount of overhead in primary address space for control structures that track the state of each on-disk buffer. The amount of storage required for the control information can be approximated by taking the desired metadata cache size and dividing by 64 to determine the amount of bytes for metadata cache control information. Since the cache is limited by zFS primary storage, the metadata cache control information is not a significant source of storage usage. The same calculation can be used to determine the amount of primary storage the control information will occupy for the backing cache.

IBM

Metadata/Backing Cache III: F ZFS,QUERY,LFS

Metadata Caching Statistics

Buffers	(K bytes)	Requests	Hits	Ratio	Updates
12800	102400	103268428	101985745	98.7%	24902311

Metadata Backing Caching Statistics

Buffers	(K bytes)	Requests	Hits	Ratio	Discards
262016	2096128	1063370	821113	77.2%	0

Report shows sizes, request rate and hit ratios for both caches, and also zFS IO requests by type.

Good performance for both caches, near 99% for metadata and 77% for backing cache.

I/O Summary By Type

Count	Waits	Cancels	Merges	Type
266415	259768	0	2311	File System Metadata
582931	10666	0	150777	Log File
0	0	0	0	User File Data

FYI: zFS uses IO queues, and merges adjacent IOs to reduce number of DFSMS IO requests

- Summary:
  - Backing cache eliminates almost half of request to zFS IO sub-system – GOOD!

IO requests are broken down into type, this workload was a pure directory workload (no user file IO)

Also shows number of times a task had to wait for an IO to complete

15

The f zfs,query,lfs provides much data, one section shows a simple report on metadata and backing cache performance. In this example (which was from an intense pure-directory read/write workload: file and directory creations, renames, removals etc…), the metadata cache got a hit ratio near the desired 100% and the 2GB backing cache was useful, it had a hit ratio of 77% and thus saves over 820,000 disk reads. The IO summary shows approx. 850,000 requests were made to disk for this workload.

One thing to note about zFS IO is that zFS uses an IO queue for every DASD volume that contains a zFS mounted file system on it, that queue is used for IO prioritization and merging of adjacent IOs. The merging of IOs usually occurs for log file IOs because the log file is in a contiguous region for each file system. The log file system uses asynchronous write-behind which is why there were not many IO waits for log file writes; however, there was substantial waiting for metadata IOs, this is examined further by the next report on the next slide.

IBM

Metadata/Backing Cache IV: F ZFS,QUERY,LFS ... continued from prior slide

I/O Summary By Circumstance

Count	Waits	Cancels	Merges	Circumstance
180	**0**	**0**	**0**	**Metadata cache read**
0	0	0	0	User file cache direct read
0	0	0	0	Log file read
....				
0	0	0	0	Metadata cache file async write
2569	636	0	0	Metadata cache sync daemon write
0	0	0	0	Metadata cache aggregate detach write
0	0	0	0	Metadata cache buffer block reclaim write
256028	**256020**	**0**	**2311**	**Metadata cache buffer allocation write**
0	0	0	0	Metadata cache file system quiesce write
7637	3111	0	0	Metadata cache log file full write
582952	**10666**	**0**	**150777**	**Log file write**

Metadata cache reads near 0, GOOD

High frequency of buffer allocation writes indicates cache smaller than amount of data being updated

→ If possible, try raising metadata cache size to see if these IOs can be reduced

Log file writes dominate IO, which is expected for a heavy directory workload

Ideal Situation: Near zero disk reads, almost all writes are log file writes and log file full writes. If this occurs, the caches are tuned as optimally as possible.

16

This slide shows a portion of the "IO Summary By Circumstance" section of the F ZFS,QUERY,LFS report. There are two important lines in this report when analyzing metadata cache performance and the effect of a change in cache size:

•Metadata cache reads – The number of reads to disk of metadata, since this is a very small value, the metadata and backing caches did a good job of eliminating reads in the steady-state.

•Metadata cache buffer allocation writes – This is the number of times that a buffer was not found in the metadata cache and the oldest buffers in the cache were dirty, forcing a write of those buffers and an IO wait so the buffers could be re-used. Thus although the read hit ratio is very high, the workload could still benefit from an increased metadata cache (writes are only performed from the metadata cache) to reduce the occurrence of the case where the oldest buffers are dirty. The user would have to look at the f zfs,query,storage report to determine if a cache increase was possible, and then re-evaluate the performance after the increase is made to see if these allocation write request rates could be reduced. Because the backing cache existed, those metadata cache misses were satisfied from the backing cache and no disk reads were required, just a write and wait of the oldest dirty buffer.

As with most journaling file systems, log file writing dominates over metadata IO, this is expected for all workloads that update a significant amount of metadata.

IBM

DASD IO I: Looking For Bottlenecks

- The first step to good zFS performance is a properly sized user file and metadata/backing caches
  - These reduce disk IO rates making less stress on the channels, control units and DASD
- Another source of response time degradation:
  - High-frequency file systems are all located on the same channel, control unit and/or DASD,
    - AND
  - The rate of IO is causing too much contention on those devices.
- **RMF** provides reports which can be used to check for DASD, control unit and channel contention and guidelines for resolving DASD issues:
  - **Chapter 4** of ***z/OS RMF Performance Management Guide*** describes how to diagnose DASD contention issues in detail
  - RMF is preferred over the zFS queries for analyzing DASD performance but:
  - zFS queries can help, by identifying the file systems that are causing the most IO such as:
    - **F ZFS,QUERY,IOBYDASD** - Shows zFS rates and average IO wait time per DASD volume
    - **F ZFS,QUERY,LFS** - Shows DASD IO rates per file system and overall average IO wait time for zFS tasks
    - RMF has this zFS information in its reports too, so you could exclusively use RMF

17

The performance monitoring of the DASD sub-system is outside the scope of this presentation, a good reference is chapter 4 of the z/OS RMF Performance Management Guide. This chapter has much information on the hardware concepts and shows example RMF reports and how to analyze the data.

This does not mean that the zFS queries could not be of some help. Generally speaking, the zFS DASD reports should echo some of the same results that an RMF report would show, and the zFS reports can show the file systems that are causing the DASD activity. Note that RMF has zFS performance information available too, and thus RMF could be used exclusively to monitor DASD performance of zFS file systems.

DASD IO II: F ZFS,QUERY,IOBYDASD

zFS I/O by Currently Attached DASD/VOLs

DASD VOLSER	PAV IOs	Reads	K bytes	Writes	K bytes	Waits	Average Wait
------	---	----------	----------	----------	----------	----------	------------
INFON7	2	0	0	86101	1094272	34269	11.675
INFON5	2	0	0	88480	1167848	34398	7.619
INFON3	2	0	0	82965	1066328	32128	11.436
INFON1	2	0	0	92100	1160816	37986	11.130
INFO01	2	0	0	54	480	17	3.130
INFON8	2	0	0	82161	1046104	31950	7.649
INFON6	2	0	0	85081	1089512	33985	7.087
INFON4	2	0	0	92351	1144528	36431	8.025
INFON2	2	0	0	86966	1150952	29270	14.761

Total number of waits for I/O: 270434

Average wait time per I/O: 9.844

- zFS Average Wait is total wall clock time a task wait for an IO in zFS.
  - It is not the same as DASD response time, though it is influenced by it.
    - The IO could be in-progress by the time a zFS task decides to wait, making the ZFS time shorter than DASD response time.
    - This is wall clock time, so it includes all processing by z/OS, any queues, the channels, DASD, the time to dispatch the waiting task, so it can also be longer than DASD response time.

18

The zFS IOBYDASD shows the IO request rates made by zFS to the DASD volumes, the number of times a task decided to wait on an in-progress IO and the average wall clock wait time for tasks waiting on IO completion. This report can be used to identify high frequency volumes and excessive wait times.

zFS uses an internal IO queue for each DASD volume to queue IOs in excess of what the DASD can handle in parallel.

The PAV IOs field is not the same number as the amount of IOs the DASD can handle in parallel. This number is the number of non-priority IOs that zFS will send to the disk. Once this number is in-progress zFS will queue any additional IOs to that DASD volume that are low-priority. If a high-priority IO arrives for the DASD volume and the PAV IOs is reached (2 in this example) but the device PAV value has not been reached, the high priority IO will be sent immediately to DFSMS. If zFS has submitted the device PAV value IOs and those IOs are not complete yet, any IO, be it high priority or low-priority is queued; though, high priority IOs are always queued ahead of low priority IOs.

This queueing allows zFS to ensure high priority IOs are handled ahead of low-priority IOs and allows zFS to merge adjacent IOs and cancel an IO if it likes.

A high priority IO is any IO that a task has decided to wait on, or has indicated it will soon be waiting on.

DASD IO III: F ZFS,QUERY,LFS – IO by aggregate

zFS I/O by Currently Attached Aggregate

DASD VOLSER	PAV IOs	Mode	Reads	K bytes	Writes	K bytes	Dataset Name
------	---	----	----------	----------	----------	----------	------------
INFO01	2	R/W	0	0	54	480	OMVS.ZFS.ROOT
INFON1	2	R/W	0	0	92100	1160816	NOTEBNCH.MAIL.INFON1
INFON2	2	R/W	0	0	86966	1150952	NOTEBNCH.MAIL.INFON2
INFON3	2	R/W	0	0	82962	1066184	NOTEBNCH.MAIL.INFON3
INFON4	2	R/W	0	0	92332	1144272	NOTEBNCH.MAIL.INFON4
INFON5	2	R/W	0	0	88480	1167848	NOTEBNCH.MAIL.INFON5
INFON6	2	R/W	0	0	85081	1089512	NOTEBNCH.MAIL.INFON6
INFON7	2	R/W	0	0	86091	1094144	NOTEBNCH.MAIL.INFON7
INFON8	2	R/W	0	0	82146	1045976	NOTEBNCH.MAIL.INFON8
------			----------	----------	----------	----------	
9			0	0	696212	8920184	*TOTALS*

- This report shows the DASD IO rate by aggregate, and also lists the first DASD volume the file system is contained on.
- This can be used along with the RMF, DFSMS and F ZFS,QUERY,IOBYDASD to locate the high usage file systems on the hardware with high contention

IBM

Lock Contention I: Overview

- Like any parallel product, ZFS has locks to protect common resources
- zFS allows tasks in parallel to write to same file in certain cases
- zFS locks a directory in write mode for a directory update, read mode for reads
- zFS file systems have common structures which have locks, which could cause contention
- Administrators have little control over contention:
  - Cannot control what an installed application might do
  - Or where it wants its files and directories located
  - But might be able to help in some cases:
    - When possible, try to have high-usage applications use separate directories to place files in (to avoid directory lock contention)
    - Even better, use different file systems to avoid lock contention altogether since file systems have common structures like log files that could have contention on them.
- **F ZFS,QUERY,LOCK** – shows lock contention

20

An administrator cannot often control lock contention since they often install applications that were not written at their site and have to place files and directories as dictated by the application. But whenever a user has control over the placement of data or how an application behaves, the best performance will always be obtained when high-usage parallel applications or tasks write to separate file systems (avoids any lock contention on shared file system structures) or if that is not possible, separate directories and files to avoid contention on the individual objects (could still have lock contention on file system shared structures such as the log file, the file system free space manager etc…).

IBM

Lock Contention II: F ZFS,QUERY,LOCK

```
Untimed sleeps:     5947   Timed Sleeps:      0   Wakeups:      2381

Total waits for locks:                 3009481
Average lock wait time:                  1.462 (msecs)

Total monitored sleeps:                   5930
Average monitored sleep time:            1.584 (msecs)

Total starved waiters:          132
Total task priority boosts:       0

          Top 15 Most Highly Contended Locks
   Thread    Async     Spin
    Wait     Disp.    Resol.    Pct.     Description
  --------  --------  --------  -----   -----------
  2922763        0      1633  89.962%  Vnode lock
    69421        0     15503   2.612%  Log system map lock
     5378     1515     61692   2.110%  Transaction-cache main lock
     5109        0     56429   1.893%  Transaction-cache complete list lock
     2711    31041      6120   1.227%  Vnode-cache main lock
    11946     9598      7440   0.892%  Metadata-cache main lock
....
 Top 15 Most Common Thread Sleeps
Thread Wait    Pct.      Description
 ----------    -----     ----------
     5925     99.916%    Transaction GC wait
        5      0.84%     OSI cache item cleanup wait
        0      0.0%      CTKC user file pending IO wait
```

Shows task lock waits and waits for events to occur and average wait time in milliseconds

Most highly contended locks - used by zFS level-2.

Sleeps are like lock waits, the task is waiting for something to occur (in this case, waiting to begin a transaction to update disk)

21

The lock contention report is used when diagnosing issues with excessive lock contention. This report, often used by IBM level-2 to analyze a customer situation, can show areas of improvement for zFS. zFS has reduced lock contention via fixes in the service stream if the fix is small, and made larger improvements on release boundaries. zFS is continually reducing lock contention with each release.

zFS does not always make a task wait when there is contention for a lock. It will try two techniques to avoid a task suspension (and processor context switch):

- Short term spinning – it will loop a short time, this is very effective for locks held for a short time
- Asynchronous queuing – zFS will queue its resource update request to current lock holder and let them make the update when possible, this depends on the nature of the update.

IBM

Lock Contention III – Locks You Might be Able To Configure Around

•Calculating average lock wait time per zFS call:

Total Lock Waits (from QUERY,LOCK) / *TOTALS* (of Count field from Query,KNPFS reports) X Avg. Lock Wait Time (QUERY,LOCK)

- Generally speaking, the only lock contention an administrator can control are when:
  - Above value is a large percentage of the average zFS response time (shown in *TOTALS* line of QUERY,KNPFS). AND the locks with contention are:
    - **Vnode Lock/Vnode Cache Access Lock** – Generally means many applications are consistently hitting the same directory
    - **Log System Map Lock/Anode fileset handle…/Anode bitmap….** – If you see any lock with these characters in their name, shows high lock contention on a particular file system.

      → Check F ZFS,QUERY,FILESETS and your application to determine if you can identify high contention directory and/or file systems.
      → If unable to identify, talk to IBM and we can help identify them via dumps. An F ZFS,DUMP will be all the documentation needed.

22

The administrator can only control zFS lock contention on file system specific structures if they partition their workload to use separate file systems (though you often do not need more DASD space, you simply split work amongst two datasets instead of one, where each dataset is smaller than the original).

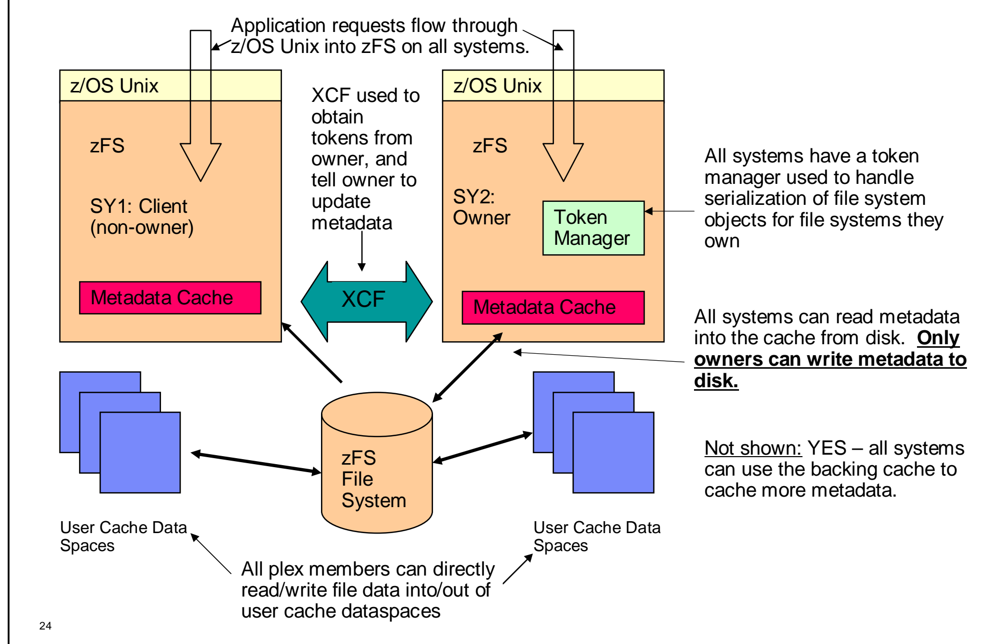
If these locks are high contenders and you cannot figure out which directories and/or file systems are receiving the contention, you can issue an F ZFS,DUMP and provide that information to IBM to get an exact answer. If some other lock is a high contender AND lock wait time is a high percentage of response time AND the zFS response time is deemed inadequate, then you can contact IBM level-2 for guidance.

Additional Items

- **Large Directory Performance Non-optimal (FIXED IN z/OS 2.1)**
  - zFS uses linear search to find names in a directory
  - zFS has sub-optimal directory performance in general:
    - >50,000 names in a directory (@4MB in size) – must use HFS
    - >20,000 names in a directory (@2MB in size) – might want to use HFS
    - zFS greatly outperforms HFS for file IO, so need to factor in the file IO rates vs. directory IO rates for a file system that has larger directories in it and make a choice
    - **Largedir.pl** tool available to find directories not suitable for zFS at **http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html**
      - Takes a long time to run for a whole system, may want to focus it on suspect file systems
  - **z/OS 2.1 zFS provides the ability to have over 2 billion names in a directory.**
  - In the meantime – keep those metadata and backing caches big to avoid disk IO
- **z/OS Unix Sysplex Sharing**
  - **Tuning zFS in this environment is the same as single system tuning**
    - Follow the guidelines presented in the prior slides of this presentation
  - ***z/OS UNIX System Services Planning Guide*** contains information on z/OS Unix Sysplex Sharing Tuning:
    - Try to ensure ownership of file system matched to the system that does the most requests to that file system
    - Use UNMOUNT for system specific file systems in case of a crash to avoid movement to a system that will never access that file system.
    - Use AUTOMOVE for non-system specific file systems so they are moved if a crash occurs.
    - Refer to the appropriate z/OS documentation for more information.

23

The largedir.pl tool is available from the listed web site. It is written in prl and takes a long time to run on an entire file system tree, better to run it against a smaller portion of the tree, such as a suspect file system. An alternative would be to simply to issue a find command and use the –exec option to issue an ls –l command against the name and pipe the output to a file. Any directory whose size is 4MB or more will definitely suffer the slow search time and likely is a better candidate for HFS. Anything over 2MB in size is potentially better served by HFS, though at this size one needs to determine if the file system is heavy on file IO. zFS greatly outperforms zFS for file IO, so that has to be factored into the decision.



With z/OS 11 and significantly enhanced (and simplified for the administrator) in z/OS 13, a file system can be mounted in RWSHARE mode, which means it will be using zFS sysplex sharing instead of z/OS Unix Sysplex Sharing. RWSHARE refers to a file system using zFS sysplex sharing and NORWSHARE refers to a file system using z/OS Unix Sysplex Sharing.

With zFS sysplex sharing, all plex members can read and write files into their user caches to avoid calling the owner for ever file read or write request. z/OS 13 zFS has full asynchronous write-behind and read-ahead for all plex members, regardless of who owns the file system. All plex members can also directly read metadata, such as directory contents and other control information related to files and directories; however, only sysplex owners can update metadata and can access the primary file status of an object. All plex members, regardless of owner status, can use the backing cache to cache more metadata for an RWSHARE file system.

Sysplex serialization to the objects in a file system is handled via tokens, which are in effect, sysplex locks. If an application want to process a file F, then if the system does not have a token with the appropriate read or write access, then it has to call the token manager to get one. For owners, this is a simple function call, and for clients, this is an XCF request to obtain them. Once the token with proper access rights is obtained, that system can cache data for that object. For clients, this means that XCF messages to owners are avoided since they have data in their cache. Clients have full read-ahead and write-behind logic for files; for directories they still send synchronous update requests to the owner since the owner is only allowed to update metadata.

IBM

zFS Sysplex Sharing II: RWSHARE Summary (z/OS 13)

- **Performance Compared to z/OS Unix Sysplex Sharing (NORWSHARE)**
  - **Large File (database) Random Update Workload:**
    - This workload randomly updates a large file, similar to a database access.
    - **9X faster** on non-owners with R13 RWSHARE as opposed to R12 NORWSHARE.
  - **Sequential File Creation Workload:**
    - This workload creates many sequentially written files (common write pattern in the field)
    - **16X faster** on non-owners with R13 RWSHARE as opposed to R12 NORWSHARE.
  - **Directory Update Workload (Significantly improved with z/OS 2.1):**
    - This workload has many processes repeatedly adding, removing, renaming and searching for files in a directory, not a typical customer environment.
    - **25% faster** on non-owners with R13 RWSHARE as opposed to R12 NORWSHARE.
  - **Cached Directory Read Workloads (Significantly improved with z/OS 2.1):**
    - **15X-20X faster** on non-owners with R13 RWSHARE as opposed to R13 NORWSHARE.
- **Some environments cannot use RWSHARE:**
  - z/OS SMB Server – cannot export file systems that are RWSHARE.
  - Fast Response Cache Accelerator support of the IBM HTTP Server for z/OS V5.3
- **If using file systems created before z/OS 9:**
  - Recommend IBM APAR OA39716 to improve sysplex client performance
- **Areas of Improvements for z/OS RWSHARE Support:**
  - Number of objects that can be cached due to primary address space
    - → This can cause clients to call server more to re-obtain lost tokens
  - Cold startup of servers on non-owners not as fast as desired
    - → If they access lots of objects not already cached at client, need to obtain a token for each new object accessed.

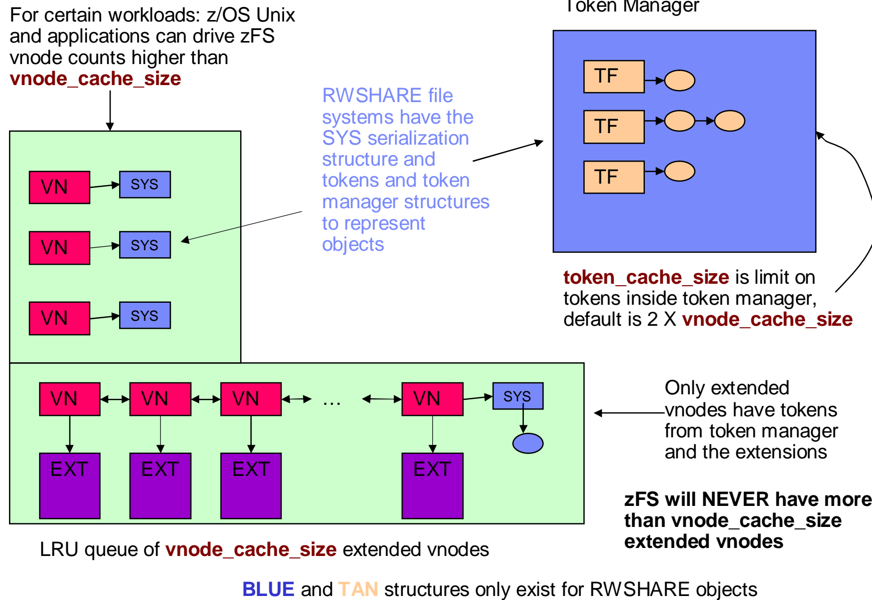
25

zFS sysplex sharing is much, much faster when files are read and written on non-owner systems in a pure z/OS 13 sysplex running RWSHARE, no matter what the access pattern to the file is. Directory update workload (removing files, renaming files etc…) run faster than z/OS Unix Sysplex Sharing, but only by a small amount. The reason is that only the owner can update metadata for an object and the client must send the request synchronously to the owner with the current zFS design.

The environments listed on the slide cannot use zFS sysplex sharing; which is why zFS allows the user to selectively choose which file systems are RWSHARE and which file systems are NORWSHARE. Additionally, because zFS is storage constrained in its primary address space, and because caching of objects and the associated sysplex token with that object is essential to avoiding XCF communications to the server to re-obtain a lock due to cast-out for low-memory, some customer environments might be limited in how many file systems could be RWSHARE.

One potential issue with z/OS sysplex sharing is the fact that due to zFS primary address space constraints, it limits the vnode cache size, and hence limits the number of objects cache-able at clients. Some workloads might need a larger object cache to be successful.

If you migrated to zFS before z/OS 9, you might have some directories that store entries in an old format that does not contain the full FID – inode/uniquifer for the name, this requires the client to query the full FID from the server on lookup requests which can have a significant performance impact. If the user has been using zFS for a long time (before z/OS 9) they might have some of these old



The most recently used objects by applications and z/OS Unix will have vnodes inside the zFS address space. zFS will guarantee that no more than vnode_cache_size vnodes have extended information. This reduces zFS primary address space storage if z/OS Unix/applications allocate more vnodes than vnode_cache_size. Essentially zFS does not have direct control over how many vnodes z/OS Unix will hold, and zFS cannot free the storage for a vnode unless given permission by z/OS Unix. zFS will request that z/OS Unix release vnodes any time they are near or exceed vnode_cache_size but z/OS Unix may not always honor that request or may delay in honoring that request. One thing that forces many vnodes to be allocated is if applications have many open files. The base zFS vnode is approx. 224 bytes and an extended vnode totals approx. 1K in size.

If an application calls zFS for a vnode that does not have an extension, then zFS will steal the extension (un-caching data from user cache, and if a sysplex client, un-caching data from metadata/backing caches) from the oldest vnode in the LRU queue of vnodes with extensions. This is extra path length overhead, but should not occur very often since the tendency of applications is to continue working with the same files and directories for a period (sometimes very long) of time.

For the most recently accessed objects in RWSHARE file systems, each vnode, whether its extended or not, will have a sysplex serialization structure attached to it, this structure is approx. 120 bytes. Extended RWSHARE file system objects will also likely have tokens, though those are fairly small objects. With RWSHARE, there is a token manager on each system that tracks the tokens held by the various client systems for file systems owned by that system. This adds more storage in the zFS primary address space, each object tracked by the token manager uses approx. 228 bytes of storage and each token held by a client system requires approx. 100 bytes. An RWSHARE object will likely use over 50% more memory than a NORWSHARE object inside zFS. This means that since zFS is constrained in its primary address space, it can cache less objects with RWSHARE file systems than with NORWSHARE file systems.

IBM

Object Caching II: Vnode Cache/Token Cache Recommendations

- **NORWSHARE File Systems and file systems mounted R/O:**
  - **vnode_cache_size** not as important to tune because if a vnode does have an extension, or needs to be newly created, we can steal from the oldest in the LRU queue, and we can quickly instantiate the vnode from the metadata cache.
    - If the status information for the object is not in the metadata cache it will require a disk read. → **So invest in metadata/backing cache storage.**
  - A vnode cache miss often just uses a bit more CPU.
  - Tune vnode_cache_size last – Ensure user file and metadata caches optimally tuned.
- **RWSHARE File Systems:**
  - **vnode_cache_size** is much more important, especially for sysplex clients.
    - **If a vnode does not have an extension or does not exist** in the cache for the desired object, it does not have a token, which means one will have to be obtained from the token manager. **For clients it means an XCF communication**.
    - Due to storage constraints, its likely dangerous to push the vnode_cache_size much past 100,000 in size. The default is 32,768.
    - **Best to selectively choose the best candidate file systems for RWSHARE usage (highest usage file systems accessed by more than one plex member at a time)**
    - → ftp://public.dhe.ibm.com/s390/zos/tools/wjsfsmon/wjsfsmon.pdf - this tool will show which R/W mounted file systems are accessed by more than one plex member
  - **token_cache_size** – The default of double the vnode_cache_size is likely sufficient in many cases.
    - If your plex has a large number of members, increase it to reduce garbage collection.

27

The token manager token limit is honored as much as possible. Any time the token manager is approaching its limit it will invoke garbage collection (even making new token requests wait for garbage collection to complete) and reclaim tokens proportionally form the other plex members based on how many tokens each plex member holds. A plex member will process a garbage collection request by stealing from the least recently used vnodes in its cache.

IBM

Object Caching III: F ZFS,QUERY,LFS – Vnode Cache Statistics

zFS Vnode Cache Statistics

Vnodes	Requests	Hits	Ratio	Allocates	Deletes
43119	11868292	8616915	72.605%	0	46880

zFS Vnode structure size: 224 bytes
zFS extended vnodes: 32768, extension size 724 bytes (minimum)
Held zFS vnodes: 220 (high 43051) Open zFS vnodes: 15 (high 8080) Reusable: 38940

LRU queue items: 32768
Total osi_getvnode Calls: 3421429 (high resp 0) Avg. Call Time: 0.005 (msecs)
Total SAF Calls: 116543153 (high resp 0) Avg. Call Time: 0.001 (msecs)

z/OS Unix pushed zFS past its limit, 43,119 base vnodes exist, only 32K have extensions

This is vnode_cache_size, they are using default of 32K vnodes

Number of vnodes held by z/OS Unix (currently) and high-water mark, including number of open files and high water mark for open files

This monitors the security product performance, important for response time to be just a few microseconds.

Larger response times likely due to excess auditing or an issue with the security product.

Hit ratio in this report not so important to monitor, it will vary greatly in cases where many new objects are accessed. Other reports will show information related to object caching shown later.

28

This is an example of a case where z/OS Unix and applications force zFS to allocate more vnodes than the vnode_cache_size. zFS still limits the number of extended vnodes to the vnode_cache_size limit and uses an LRU queue to determine which vnode to re-use if a vnode does not exist or does not have an extension.

IBM

Sysplex Statistics I: F ZFS,QUERY,KNPFS - Sysplex Client Summary

PFS Calls on Client

Operation	Count	XCF req.	Avg Time
---------	----------	----------	----------
zfs_opens	885098	0	0.020
zfs_closes	885110	0	0.010
zfs_reads	12079	0	0.157
zfs_writes	0	0	0.000
zfs_ioctls	0	0	0.000
zfs_getattrs	2450523	8	0.009
zfs_setattrs	313031	656	0.020
zfs_accesses	11495	0	0.018
zfs_lookups	**13764811**	**1190897**	**0.287**
zfs_creates	876507	876556	5.625
zfs_removes	1240556	1240621	2.117
zfs_links	157216	157216	2.567
zfs_renames	155164	155165	1.890
zfs_mkdirs	157971	157971	6.031
zfs_rmdirs	**155108**	**155109**	**2.164**
zfs_readdirs	11322	3053	11.345
zfs_symlinks	157398	157398	4.295
zfs_readlinks	68	58	0.871
zfs_fsyncs	0	0	0.000
zfs_truncs	0	0	0.000
zfs_lockctls	0	0	0.000
zfs_audits	33	0	0.015
zfs_inactives	2698174	0	0.020
zfs_recoveries	0	0	0.000
zfs_vgets	0	0	0.000
zfs_pfsctls	0	0	0.000
zfs_statfss	0	0	0.000
zfs_mounts	0	0	0.000
zfs_unmounts	0	0	0.000
zfs_vinacts	0	0	0.000
---------	----------	----------	----------
TOTALS	23931664	4094708	0.602

- Lookup requests have over 1 million XCF calls, likely to get token for a vnode not found in cache. Could make vnode_cache_size larger if memory permits to try and reduce these.
- But due to client caching, over 12 million lookup requests satisfied by client metadata/vnode cache.

Directory operations are sent synchronously to server.

29

The KNPFS report will also show call counts for file systems that are not locally owned and are mounted RWSHARE. In this case, the XCF Req. column shows the number of operations that required one or more calls to the owner of the file system. For a directory update operation it will generally be 1-1. For read operations there will typically be a large amount of caching at sysplex clients to reduce XCF call traffic. This is shown in the open/close/getattr/lookup/read rows since those are all read operations and are generally getting a good number of cache hits. Note that since some lookup calls (which are directory searches to find a name in a directory) result in XCF calls, a larger vnode_cache_size, if possible might reduce those cases and improve lookup performance.

Note that this is a canned performance test workload and not representative of a real world customer workload. Typical customer workloads do very light directory update requests and much, much more file IO, but this report does show a good example of how a possible tuning improvement might help the workload.

IBM

Sysplex Statistics II: F ZFS,QUERY,STKM – Token manager statistics

```
Server Token Manager (STKM) Statistics
          -------------------------------------
Maximum tokens:      200000     Allocated tokens:      61440
Tokens In Use:        60060     File structures:       41259
Token obtains:       336674     Token returns:        271510
Token revokes:       125176     Async Grants:             64
Garbage Collects:         0     TKM Establishes:           0
Thrashing Files:          4     Thrash Resolutions:      131

                 Usage Per System:
System      Tokens   Obtains  Returns   Revokes   Async Grt Establish
--------  --------- --------- --------- --------- --------- ---------
DCEIMGHR     18813    161121    134907     70275         0         0
ZEROLINK         0     66055     66054         5        64         0
LOCALUSR     41247    109499     70549     54974         0         0
```

Shows token limit, number of allocated tokens, number of allocated file structures and number of tokens allocated to systems in plex

Number of times tokens had to collected from plex members due to tokens reaching limit – if high then might want to update token_cache_size

Thrashing files indicates objects using a z/OS Unix-style forwarding protocol to reduce callbacks to clients – check application usage

- Shows tokens held per-system and number of token obtains and returns since statistics last reset.
- ZEROLINK – pseudo-sysplex client used for file unlink when the file still open – used to know when file fully closed sysplex-wide to meet POSIX requirement that a file's contents are not deleted, even if its been unlinked, if processes still have file open.

30

The STKM report shows token usage for file systems owned by the plex member. In terms of zFS tuning, there are two things to look for:

•Excessive garbage collection – If the garbage collection count is high, then the plex is desiring more tokens than the token limit. If there is room in the zFS address space, try increasing the token_cache_size to reduce garbage collection (which is simply extra system overhead).

•Thrashing Files – Indicates that more than one plex member is attempting to access the same object and at least one of them is continually writing. In this case zFS will use an access protocol very much like z/OS Unix does for its sysplex sharing, but just for the thrashing object. This reduces/eliminates token revoke callbacks and yields better performance. Unfortunately, zFS does not list the thrashing objects here, there is no command to show which directories or files have continued sysplex contention.

IBM

Sysplex Statistics III: F ZFS,QUERY,CTKC

SVI Calls to System **PS1**

SVI Call	Count	Avg. Time
GetToken	**1286368**	**1.375**
GetMultTokens	0	0.000
ReturnTokens	26	0.050
ReturnFileTokens	0	0.000
FetchData	0	0.000
StoreData	540	1.566
Setattr	0	0.000
FetchDir	7140	6.291
Lookup	0	0.000
GetTokensDirSearch	0	0.000
Create	1320406	3.736
Remove	1499704	1.595
Rename	166498	1.448
Link	169176	1.549
ReadLink	0	0.000
SetACL	0	0.000
…..		
FileDebug	0	0.000
TOTALS	4449858	2.167

Shows requests a plex member sends to other plex members for objects in file systems owned by other members and average response time in milliseconds. Includes XCF transmission time.

Might be able to reduce GetToken calls by raising **vnode_cache_size** (if zFS primary storage allows it)

Sysplex Statistics IV: F ZFS,QUERY,SVI

SVI Calls from System **PS2**

SVI Call	Count	Qwait	XCF Req.	Avg. Time
GetToken	1286013	0	0	0.259
GetMultTokens	0	0	0	0.000
ReturnTokens	26	0	0	0.050
ReturnFileTokens	0	0	0	0.000
FetchData	0	0	0	0.000
StoreData	540	0	0	0.081
Setattr	0	0	0	0.000
FetchDir	7140	0	0	4.997
Lookup	0	0	0	0.000
GetTokensDirSearch	0	0	0	0.000
Create	1321096	0	0	2.371
Remove	1499689	0	177	0.645
Rename	166500	0	0	0.509
Link	169608	0	0	0.538
ReadLink	0	0	0	0.000
SetACL	0	0	0	0.000
….				
LkupInvalidate	0	0	0	0.000
FileDebug	0	0	0	0.000
TOTALS	4450612	0	177	1.044

Shows calls received by indicated plex member:

- Qwait non-zero when all server tasks are busy
- XCF Req. means server had to reclaim tokens from other plex members to process request.
- Avg. Time in milliseconds shown for server to process request.

IBM

Going Forward.

- **A valuable monitoring process:**
  - If possible at your site, issue:
    - **F ZFS,QUERY,ALL**
    - **F ZFS,RESET,ALL**
  - Every 30 minutes or so
    - → Now zFS job output and system log have a running history of zFS performance, good to look back at a reported performance problem, very useful for IBM level-2 if a performance problem exists.
- **IBM working on solutions to:**
  - **Directory scale-ability fixed in z/OS 2.1**
  - Make more intelligent cache defaults for zFS, based on system memory
  - Improve queries,
    - Example: showing thrashing objects in a sysplex
  - Improve scale-ability by:
    - Reducing amount of storage required to track and cache objects and tokens for RWSHARE
    - Run zFS in 64 bit mode to eliminate primary address space storage constraints which prevent customers from running with really big caches, particularly vnode caches for RWSHARE.
    - Reduce lock contention on file system specific structures in high directory write workloads.

33

IBM

z/OS 11 and 12 vs. z/OS 13 zFS

z/OS 11 and 12 RWSHARE specific support:

- **Reduced caching capacity** – sysplex clients cannot store directory contents in backing cache
- **Do not support write-behind or direct disk IO for sysplex clients**
  - As a result have reduced performance
  - Stress owners more
- **Store user file data in a separate set of data spaces than user cache:**
  - Called **client_cache_size**
    - → Must tune both user_cache_size and client_cache_size and estimating amount of memory to assign to locally owned access and sysplex client access
- **Do not handle thrashing directories quite as well as z/OS 13**

z/OS 11 and 12:

- **Partition directory data from metadata on owner systems, single systems and for NORWSHARE systems, placing in a cache called the directory cache:**
  - Tune via **dir_cache_size**
  - There is no dynamic tuning for directory cache, requires zFS restart
  - Should define this to be larger and metadata cache to be smaller to make directory operations more efficient for these releases and avoid data copying.

34

Generally speaking, it is highly recommend to migrate to z/OS 13 when using RWSHARE sysplex support, it simply performs much better and is easier to tune. The R11 and R12 support does not allow caching of metadata in the backing cache for objects that are not owned on the local system, this can greatly reduce the amount of directory buffers a sysplex client system could cache. The R11 and R12 support also does not support write-behind and most file writes are going to be synchronous calls to the owner. Any time file data is missing from a client cache it has to request that data from the server, which means file contents flowing over the XCF channels and the server having to spend CPU to obtain and return that data. The R11 and R12 sysplex code requires that the administrator specify how much cache storage is to be used to hold file data that is remotely owned, and file data that is locally owned via the client_cache_size parameter. This either requires more system memory or could reduce cache hits if a system is more heavily skewed either to local file requests or remote file requests.

Additionally, for all zFS systems: single system, NORWSHARE and RWSHARE, the administrator has yet another tuning option to specify: dir_cache_size which partitions directory metadata from other metadata. zFS on those systems could keep the contents of a directory page in two places: the metadata cache and the directory cache. Since the directory code interfaces directly with the directory cache, its best to ensure dir_cache_size is reasonably large for directory intensive workloads. A miss from the directory cache means a search of the metadata cache and a miss there means a disk read into the metadata cache and then a memory transfer of the block to the directory cache. The directory cache cannot be dynamically configured via zfsadm config. Because z/OS 13 removed this cache it performs slightly better than z/OS 11 and 12 for directory operations.

IBM

Publications of Interest

- z/OS UNIX System Services Planning (GA22-7800)
  General Administration of z/OS UNIX file systems

- z/OS Distributed File Service zSeries File System Administration (SC24-5989)
  zFS Concepts and zfsadm command for zFS

- z/OS Distributed File Services Messages and Codes (SC24-5917)
  IOEZxxxt messages and X'EFxxrrrr' reason codes for zFS

- z/OS RMF Performance Management Guide (SC33-7992)
  Describes how to monitor DASD performance

IBM

System z Social Media Channels

- Top Facebook pages related to System z:
  - IBM System z
  - IBM Academic Initiative System z
  - IBM Master the Mainframe Contest
  - IBM Destination z
  - Millennial Mainframer
  - IBM Smarter Computing
- Top LinkedIn groups related to System z:
  - System z Advocates
  - SAP on System z
  - IBM Mainframe- Unofficial Group
  - IBM System z Events
  - Mainframe Experts Network
  - System z Linux
  - Enterprise Systems
  - Mainframe Security Gurus
- Twitter profiles related to System z:
  - IBM System z
  - IBM System z Events
  - IBM DB2 on System z
  - Millennial Mainframer
  - Destination z
  - IBM Smarter Computing
- YouTube accounts related to System z:
  - IBM System z
  - Destination z
  - IBM Smarter Computing
- Top System z blogs to check out:
  - Mainframe Insights
  - Smarter Computing
  - Millennial Mainframer
  - Mainframe & Hybrid Computing
  - The Mainframe Blog
  - Mainframe Watch Belgium
  - Mainframe Update
  - Enterprise Systems Media Blog
  - Dancing Dinosaur
  - DB2 for z/OS
  - IBM Destination z
  - DB2utor

© 2012 IBM Corporation

This document was created with Win2PDF available at http://www.win2pdf.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.