#SHAREorg

SHARE
Technology · Connections · Results

zFS Diagnosis I: Performance Monitoring and Tuning Guidelines

Scott Marcotte (smarcott@us.ibm.com)

IBM

August 8, 2012 3:00 PM

Session Number 11788

SHARE in Anaheim 2012

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	20-21
Additional Items	22
Sysplex Sharing	23-24
Object Caching	25-27
Sysplex Statistics	28-31
Going Forward	32
z/OS 11/12 Summary	33

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**

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**

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**

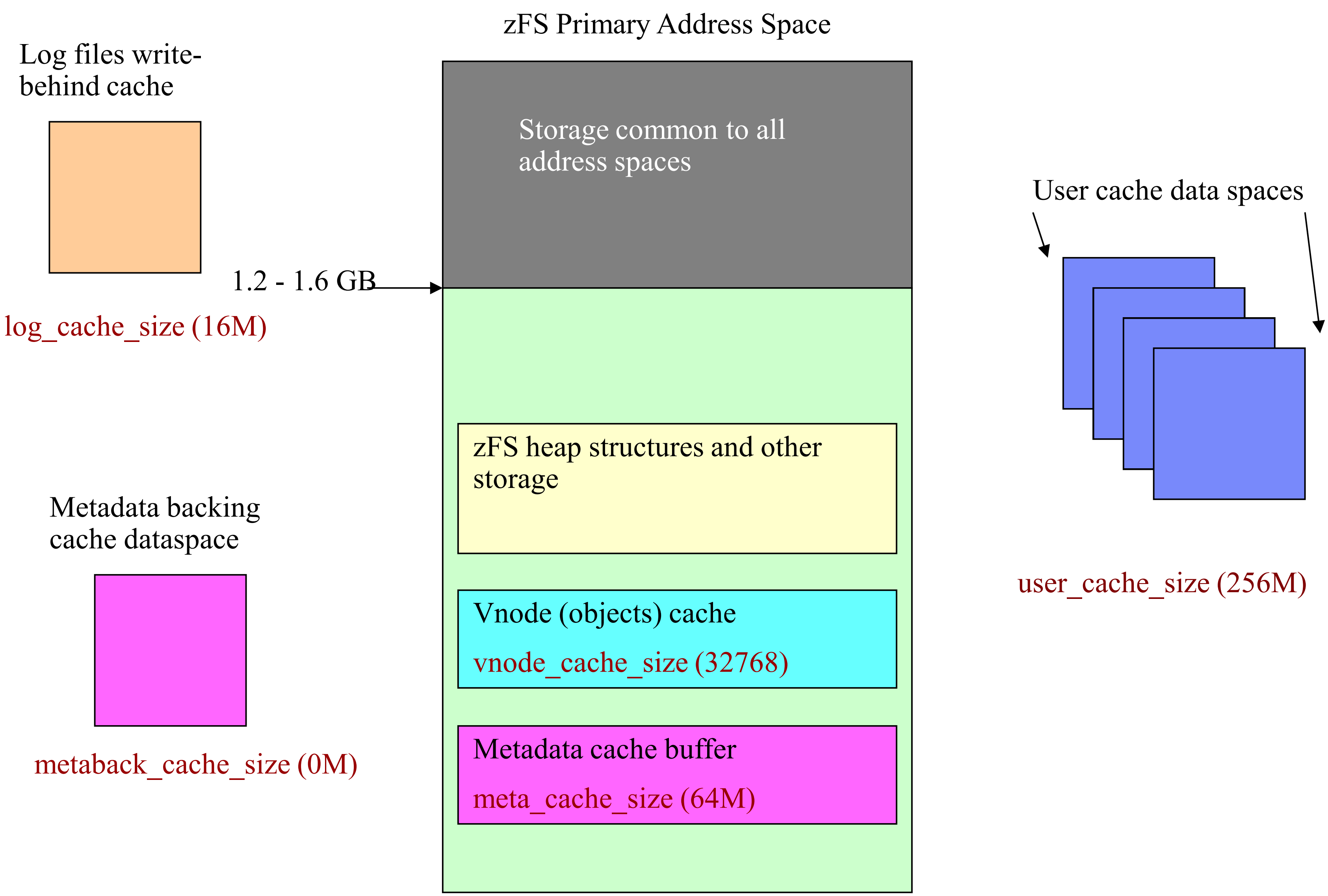
- This report shows all of the calls made to zFS since last statistics reset or since start of zFS
- **Boldface** are write operations

………

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.

Storage I: zFS System Storage Layout (z/OS 13)



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

Storage III: Monitoring zFS Storage continued…

```
Heap Usage By Component

        Storage Usage By Component
        --------------------------
Bytes           No. of  No. of
Allocated       Pieces  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.

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

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

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

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

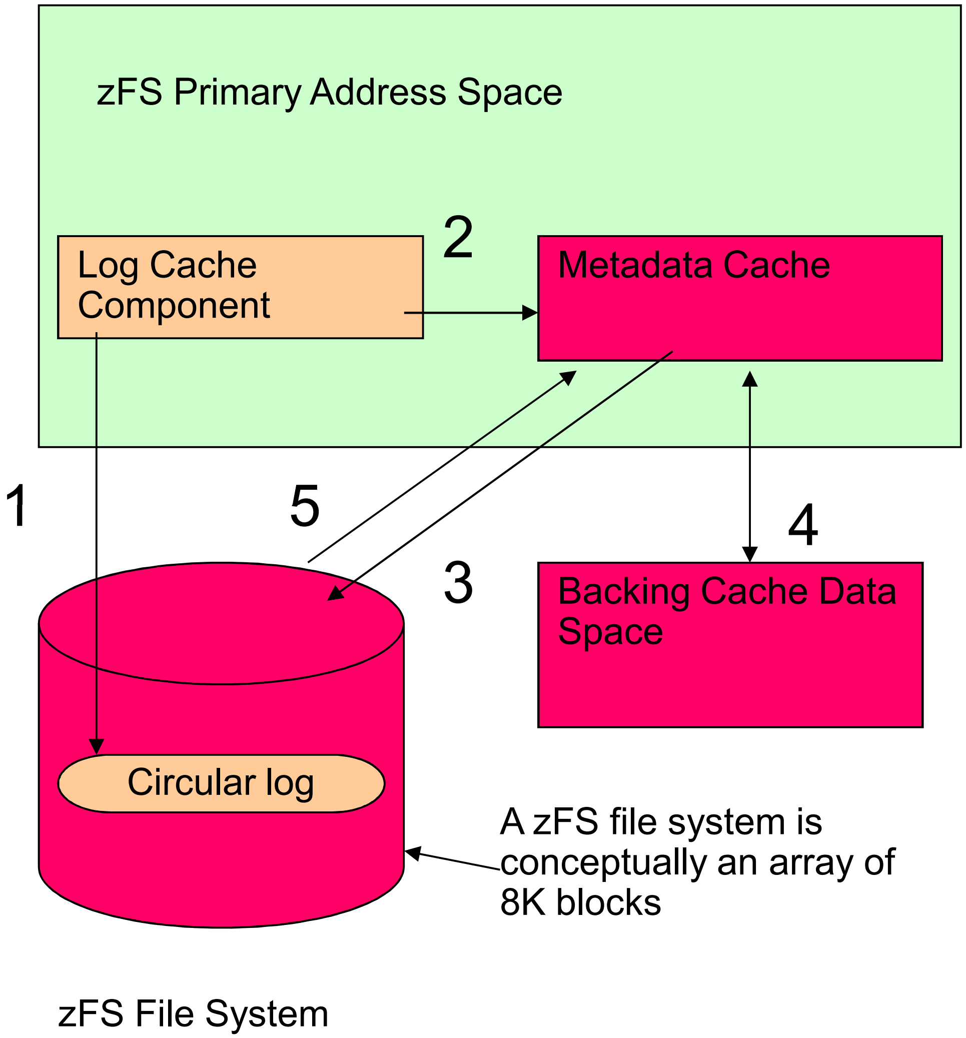
```
Page Management (Segment Size = 64K) ) (Page Size = 4K)
----------------------------------------------------
Total Pages          65536    Free          65451
Segments             16384
Steal Invocations     2405    Waits for Reclaim     0

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.**

Metadata/Backing Cache I: Background



1. Every zFS file system has a circular log file managed by log cache component that contains transactional updates to metadata
2. When the log file becomes full, the log component tells the metadata cache to write out dirty data so the log can be over-written
3. The metadata cache writes out dirty data so that the log can be over-written with new transaction data
4. Any time the metadata cache needs to make room for new data, it casts oldest buffers out to backing cache (if it exists)
   - Will check the backing cache to see if a block exists in that cache to avoid disk reads
5. **If during a read a block is not in backing cache and not in meta cache:**
   - **Will have to read from disk <u>(this is what users have some control over)</u>**

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

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

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.

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

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.

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

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

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

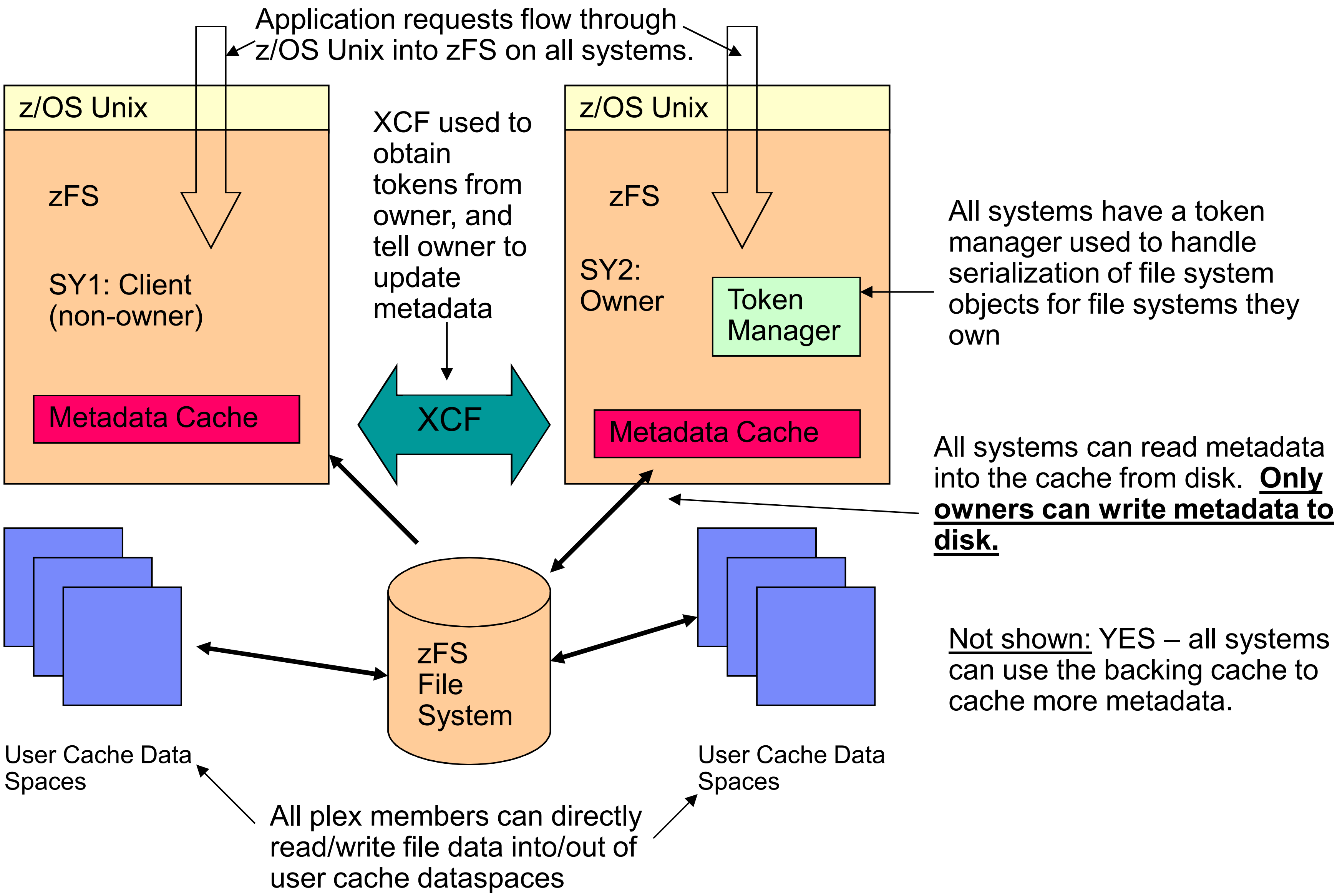
Sleeps are like lock waits, the task is waiting for something to occur (in this case, waiting to begin a transaction to update disk)

Additional Items

- **Large Directory Performance Non-optimal**
  - 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
  - IBM is working on a solution for directory scale-ability
  - 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.

zFS Sysplex Sharing I: RWSHARE Mounted File System (z/OS 13)



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:**
    - 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:**
    - **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

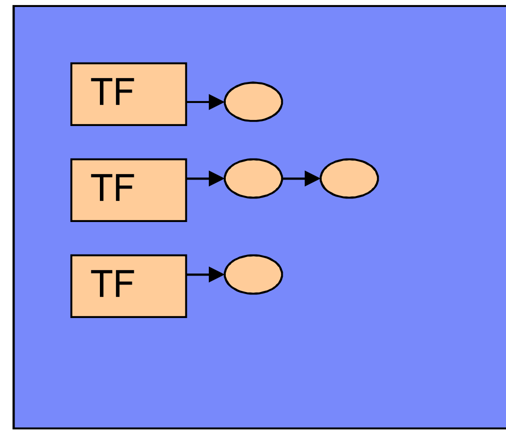
- **If using file systems created before z/OS 9:**
  - Recommend IBM APAR OA39716 to improve sysplex client performance

- **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.

Object Caching I: Vnode and Token Caches Overview

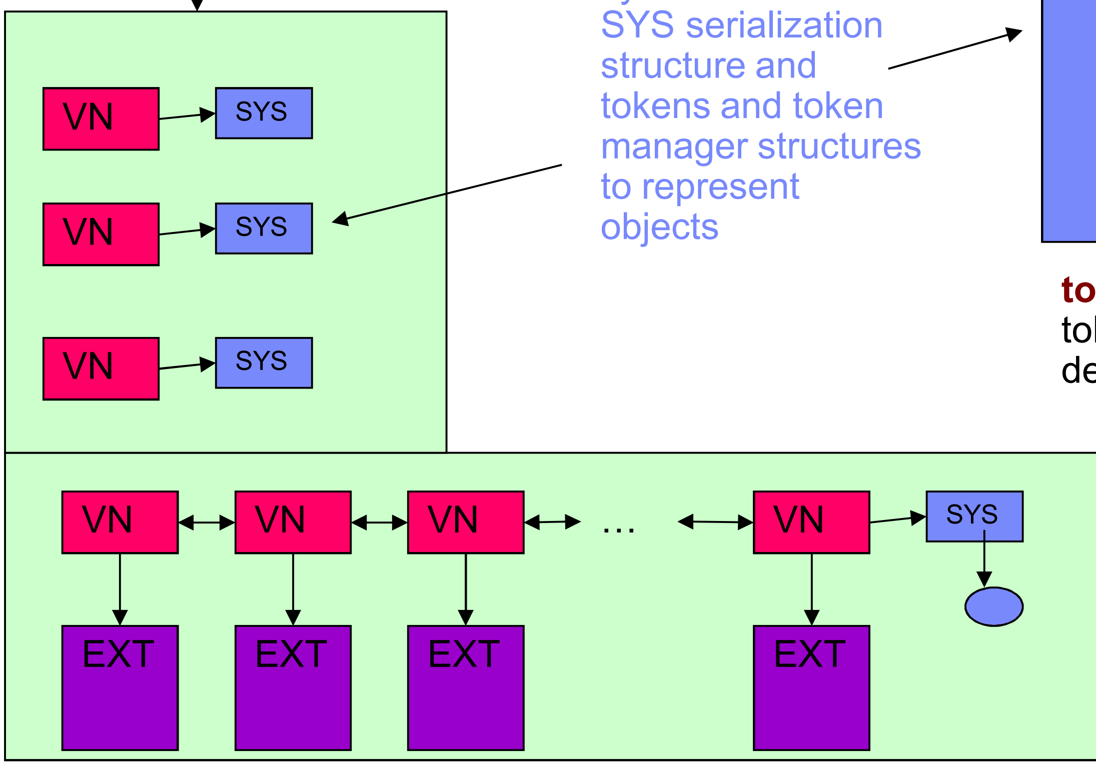
For certain workloads: z/OS Unix and applications can drive zFS vnode counts higher than **vnode_cache_size**

Token Manager



RWSHARE file systems have the SYS serialization structure and tokens and token manager structures to represent objects

token_cache_size is limit on tokens inside token manager, default is 2 X **vnode_cache_size**



Only extended vnodes have tokens from token manager and the extensions

zFS will NEVER have more than vnode_cache_size extended vnodes

LRU queue of **vnode_cache_size** extended vnodes

BLUE and TAN structures only exist for RWSHARE objects

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.

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.

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.

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.

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.

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:**
  - Fix directory scale-ability problems with zFS
  - 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.

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.

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

← QR Code