#SHAREorg

SHARE

zFS Diagnosis II: Problem Determination and File System Monitoring

Scott Marcotte

IBM

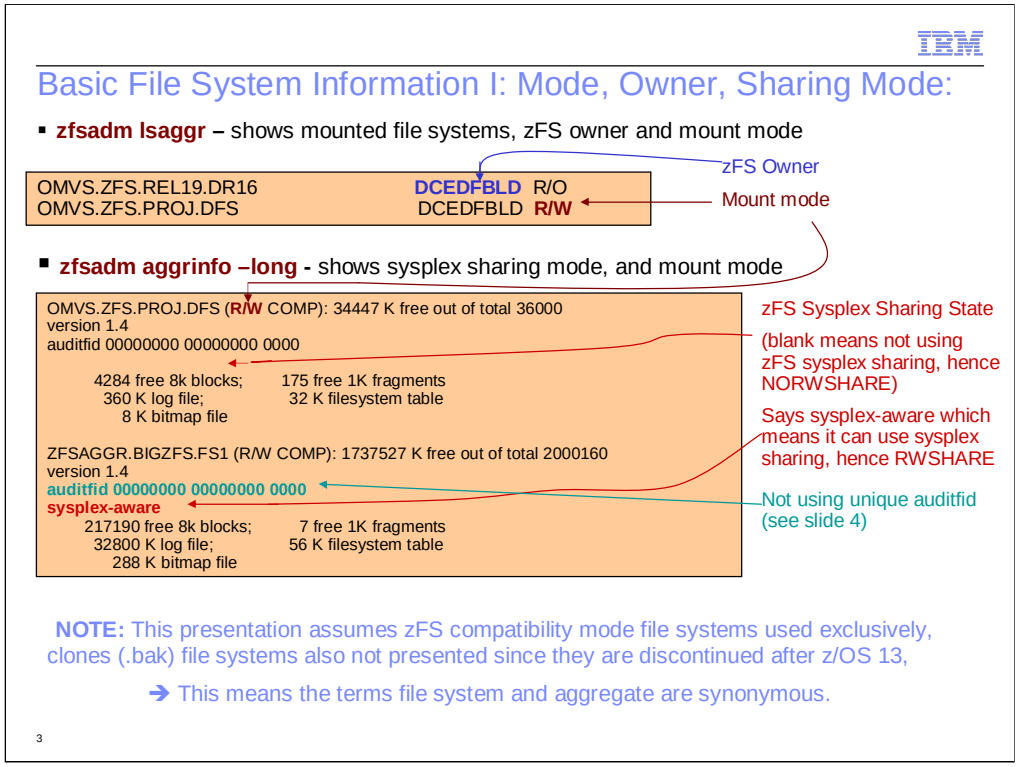
February 7, 2013 11AM
Yosemite B
Session Number 12731

smarcott@us.ibm.com

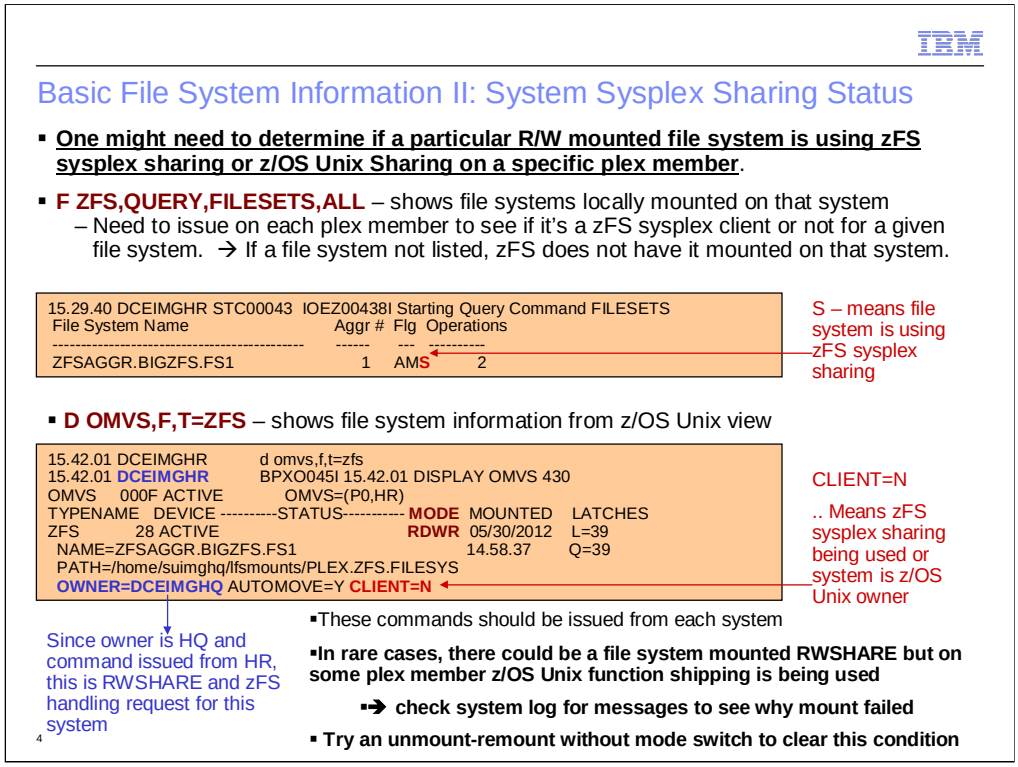
SHARE in San Francisco 2013

Topics

Title	Slides
File System Monitoring	3-4
Auditing	5
Disk Space Monitoring	6-9
File System Backup/Quiesce	10-12
File System Copy	13
Cancel/Force	14
System Outages	15-17
Disk I/O Errors	18
XCF Errors	19-20
Enabling Debugs	21
zFS Software Errors	22-25
System Hangs	26-28
Storage Usage	29-30
z/OS 2.1 Updates	31



In a sysplex, its often important to know if a file system is using zFS sysplex sharing or not. The commands shown on this slide can be used to quickly obtain a list of zFS mounted file systems, their mount mode and current zFS owner. Note that z/OS Unix also has an owner and who that owner is very relevant, particularly for file systems that are not using zFS sysplex support (also called NORWSHARE). For zFS file systems that are mounted RWSHARE, also known as sysplex-aware, knowing the zFS owner is important. Regardless of the sysplex sharing mode, the owner is the system that updates the file system metadata (metadata is any on-disk data that is not the contents of a user file, an example would be the contents of a directory or an ACL). The zfsadm aggrinfo command can also be used to show whether the file system is mounted RWSHARE or not.



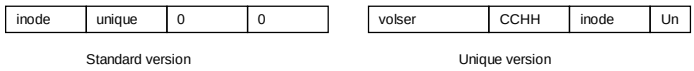
At mount time a R/W mounted file system will be mounted as RWSHARE or NORWSHARE in a sysplex. If the file system is mounted NORWSHARE then z/OS Unix sharing is used and z/OS Unix function ships requests from non-owner systems to owner systems. In this case the CLIENT=Y will be specified for each member that is not the owner, and CLIENT=N will be displayed on the system that is the owner of the file system.

If a file system is RWSHARE then zFS on each sysplex member will receive a mount of the file system. The first system receiving the mount will become the zFS owner and all other systems will become zFS clients for that file system. In this case the D OMVS,F would show all plex members as CLIENT=N for that file system.

With the F ZFS,QUERY,FILESETS command, any plex member that has successfully processed its mount will result in a listing in this command's output. But if for some reason a plex member failed the mount of the file system, then that file system would not show up in the query,filesets command issued on that member and the D OMVS,F command would show CLIENT=Y for that plex member. In this case we have an RWSHARE file system but on a plex member z/OS Unix is function shipping rather than calling ZFS to do its caching. This leads to extra system overhead and meant that the mount failed. The system log should be examined for messages that could explain why the mount failed, such as IO errors or unavailable DASD or a zFS specific error. An unmount-remount without switching modes might be able to clear the condition to see if the new mounts succeed on this plex member or at least re-do the mount to see why it fails on this plex member.

Auditing

- SMF type-80 records and RACF message ICH408I use a 16 byte identifier to identify an object that had some sort of authorization failure or state change of interest to the operator
- zFS audit identifiers take one of two formats:



Standard version Unique version

  - The standard (default) version is not acceptable for truly identifying the object.
    - → The unique version uses the volume serial and CCHH of the first extent of the file system, plus the inode and the low order part of the uniquifier to identify the object
  - www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty2.html - **auditid** tool available on z/OS Unix Tools website to display the object's pathname but needs the unique audit id version.
  - To ensure the unique version is used you can do the following:
    - **zfsadm format –newauditfid** – This option will store the proper unique audit id inside the new file system and it will be used in auditing records and messages.
    - **zfsadm setauditfid** – This will store the audit fid inside the specified file system.
    - **convert_auditfid=on** – zFS startup parameter will auto-convert any mounted file system not using the unique method to begin using it (also: **zfsadm config –convert_auditfid**).
      - **→On by default in z/OS 2.1 and newly formatted file systems use full auditfid**
  - **zfsadm aggrinfo –long** – will show the audit fid of the file system (the first 10 bytes of audit id)

5

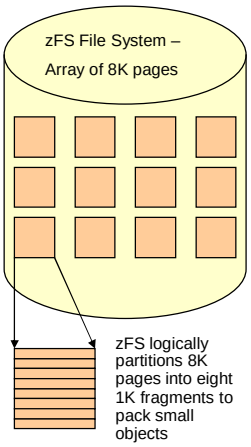
Auditing is used for a number of purposes including important state changes, such as the deletion of an object or an authorization failure on an attempt to access an object by a z/OS Unix process. Audit identifiers are 16 byte objects and in older releases of zFS only the first 8 bytes are used, which show the inode and uniquifier (8 byte fid that identifies an object inside a file system) of the file system object, but does not properly show which file system contains the object. Later releases of zFS change the audit id so that the first 10 bytes show the file system by saving the volume serial and CCHH of the first extent of the file system and also contains the inode and the low order portion of the uniquifier. This allows someone to find out the file system and name of the object of the audit record. There is an auditid tool available in the listed website that will take the auditid and show the full pathname of the file.

To ensure that the better version of the audit id is used, one can specify the –newauditfid option on the zfsadm format and/or IOEAFGFMT command for new file systems. Old file systems can be specifically updated to use the new version via the zfsadm setauditfid or the administrator can enable automatic conversion via convert_auditfid=on zfs startup parameter. Also, the zfsadm config –convert_auditfid command can be used to dyamically enable it without the need for a zFS restart.

The audit fid being used for the file system is shown in the zfsadm aggrinfo –long command.

Disk Space Monitoring I: Disk Layout

- zFS file system space monitoring more complicated due to way files stored on disk:

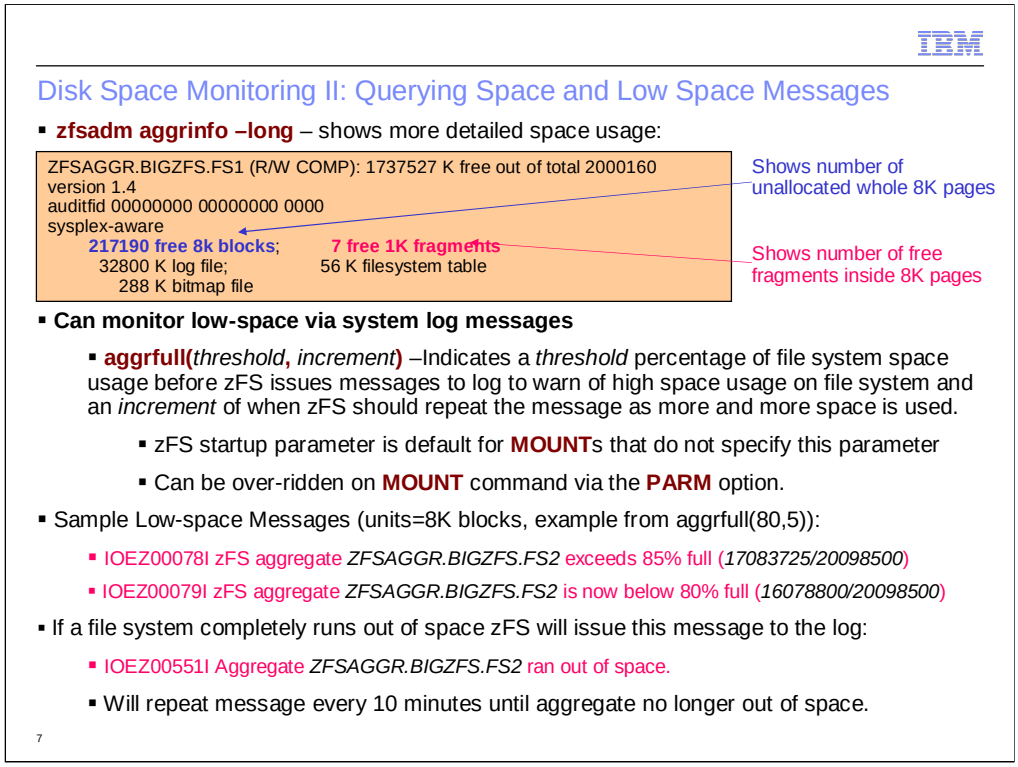


- **File system objects stored 3 ways**:
  - **Inline** – Object never grew bigger than 52 bytes, stored inside the object's inode on disk.
  - **Fragmented** –Object never grew bigger than 7K, can be put into fragments in a page and stored with other small objects:
    - R13 zFS only uses fragmented method for ACLs and symbolic links, aggressively packing them in the same pages. Prior releases stored small files and directories in same page but was not efficient at it.
  - **Blocked** – Any object that grew bigger than 7K (or for R13 any files bigger than 52 bytes and all new directories) is simply a sparse array of pages. Must have empty pages to extend.
- **df** - command not always good to determine space usage, might indicate space free (free fragments) but might not show the fact that there are no free 8K pages.
  - Because of R13 allocation methods and aggressive packing of small objects, df will be more accurate and usable if the file system was created and used with R13 or later.

6

zFS considers an aggregate an array of 8K pages, each page is actually two 4K VSAM control intervals. Every object on disk has a structure called an anode that contains attributes about the object (like size, mtime, atime, permissions, link count etc…) and points to where the object's data resides in the file system. Objects are stored in three ways. Inline is used only for small objects that never grew bigger than 52 bytes in size, the object's data is stored inside the anode and takes no extra space on disk. For larger objects, z/OS 13 zFS will pack ACLs (that are less than 7K in size, which is the typical case) and symbolic links into the same disk blocks to conserve space. Prior releases will store small directories and files into fragments but they did not attempt to place them in the same block which could still reduce the number of free 8K blocks for larger objects. Any object larger in size than 7K will be stored into 8K blocks, fragments are not used and to extend larger files requires whole 8K blocks to be free, any partially empty 8K blocks are not usable.

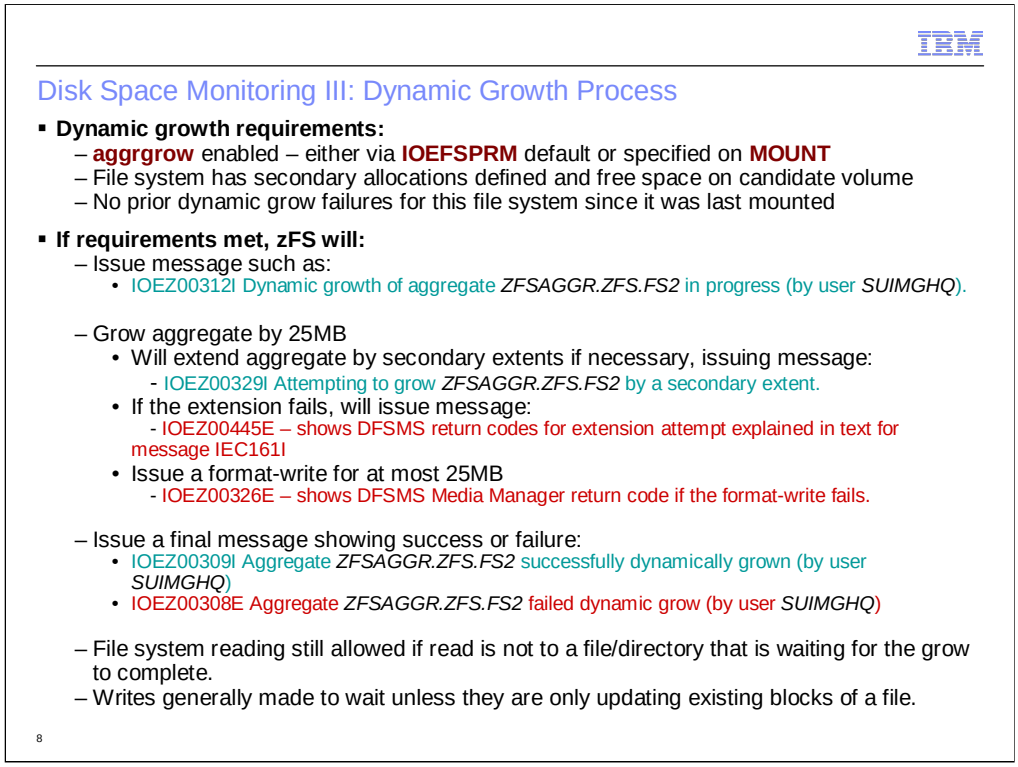
This makes the df command less useful because you can run into situations where there are free fragments but no whole 8K blocks free, which are required when storing larger objects. If an aggregate was created using z/OS 13 format and populated and used only on z/OS 13 or later, this situation is greatly reduced since all files are either stored inline, which uses no extra space, or blocked, and all directories are stored blocked. ACLs and symbolic links are tightly packed into the same blocks which typically leaves the free fragment count very low and the df command more useful.



The zfsadm aggrinfo –long is the most informative command to use to determine how much free space there is in a zFS file system. It shows the number of whole 8K pages that are unallocated on the disk. As long as this number is not 0 any object can be grown in size, as soon as its 0 then any larger object cannot be grown in size. The command also shows the amount of space used for file system control objects.

The user can monitor the cases where a zFS file system is low on space via the aggrfull IOEFSPRM option and they can override the default on the MOUNT command. There are two parameters required: threshold is the percentage of use of the file system space that will cause the message to be displayed, and increment, which is the percentage increment when the message will be repeated. zFS has mechanisms in place to prevent a flood of messages but a file system that is monitored that is low on space could result in many messages to the system log that indicate its low on space.

If a file system has no space to handle an update to the object it will issue message IOEZ00551I to indicate that it ran out of space. If dynamic growth is enabled for that file system (discussed on next slide) then it will be attempted for that file system and additional messages will be issued to show the status of the grow operation. If dynamic growth is not enabled then that means an application saw an error.

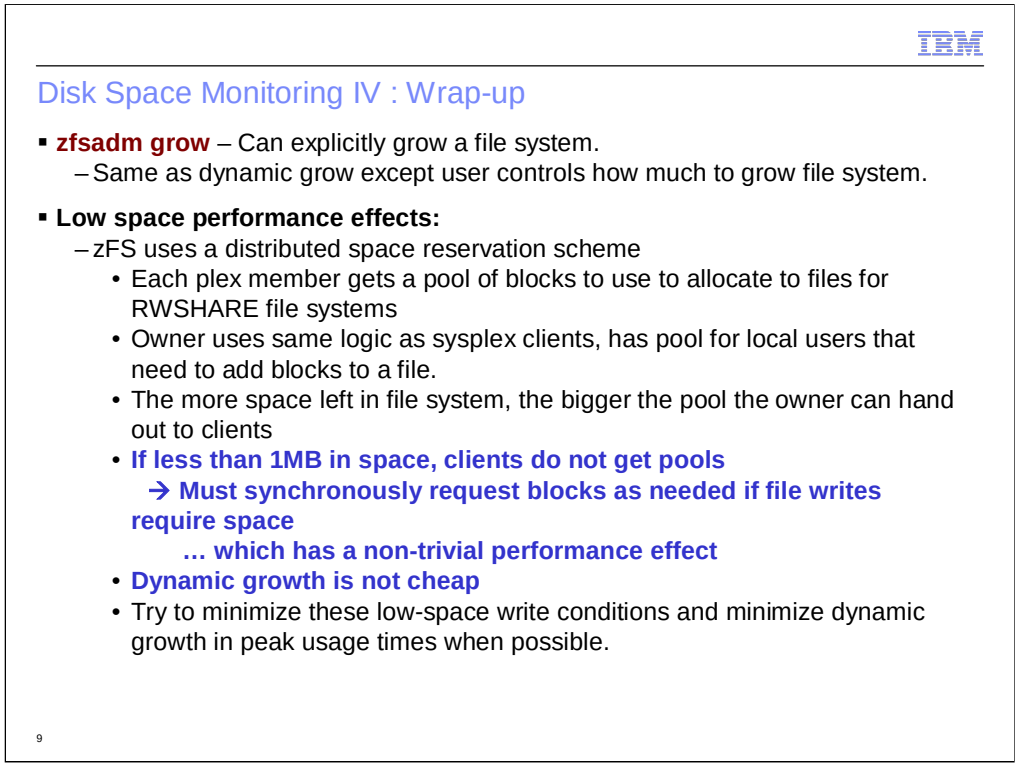


Dynamic growth is on by default in z/OS 13 zFS, and off by default for prior releases. Specifying aggrgrow in the zFS kernel startup parameters (IOEFSPRM) or by specifying it on the MOUNT statement (or if z/OS 13, taking the default) will enable dynamic aggregate growth for the file system. A file system must have a secondary allocation defined to allow zFS to extend the dataset and zFS will remember if dynamic growth for a file system has failed, and will not attempt further growth until the file system is un-mounted and re-mounted.

zFS will issue several messages during the dynamic growth progress to show success or failure of the operation. zFS will grow by no more than 25MB at a time. It will extend the dataset if necessary and format at most 25MB of it, it will loop extending if necessary if the seocndary extent is small. So if the secondary allocation was only 1MB, it would have to loop extending and formatting 25 times. If the secondary allocation was 100MB, the first growth attempt would extend the dataset by 100MB and it would format 25MB of it. If the file system ran out of space again, since the extent already exists with 75MB of unused space, the next 25MB would simply be formatted. This ensures a grow does not hang up users for long.

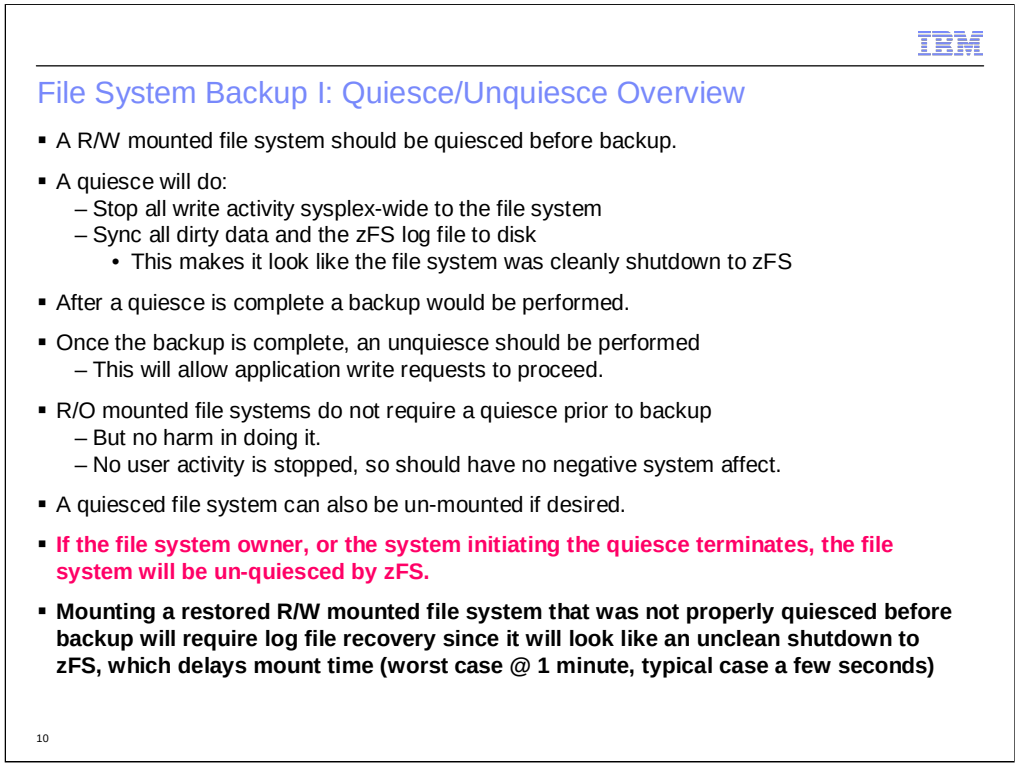
Generally speaking, if the growth fails, it will likely be because zFS could not allocate another extent and DFSMS returned an error to zFS on the extend attempt. In this case zFS will issue message IOEZ00445E to show the DFSMS return codes and system message IEC161I will likely precede it and the message documentation for that message should explain why the extension failed.

zFS allows all file system read attempts while the growth is in progress unless those reads are to a file/directory that is waiting on the dynamic growth. File system writes that require new blocks added to a file or directory are made to wait until the dynamic growth is finished.



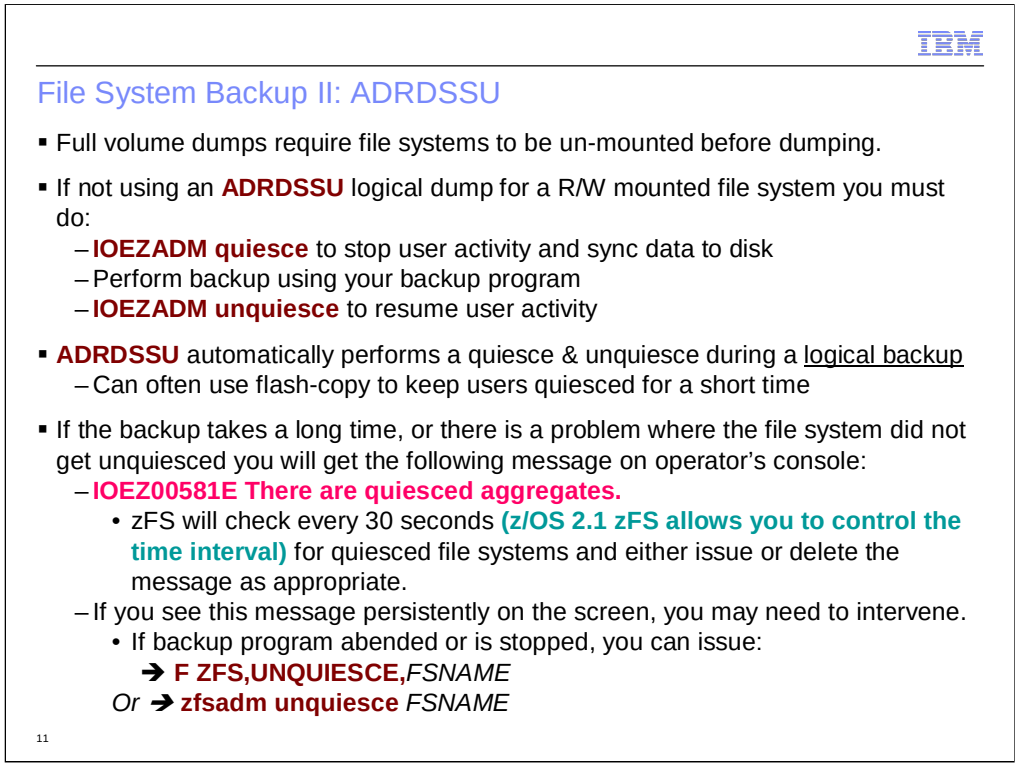
The administrator can also explicitly grow a file system and they can determine how big to make it and grow it by a large amount if desired. This works very similar to dynamic grow, file reads are allowed and file writes that are not waiting on the grow are allowed. If the file system was not out of space then a file write would not be waiting, so typically file operations run fully in parallel with the user-initiated grow operation.

zFS uses a distributed reservation scheme in z/OS 13 and later to allow sysplex clients (and owners have a similar pool) to assign file system blocks to files without communication with the owner. The more free space on disk the bigger the pool. If the total amount of unallocated space on disk is less than one megabyte, the clients do not receive any pool and every single write request that requires space must synchronously obtain it from the owner; this would go on until more space becomes available either by file system growth or by removal of files/directories that free up space. Thus performance will be better if this situation is minimized during peak usage. Of course when a file system is being newly populated a common trick is to start with a small file system and dynamically grow it to save space; this is fine since its usually a one-time occurrence for the file system. But the user should keep in mind that zFS, like many other file systems, cannot make as many performance optimizations to a file system that is low on space than one that has a reasonable amount of space. Prior releases of zFS also have reduced performance for file systems that are low on space, they generally have reduced file write performance compared with z/OS 13 zFS, on both the owner and especially on non-owners.



zFS is a logging file system and performs asynchronous write-behind. If a user would like to backup a zFS file system, then its heavily recommended that the file system be quiesced before the backup program is started, if the backup program is one that copies whole datasets (hence whole file systems). A quiesce of the file system will sync any dirty data to disk and sync the zFS transaction log file to disk. This will make the dataset that is backed up look like it was cleanly un-mounted by zFS and ensure that the restored dataset will mount without the need for log file recovery (which makes the mount operation faster) and also ensures that no uncommitted updates were lost. R/O mounted file systems generally do not require a quiesce but there is no harm in doing one, and it is a good way of noting that a backup is in progress in case someone tries to un-mount it (possibly to later mount it R/W).

zFS has logic to ensure that a quiesced file system will be unquiesced if either the owner system or the system that initiated the quiesce goes down in a sysplex environment.



There are many methods that can backup a file system or its contents. There are incremental backup programs that dump files that have changed since last backup and full-DASD backups. In z/OS a typical backup program called ADRDSSU can backup an entire dataset, often using flash-copy which uses copy-on-write techniques to keep activity to the dataset suspended for a very short amount of time, depending on the hardware/software installed at the site. If you are using a backup program that dumps an entire dataset that is not ADRDSSU, or you are performing a physical dump of the dataset using ADDRDSU, you should update your job so that it will call the IOEZADM quiesce/unquiesce functionality to ensure user activity is stopped and that any dirty data is synced to disk and the log file on disk is closed. This yields a consistent file system that does not require log file recovery when the file system is later restored. ADRDSSU will automatically call zFS to quiesce before the backup and then unquiesce after the backup if the file system is mounted. Note that backing up a R/W mounted file system without a proper quiesce will look to zFS to be an unclean shutdown of the file system and zFS will perform log file recovery to ensure the file system is consistent which delays mount time; it also means that some recent updates made around the time of the original backup may have been lost since the transactions could have been rolled back during log file recovery.

File System Backup III: Finding Quiesced File Systems

- The following commands can be used to find which file systems are currently quiesced:
  - **F ZFS,QUERY,FILESETS,ALL**

```
10.43.12 DCEIMGHQ STC00044  IOEZ00438I Starting Query Command FILESETS
File System Name                        Aggr #  Flg  Operations
--------------------------------------- ------  ---  ----------
ZFSAGGR.BIGZFS.FS1                           1  AMQSL          43
```

  - **zfsadm lsaggr**

```
IOEZ00106I A total of 1 aggregates are attached
ZFSAGGR.BIGZFS.FS1                      DCEIMGHQ  R/W QUIESCE
```

  - **zfsadm aggrinfo**

```
IOEZ00369I A total of 1 aggregates are attached to the sysplex.
ZFSAGGR.BIGZFS.FS1 (R/W COMP QUIESCED): 1737527 K free out of total 2000160
```

- **F ZFS,QUERY,FILESETS** often the best
  - Operator command, always available
  - zfsadm may be inaccessible because the file system containing it is quiesced
  - Issue it on the system that shows the message **IOEZ00581E**
- If repeated issuances of the query show the same file system is always quiesced AND the message **IOEZ00581E** is not going away (it is refreshed or deleted every 30 seconds) then:
  - Check to see if backup still running and if not:
    - Issue **zfsadm unquiesce** or **F ZFS,UNQUIESCE** to unquiesce the file system

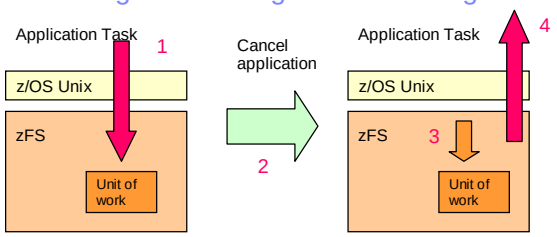
12

Its important to note that the IOEZ00581E is only displayed if a locally mounted file system is quiesced for a long time, so its important to look at the various system consoles on each plex member and issue the F ZFS,QUERY,FILESETS on the plex members that display the message. Zfsadm commands may be easier since they can be issued on any system and show a sysplex-wide view, but if the file system they are mount in, or a file system in the path to the zfsadm binary is quiesced, then you will be hung waiting to find and load the zfsadm binary. The modify command is immune to which file systems are quiesced. Thus if the zfsadm command hangs, use the modify command instead.

File System Copying and Multi-System R/W Access

- zFS uses a scheme to detect multiple R/W access to a file system from multiple systems not in the same sysplex.
  - zFS periodically writes information to first block of file system showing system usage:
    - Includes system name and plex name and timestamp.
  - At un-mount time and system shutdown time, zFS will clear this information to show the file system was cleanly un-mounted and not currently mounted R/W on a system.
  - At mount time, zFS will examine this information to see if another system might have access.
    - If it sees the first block is non-zero
      - Will delay mount for 65 seconds to determine if another system is updating this block.
      - Will issue message IOEZ00807I to indicate its waiting
- As a result, you can cause zFS to wait during mount unnecessarily and you can experience z/OS UNIX latch contention if you fail to unmount (detach) a zFS aggregate before copying it or moving it to another system not in the same sysplex.

Canceling and Forcing Users Running in zFS



1. Application calls for the file system run on the application task
2. User decides to cancel application
3. zFS recovery is notified and switches work to zFS task
4. Application task immediately freed which allows rest of cancel processing to occur more quickly

→ **Unit of work has to be pushed forward or rolled back**

- Application calls into zFS run on the application tasks and there is a unit of work associated with that call.
- When an asynchronous abend, such as a cancel, is delivered to zFS it will dispatch the recovery onto a zFS task which frees up the application task and allows the application to end quickly.
- The unit of work must either be rolled back or pushed forward, it rarely can be ignored.
  - A shared file system such as zFS has common file system objects, an application task can touch global file system structures and other objects that are not its own
  - Pushing forward is common, especially if transactions are started, just as much work to push forward as to roll back
- **Canceling user tasks does NOT break a hang inside zFS if the hang is the fault of zFS.**

14

Any time an application makes a file call into zFS, there is a unit of work associated with that call that exists inside zFS. Because zFS is a shared file system, one user can often be updating or manipulating other user's data. For example, if an application wants to read a file but its not in the user file cache, it might have to reclaim space from the oldest data in the cache. If that oldest data is dirty and not written to disk then it has to write it. Thus an application task is affecting other users files. Another example are the shared structures that exist in every file system, such as the allocation map that tracks free space in the file system. An application task might need to extend a file and obtain free space from this map and update the map; if its partially updated the map and then gets cancelled, that map structure has to be completely left consistent and intact for other users. Thus the processing performed by the application call has to be rolled back or pushed forward. Generally speaking, if the application call has started a transaction that might update the disk, it will have to be pushed forward.

With z/OS 13 zFS, in order to allow the cancel or the abend of the task to complete quickly, zFS dispatches a zFS task to handle recovery of the unit of work that represents the application call. This immediately allows the application task to continue with recovery in other components like z/OS Unix since z/OS Unix is always a participant in any call to zFS.

If there is hang inside zFS, and that hang is somehow due to a zFS software error, canceling the applications that have called into zFS will very likely have no effect on the hang and may even lose some of the documentation needed to diagnose the problem. This is because the unit of work cannot be ignored and whether its rolled back or pushed forward if it encounters the source of the hang it would block. There are much more effective ways of breaking a hang inside zFS and at the same time guaranteeing first-failure-data-capture (FFDC) and a true resolution of the problem.

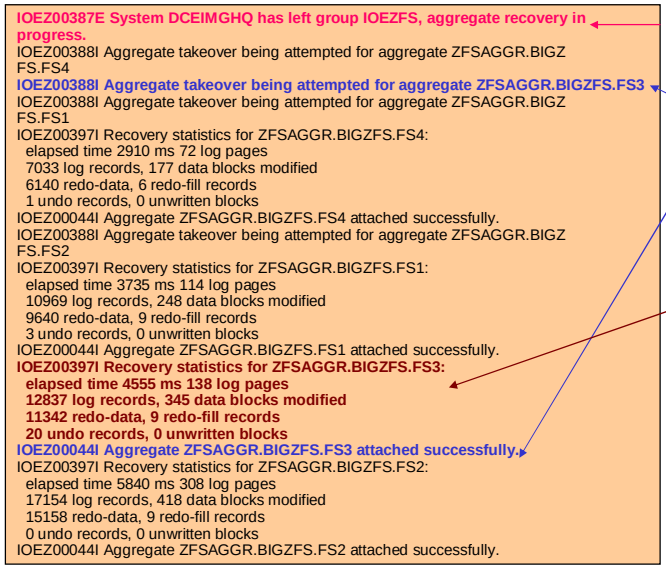
System Outages I: zFS Recovery Fundamentals

- **zFS considers a system down when XCF notifies the group exit for the other members in the sysplex.**
- When zFS is told by XCF that another system is down, zFS knows there are no stray IOs in progress to the disks because:
  - If whole system down, the fencing support in the sysplex software/hardware guarantees no IO in progress from the down member.
  - If only zFS took an outage:
    - If zFS aborts, it will ensure no IOs in progress to disk before leaving its XCF group.
    - If zFS was cancelled, XCF and address space recovery ensures no IO is in progress before it tells other members that zFS went down.
- Because of IO fencing:
  - zFS on other plex members can assume ownership of RWSHARE file systems owned by the down member (z/OS Unix moves ownership for NORWSHARE file systems)
    - zFS members will race for ownership takeover, using GRS serialization to determine who wins race for a specific file system
    - The winner of the race assumes ownership of the file system, confidently performing log file recovery to put the file system in a consistent state

System Outages II: zFS Recovery Processing

- **When a system goes down:**
  - A zFS member implicitly releases any tokens held by the down system for file systems it owns.
  - zFS members implicitly un-quiesces a quiesced file system if the file system was owned or the quiesce was initiated by the down member.
  - Will make local calls to file systems owned by the down member wait until someone assumes ownership of the file system.
  - Will race for ownership, and if a system wins the race for a specific file system:
    - Will perform log file recovery putting the file system in a consistent state.
    - Allow other plex members to re-obtain tokens for any open files they have.
    - Tell other members to allow their stopped application tasks to resume processing
- **Application point of view:**
  - Applications running on other members are stopped by zFS until ownership of their target file system is assumed by one of the members
  - Applications that had open files:
    - May see IO errors reported if zFS on that system thinks data may have been lost in the file; otherwise if zFS is confident of no errors, they will see no errors.
  - Due to transaction rollback during log file recovery, newly created objects might not exist, in this case an application could see IO errors reported due to lost object during the crash.
  - If no system could assume ownership (should be very rare), stopped applications will be resumed but they will get errors for all calls until some zFS can successfully take ownership of the file system (each zFS member will re-try periodically).

System Outages III: zFS Messages Issued During a System Outage



Each zFS member issues message to show when it was told member went down

Any time a zFS member wins race to assume ownership of an RWSHARE file system, it will tell you.

And also tell you when its successfully assumed ownership

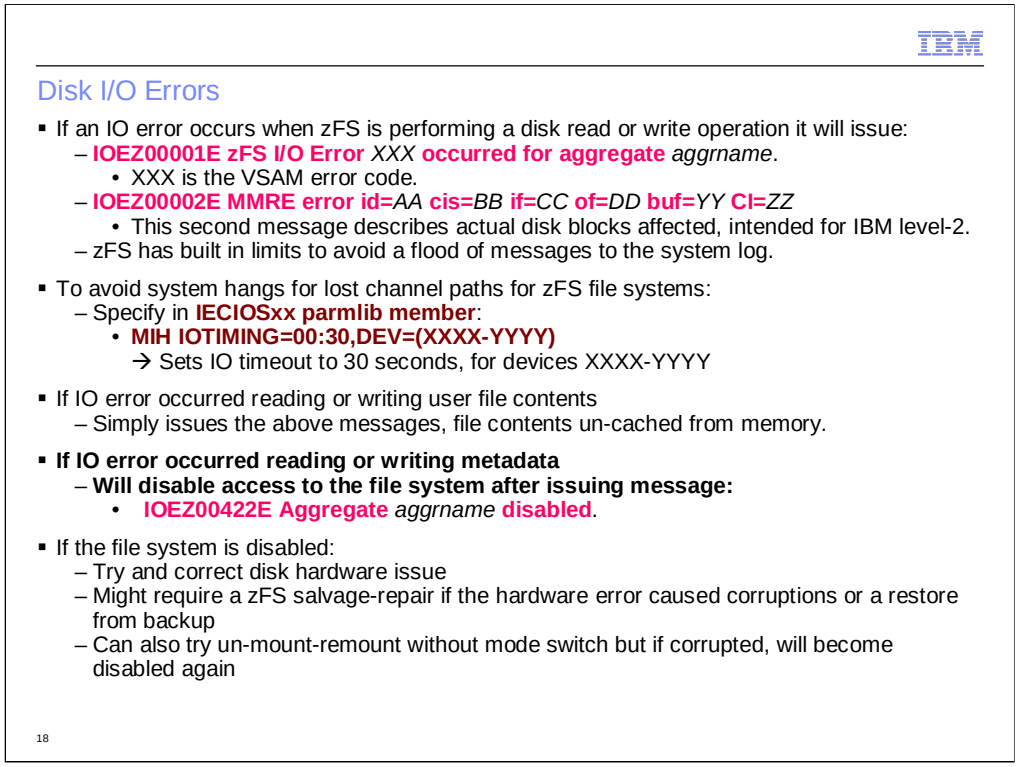
Log file recovery often required for file systems that were being updated when owner went down – undo records means transactions rolled back.

Not shown:

- **IOEZ00589E** – system could not open file system to become owner.
- **IOEZ00389E** – Unexpected error prevented system from assuming ownership.

17

Shown on this slide are the messages specifically related to down-system recovery processing. While other errors could occur during recovery processing (such as an unexpected disk IO error for example), these messages show the general flow of ownership assumption and whether the system successfully assumed ownership or not.



zFS will always issue messages IOEZ00001E and IOEZ00002E for disk IO errors that occur to a file system; zFS has limits to keep the number of messages shown per-file system to a reasonable amount. IOEZ00002E is intended for IBM service personnel.

If all channel paths are lost to a device, zFS will hang waiting for DFSMS to tell it that an IO is complete (the IO will remain in the system queues) unless IO timeout is specified. This is accomplished by updating the IECIOSxx parmlib member to specify an IO timeout for the device ranges of devices that contain zFS file systems. This will ensure that timed-out IOs for lost channel paths are handled like any other IO error.

If the IO error occurs when reading or writing the contents of a user fie, there will be no additional action taken, though zFS will remove the cached contents of that file from memory. If the IO error occurs with any file system metadata structures (such as the contents of a directory) it will disable the aggregate for all access. In this latter case message IOEZ00422E will be issued.

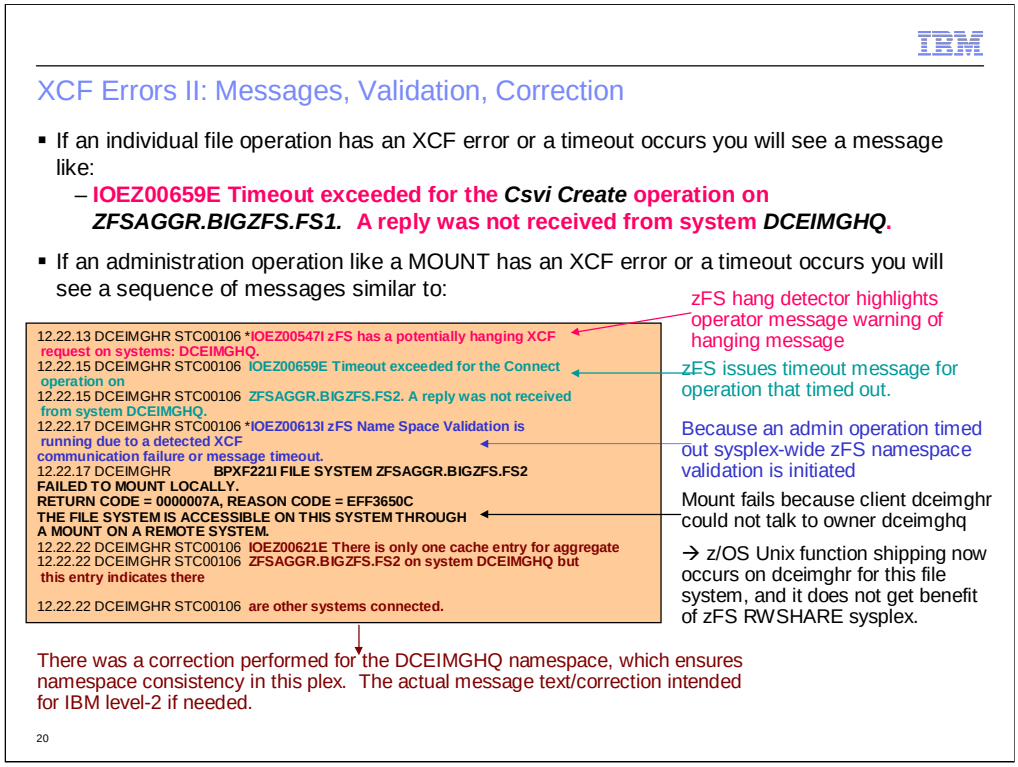
Once the hardware error is resolved the administrator could attempt an un-mount re-mount without mode switch which would un-disable the file system and allow user activity to resume. This will work only if the hardware error did not cause permanent damage to zFS. If the hardware error is something that will take a long time to resolve, it might be best to simply un-mount the file system and then possibly restore from a backup, if possible. Once the hardware comes back online the administrator could run the stand-alone salvage program (with no options specified which means both verification and repair of the file system should be performed) to search for and repair errors. The current salvage program available with z/OS 13 requires re-runs until it says the file system is clean.

XCF Errors I: Overview

- **XCF error types:**
  - Communication error transmitting or replying to another member
    - This has been extremely rare during the life of zFS sysplex support.
  - More likely is that a system went down/going down just prior to message transmit OR
  - A hang or slowdown that exists on a plex member, these show as timeout errors in zFS.
- **File Operations**:
  - Client to Owner Transmit/Reply Error or Timeout
    - Generally speaking the individual operation will fail.
    - The client will un-cache any data for that object.
  - Owner to client callbacks
    - Owner might have to reclaim tokens held by clients for an object
    - If it cannot communicate to clients, tokens cannot be reclaimed and new accesses to object for all other members fail until communication can be established.
- **Administration Operations**:
  - Administration operations are operations like mount, un-mount, quiesce etc…
  - A lost transmit or reply could leave zFS name-spaces out of sync with each other.
    - In this case zFS will dynamically repair via validation and correction
    - Any new administration operation is made to wait until zFS name space issues corrected due to lost communication packets.
- Hangs and slowdowns appear to other members as a transmit or reply failure
  - Will treat similarly to the above
  - Administration operations have a 15 minute timeout and file operations a 10 minute timeout.

19

The one difference between a hang/slowdown and a lost transmit or reply packet is that there is still a task doing that work on the member that took too long processing the message; zFS handles those cases and will ensure that zFS systems stay consistent with each other and prevent improper sysplex access to file system objects and will repair zFS file system name-spaces.



As shown on later slides, zFS has a hang detector that highlights messages on the operator console to warn when another system is taking too long for a message transmitted to them. In this case we transmitted a desire to be an RWSHARE client for the file system that was being mounted, and it somehow got hung-up on DCEIMGHQ (maybe that system is having a problem). The zFS hang detector will un-highlight the message when the hang has passed. Any time a file or administration related communication has a timeout error, zFS issues message IOEZ0659E to indicate the actual operation that timed-out. zFS has code to prevent too many messages from flooding the system log in cases of extreme hang-ups.

Since this was an administration command that timed out, that means there could potentially be an inconsistency in the namespaces between DCEIMGHQ and DCEIMGHR (in this example, DCEIMGHQ thinks DCEIMGHR is an RWSHARE client but since DCEIMGHR got a failure on the XCF call to DCEIMGHQ they failed the mount and do not consider themselves an RWSHARE client) and zFS namespace validation occurs. Since the mount of the file system on DCEIMGHR failed, that means z/OS Unix function shipping will be used for that file system instead of zFS RWSHARE support for the system DCEIMGHR. That is not a desirable case, and this will clearly show in the D OMVS,F output that DCEIMGHR is a z/OS Unix client.

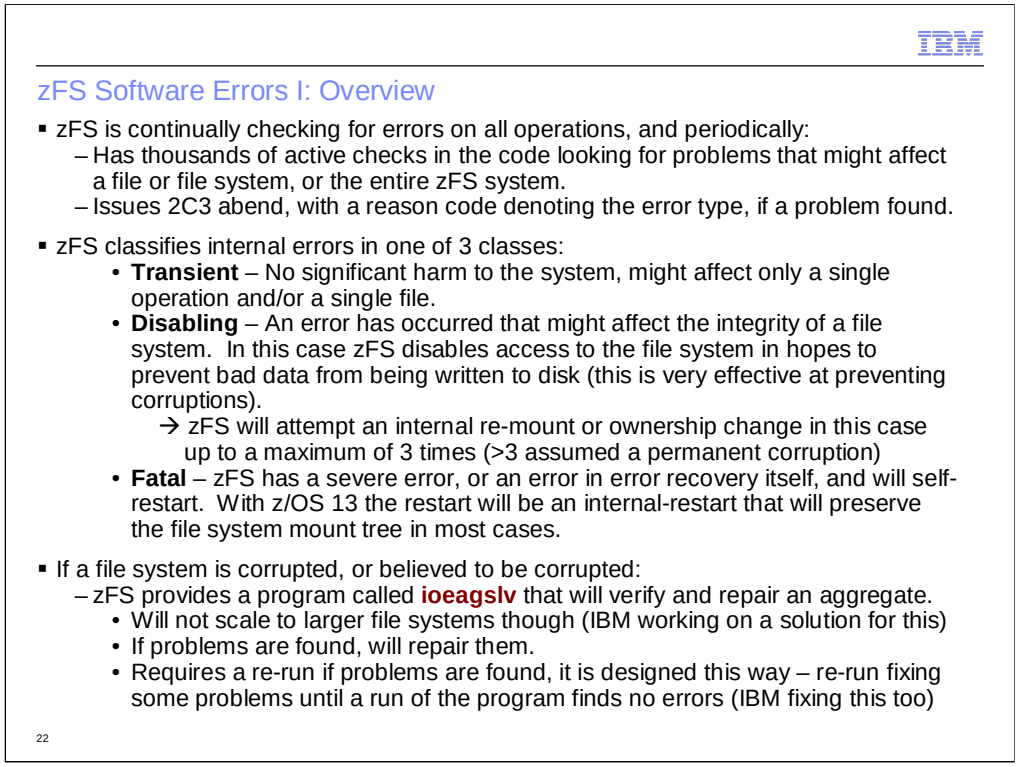
Any time validation finds a problem in a system's namespace it will correct the problem and issue messages to show what it corrected. Those messages are more useful to IBM level-2 than to a user, but the user can at least take note that the inconsistency was corrected which ensures proper future sysplex function.

Enabling Debug and Obtaining Dumps

- If you have a problem, IBM level-2 might ask for debug classes enabled.
  - Can have debugs enabled at startup by specifying dataset in IOEFSPRM:
    - **DEBUG_SETTINGS_DSN=***SUIMGHQ.PRIVATE.PARMS(ZFS)*
    - Which would contain statements such as this:
      IOEDEBUG=(C=*DIR***,P=M,L=127)**
      IOEDEBUG=(C=*CTKC***,P=M,L=127)**
  - Can also dynamically enable or disable debug classes like this:
    - **F ZFS,IOED,C=***DIR***,P=M,L=127** -- this enables the DIR debug class
    - **F ZFS,IOED,C=***DIR***,P=N,L=0** -- this disabled the DIR debug class
- IBM level-2 might ask for you to increase your trace table size.
  - Can specify the size only in the startup parms
    - **trace_table_size=***500M*
  - Cannot dynamically re-size the trace table.
  - Default is somewhat small, if possible, maybe make it larger in case you hit a bug.
- IBM level-2 or user might decide that dumps are needed for a situation (such as unexpected failure of a file command or operation):
  - Can obtain one at any time via:
    - **F ZFS,DUMP**
  - This dump command will do the following:
    - Dump all zFS sysplex members
    - Dump both z/OS Unix and zFS storage to allow for debugging of most problems.

21

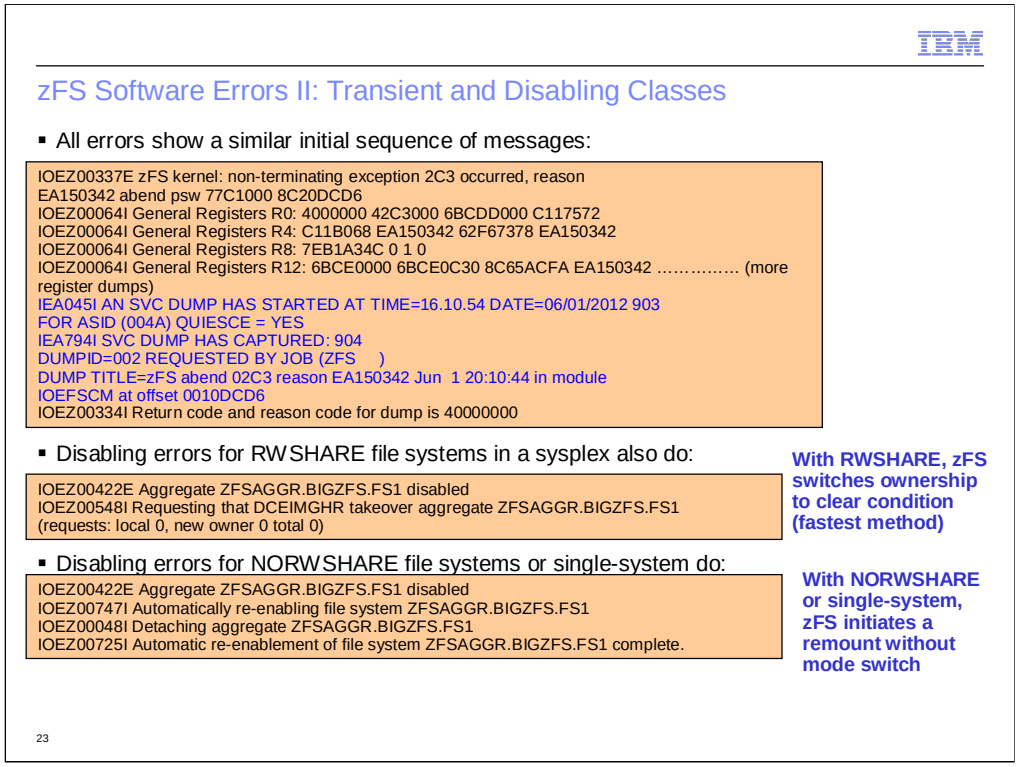
A customer would not generally enable zFS debug classes unless requested by IBM service. IBM service would provide instructions similar to those listed on this slide for ensuring proper documentation is obtained and request a dump (if one is not being auto-generated by zFS) via the F ZFS,DUMP command. One thing to keep in mind is that the zFS trace table currently cannot be dynamically re-sized (this should be fixed in a future release).



zFS provides various types of software error recovery. Problems that do not have any serious impact to the system or a file system will simply result in a dump and possibly an error received by an application. If a problem is found with an in-memory copy of metadata for a file system, any IO to that file system is stopped and operations to that file system are disabled. If zFS is running in a single system environment or the file system is a NORWSHARE file system, zFS will initiate an internal remount without mode-switch which will cause all data related to that file system to be purged from memory and a new mount will occur which will ensure memory for that file system is newly initiated.

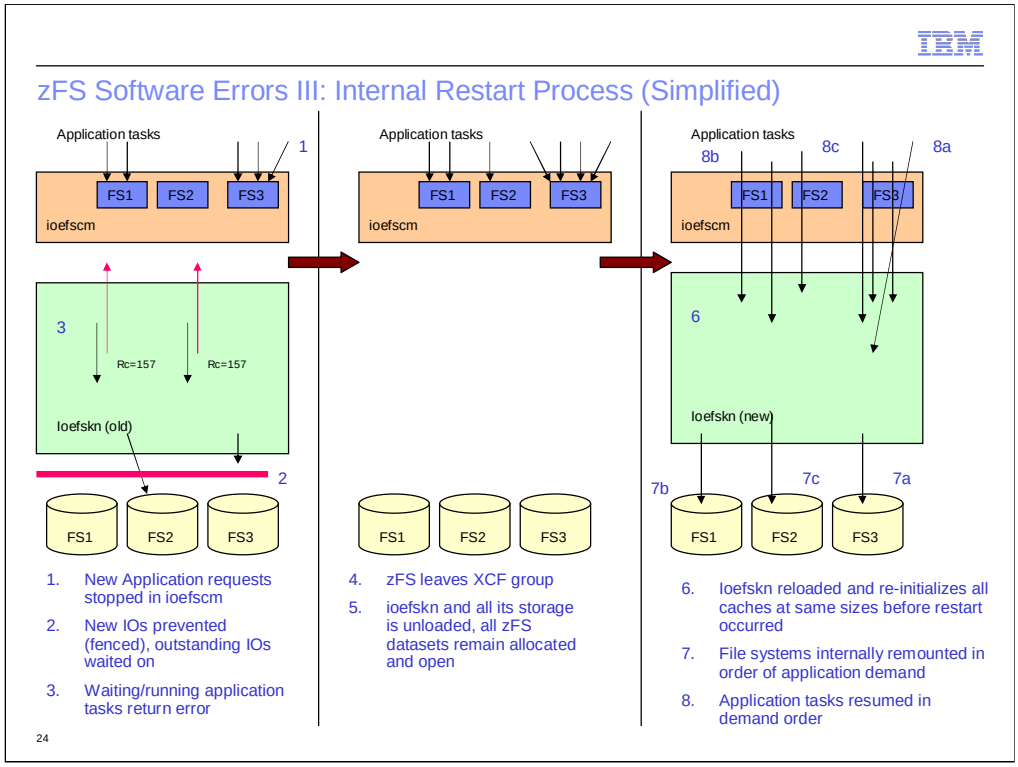
If the problem is a fatal problem, one that affects many file systems or all of zFS, or there is an error itself in error recovery or an unhandled error is found, zFS will restart. With z/OS 13 this restart will be an internal restart which will stop user activity, re-initialize zFS and then re-mount any mounted file systems and resume user activity preserving the mount tree and ensuring that the error condition is corrected.

Actual disk corruption errors require that the file system be restored from a backup or corrected via the zFS salvage program (ioeagslv). This program requires that it be run repeatedly until a run says no errors are found in the file system (typically its about 3 times). IBM is working on providing a better salvage program that scales to the largest zFS file systems and only requires one run to correct a problem file system.



Transient errors only issue the PSW/register dump messages and show the return and reason code for the x2C3 dump zFS issues.

For errors that require zFS to disable access to the file system, zFS will move ownership to another sysplex member if the file system is an RWSHARE file system and there is at least one other plex member capable of assuming ownership of the aggregate. This results in a clearing of the in-memory structures on the original owner which clears the error condition. In all other cases zFS would perform an internal remount with mode switch. In this case z/OS Unix would internally un-mount the file system resulting in zFS clearing its memory of any data related to the file system and a fresh mount would occur for the file system. While the resolution of the problem occurs, any applications requesting access to the file system are made to wait.



zFS internal restart is designed to handle fatal errors; hence, errors that significantly affect zFS operation or errors in error recovery (leaving something in an inconsistent state) and is a method to avoid a system-IPL.

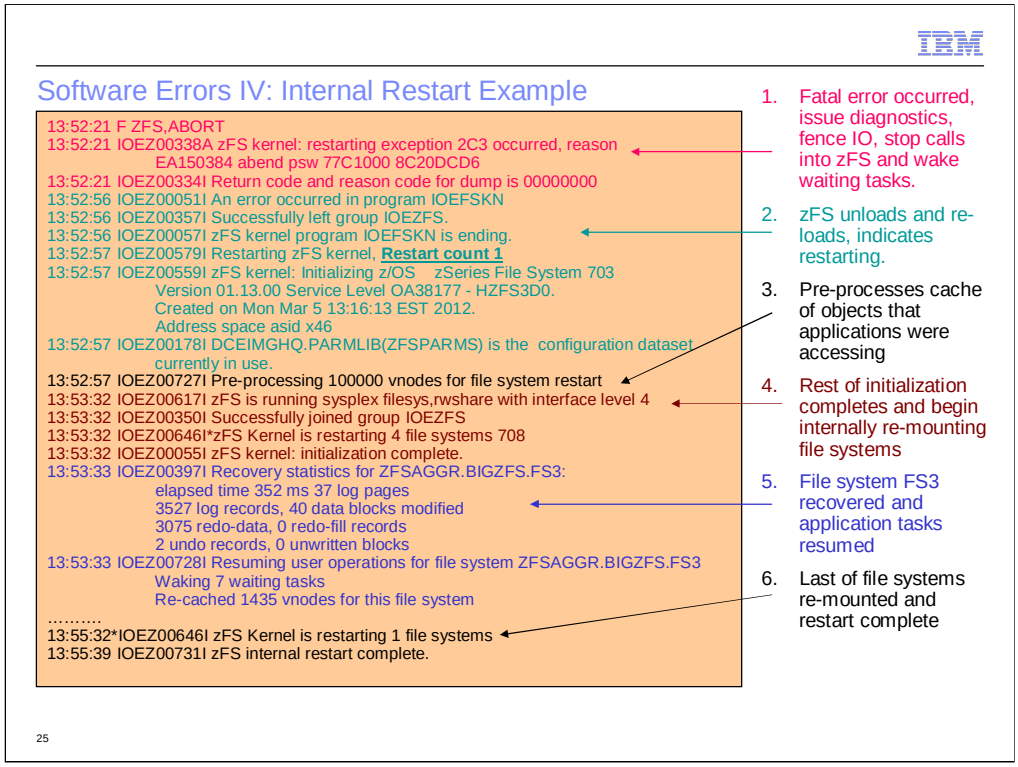
1. zFS will, as quick as possible, make new application requests wait tracking how many tasks are waiting for each currently mounted file system. Ioefscm owns the file system and vnode storage, so that storage remains.
2. Disk IO is fenced, no new IO is allowed and any outstanding IOs are waited on.
3. And application tasks running/waiting inside ioefskn code are made to return with error (a common error code will be 157 – EMVSERR).
4. zFS will then leave the XCF group, other systems would detect the system went down and will try and assume ownership of file systems owned by this system.
5. Ioefskn tasks are stopped and ioefskn and all its storage (which is all zFS storage with exception of the file system and vnode structures accessed by z/OS Unix) is freed.
6. Ioefskn is then re-loaded and initialized so all caches and storage start clean.
7. File systems will be internally re-mounted in application demand order and as soon as an application task's file system is ready that application task can resume processing.

This internal restart process is opaque to z/OS Unix. no file systems are unmounted or lost. Some file systems may be taken over by other plex members but if most of the access is from the system that restarted, zFS usage based movement will move the file system back.

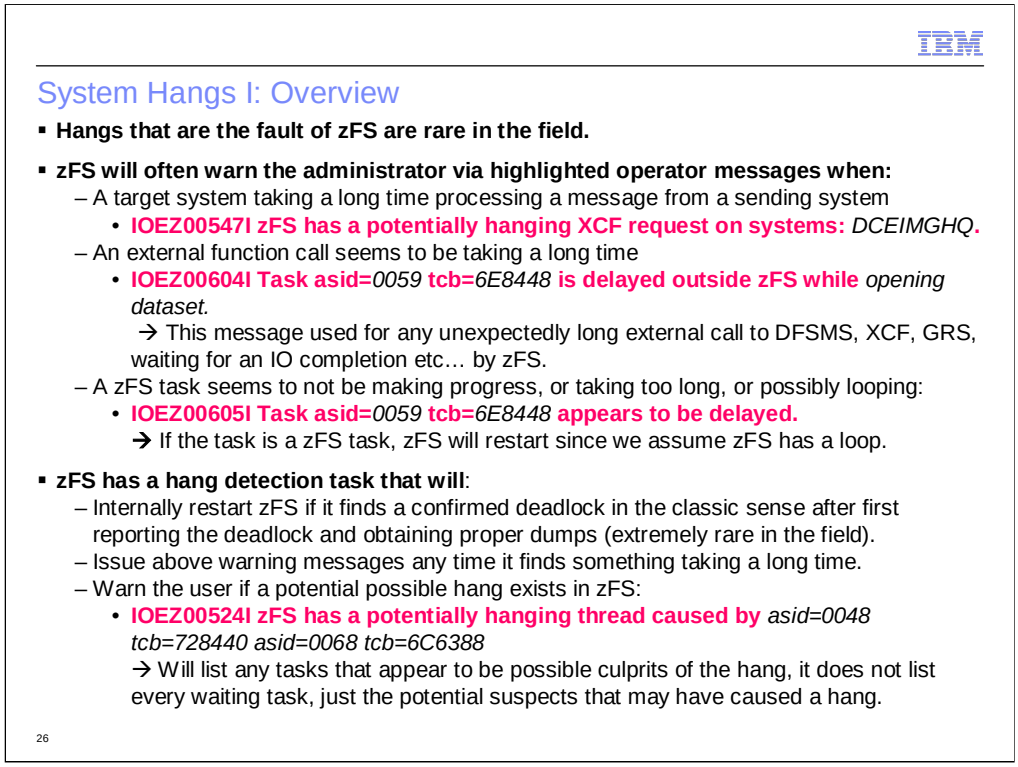
This ensures the fasted possible resolution to the error, the error condition should truly be cleared. Some applications will see errors if they had an open file at the time, this scheme preserves the file system tree but some applications will see errors at the time of restart.

Note that internal restart does not re-read the IOEFSPRM file, all program settings, cache sizes and options are the same as they were at the time of the zFS restart.

Note that an internal restart tries to leave things as they were at the time of the restart; this means that when zFS restarts it does not re-read the IOEFSPRM dataset, but instead initializes caches at the same size as they were before the restart occurred.



Its important to note that the application users of file system ZFSAGGR.BIGZFS.FS3 are running again as soon as message IOEZ00728I is issued. This restart was from a high stress workload against a zFS with very large caches and applications that were accessing a very large number of objects.



zFS has a hang detector task that is waking up every 20-30 seconds and checking for anything that is taking too long or running too long. It will warn the administrator via highlighted operator messages if anything is taking too long or if another system is taking too long processing a request from this system. These messages appearing at the operator's console are a sign of a possible problem or a performance issue. If the hang detector ever finds a true deadlock in the classic parallel programming definition, it will obtain a dump and initiate an internal restart since zFS has a fatal error where it will leave the system hung (this is extremely rare). If the hang detector finds signs of a possible deadlock (the hang detector does not have every piece of information sysplex-wide to confirm in all cases) it will simply issue message IOEZ00524I and indicate the likely suspect tasks that are involved in the potential hang. If this message is not being deleted and the hang does not appear to be going away, a hang break by the administrator might be required.

System Hangs II: Diagnosis

- **F ZFS,QUERY,THREADS,OLDEST** (APAR #XXXXXX)
  - Lists the tasks (and system name) that is waiting the longest sysplex-wide
  - If repeated issuance of this command shows same tasks (nothing changing) you have a likely hang

```
19.42.59 DCEIMGHQ          f zfs,query,threads,oldest
19.42.59 DCEIMGHQ STC00158  IOEZ00438I Starting Query Command OLDEST.
          Oldest zFS and z/OS UNIX Tasks
          ------------------------------
STK/Recov   TCB        ASID  Stack       Routine              State
--------   --------    ----  --------    --------             --------
System DCEIMGHQ
612E8820  005FF188  004A  611843E8  ZFSRDDIR            WAITLOCK
612A4000
     since  Jun  4 23:42:59 2012  Current DSA: 61185258
     wait code location offset=04E2 rtn=get_evp_locked
  lock=71FA79F0 state=E2F82001 owner=(62F82000 0077 5CEBF8)
     lock description=Vnode-cache access lock
Operation counted for OEVFS=667A4150  VOLP=63162950

fs=ZFSAGGR.BIGZFS.FS3
```

Shows system name, time it started waiting and file system its using

- zFS uses GRS latches and will ENQUEUE GRS resources for certain administration operations (such as MOUNT, QUIESCE, performance based aggregate movement),
  - **D GRS,C** – will include zFS ENQUEUEs and latches
    - Look for EXCL access on **IOEZJOIN** and/OR **IOEZNS** resources
      → Means a system is delayed joining or leaving sysplex or administration commands timed-out and a namespace validation is pending
- If message **IOEZ00604I** is shown, look at system log for problems with the catalog, DFSMS, XCF, GRS, or problems communicating with a sysplex member based on the message text.

27

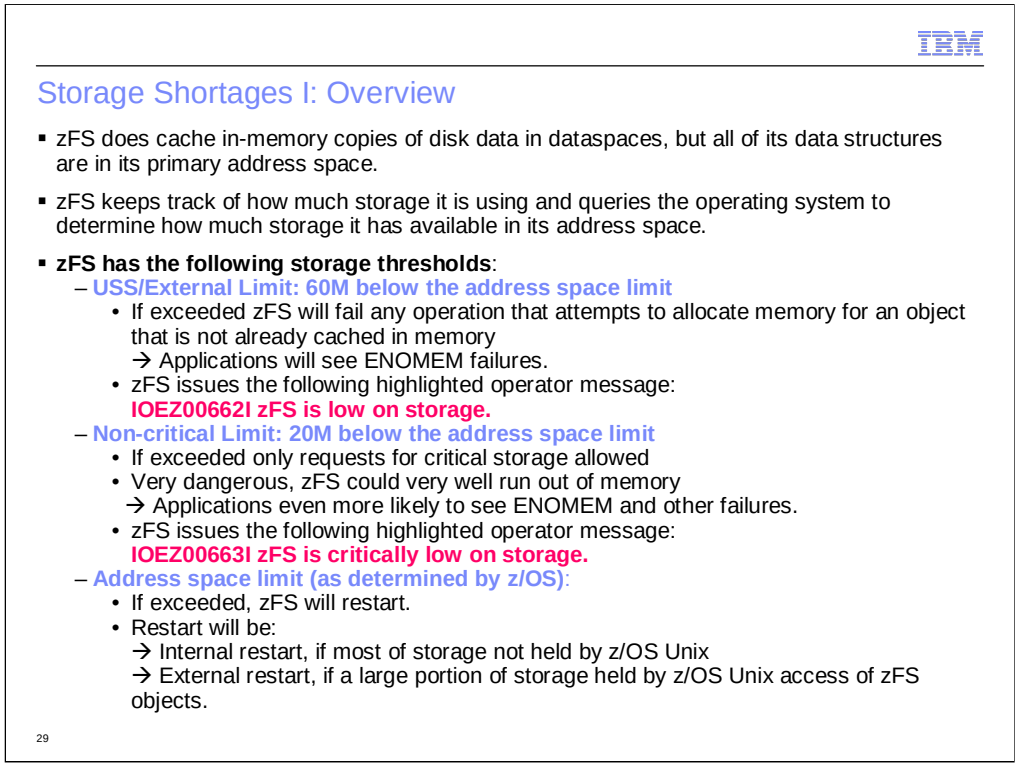
Generally speaking the F ZFS,QUERY,THREADS,OLDEST would be used to determine which system had the oldest waiting task. If repeatedly issuing the command provides the same exact output, or the command indicates it cannot communicate with a particular sysplex member, then those sysplex members are likely the source systems of a sysplex-wide hang. This command also works single-system, if repeated issuance of the command shows the same output over a “long” period of time, then its likely there is a hang. The zFS hang detector will also likely be issuing warning messages but zFS sysplex is one large shared file system, so multiple system can sometimes be complaining about each other.

Another source of analysis can be the use of the D GRS,C command. This is useful to show cases where there is contention with zFS sysplex administration options; it is not as valuable as the query,threads command since this command really factors every last task running inside zFS sysplex-wide. But it is another method of showing contention.

If message IOEZ00604I is shown on the operator’s console, then it could be the case that a component that zFS is calling is the source of a hang. A zFS task calling an external component might hold a lock in zFS, which makes other tasks wait, though the ultimate issue is an external component. The diagnosis procedure of the component shown by the message should be used, possibly consulting the system log or issuing additional command to find out what is wrong.

System Hangs III: Resolution

- **If the hang is due to a component that zFS calls due to message IOEZ00604I**
  - → Refer to that components documentation for a resolution procedure
    - … fixing zFS is not going to fix the external component
    - Canceling the listed tasks/jobs may break the hang in some cases, depending on the component, unless the listed task is a zFS task, then you have to use hangbreak.
- **But if the hang looks to be in zFS itself then the recommended procedure is**:
  - Use **F ZFS,QUERY,THREADS,OLDEST** to determine oldest hanging system
  - Issue **F ZFS,HANGBREAK** on that plex member (or single-system)
    - → This will cause an internal restart after ensuring proper documentation obtained, this should get dumps and solve the problem.
    - (If for some reason an internal restart does not work or it gets hung (this should be rare), then perform a system IPL after obtaining a dump of the zFS and z/OS Unix address spaces)
  - Remember that canceling user tasks/jobs will normally not break a zFS hang
  - When the internal restart is complete, check to see that hang is resolved, if not then repeat these steps with next oldest system.
- **Can initiate a sysplex-wide internal restart**:
  - **F ZFS,ABORT,ALL**
    - → This will do an internal restart on all plex members which ensures every zFS starts with fresh storage.
    - Generally an approach of last-resort.



Because zFS is a 31 bit program, it is constrained to hold all of its data structures below the bar. It does keep in-memory copies of disk blocks in data spaces, but the tracking structures for those caches and all other structures are inside the zFS primary address space. zFS queries the virtual storage management (VSM) component of z/OS to determine how much storage it has in its address space. zFS has defined 3 limits which if exceeded, will alter zFS function:

1. USS/External Limit – 60M below the VSM limit, if this is exceeded zFS will issue message IOEZ00662I. It will also fail any application request that tries to access or create a file system object not already cached in zFS memory. zFS keeps an LRU cache of the most recently accessed objects by applications. If zFS has to allocate storage for a new object it will simply fail if this limit is exceeded. Thus if message IOEZ00662I is on the operator's console, applications likely have seen failures.
2. Non-critical Limit – 20M below the VSM limit, if this is exceeded, only certain types of storage allocations are allowed internally in zFS. This means that most operations, even operations to currently cached objects will fail.
3. Address space limit – the VSM limit, if met/exceeded zFS will restart. If the reason that zFS ran out of storage is an excessive number of objects held by z/OS Unix (z/OS Unix has a hold on cached objects it accesses, zFS cannot delete these structures even in an internal restart, and thus an internal restart would not correct the condition in this case) then it will simply do an external restart. An external restart means that zFS will completely stop and the address space will end and then its up to z/OS Unix to restart zFS based on the rules of z/OS Unix. An internal restart would occur if z/OS Unix was not responsible for the low-storage condition (hence likely some zFS storage leak) and an internal restart would likely correct the condition and therefore would solve the problem and leave the file system mount tree intact.

Storage Shortages II: Diagnosis and Actions

- **F ZFS,QUERY,STORAGE** – shows zFS storage use

```
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
```

Address space limit determined by VSM

Non-critical limit

USS/External limit

Storage currently allocated

- **If storage dangerously close or exceeds USS/External Storage Access Limit:**
  - Try reducing cache sizes (**meta_cache_size/metaback_cache_size/user_cache_size**) to free some storage via **zfsadm config**.
  - If cache reduction does not solve problem, report it to IBM service.
- **zFS should be run with a region size of 0M**

30

If zFS is low on storage, one possible method of freeing some memory is to reduce cache sizes via the zfsadm config command. The query storage command will also show a per-component breakdown (not shown here) which can be used to determine which cache is the best to reduce in size. Generally speaking, when increasing caches due to performance tuning, be careful not to make them too large for zFS primary storage; you want to prevent a storage shortage rather than correct one.

z/OS 2.1 Improvements for Diagnosis

- **D OMVS,F** – shows whether zFS file system is quiesced
- **Full Audit Fid** – for all newly formatted file systems and convert_auditfid=on by default
- **New Salvage (ioeagslv)** – For file system recovery.
  - Fixes scale-ability problems with old salvage
  - Proper IBM messages
  - Repairs in one-pass, not multiple passes as the old one.

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 MVS System Commands (SA22-7627)
  Describes D OMVS command

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.