

Storage Scale User Group: San Jose 2025

A brief introduction to debug common Storage Scale issues

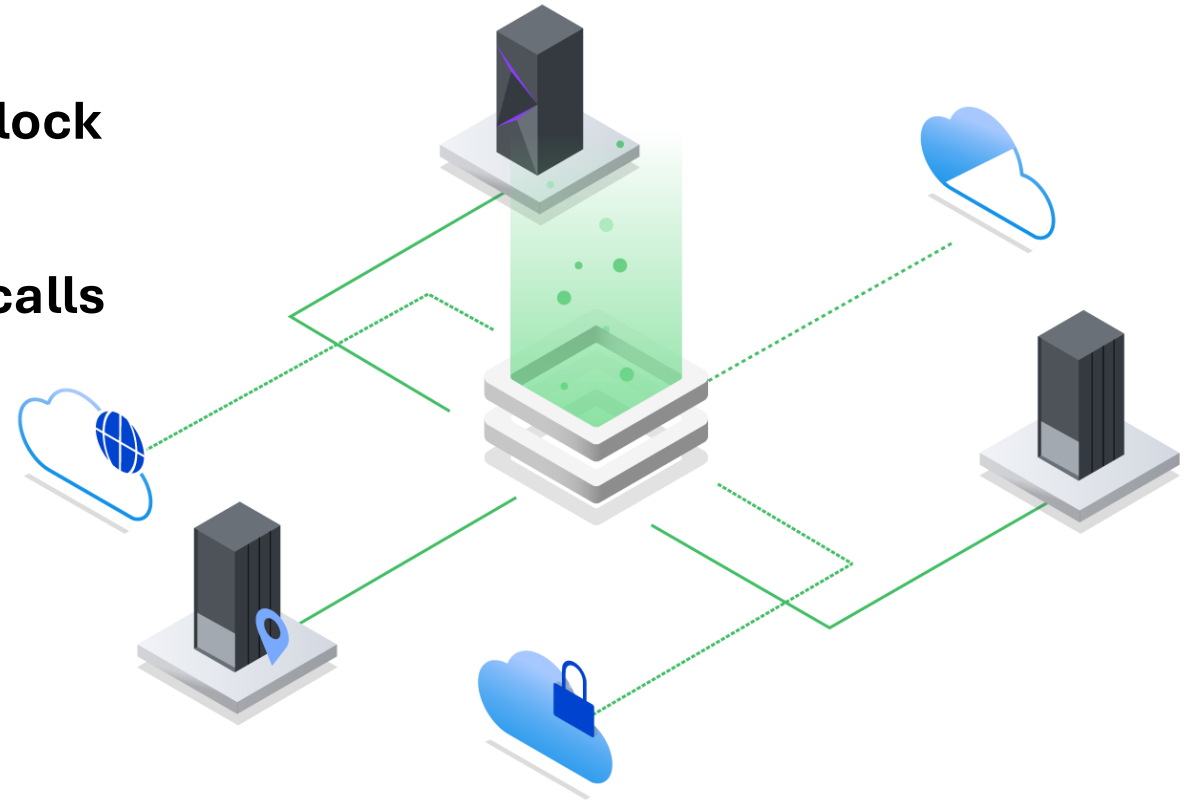
Yulin Chen
Storage Scale Support, North America Team

2025.08.07



Agenda

- **Debugging tmMsgTellAcquire1 longwaiters/deadlock**
- **Debug to a long waiter triggered by user app vfs calls**
- **FGDL and rename/mkdir**
- **Q&A**



Debugging tmMsgTellAcquire1 longwaiters/deadlock

- **Token Acquisition Process in Spectrum Scale (GPFS)**
- When a user application attempts to access a GPFS file, it must first obtain a token from the token manager—this is the initial step in the access process. Tokens must be acquired before accessing certain objects such as files or inodes.
- VFS calls:
open, lookup, rdwr, getattr, setattr, rename, mkdir, fcntl, chmod, chown ... etc

- **Token Acquisition Workflow:**

1. The client sends a `tmMsgTellAcquire1` RPC to the token manager responsible for the target filesystem/inode.
2. The token manager checks for conflicts with tokens already held by other nodes:
 - **If no conflicts exist**, the token is granted.
 - **If conflicts are detected**, the token manager returns a *copyset*—a list of nodes holding conflicting tokens.

3. **Handling the Copyset:**

- **In a single-cluster setup, or when only one client cluster accesses a remote cluster, the client itself performs the revocation by sending `tmMsgRevoke` RPCs to the listed nodes.**
 - **In a multi-cluster environment, where multiple client clusters access a shared (home) file system, the token revocation is managed by the token manager(s) in the home cluster.**
4. Once revocation is complete, the client issues a “second tell request” by sending a `tmMsgTellAcquire2` RPC to the token manager.
 5. Block reservation (BR) tokens also involve their own RPCs.

- **Note:**

Token-related RPCs (such as `tmMsgTellAcquire1`, `tmMsgRevoke`, and `tmMsgTellAcquire2`) constitute a significant portion of overall Storage Scale RPC traffic.

- [1 . internaldump.2025-04-16_11.59.01.39524.deadlock.fyre-compute02](#)

===== dump waiters =====

Current time 2025-04-16_11:59:01-0500

...

Waiting **3904.2938 sec** since 2025-04-16_11:54:53, monitored, thread 64614 **SharedHashTabFetchHandlerThread**: on ThCond 0xC1011C2E0 (MsgRecordCondvar), reason 'RPC wait' for **tmMsgTellAcquire1**

msg_id 1216422397 1/1 pending on node **scale-mgr3-vm1**

tmMsgTellAcquire1: token mode **rs** Inode key 12C9CB6507976106:**000000018DE7E200**:0000000000000000 flags: MULTIACQUIRE MQ:39

- [2 . internaldump.2025-04-16_11.59.01.1376183.deadlock.scale-mgr3-vm1](#)

===== dump selected_tokens =====

Current time 2025-04-16_11:59:01-0500

...

Selected: SubToken::mostWanted: SubToken: Inode stFlags 0x80 state 0x3 stTransNode 245

Token: 1:7 12C9CB6507976106:**000000018DE7E200**:0000000000000000 ref 431 @ 0x7FFFC5405708

SubToken: Inode stFlags 0x80 (qm) state **COPYSET** stTransNode **<24>**

<24> --> <c0n24> 10.10.103.134/0 (fyre-compute09)

- [3 . internaldump.2025-04-16_11.59.03.19883.deadlock.fyre-compute09](#)

OpenFile: 12C9CB6507976106:**000000018DE7E200**:0000000000000000 @ 0x18081D6CFE0

cach 1 ref 1 hc 3 tc 2 mtx 0x18081D6D010

Inode: invalid eff token **nl @** 0x18081D6D190, ctMode nl State **COPYSET** seq 1830042

lock state [acq] x [] flags [wka] hasWaiters 1 0

Mnode: valid eff token ro @ 0x18081D6D1D8, ctMode ro seq 1830045

DMAPI: invalid eff token nl @ 0x18081D6D148, ctMode nl seq 1830038

SMBOpen: invalid eff token nl @ 0x18081D6D088, ctMode nl seq 1830038

lock state [(nil) D:] x [] flags []

SMBOpLk: valid eff token wf @ 0x18081D6D0F0, ctMode wf Flags 0x30 (pfro+pfxw) seq 1830040

BR: @ 0x18081D6D230, ctMode nl seq 1830042

Fcntl: @ 0x18081D6D268, ctMode nl Flags 0x8 (ma) seq 1830041

inode 6675751424 snap 0 INVALID nlink 373 genNum 0x72AF4528 mode 0200042750: **drwxr-s---**

...

==== dump waiters =====

- Current time 2025-04-16_11:59:01-0500
- Waiting 3894.9832 sec since 2025-04-16_12:03:53, monitored, thread 32781 **MkdirHandlerThread**: for **tmMsgRevoke** on node 10.10.103.89 **<c0n17>**
- **<c0n17> ---> 10.10.103.89/0 (fyre-compute07)**

- [4 . internaldump.2025-04-16_11.59.03.19883.deadlock.fyre-compute07](#)

===== dump selected_files =====

Current time 2025-04-16_11:59:03-050

OpenFile: 12C9CB6507976106:**000000018DE7E200**:0000000000000000 @ 0x18096703098

cach 1 ref 1 hc 3 tc 4 mtx 0x180967030C8

Inode: valid eff **token xw** @ 0x18096703248, ctMode xw seq 25362

lock state [xw] x [] flags [dmn rvk] **writer 17505** hasWaiters 0 1 -----> **xw token this dir inode was hold by thread 17505**

Mnode: valid eff token ww @ 0x18096703290, ctMode ww seq 25365

DMAPI: invalid eff token nl @ 0x18096703200, ctMode nl seq 25359

SMBOpen: invalid eff token nl @ 0x18096703140, ctMode nl seq 25359

lock state [(nil) D:] x [] flags []

SMBOpLk: valid eff token wf @ 0x180967031A8, ctMode wf Flags 0x30 (pfro+pfxw) seq 25361

BR: @ 0x180967032E8, ctMode nl Flags 0x30 (pfro+pfxw) seq 25363

treeP 0x180C3786218 C btFastTrack 0 1 ranges mode RO/XW:

BLK [0,INF] mode XW node <320>

Fcntl: @ 0x18096703320, ctMode nl seq 25361

inode 6675751424 snap 0 USERFILE nlink 462 genNum 0x72AF4528 mode 0200042750: **drwxr-s---**

...

===== dump kthread =====

Current time 2025-04-16_11:59:07-0500

...

Stack for process **17505 (python)** kernel **thread 17505**

[<0>] cxiWaitEventWait+0x208/0x330 [mmfslinux]

[<0>] ThCond::internalWait(KernelSynchState*, unsigned int, void*)+0x58/0x250 [mmfs26]

[<0>] ThCond::kWait(int, char const*)+0x1d8/0x410 [mmfs26]

[<0>] KernelMailbox::sendToDaemonWithReply()+0x250/0x3e0 [mmfs26]

[<0>] kShHashFetch(KernelOperation*, CacheObj*, unsigned short, short, int, void*, int, unsigned int)+0x21f/0x250 [mmfs26]

[<0>] lockFileHeldM(KernelOperation*, OpenFile*, FileUID, LkObj::LockModeEnum, int, unsigned int*, unsigned int)+0x256/0x550 [mmfs26]

[<0>] lockFile(KernelOperation*, OperationLockMode, FileUID, LkObj::LockModeEnum, OpenFile**, WhatLocked*, int)+0x332/0x620 [mmfs26]

[<0>] gpfsAccess(gpfsVfsData_t*, cxiNode_t*, int, int, void*, int*, ext_cred_t*)+0x400/0xc30 [mmfs26]

[<0>] gpfs_i_permission+0x1ab/0x370 [mmfslinux]

[<0>] inode_permission+0xaa/0x140

[<0>] vfs_mkdir+0xa2/0x1b0

[<0>] bpf_trampoline_10737456105_1+0x68/0x1000

[<0>] vfs_mkdir+0x5/0x1b0

[<0>] do_mkdirat+0x7d/0xf0

[<0>] do_syscall_64+0x5b/0x1b0

[<0>] entry_SYSCALL_64_after_hwframe+0x61/0xc6

- **To breakup this deadlock:**

- 1 . Kill python process/reboot mmfsd manually

- 2 . Auto breakup

`mmhealthPendingRPCExpelThreshold`

Raises an mmhealth WARNING event rpc waiters expel, and expels a node when pending RPC messages on the node pass the threshold time, which is specified in seconds. The default threshold value is 0. Nodes do not expel automatically.

`mmhealthPendingRPCWarningThreshold`

Raises an mmhealth WARNING event rpc waiters. The default threshold value is 180 seconds. To disable this option, set the threshold value to 0.

- <https://www.ibm.com/docs/en/storage-scale/5.2.3?topic=reference-mmchconfig-command>

Debug to a long waiter triggered by vfs call

- Debugging IO/VFS pattern from Scale perspective - mailbox

A **mailbox** is a **software structure** (often a queue) that allows **synchronous/asynchronous, message-based communication** between processes or threads:

- **Daemon mmfsd ↔ User application daemon in user-space**
- **Daemon mmfsd inside communication**

- **Synchronous/Asynchronous:** The sender and receiver do not need to synchronize (unlike pipes).
- **Buffered:** Messages are stored in a queue-like structure.
- **Non-blocking / Blocking options:** Readers may poll or wait.
- **Kernel/User-level:** Mailboxes can be implemented in either space, sometimes using shared memory or kernel-managed queues.

- **Example:**

===== dump kthread =====

Current time 2025-04-16_11:59:07-0500

...

Stack for **process 17505** (python) **kernel thread 17505** ----→ **17505 = 0x4461**

[<0>] cxiWaitEventWait+0x208/0x330 [mmfslinux]

[<0>] ThCond::internalWait(KernelSynchState*, unsigned int, void*)+0x58/0x250 [mmfs26]

[<0>] ThCond::kWait(int, char const*)+0x1d8/0x410 [mmfs26]

[<0>] **KernelMailbox::sendToDaemonWithReply()+0x250/0x3e0 [mmfs26] -----→ send mb request to gpfs daemon in blocking mode**

[<0>] kShHashFetch(KernelOperation*, CacheObj*, unsigned short, short, int, void*, int, unsigned int)+0x21f/0x250 [mmfs26]

[<0>] lockFileHeldM(KernelOperation*, OpenFile*, FileUID, LkObj::LockModeEnum, int, unsigned int*, unsigned int)+0x256/0x550 [mmfs26]

[<0>] lockFile(KernelOperation*, OperationLockMode, FileUID, LkObj::LockModeEnum, OpenFile**, WhatLocked*, int)+0x332/0x620 [mmfs26]

[<0>] gpfsAccess(gpfsVfsData_t*, cxiNode_t*, int, int, void*, int*, ext_cred_t*)+0x400/0xc30 [mmfs26]

[<0>] gpfs_i_permission+0x1ab/0x370 [mmfslinux]

[<0>] inode_permission+0xaa/0x140

[<0>] vfs_mkdir+0xa2/0x1b0

[<0>] bpf_trampoline_10737456105_1+0x68/0x1000

[<0>] **vfs_mkdir+0x5/0x1b0 -----→ mkdir()**

[<0>] do_mkdirat+0x7d/0xf0

[<0>] do_syscall_64+0x5b/0x1b0

[<0>] entry_SYSCALL_64_after_hwframe+0x61/0xc6

- As mmfsd receive a mailbox request, a gpfs thread (waiter) will be generated to handle this request:

===== dump thread =====

Current time 2025-04-16_11:59:02-0500

...

SharedHashTabFetchHandlerThread: desc 0x7FF3700023A0 handle 0xF50B03FA parm 0x560005 highStackP 0x7FF9C4343510

pthread 0xC4345700 kernel thread id 66404 (slot -1) pool 3 ThPoolMBHandler

per-thread gbls:

0:0x180604E6190 1:0x0 2:0x0 3:0x7 4:0x56 5:0x1 6:0x0 7:0x0

8:0x0 **9:0x4461** 10:0x4 11:0x0 12:0x1 13:0xA800003 14:0x0 15:0x0

--→ **The 9th id is user app thread id in hex 0x4461 = 17505**

16:0x0 17:0x0 18:0x7FF3C8026AC0 19:0x0 20:0x0

waiting for tmMsgTellAcquire1 (Medium cl:2 fs:-1 1/1 pending)

waiting for 230.656229000 seconds

...

Traceback:

...

#13: 0x0000555558FAB50 HandleMBHashFetch(MBHashFetchParms*) + 0x120 at ??:0

#14: 0x0000555558E9530 **Mailbox::msgHandlerBody(void*) + 0x350 at ??:0**

#15: 0x0000555558CB9B2 Thread::callBody(Thread*) + 0x42 at ??:0

#16: 0x0000555558B89B0 Thread::callBodyWrapper(Thread*) + 0xA0 at ??:0

#17: 0x00007FFFF79B41CF start_thread + 0xEF at ??:0

#18: 0x00007FFFF66D3DD3 __GI___clone + 0x43 at ??:0

- This Python process was calling a mkdir() VFS call and sent a blocking mailbox request to the mmfsd daemon using KernelMailbox::sendToDaemonWithReply()—a method used in IBM Storage Scale for inter-process communication between the kernel and user space.

==== dump kthread =====

Current time 2025-04-16_11:59:07-0500

...

Stack for process 17505 (python) kernel thread 17505 -> 17505 = 0x4461

- The mailbox message was delivered to mmfsd, which then spawned the SharedHashTabFetchHandlerThread to process the mkdir request. However, this handler thread became stuck while waiting on a token request, causing the mkdir call in python process to hang.

==== dump thread =====

Current time 2025-04-16_11:59:02-0500

...

SharedHashTabFetchHandlerThread: desc 0x7FF3700023A0 handle 0xF50B03FA parm 0x560005 highStackP 0x7FF9C4343510

pthread 0xC4345700 kernel thread id 66404 (slot -1) pool 3 ThPoolMBHandler

per-thread gbls:

0:0x180604E6190 1:0x0 2:0x0 3:0x7 4:0x56 5:0x1 6:0x0 7:0x0

8:0x0 **9:0x4461** 10:0x4 11:0x0 12:0x1 13:0xA800003 14:0x0 15:0x0 -> 0x4461 = 17505

16:0x0 17:0x0 18:0x7FF3C8026AC0 19:0x0 20:0x0

FGDL and Rename/mkdir vfs calls

- **Fine-Grained Directory Locking (FGDL)** is an internal optimization in **IBM Storage scale** that improves performance and scalability of **concurrent updates to shared directories**.

Previously GPFS used byte-range tokens on whole directory blocks. Such locking mechanism can become a bottleneck when **many clients or threads access the same directory concurrently**, leading to:

High lock contention

Reduced parallel performance

Metadata operation delays

- **FGDL addresses this by introducing fine-grained locks**, allowing:

Multiple clients to **operate on different parts of the same directory simultaneously**

Reduced lock contention

Improved **throughput for directory-intensive workloads**

- **How FGDL Works**

Collaborating with token manager and meta-node, FGDL uses a new byte-range tokens to address individual entries of the hash-based organization of GPFS directories. The directory meta-node responds for coordinating access to the directory's disk blocks, while token manager maintains tokens, using directory name hash function values as offset that help address individual entries.

Lock granularity is managed by GPFS **without requiring application changes**.

- However, two VFS operations—rename and mkdir—are not supported by Fine-Grained Directory Locking (FGDL). These operations require locking the entire parent directory, which can block other nodes from accessing different files within the same directory. This behavior may negatively impact metadata-intensive or highly concurrent workloads.

- Trace example:

```
[root@proto2 testdir]# pwd
```

```
/gpfs/gpfs1/testdir
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]# ls -ld /gpfs/gpfs1/testdir
```

```
218624 /gpfs/gpfs1/testdir
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]# ls -li testfile
```

```
218625 testfile
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]# mmtrace start; mv testfile testfile_1; mmtrace stop
```

```
mmtrace: move /tmp/mmfs/lxtrace.trc.proto2-hs.cpu0 /tmp/mmfs/trcfile.2025-07-28_14.55.48.180333.proto2-hs.cpu0
```

```
mmtrace: formatting /tmp/mmfs/trcfile.2025-07-28_14.55.48.180333.proto2-hs to /tmp/mmfs/trcrpt.2025-07-28_14.55.48.180333.proto2-hs.gz
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]# zgrep "FSOperation::createLockedFile enter: inode" /tmp/mmfs/trcrpt.2025-07-28_14.55.48.180333.proto2-hs.gz
```

```
0.056429 177839 TRACE_FS: FSOperation::createLockedFile enter: inode 218625 snap 0 fileset 0, lock mode wanted ro
```

```
0.056455 177839 TRACE_FS: FSOperation::createLockedFile enter: inode 218624 snap 0 fileset 0, lock mode wanted wf
```

```
0.056480 177839 TRACE_FS: FSOperation::createLockedFile enter: inode 218625 snap 0 fileset 0, lock mode wanted xw
```

```
[root@proto2 testdir]# ls -ld /gpfs/gpfs1/testdir
```

```
218624 /gpfs/gpfs1/testdir
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]# mmtrace start; mkdir testsubdir; mmtrace stop
```

```
mmtrace: move /tmp/mmfs/lxtrace.trc.proto2-hs.cpu0 /tmp/mmfs/trcfile.2025-07-28_14.56.59.182129.proto2-hs.cpu0
```

```
mmtrace: formatting /tmp/mmfs/trcfile.2025-07-28_14.56.59.182129.proto2-hs to /tmp/mmfs/trcrpt.2025-07-28_14.56.59.182129.proto2-hs.gz
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]#
```

```
[root@proto2 testdir]# zgrep "FSOperation::createLockedFile enter: inode" /tmp/mmfs/trcrpt.2025-07-28_14.56.59.182129.proto2-hs.gz
```

```
0.092563 26449 TRACE_FS: FSOperation::createLockedFile enter: inode 218624 snap 0 fileset 0, lock mode wanted xw
```

```
0.092593 26449 TRACE_FS: FSOperation::createLockedFile enter: inode 218626 snap 0 fileset 0, lock mode wanted xw
```

```
[root@proto2 testdir]#
```

- **App pattern:**

Different processes/nodes attempt to access different files and sub-directories under the same directory AND...

The directory contains lots of files and sub-directory AND ...

Heavy IO loads mix with many mkdir/rename calls

Performance may deteriorate further if network or disk I/O latency increases.

- **Best practice:**

Design workloads to reduce rename/mkdir in shared directories.

Use sharded directory structures to minimize contention (e.g., per-task or per-process directories).

- Q&A