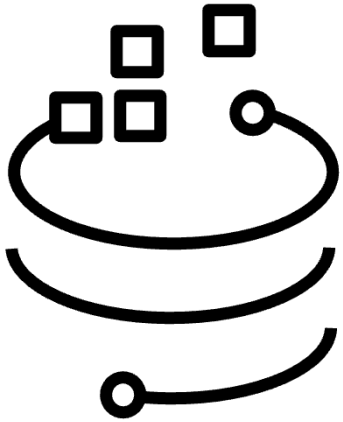
Storage Scale Network Related Topics

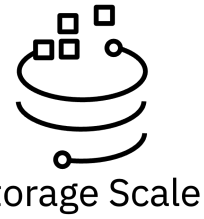
V. 1.06

John Lewars
Storage Scale Performance Architect



Storage Scale

Disclaimer



IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

IBM reserves the right to change product specifications and offerings at any time without notice. This publication could include technical inaccuracies or typographical errors. References herein to IBM products and services do not imply that IBM intends to make them available in all countries.

Storage Scale's Dependence On the Network

Why Is Storage Scale dependent on the network?

- Storage Scale is a clustered file system. It depends on timely and reliable TCP/IP communication between all nodes in the cluster to ensure data integrity and good performance, while preventing hangs and deadlocks.
- The best performance will generally be achieved with RDMA, which also requires low latency and reliable data transfers over the network, but TCP/IP is always a functional requirement for Storage Scale clusters.
- Nodes coordinate over the network to perform different tasks/roles as required to achieve scalable performance and resiliency.

Timely communication between nodes is required for several reasons, including:

- **Handling of disk I/Os.** Since not all nodes typically have direct access to disk, those nodes without disk access need to be communicate with nodes that serve disks directly. For example, NSD (Network Shared Disk) clients need to be able to communicate with NSD servers to perform file system I/O.
- **Ensuring data consistency/integrity.** Since the file system must serialize access to the data that multiple nodes operate on, distributed locks and tokens must be used to ensure consistency of both data and metadata, and token manager nodes are responsible for controlling access to tokens.

Storage Scale Expels – What are They?

- When a node is expelled from a given Storage Scale cluster, the expelled node is impacted as follows:

  1. All the file systems mounted from the storage cluster will be unmounted from the node.

  2. All the tokens for the storage cluster will be revoked from the node.

  3. Applications running on the node that are using the impacted file system(s) will typically die because:

     a) I/O requests to these file systems will fail with an EIO (input/output), ENOENT (no such file or directory), or ESTALE (Stale File Handle) error.

     b) Applications may also potentially receive a terminating SIGBUS signal if they have memory mappings or execute code residing on files that become unavailable after the unmount.

- An expelled node may be referred to as having “lost cluster membership”.

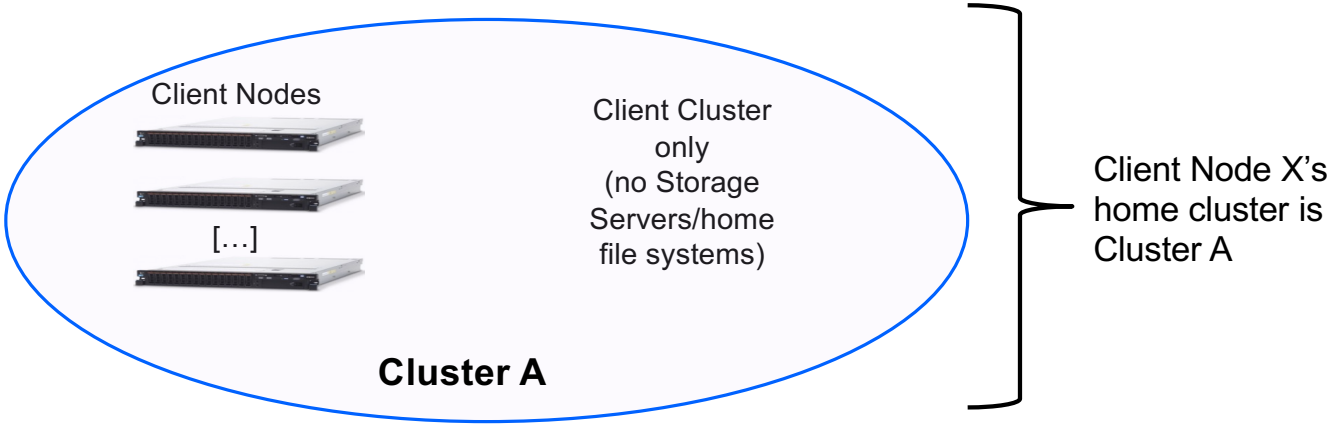
- Sometimes an expelled node may be described as having “lost quorum” but the use of ”quorum” in this context is discouraged as it differs from how “quorum” is used in the terms “cluster **node quorum**” and “**file system descriptor quorum**”.

- To ensure stability and good performance, Storage Scale requires reliable communication between all nodes using a file system.

  - Expels are required to prevent unresponsive nodes from causing delays or hangs – nodes that can’t achieve Storage Scale’s communication requirements within a cluster will be expelled from the cluster

Storage Scale Expels – What are They?

- A node may be expelled from one cluster and lose access to that cluster's file systems, while still maintaining membership in another cluster.



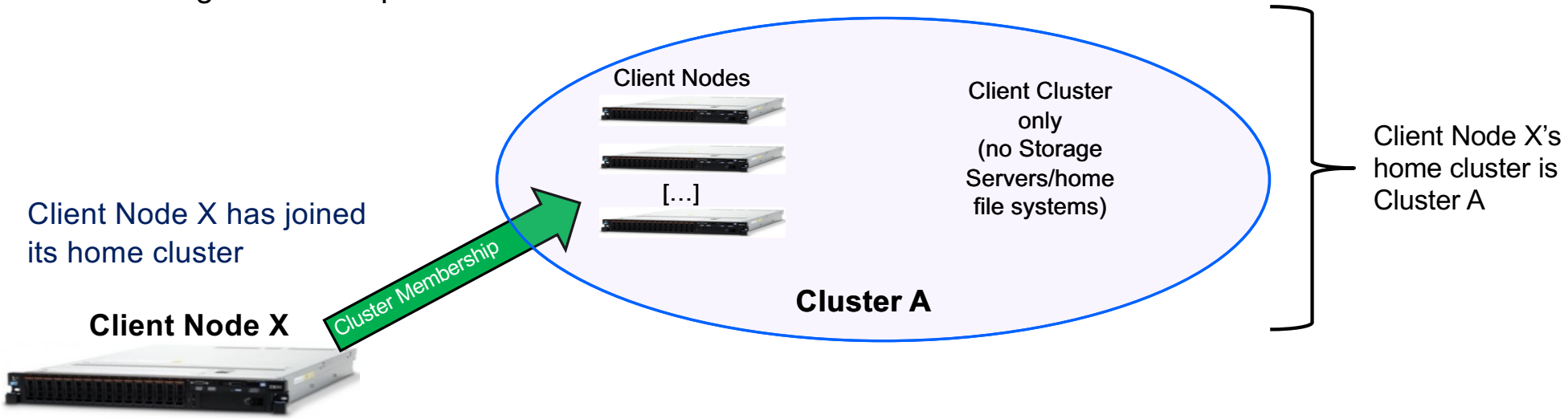
- In this example, node X's home cluster is a client cluster with no file systems

- Any node must join its own home cluster to receive up-to-date configuration info before it can:

1. submit I/O to any home cluster file system(s)
2. join any remote cluster

Storage Scale Expels – What are They?

- A node may be expelled from one cluster and lose access to that cluster's file systems, while still maintaining membership in another cluster.



- After any node joins its own home cluster it is eligible to:

1. submit I/O requests to any home file system(s) that might be defined in the home cluster (there are no such file systems in this example)

2. join remote clusters that have their own file systems that the client has authorization to mount

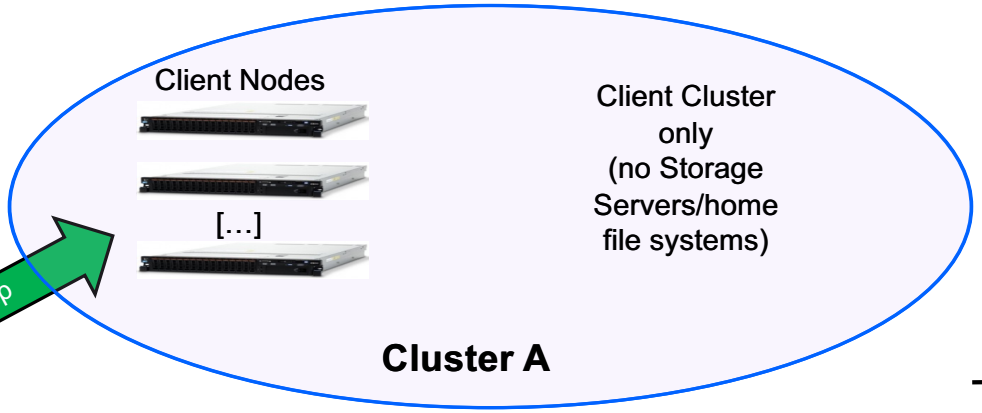
Storage Scale Expels – What are They?

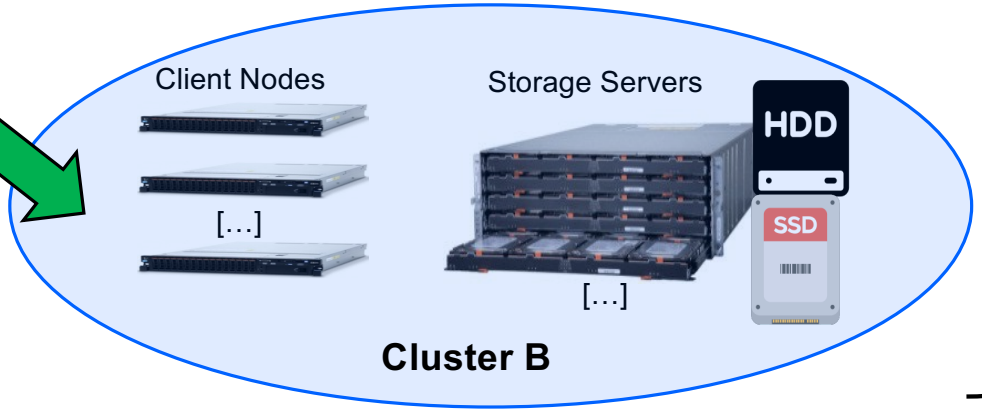
- A node may be expelled from one cluster and loose access to that cluster's file systems, while still maintaining membership in another cluster.

- Node X joins the remote Cluster B and can access Cluster B's home file system(s)

Cluster Membership

Cluster Membership





- Once a client has joined a remote cluster its membership in the remote cluster is no longer contingent on maintaining its home cluster membership.

Client Node X's home cluster is Cluster A

Client Node X is now a member of Cluster B, in most[1] respects, just like nodes which have Cluster B as their home cluster.

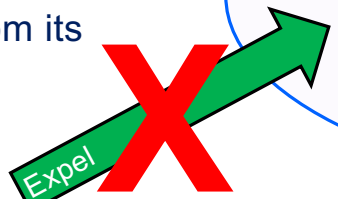
1 Node X cannot act as a manager node for cluster B.

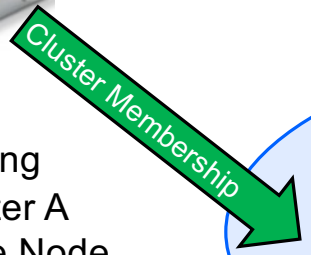
Storage Scale Expels – What are They?

- A node may be expelled from one cluster and loose access to that cluster's file systems, while still maintaining membership in another cluster.

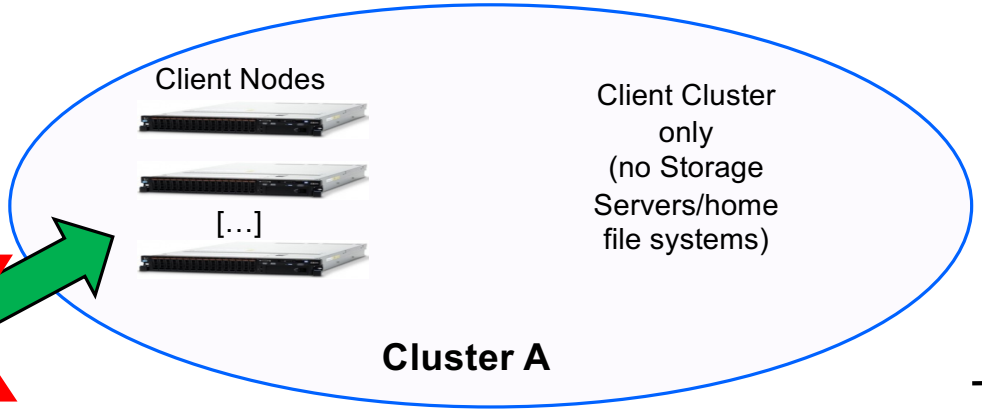
- Node X is expelled from its home Cluster A

Client Node X

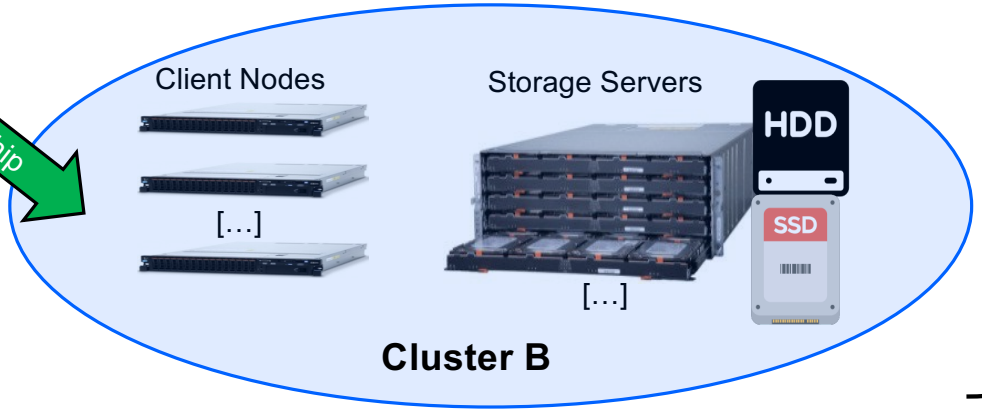




- Client Node X's being expelled from Cluster A does not impact the Node X's access to Cluster B's home file system(s)



Client Node X was expelled from Cluster A, but this does not impact its membership in other clusters



Client Node X is still a member of Cluster B and can access Cluster B's home file system(s)

Storage Scale Expels – What are They?

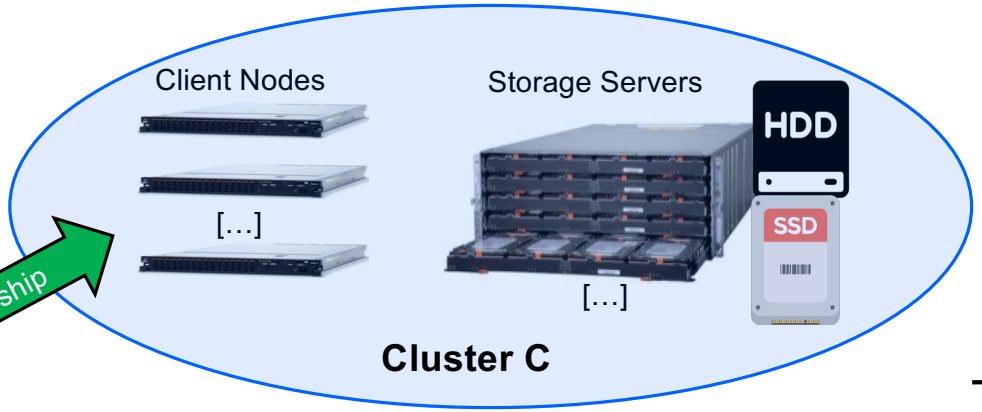
- A node may be expelled from one cluster and loose access to that cluster's file systems, while still maintaining membership in another cluster.

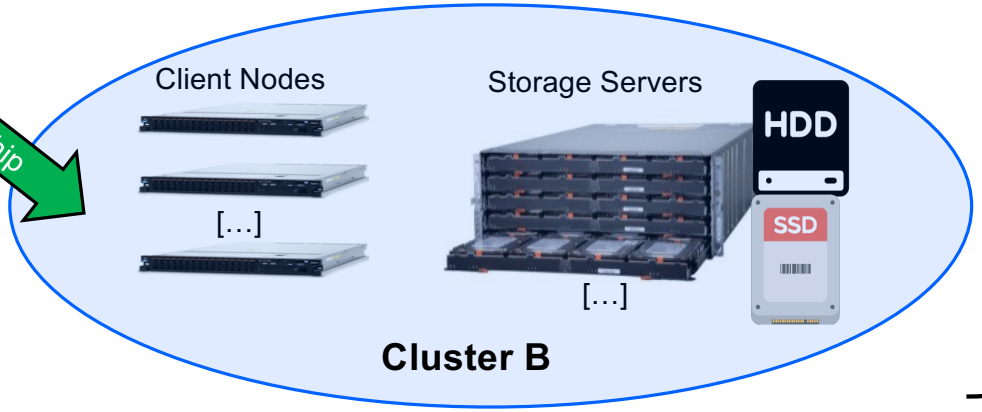
- Node X has joined two remote clusters (Cluster C is a second remote cluster with its own storage)

Client Node X

Cluster Membership

Cluster Membership





- Membership in one remote cluster is not contingent on membership in another remote cluster.

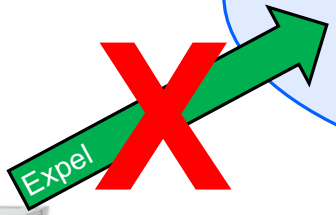
Client Node X is a member of Cluster A, in most respects, just like any native member of cluster A, but Node X will not act as a manager node in cluster A.

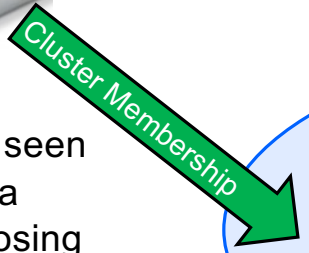
Client Node X is a member of Cluster B, in most respects, just like any native member of cluster B, but Node X will not act as a manager node in cluster B.

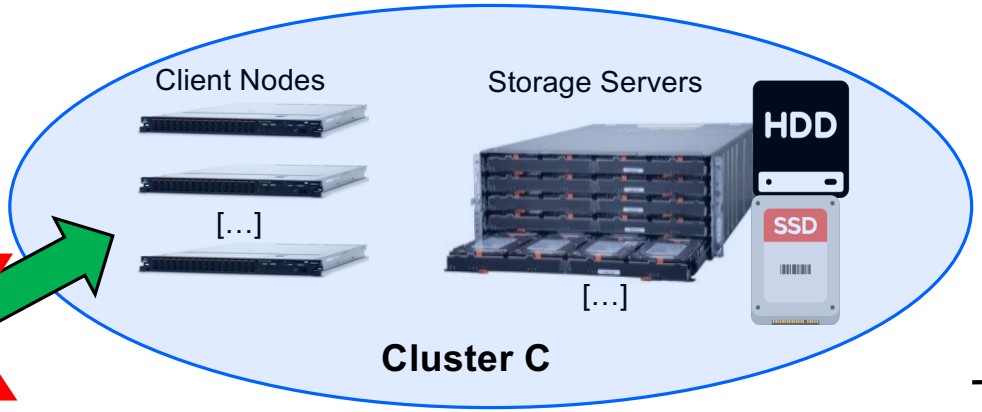
Storage Scale Expels – What are They?

- A node may be expelled from one cluster and loose access to that cluster's file systems, while still maintaining membership in another cluster.

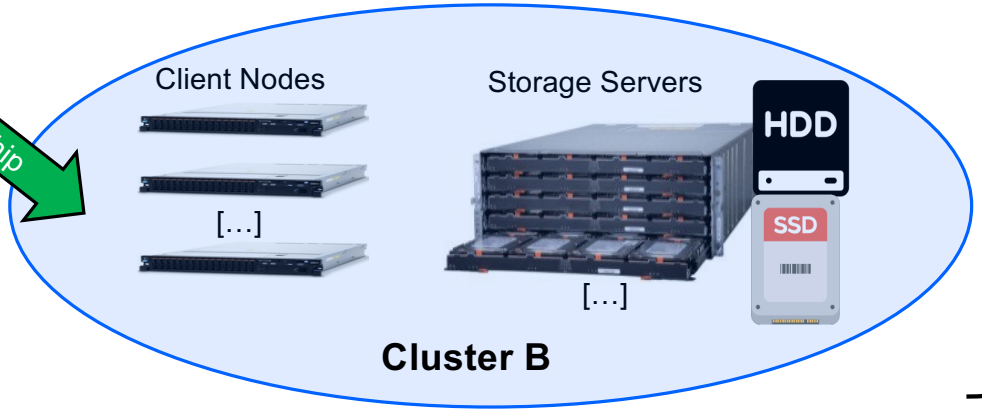
- If Client Node X is expelled from one remote cluster but can still submit I/O to file systems belonging to the other cluster







Client Node X was expelled from Cluster C, but this does not impact its membership in other clusters.



Client Node X is still a member of Cluster B.

- Just like the behavior seen when membership in a home cluster is lost, losing membership in one remote cluster doesn't impact another remote cluster's membership

Types of Expels

1. **Disk lease timeout related expels.**

   - All nodes must renew their disk leases with the cluster manager by sending it a ccMsgDiskLease RPC.
   - Failure to renew the lease will first cause a node to be unable to submit I/O requests and then, shortly after this, the node may be expelled from the cluster by the cluster manager if it does not renew its lease.

2. **Node-requested Expels (also referred to as RPC timeout related node requested expels)**

   a. After a source node sends an RPC request to a target node, on some interval the sender will send a commHandleCheckMessages RPC to verify that the request is still pending, and, if the recipient cannot reply in a timely manner to the RPC to confirm that the request is still pending, the sender (source) of the request will request the cluster manager to expel the recipient (target) of the request.

   b. If a source node cannot establish a connection to that target node in a timely manner, the node that cannot establish the connection will request the cluster manager to expel the target node

   - For cases (a) and (b), the cluster manager will choose to expel either the source or destination node

3. **Expels requested by the mmexpelnode command or, using the same mechanism, enabling the mmchconfig mmhealthPendingRPCExpelThreshold option to recover nodes not responding to token revoke requests**

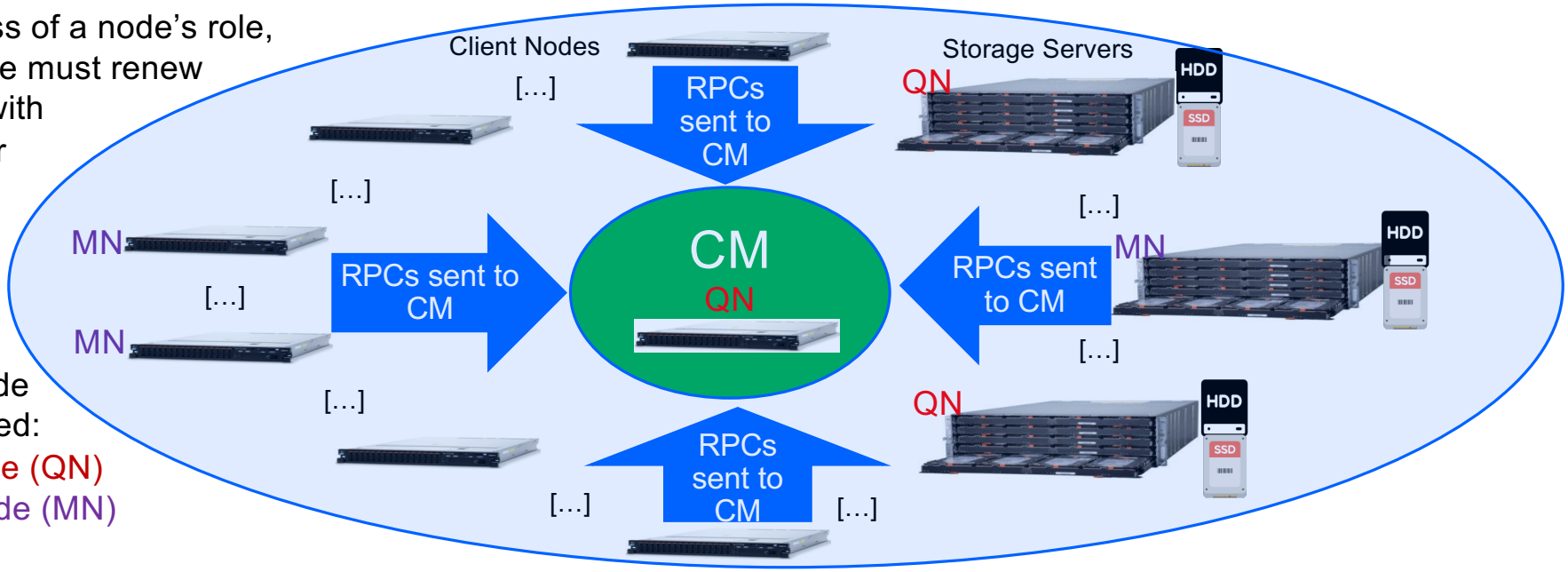
   - A Storage Scale admin can expel a node by running the **mmexpelnode** command directly or automating the command to run.
   - The **mmchconfig** mmhealthPendingRPCExpelThreshold option provides a way of automating expelling of nodes when they don't respond to token revoke requests, thereby potentially avoiding cluster-wide hangs.

Recovery From Expels and Making Expels Persistent

- When a node is expelled from a cluster, it will attempt to rejoin by sending a probe request to the quorum nodes, which will respond to the probe request by pointing the expelled probing node to the cluster manager.
- After the cluster manager has expelled a node and all recovery protocols are run, the cluster manager will allow the expelled node to rejoin the cluster provided that the expel has not been requested to be persistent, which occurs when the expel is initiated by running the mmexpelnode command with the default options (when mmexpelnode is run with the **-o** or **--once** option, then the expel is not requested to be persistent).
  - An expel that has been requested to be persistent can be reset (no longer requested to be persistent) by any of these conditions:
    a) Running the mmexpelnode command with the **-r** or **--reset** options removes the specified node(s) from the persisted list of of expelled nodes.
    b) If cluster node quorum is lost (see Cluster quorum with quorum nodes) the list of expelled nodes is lost.
    c) On clusters running any code level earlier than 5.2.1, or (on newer code levels) if the mmchconfig option **disablepersistExpelList=yes** is set, then the persisted list of expelled nodes will be lost when the cluster manager moves to a new node, which can occur when:
       i. the cluster manager fails, and a new manager takes over
       ii. the cluster manager is deliberately changed by running the mmchmgr command with the **-c** option

Details on Disk Lease Timeout Related Expels

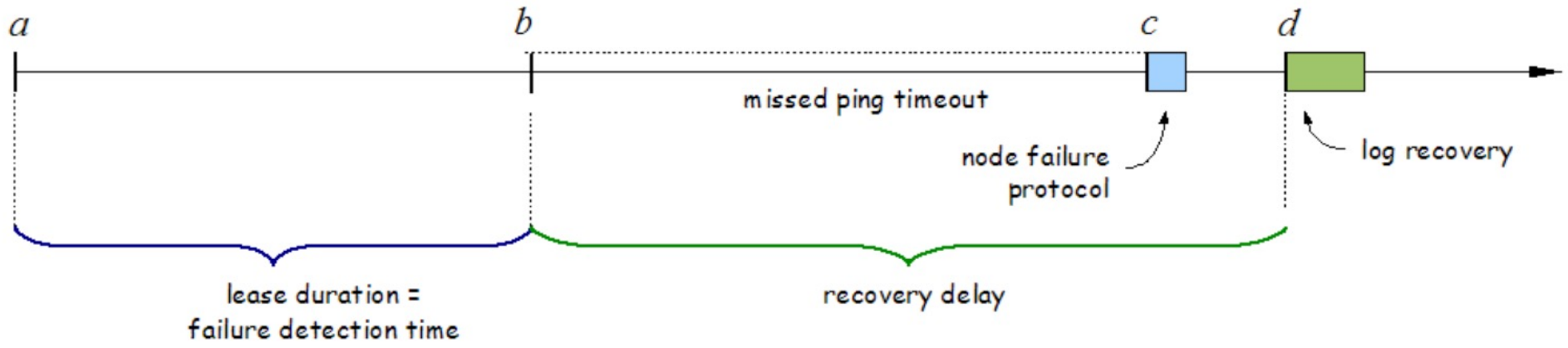
- In each cluster, the cluster manager (CM) manages ‘disk leases’; disk leases are needed to submit I/O requests to a given GPFS cluster.
- Each node (regardless of its role) needs to renew its lease with the cluster manager (CM) node at regular intervals (leases are 35 seconds by default) or the node will no longer be able to submit I/O requests to the cluster.
- Shortly after lease expiration occurs, the CM may end up expelling the node if it still hasn’t renewed its lease.
- Regardless of a node’s role, every node must renew its lease with the cluster manager to avoid being expelled.
- Example Node Roles depicted:
  Quorum Node (QN)
  Manager Node (MN)



Details on Disk Lease Timeout Related Expels

- From the researcher Frank S., here is an iconic diagram that explains the lease timeout flow when disk leasing is in effect (in this presentation we'll focus only on the disk leasing case, which is the typical lease configuration):

With disk leasing



- The length of a non-quorum node's lease (in seconds) is defined by the configuration options **leaseDuration** and **failureDetectionTime**. Based on IBM test experience, we recommend always tuning **failureDetectionTime** and never changing **leaseDuration** via mmchconfig (setting **failureDetectionTime** will also change **leaseDuration).**

- A node is not expelled until after its lease expires (lease duration=failure detection time in seconds) and then the 'missed ping timeout' window completes. In the 'missed ping timeout' window, the cluster manager (CM) will send ICMP datagrams (pings) to the node with the expired lease. The length of the 'missed ping timeout' window is dependent the node's response to the CM's pings as per the details on the next chart.

Details on Disk Lease Timeout Related Expels

- If a node fails to respond to pings from the cluster manager, the length of the ‘missed ping timeout’ window is defined by the mmfsd computed value **missedPingTimeout**, which defaults to 30 seconds. GPFS sets the value of **missedPingTimeout** based on the value of GPFS configuration parameters, taking the maximum of **minMissedPingTimeout** and **leaseRecoveryWait** (but never exceeding **maxMissedPingTimeout).**

- If a node responds to pings from the cluster manager, the length of the ‘ping timeout’ window is defined by the GPFS configuration parameter **totalPingTimeout**, which defaults to 120 seconds.

- For details on the meaning of the lease tuning parameters and how they are derived, see the Backup section of this presentation for slides on “**Lease Configuration Parameters As they Apply to Disk Leasing Described”**

- When trying to run Storage Scale on a network that is less resilient (e.g. has a packet loss problem), some customers choose to increase the length of the ’missedPingTimeout’ window to allow nodes to be given more time before they are expelled (after their lease expires). Here’s one such set of tuning options that’s been tested in the field to try to ride over short windows in which network connectivity may break (these parameters require a restart of Storage Scale and changing **failureDetectionTime** requires restarting the entire cluster):

```
mmchconfig minMissedPingTimeout=60 # increase missedPingTimeout=60 and potentially
                                   # ride over network outages without an expel
mmchconfig failureDetectionTime=60 # to increase the lease duration from 35 to 60s
```

Checking Storage Scale Lease Related Configuration Values

- **Details on how the Storage Scale Lease Configuration Parameters can be found in the backup section.**
- Here's how to dump out the relevant lease tuning (mmchconfig) configuration values using mmdiag (mmlsconfig can also be run to query each of these values).

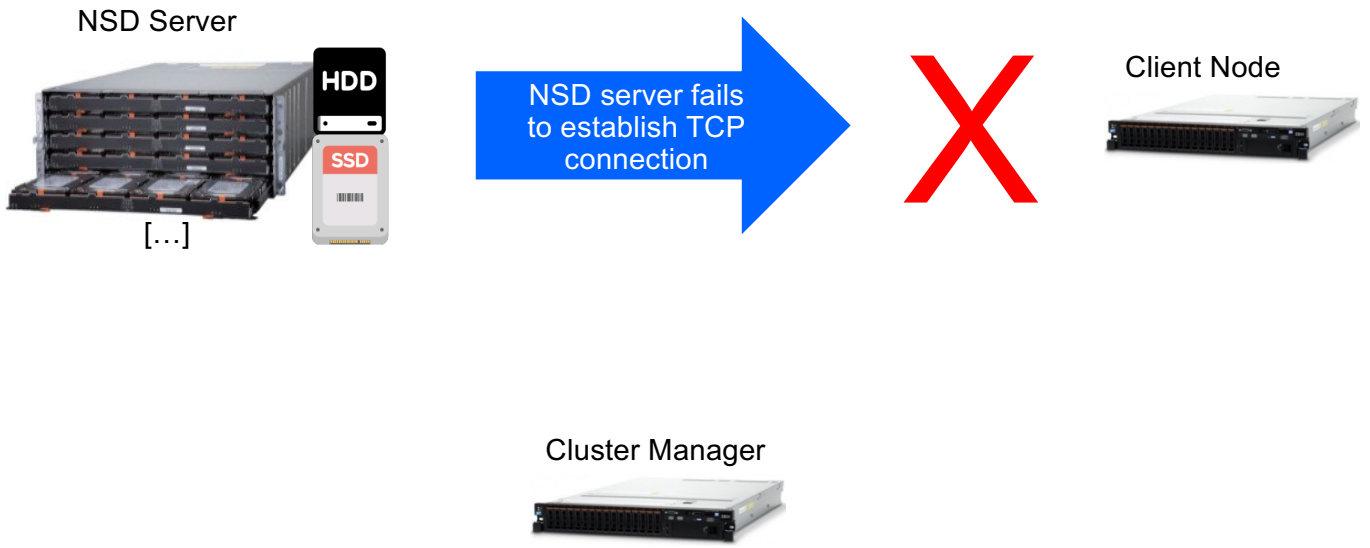
```
# mmdiag --config | egrep \
"failureDetectionTime|leaseDMSTimeout|leaseDuration|leaseRecoveryWait|PingTimeout"
failureDetectionTime -1
leaseDMSTimeout -1
leaseDuration -1
leaseRecoveryWait 35
maxMissedPingTimeout 60
minMissedPingTimeout 3
```

Details on Node Requested Expels

- Step 1: In this example, an NSD server node attempts to establish can't establish a connection to a client node.

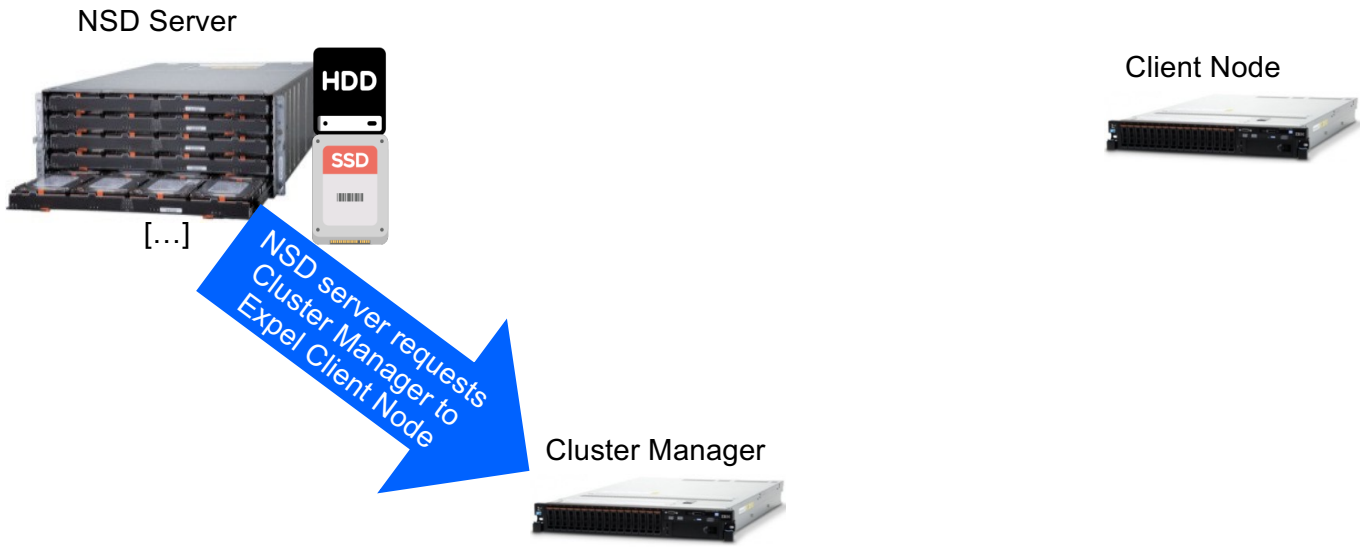
Details on Node Requested Expels

- Step 2: The NSD Server's attempt to connect to the Client Node fails (the TCP timeout was hit, with an ETIMEDOUT error returned, and the connection attempt was retried without success)



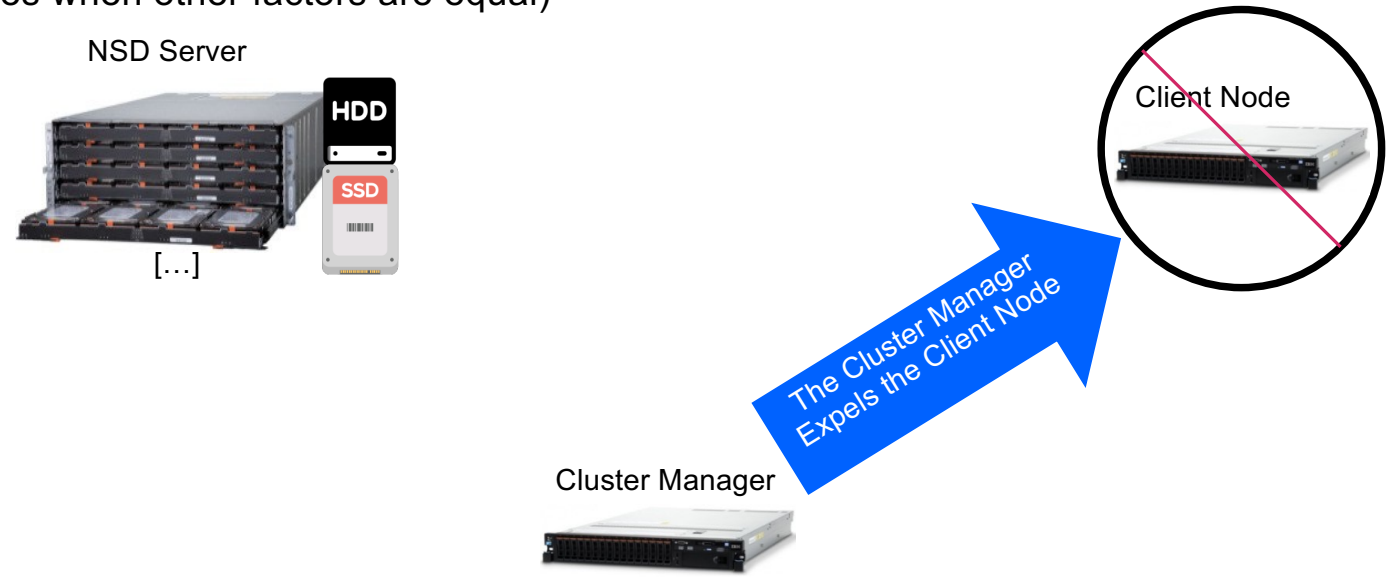
Details on Node Requested Expels

- Step 3: The NSD Server will send an RPC to the Cluster Manager requesting it to expel the Client Node.
  - The Cluster Manager needs to decide if it should expel the NSD Server making the expel request or the target Client Node.



Details on Node Requested Expels

- Step 3: Having received a request from the NSD Server to expel the depicted Client Node, Cluster Manager follows an algorithm (described in an upcoming slide).
  - In this example, both the NSD Server and the Client Node are non-quorum home cluster nodes that have not been the recent source or target of other expels requests, so the cluster manager decides to expel the Client Node (NSD servers are favored over Client Nodes when other factors are equal)



Details on Node Requested Expels (Longer Timeout Flow 1/2)

- Every 90 seconds the EEWatchDogThread in the GPFS daemon will look for pending messages outstanding to all destinations.
- For every destination that has pending messages, the GPFS daemon will send a commMsgCheckMessages RPC to verify the destination is aware of all pending messages.
- The commMsgCheckMessages RPC verifies both (1) network connectivity is good and (2) the full list of RPCs that are pending is acknowledged (verifying that no messages have been lost)
- The destination must reply and acknowledge ALL messages are being worked within the timeout as described below (if a network error leads to a new connection being established, then the timeouts described in the upcoming **Shorter Timeout Flows** slide will come into play)
- The commMsgCheckMessages RPC verifies both (1) network connectivity is good and (2) the full list of RPCs that are pending is acknowledged (verifying that no messages have been lost)
- For non-quorum nodes, the timeout on the commMsgCheckMessages is defined to be the minimum of (10 * **leaseDuration)** and (300). As the default value for **leaseDuration** is 35, the timeout typically ends up being 300 seconds on a non quorum node (quorum nodes use 2/3rd the typical lease length).

Details on Node Requested Expels (Longer Timeout Flow 2/2)

- The timeout on commMsgCheckMessages includes sending a single retry of the message to allow for timing holes in how the pending messages are viewed on the source and target nodes.

- If the target of a given message has not acknowledged every RPC that we've attempted to verify is pending via the commMsgCheckMessage RPC (with 300 second timeout) then the source node will request the cluster manager to expel the target node.

- If the source node does request the cluster manager to expel the target node, the cluster manager then needs to decide whether it should expel the source or destination node (more on that process on the later "**Node Requested Expels - Which Node Gets Expelled?"** chart).

Details on Node Requested Expels – Shorter Timeout Flows

Note that there are potentially shorter timeouts that can lead to node requested expels before hitting the 300 second timeout that occurs when communicating with destinations that have connections already established. For this reason, consider tuning the system to minimize reconnects (described later).

If a node needs to (1) reconnect to, or (2) send an RPC message to another node which it has no established connection to, a shorter timeout can be hit in the code paths that establish new connections. Two examples of these shorter timeout cases, for TCP connections, are as follows:

1. When a new TCP connection must be established to another host, if ARP resolution is not working and a valid ARP entry is not present for the destination IP, a ”No Route To Host”* error will occur establishing the connection, by default after 3 seconds of ARP retries (controlled by the applicable sysctl mcast_solicit parameter). Storage Scale will retry establishing the connection, but the RPC could timeout in as little as 6 seconds if ARP is not working and the default tuning is in effect.

2. If ARP is working but TCP connections cannot be established, an RPC timeout can, with default TCP tuning, occur in about two minutes (Storage Scale will retry the TCP connection if it times out on connect). This ~60 second timeout covers more than the TCP connection timeout, as Storage Scale needs to run additional (TLS) authentication code in gskit after the connection is established.

* Note that “No route to host” (EHOSTUNREACH) errors can be seen in other flows and one possible source of this error can be a firewall misconfiguration that blocks Scale messages.

Details on Node Requested Expels – Shorter Timeout Example 1

- Example of (1 on previous **Shorter Timeout Flows** slide) timeout establishing new TCP connection when ARP doesn't resolve (no ARP entry causes "No route to host", EHOSTUNREACH, error):

```
2024-10-05_19:13:47.152+0200: [N] Connecting to 10.200.42.204 fscc-sr650v2-4
<c0n21>:[0]
2024-10-05_19:13:50.251+0200: [N] Close connection to 10.200.42.204 fscc-
sr650v2-4 <c0n21>:[0] (No route to host)
2024-10-05_19:13:50.251+0200: [N] Retry connection to 10.200.42.204 fscc-
sr650v2-4 <c0n21>:[0]
2024-10-05_19:13:53.323+0200: [N] Close connection to 10.200.42.204 fscc-
sr650v2-4 <c0n21>:[0] (No route to host)
[…] >> Network connectivity established at this point (ARP works at 19:13:54 but
the client has already decided it will send an expel request at 19:15:05 […]
2024-10-05_19:15:05.620+0200: [N] Request sent to 9.152.187.70 (fscc-sr650-49)
to expel 9.152.185.166 (fscc-sr650v2-4) from cluster GNRocks.fscc-ess3500-1-a
```

- Connectivity is down for only 6.1 seconds (from 19:13:47.152 - 19:13:53.251) before the client makes the decision to send a request for fscc-sr650v2-4 be expelled, but it delays sending the expel request for fscc-sr650v2-4 to the cluster manager to give the cluster manager time to discover if there is a network issue impacting fscc-sr650v2-4, which might prevent it from renewing its lease.
- The total time taken from introducing the network connectivity failure to the time that the expel request is sent to the cluster manager is 1 minute + 18 seconds (19:15:05 - 19:13:47).

Details on Node Requested Expels – Shorter Timeout Example 2

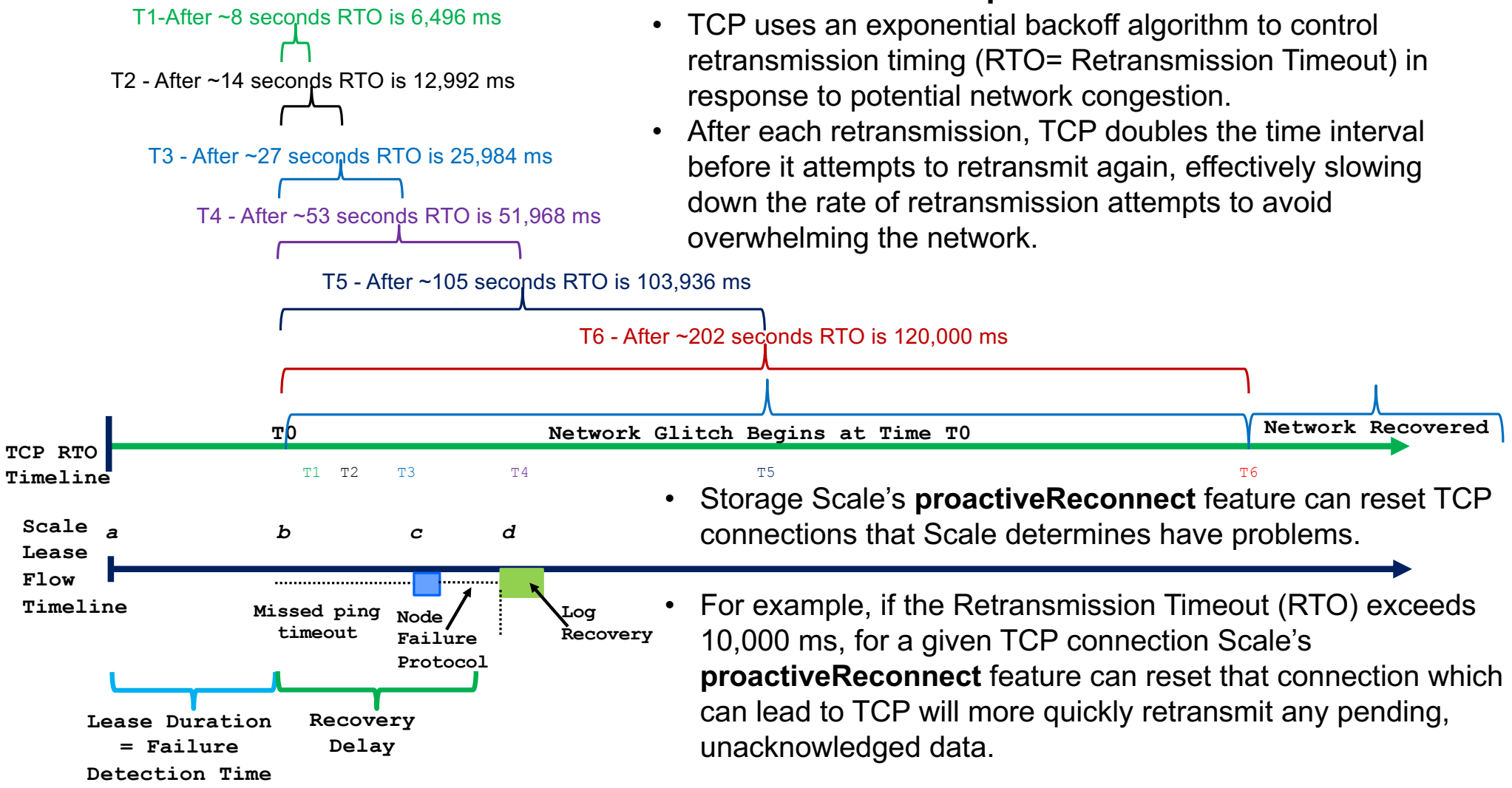
- Example of (2 on previous **Shorter Timeout Flows** slide) timeout establishing new TCP connection when we have a valid ARP entry for the node being connected to (fscc-sr650v2-4):

```
2024-10-05_17:21:11.732+0200: [N] Connecting to 10.200.42.204 fscc-sr650v2-4
<c0n21>:[0]
2024-10-05_17:22:15.241+0200: [N] Close connection to 10.200.42.204 fscc-
sr650v2-4 <c0n21>:[0] (Connection timed out)
2024-10-05_17:22:15.242+0200: [N] Retry connection to 10.200.42.204 fscc-
sr650v2-4 <c0n21>:[0]
2024-10-05_17:23:18.775+0200: [N] Close connection to 10.200.42.204 fscc-
sr650v2-4 <c0n21>:[0] (Connection timed out)
2024-10-05_17:23:30.200+0200: [N] Expel data is not collected on any node. It
was collected recently at 2024-10-05_16:54:34+0200.
2024-10-05_17:23:30.200+0200: [N] Request sent to 9.152.187.70 (fscc-sr650-49)
to expel 9.152.185.166 (fscc-sr650v2-4) from cluster GNRocks.fscc-ess3500-1-a
```

- Connectivity is down for 2 minutes and 7 seconds (from 17:21:11.732 – 17:23:18.775) before the client makes the decision to send a request for fscc-sr650v2-4 be expelled (the expel request is sent 12 seconds later).

- The total time taken from introducing the network connectivity failure to the time that the expel request is sent to the cluster manager is 2 minutes and 19 seconds (17:23:30 - 17:21:11).

Proactive Reconnect and RTO/Expel Timelines

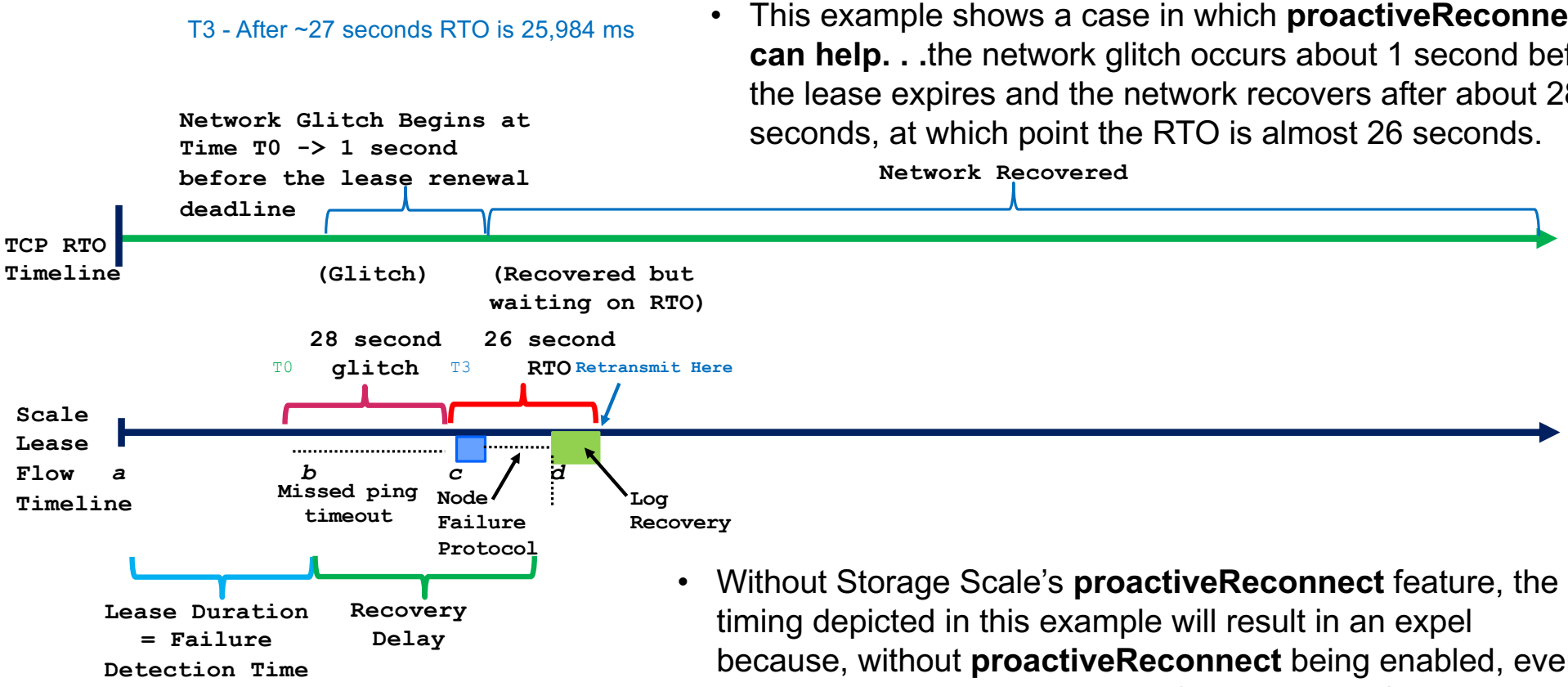


- TCP uses an exponential backoff algorithm to control retransmission timing (RTO= Retransmission Timeout) in response to potential network congestion.
- After each retransmission, TCP doubles the time interval before it attempts to retransmit again, effectively slowing down the rate of retransmission attempts to avoid overwhelming the network.

- Storage Scale's **proactiveReconnect** feature can reset TCP connections that Scale determines have problems.

- For example, if the Retransmission Timeout (RTO) exceeds 10,000 ms, for a given TCP connection Scale's **proactiveReconnect** feature can reset that connection which can lead to TCP will more quickly retransmit any pending, unacknowledged data.

Proactive Reconnect and RTO/Expel Timelines



- This example shows a case in which **proactiveReconnect can help. . .**the network glitch occurs about 1 second before the lease expires and the network recovers after about 28 seconds, at which point the RTO is almost 26 seconds.

- Without Storage Scale's **proactiveReconnect** feature, the timing depicted in this example will result in an expel because, without **proactiveReconnect** being enabled, even though the network recovers before the expel (c – Node Failure Protocol), any pending data for the connection won't be retransmitted until the 26 second RTO expires

Avoiding the “Shorter Timeout” Node Requested Expels

- As shown on the previous slides, short interruptions to network connectivity can lead to node requested expels if the network issues coincide with windows in which new network connections need to be established, and the long-term plan is to address this issue via design changes

- Until we have code changes to address the shorter timeouts in the connection establishment path, we can try to minimize how often nodes will establish new connections as follows:

  - Disable **proactiveReconnect** (e.g., mmchconfig proactiveReconnect=no -i) to prevent Storage Scale from reconnecting established connections (in 5.2.2.0, **proactiveReconnect** is disabled for all new clusters)

  - Disable idle sockets from being disconnected (“mmchconfig **idleSocketTimeout**=0 -i”, which becomes the default for this tuning option in 5.1.9.0)

  - Consider enabling the **allToAllConnection** option to force Storage Scale to establish connections to other nodes shortly after mmfsd comes up (this makes connection failures less likely to happen in a cluster that is in steady-state production, with all nodes already running jobs): mmchconfig **allToAllConnection**=all # means to connect to both local and remote cluster mmfsd’s (See: https://www.spectrumscaleug.org/wp-content/uploads/2020/09/004-spectrum-scale-performance-update.pdf)

Node Requested Expels - Which Node Gets Expelled?

- When one node requests the cluster manager to expel another node, the cluster manager will decide if it should expel the node making the request or the target node of the request.

- In order of priority, the following criteria are used to determine which node to expel by default:
  1. The cluster manager is always favored and will never be expelled.
  2. Starting with the 5.1.9 code, Storage Scale will keep track of recent expel requests and will prefer to expel a node that has recently been the target or source of multiple expel requests over expelling a node that has recently only been involved in a single expel request.
  3. Non-quorum nodes are preferred to be expelled over quorum nodes.
  4. Nodes that have joined the local (home) cluster from a remote cluster are preferred to be expelled over nodes that are part of the local (home) cluster.
  5. Non-manager nodes are preferred to be expelled over manager nodes.
  6. Nodes managing fewer file systems are preferred to be expelled over those managing more file systems.
  7. Non-NSD server nodes are preferred to be expelled over NSD server nodes.
  8. Nodes that have joined the local (home) cluster for a shorter time are preferred to be expelled over those that have been part of a cluster for longer.

- When evaluating if a node is a quorum, manager, NSD server, or file system manager node, the cluster manager will only consider the role of the node in the cluster in which the expel is to be done.

Node Requested Expels – The Expel History Feature

- In Storage Scale 5.1.9, when the cluster manager (CM) receives an expel request from a client node, before deciding which node it should expel, it will wait for at least 60 seconds by default to see if additional expel requests are received (this initial delay interval can be configured via the undocumented mmchconfig option **expelHistoryTimeout**, which defaults to 60 seconds).

- During the window in which the cluster manager waits for new expel requests to arrive, it tracks the number of expel requests that have been received during the **expelHistoryTimeout** window. (The CM defers its expel decision while waiting for new expel requests to arrive.)

- After the cluster manager waits for **expelHistoryTimeout** seconds for new expel requests to arrive, it then also waits for up to four smaller intervals (the length of the interval is controlled by the undocumented mmchconfig option **expelHistoryWaitInterval**, which defaults to 5 seconds) to see if the number of expel requests pending continues to change.

- If the threshold of four **expelHistoryWaitInterval** windows is reached, or the expel request counts remain static after any wait interval, the CM will proceed with the expel process provided that expelling the node chosen in this flow will not eliminate the last quorum node (this flow will not allow node quorum to be lost by expelling the last remaining node needed for quorum).

- The expel history feature can be disabled via the undocumented mmchconfig option: **mmchconfig disableExpelHistory=1 -i # no restart of mmfsd is required with the -i option**

Customizing the Handling of Node Requested Expels

- GPFS provides a callback mechanism to allow over-riding the default criteria for choosing which node will be expelled in the case of a node requested expel
- The callback can also be used to take action on a node requested expel, such as calling-out an admin, shutting down services, generating an alert, etc.
- Full details of mechanism are described in the script /**usr/lpp/mmfs/samples/expelnode.sample** – key points from that script are:

```
# When invoked, the script is provided information on both nodes involved: the
# node chosen for being expelled and the other node. The script is given the
# chance, via its exit value, to reverse the GPFS daemon's original decision.
#
# The script is invoked with the following parameters:
#              <Node name of GPFS's chosen node>
#              <Node name of the other node>
#              <Node spec for GPFS's chosen node>
#              <Node spec for the other node>
#              <"dry-run" indicator>
[…]
```

Customizing the Handling of Node Requested Expels

```
# The node spec contains colon-separated attributes which may be used
# in the decision of which node to expel. The following is the format of the
# node spec:
#
# [quorum:]local|remote-<cluster name>:[manager:]fsmgr_N:[nsdserver:]
# older|newer[:<others>]
#
# 'quorum'  : the node is a quorum node
# 'local'   : the node belongs to the local cluster
# 'remote-<cluster name>' : if the node belongs to a remote cluster, the
#                           remote cluster name is given
# 'manager' : the node is a manager node
# 'fsmgr_<number>' : indicates how many file systems the node is a manager for
# 'nsdserver' : the node is an NSD server
# 'older'   : the node joined the active cluster first (before the other node)
# 'newer'   : the node joined the active cluster later (after the other node)
#
# Examples:
# quorum:local:manager:fsmgr_2:nsdserver:older
# remote-my_cluster1:fsmgr_0:newer
[…]
# Exit code:   0: script agrees with the original decision: choose
#                 "originally picked node" as node to expel
#              1: script disagrees with the original decision: choose the
#                 other node (reverse choice)
#              Any other value: choose "originally picked node" as node to
#                 expel
```

Customizing the Handling of Node Requested Expels

```
# IMPORTANT: The GPFS daemon will wait until this script returns.
#            Invoking complex or long-running commands, or commands
#            that involve GPFS files, may cause unexpected and undesired
#            results, including loss of file system availability.
#
#
# To activate the code:
#
#   a) make a copy of this script, and then edit the copy and make the
#      appropriate changes
#
#   b) copy the script to /var/mmfs/etc/expelnode
#
#   c) ensure the script is executable (chmod +x /var/mmfs/etc/expelnode)
[…]

EXIT_VAL=0   # default: don't change decision

# Example of changing the default decision:
## # Change default decision if chosen node is an NSD server but the
## # other node is not
## if [[ $CHOSEN_IS_NSD_SERVER = 1 && $OTHER_IS_NSD_SERVER = 0 ]]
## then
##   echo "Chosen node is NSD server and other node is not" >> $LOG
##   EXIT_VAL=1
## fi
```

Details on Expels caused by the mmexpelnode Command

What is the mmexpelnode Command?

A system administrator may cause a node to be expelled by running the **mmexpelnode** command. Expels that are forced in this manner have the same impact as an expel that results from the more common lease timeout or RPC timeout node requested expel flows.
Running the **mmexpelnode** command can be useful when it's desired to remove a node from the Storage Scale cluster, without shutting the node down (for example, it may be desirable to temporarily expel a node that has a resource issue, such as over-committed memory).
Warning: do **not** use the -f (--is-fenced). option to **mmexpelnode**, unless you are sure that the node is down/fenced (as this option avoids the typical log recovery delay associated with an expel). It's generally recommended to avoid this --is-fenced option.

To expel a node in this manner, run **mmexpelnode** with the -N flag to specify the target node(s) for the expel, e.g.:

```
/usr/lpp/mmfs/bin/mmexpelnode -N c933f02x27
```

To bring a node back into the cluster, after it's been expelled in this manner, use the '-r' flag, e.g.:

```
/usr/lpp/mmfs/bin/mmexpelnode -r -N c933f02x27
```

Causes for Expels and Starting Points For Debugging

- Any issue that can lead to packet loss can prevent Storage Scale from establishing connections and sending the RPCs needed to avoid expels (namely ccMsgDiskLease and commHandleCheckMessages RPCs).
- Packet loss may be caused by faulty network adapters, switches, cables, interposers, etc. or the connections between these hardware components, and diagnosing such drops may requiring reseating or replacing components and working with the hardware vendor.
- Critical to this debug is the diagnosing of any packet loss by collecting counters and monitoring how packet drop counters on the nodes and switches are changing over time:
  - For instance, on the client and server nodes look for packet loss on the network interfaces used by Storage Scale, e.g., starting with ‘netstat -I’:

```
# netstat -I=ens1np0
Kernel Interface table
Iface             MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
ens1np0          9000    38047      0    120 0         35580      0      0      0 BMRU
```

  Look at the breakdown of adapter errors and drops by running ‘ethtool -S’ for the interface, e.g.:

```
# ethtool -S ens1np0
```

  You can also look for drops that are non-zero in ethtool -S output, e.g.:

```
# ethtool -S ens1np0  | egrep -i "drop" | grep -v ": 0"
```

Causes for Expels and Starting Points For Debugging

- Packet loss may also be addressed by the tuning of buffers and queues on adapters and switches
- Ring buffer drops are one of the most common issues seen in network related field issues – you can check the ring buffer sizes in 'ethtool -g' to see if they can be increased:

```
# ethtool -g ens1np0
Ring parameters for ens1np0:
Pre-set maximums:
RX:             8192
RX Mini:        0
RX Jumbo:       0
TX:             8192
Current hardware settings:
RX:             1024
RX Mini:        0
RX Jumbo:       0
TX:             1024
```

Example of increasing RX and TX ring buffers to the maximum value the adapter supports:

```
# ethtool -G ens1np0 rx 8192 tx 8192
```

Causes for Expels and Recommendations

- On client and server nodes you may find packet loss in the statistics for interrupt related queues and New Application Programming Interface (NAPI) processing - try looking for such packet loss by checking **/proc/net/softnet_stat,** e.g.:

```
# cat /proc/net/softnet_stat
001c1216 000000b0 00000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000
02fdffa5 000000aa 00000032 00000000 00000000 00000000 00000000 00000000 00000000 00000000 […]
         |||||||| ||||||||
         |||||||| sd->time_squeeze (these drops may be addressed by increasing net.core.netdev_budget
         ||||||||                                                and/or net.core.netdev_budget_usecs)
         ||||||||
         sd->dropped (these drops may be addressed by increasing net.core.netdev_max_backlog)
```

Below are suggested starting point tunings to address drops seen here:

1. **sd->dropped** (second column above), shows cases of netdev_max_backlog being exceeded.
   `sysctl -w net.core.netdev_max_backlog=10000`

2. **sd->time_squeeze** (third column above), shows cases above in which netdev_budget and/or net.core.netdev_budget_usecs has been exceeded). A starting point tuning in such cases is:
   `sysctl -w net.core.netdev_budget=600`
   `sysctl -w net.core.netdev_budget_usecs=4000`

Causes for Expels and Recommendations

- After eliminating packet loss related to hardware issues and tuning/configuration settings, another issue to check is if there are resource issues on the nodes that may impact performance.
- If the nodes are short on memory, this can impact performance – check for out-of-memory (OOM) errors that may be logged in /var/log/messages and monitor memory use while workloads are running.
- Also look for CPU contention that may impact performance – Storage Scale will log ‘critical thread’ monitoring’ issues when threads are blocked for excessive amounts of time and Linux will log 'stuck CPU’ warnings’ when threads are blocked for more than 120 seconds.
- On quorum nodes, local disk performance is important as slow I/O times to writes to the file /var/mmfs/gen/LastLeaseRequestSent may result in the quorum nodes being expelled.
- As previously noted, in some cases, the mmchconfig option **proactiveReconnect** may cause issues in that, if the network experiences stability issues, **proactiveReconnect** can end up accelerating the timeline for expels and exposing nodes to shorter expel timeouts that occur during the establishment of TCP/IP connections, so consider minimizing the establishment of new connections as recommended on a previous slide
  - The up-side of **proactiveReconnect** is that it can (1) avoid timing holes as described on the **Proactive Reconnect and RTO/Expel Timelines** slide, and (2) work-around the ’Socket Lockup’ exposure seen on older Linux levels in which TCP connections could hang as a result of Linux kernel issues (e.g. 2.6.x kernel exposures related to how TIME_WAIT and FIN_WAIT were handled during heavy retransmission)
- Also consider implementing ARP tuning, as broadcast operations can introduce more stress on the network. . . in general larger systems should consider some form of ARP tuning – traditional guidance has been to implement such tuning on systems larger than 1000 nodes, but we’ve also seen good success resolving network issues by implementing ARP tuning on much smaller clusters (even systems as large as 128 nodes)

Causes for Expels and Recommendations

- Another concern to consider is the potential for overloading the network, leading to drops on switches and adapters, particularly when very large TCP/IP windows are used and/or many to few communication patterns occur.

- Some techniques that can address congestion and overloading of a network fabric:

    1. Enabling Explicit Congestion Notification (ECN), if all switches and adapters support ECN, allowing senders to be notified of congestion without dropping packets. (This option is strongly recommended.)

    2. Using smaller TCP window sizes, which reduces the total data that can be in flight for any TCP connection.

    3. Enabling more modern congestion control algorithms, such as Google's BBR (Bottleneck Bandwidth and Round-trip propagation time, available in 4.9 kernel levels by setting: "**sysctl -w net.ipv4.tcp_congestion_control=bbr**").

    4. Enabling an alternative congestion control solution such as L4S (Low Latency Low Loss Scalable Throughput, which requires a recent kernel level, enabling ECN, and a compatible congestion control algorithm, e.g., "**sysctl -w net.ipv4.tcp_congestion_control=dctcp**").

    5. Consider Active Queue Management (AQM) scheme options, such as Proportional Integral controller Enhanced (PIE), that can be enabled to better manage queue lengths in routers and switches, allowing more proactive congestion notification and dropping of packets under congestion conditions.

ARP Tuning Options to Consider to Improve Network Stability

gc_thresh1: INTEGER Minimum number of entries to keep. Garbage collector will not purge entries if there are fewer than this number. Default: 128

gc_thresh2: INTEGER Threshold when garbage collector becomes more aggressive about purging entries. Entries older than 5 seconds will be cleared when over this number. Default: 512

gc_thresh3: INTEGER Maximum number of non-PERMANENT neighbor entries allowed. Increase this when using large numbers of interfaces and when communicating with large numbers of directly-connected peers. Default: 1024

gc_stale_time: Determines how often to check for stale neighbor entries. When a neighbor entry is considered stale, it is resolved again before sending data to it. Default: 60 (seconds)

gc_interval: How frequently the garbage collector for neighbor entries should attempt to run. Default: 30 (seconds)

base_reachable_time_ms: Once a neighbor has been found, the entry is considered to be valid for at least a random value between *base_reachable_time*/2 and 3**base_reachable_time*/2. An entry's validity will be extended if it receives positive feedback from higher level protocols. Default: 30000 (ms) (since since Linux 2.6.12 - *base_reachable_time*, which was defined in seconds, is obsolete)

mcast_solicit: The maximum number of attempts to resolve an address by multicast/broadcast before marking the entry as unreachable. Default: 3

Configuration Options to Avoid Network Related Expels

Here are the ARP tuning parameters that we found resolved stability issues on some of our largest deployments (the Coral systems):

```
net.ipv4.neigh.ib0.gc_thresh1=30000
net.ipv4.neigh.ib0.gc_thresh2=32000
net.ipv4.neigh.ib0.gc_thresh3=32768
net.ipv4.neigh.ib0.gc_stale_time=2000000
net.ipv4.neigh.ib0.gc_interval=2000000
net.ipv4.neigh.ib0.base_reachable_time_ms=120000
net.ipv4.neigh.ib0.mcast_solicit=18
```

Other large systems in the field have seen improvements applying slightly modified settings:

```
net.ipv4.neigh.ib0.gc_thresh1=30000
net.ipv4.neigh.ib0.gc_thresh2=32000
net.ipv4.neigh.ib0.gc_thresh3=32768
net.ipv4.neigh.ib0.gc_stale_time=864000
net.ipv4.neigh.ib0.gc_interval=864000
net.ipv4.neigh.ib0.mcast_solicit=9
```

net.ipv4.neigh.default.base_reachable_time_ms has not been tuned in these modified tunings because we to allow ARP entries to grow stale in a timely manner so we can detect if an entry is no longer valid (e.g. due to a LID change)

Example of Disk Lease Timeout Expel As Reported on the Cluster Manager

- One of the best ways to determine if a network layer problem is root cause for an expel is to look at the low-level socket details dumped in the ‘internaldump’ log data (mmfs dump all) saved as part of automatic data collection on Linux GPFS nodes (not on AIX or Windows)
- Some of this socket level data will also be printed to the logs in later code levels, when Storage Scale detects an issue with a socket that has outstanding messages associated with it.
- The logs also show results of pings that are sent from the cluster manager to the failing node, before the expel, but we can’t always trust the correct underlying interface will be pinged in the case of an 802.3ad xmit layer 3+4 bonded configuration, and in the case that the subnets configuration option is used (this subnets issue is fixed in Storage Scale 5.0.2). If the pings don’t go through (if “Replies received: 0”) this may indicate a node failure, a reboot, or a network connectivity issue.

```
2023-04-01_18:45:54.994-0400: [E] The TCP connection to IP address 10.3.2.3 c933f02x03
<c0n0> (socket 67) state is unexpected: ca_state=4 unacked=1 rto=4000000

2023-04-01_18:45:54.994-0400: [I] tscCheckTcpConn: Sending debug data collection request
to node 10.3.2.3 c933f02x03

2023-04-01_18:46:00.420-0400: [E] Node 10.3.2.7 (c933f02x07) is being expelled because
of an expired lease. Pings sent: 60. Replies received: 60.

2023-04-01_18:46:04.302-0400: [E] Node 10.3.2.5 (c933f02x05) is being expelled because
of an expired lease. Pings sent: 60. Replies received: 60.
```

Debugging Expels From the mmfs Logs and Dumps

- One of the best places to start debugging an expel is to look at the mmfs.log and any “internaldump” data collected from the time of the expel. (Note this approach is only valid on Linux.) TCP/IP socket level statistics provide a view of how the network is working at a low level (TCP acknowledgements are turned around by the kernel, typically in interrupt context). Using this data, look for the socket statistics in the logs and “dump tscomm” section of the data, and focus on values such as:

– rto: starts out at around 204000 microseconds on a low latency network, and, if it goes higher, this may indicate that TCP is retransmitting and backing off the retransmission time

– retransmits, backoff, lost: if any of these are non-zero, then the socket is retransmitting

– ca_state: if this is anything other than ‘open’, there has been either packet loss or a out of order packet delivery issue

- Here’s an example of a socket connection that is showing a clear problem:

```
192.168.1.13/1
     state 1 established snd_wscale 7 rcv_wscale 9 rto 120000000 ato 40000
     retransmits 0 probes 0 backoff 11 options: WSCALE
     rtt 3711 rttvar 123 snd_ssthresh 2 snd_cwnd 2 unacked 122
     snd_mss 1460 rcv_mss 973 pmtu 2044 advmss 2004 rcv_ssthresh 31074
     sacked 0 lost 120 retrans 0 fackets 0 reordering 3 ca_state 'loss'
```

- As per the previous chart, some of this socket level data will also be printed to the logs in later code levels, when GPFS detects an issue with a socket that has outstanding messages associated with it.

Debugging Expels From Logs

If there's no level socket data available (e.g. on AIX, Windows, or older Linux code levels), lease timeout related expels may offer some clue on the state of the network by looking at the cluster manager logs and checking if the pings from the cluster manager (done before an expel in the case of a lease timeout) are going through. If all the packets go through, as in this example, it may mean there's a resource, tuning, or code issue causing the expel, or it might still be a network layer problem. Here's an example:

```
2022-04-01_18:46:00.420-0400: [E] Node 10.3.2.7 (c933f02x07) is being expelled
because of an expired lease. Pings sent: 60. Replies received: 60.
```

In such cases, where's there's no evidence of a network problem, traces may be needed, but other things to check include:
1) /var/log/messages for evidence of resource issues (e.g. memory being over-committed)

2) Verify if bonding is in use (GPFS may ping the wrong underlying physical interface in the case of 802.3ad bonding with xmit mode 3+4).

If "Replies received" had been 0, this might point more at a network connectivity problem. In such cases, check gpfs.snap data for:
1) evidence of packet loss (e.g. high packet loss reported in 'ifconfig' data – we commonly see that a high "RX dropped" count may be an indication that additional receive ring buffers need to be configured
2) high retransmission levels reported in the netstat tcp statistics (also check /var/log/messages for signs of switch flow control– or pause frames– which may lead to retransmission)

Token Handling in Storage Scale

Why does Storage Scale need Tokens/Locks?

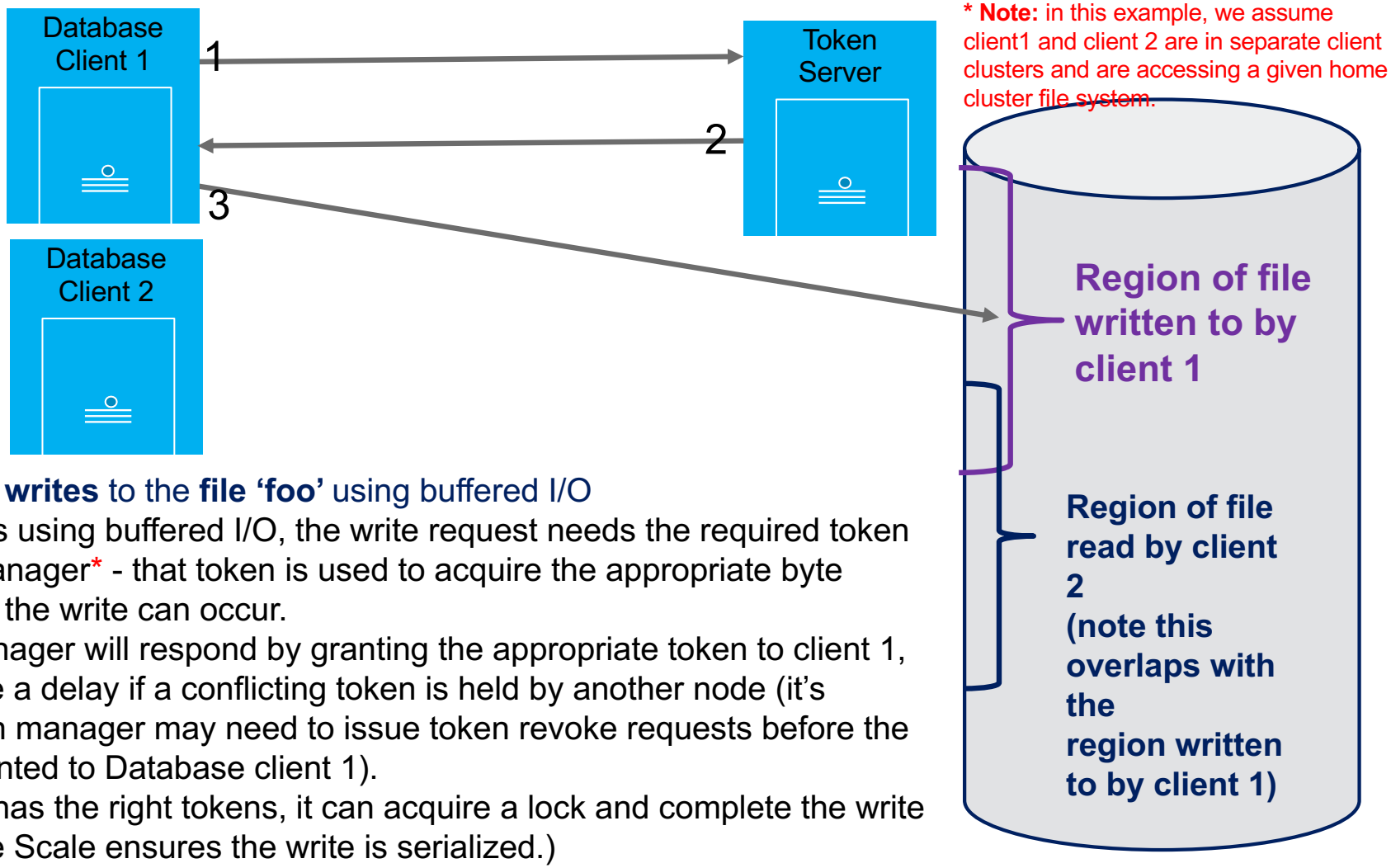
To ensure that multiple tasks running across different nodes can safely operate on shared data, tokens and locks are used to serialize access to the file system. To read and write data to the filesystem (and to do certain operations, such as become metanode) the appropriate token(s) and lock(s) must be held.

What happens if tokens/locks are not granted in a timely manner?

As tokens and locks are needed for most file system operations, if a thread can't get the required tokens/locks, it will effectively appear to be hung. If one node gets the token for an important resource (say a write token for the root directory of a Storage Scale file system) this can lead to a cluster-wide hang if the node fails to release that token.

Token Handling in Storage Scale

Buffered I/O Example (1/2)



* **Note:** in this example, we assume client1 and client 2 are in separate client clusters and are accessing a given home cluster file system.

Database client 1 **writes** to the **file 'foo'** using buffered I/O

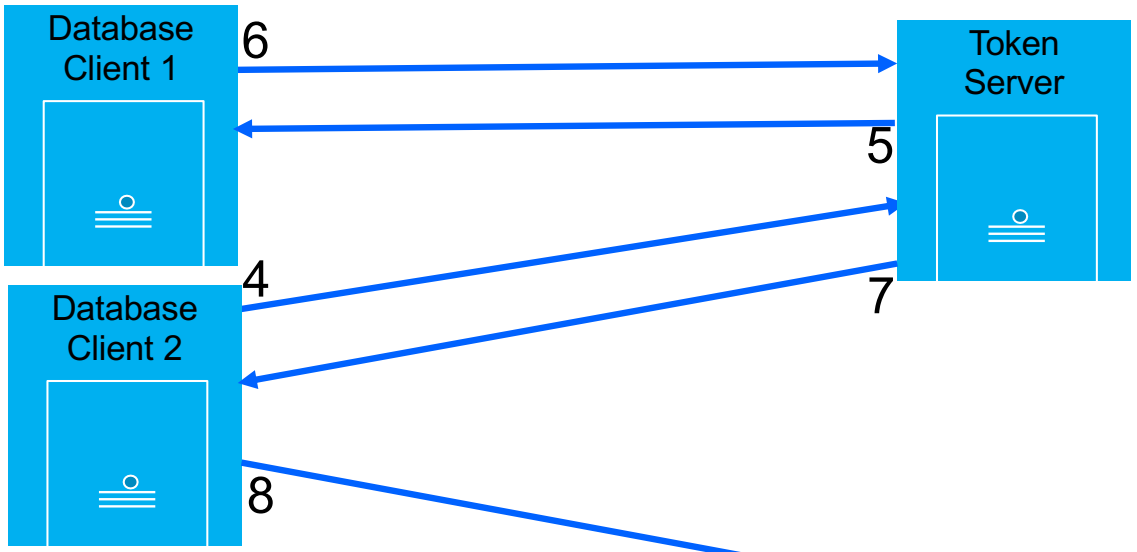
(1) If the access is using buffered I/O, the write request needs the required token from the token manager* - that token is used to acquire the appropriate byte range lock before the write can occur.

(2) The token manager will respond by granting the appropriate token to client 1, but there could be a delay if a conflicting token is held by another node (it's possible the token manager may need to issue token revoke requests before the token may be granted to Database client 1).

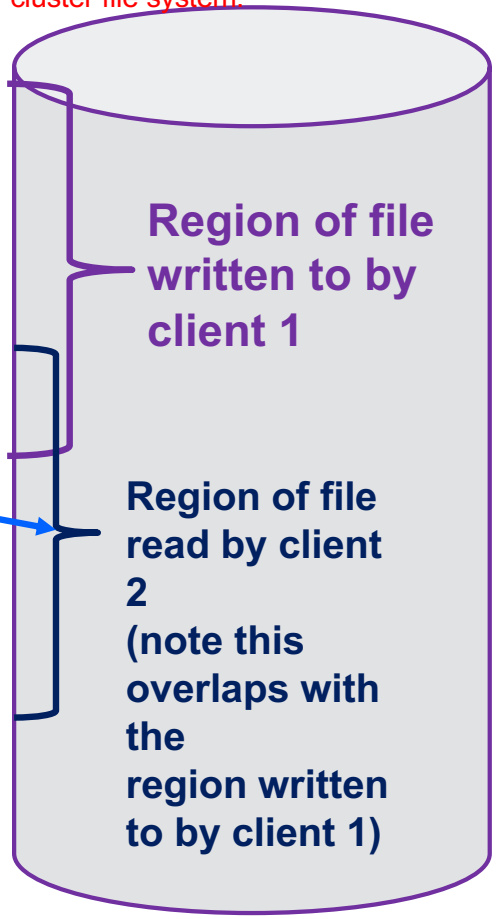
(3) Once client 1 has the right tokens, it can acquire a lock and complete the write request. (Storage Scale ensures the write is serialized.)

Token Handling in Storage Scale

Buffered I/O Example continued (2/2)



* **Note:** in this example, we assume client1 and client 2 are in separate client clusters and are accessing a given home cluster file system.



Database client 2 **reads** the **file 'foo'** in a region overlapping prev. write:

(4) Client 2's read request needs the required token and lock. It's more likely there will be a conflict when using buffered I/O; here client 2 asks the token server for a read token. *

(5) The token manager will try to grant requested token to node 2, but in this example, it must first revoke a portion of the conflicting token range that client 1 acquired in step 2.

(6) Client 1 completes its write, giving up the token requested by the token manager.

(7) The token manager responds to database client 2 with the appropriate token, allowing client 2's read request to proceed.

(8) Once client 2 has the right tokens, it can acquire a lock and complete the read request. (Storage Scale ensures the read is serialized.)

Resiliency Improvement: Health Monitoring of pending RPCs and automatic expel

Starting with 5.1.6 mmhealth has the functionality to detect pending RPCs for token revokes

- When then RPC age is passing a given warning threshold it will raise a rpc_waiters event to inform the user about the misbehaving node
- Optionally a expel threshold can be configured when mmhealth should start to expel the node automatically

There are two mmchconfig options to control the feature:

- mmhealthPendingRPCWarningThreshold (default = 180 seconds)
- mmhealthPendingRPCExpelThreshold (default = 0 means disabled)

Detection of pending RPCs - rpc_waiters warning event

The mmhealth GPFS monitor runs "mmdiag --network -Y" every 2nd monitoring cycle (default interval for GPFS monitoring is 30s, means that rpc monitoring is done every 60s).

mmdiag:pendingMessages:HEADER:version:reserved:reserved:messageId:serviceNumber:version:typeNumber:typeString:ageInSeconds:sentToNumberOfNodes:waitingForReplyFromNumberOfNodes:this:xhold:resending:cl:cbFn:sentByThreadId:sentByThreadName:sentBySenderAddress:replies:destination:node:status:error:replyLength:RDMAConnectionIndex:TCPConnectionIndex:isResending:

mmdiag:pendingMessages:0:1:::1881:::18:tmMsgRevoke:305:1:1:::::::2427972:SharedHashTabFetchHandlerThread::10.0.100.120:<c0n1>:pending:0:0:0

mmdiag:pendingMessages:0:1:::1881:::18:tmMsgRevoke:108:1:1:::::::2427972:SharedHashTabFetchHandlerThread::10.0.100.88:<c1n1>:pending:0:0:0::

mmdiag:pendingMessages:0:1:::1881:::18:tmMsgRevoke:508:1:1:::::::2427972:SharedHashTabFetchHandlerThread::10.0.100.74:<c1n0>:pending:0:0:0::

mmdiag:pendingMessages:0:1:::1881:::18:tmMsgBRRevoke:400:1:1:::::::2427972:SharedHashTabFetchHandlerThread::10.0.100.70:<c1n3>:pending:0:0:0::

mmdiag:pendingMessages:0:1:::1895:::3:commMsgCheckMessages:4:1:1:::::::2423979:EEWatchDogThread::10.0.100.120:<c0n2>:pending:0:0:1::

.....

If the mmdiag output contains some pending RPCs with an age larger than the mmhealthPendingRPCWarningThreshold , it will raise a ***rpc_waiters*** event which will change the GPFS component state to DEGRADED.

- With 5.1.6.0 tmMsgRevoke RPCs are detected only
- With 5.1.6.1 tmMsgRevoke and tmMsgBRRevoke RPCs are detected

Multiple pending RPCs will be grouped in a single rpc_waiters event. If mmhealthPendingRPCWarningThreshold is set to 0, no event will be created.

Automatic expel of unresponsive node

- Enable automatic expel by setting a threshold in seconds using **mmchconfig**. The expel threshold should be higher than the warning threshold, e.g.:

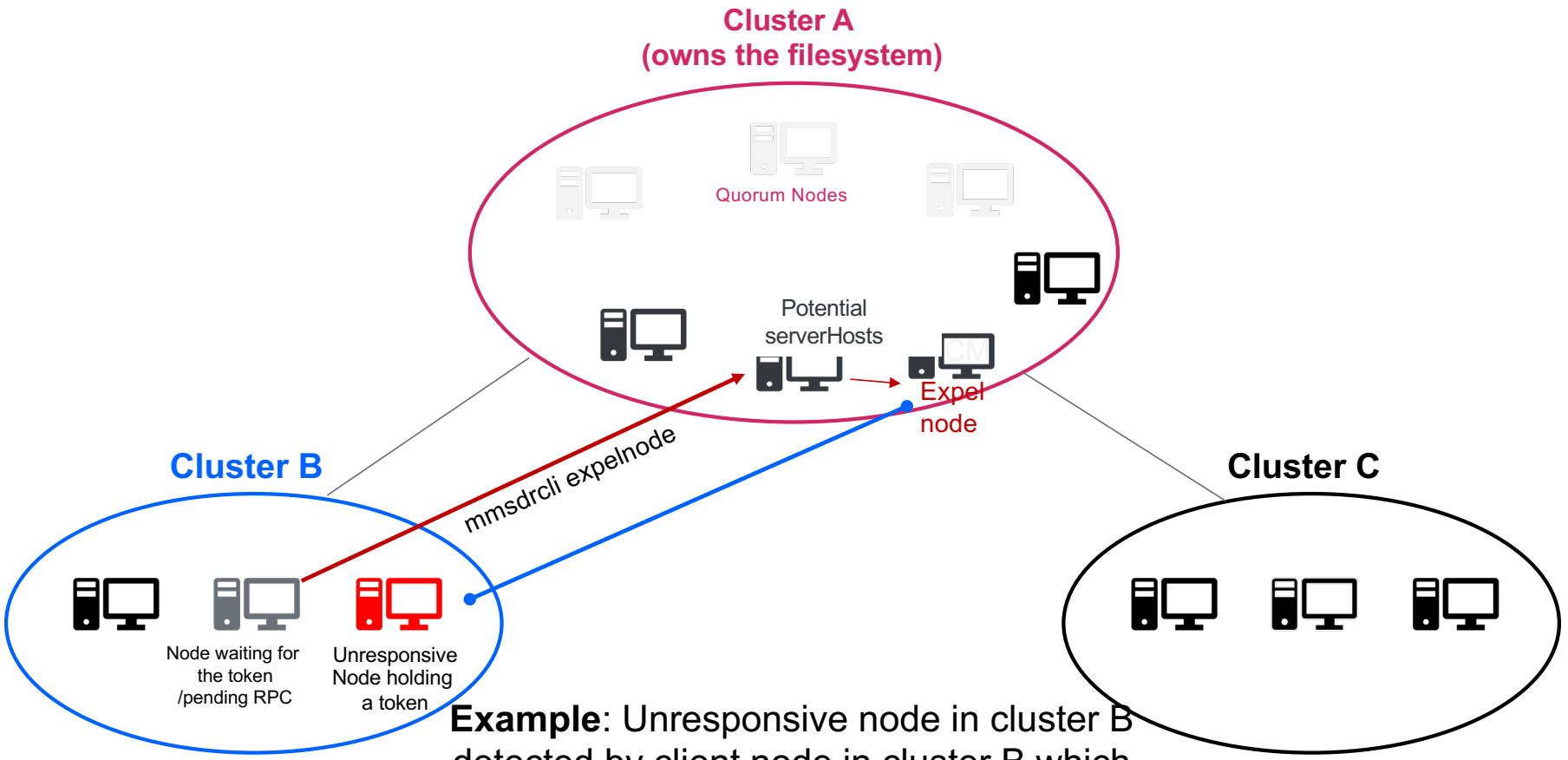
```
mmchconfig mmhealthPendingRPCExpelThreshold=300
```

- if **mmhealthPendingRPCExpelThreshold** is set to 0 (the default), no expel will be done

- If the mmdiag output contains some pending RPCs with an age larger than the mmhealthPendingRPCExpelThreshold, it will:
  - get the node to expel from the pending rpc output (victim)
  - get the node (serverHost) where we send the expel request to from the connected node list, but matching the cluster id (e.g. <c1n3>)
  - run *mmsdrcli expelnode* command with the victim and serverHost as parameters
  - raise a ***rpc_waiters_expel*** warning event (non-state-change) for the particular node expel

By default, the node will be expelled persistently-- once a node is expelled, the node will not rejoin the cluster unless "`mmexpelnode -r -N <node>`" is run.

Storage Scale – Multicluster Expel



Example: Unresponsive node in cluster B detected by client node in cluster B which will trigger expell from cluster A

Health Monitoring of pending RPCs and automatic expel

Storage Scale 5.1.6.1

- Added mmhealth support for pending RPC detection and automatic expel
  - All nodes in the cluster (s) must run >= 5.1.6.0
  - If there are nodes in the cluster with an older level:
    - Pending rpcs on those nodes will not be detected and expel requests send to those nodes fail silently
- tmMsgRevoke RPCs are monitored only

Storage Scale 5.1.7.0 improvements

- Detection of tmMsgBRRevoke RPCs added
- Retry sending expel requests to other nodes in case of failure

Storage Scale 5.1.9.0

- Prior to 5.1.9.0 some RPCs did not update the age correctly (age of 0s) and would not be detected

Storage Scale 5.1.9.4

- Fixed problem of mmsdrcli not working correctly when **sdrNotifyAuthEnabled** is not enabled (the workaround, also recommended for security reasons, is to set sdrNotifyAuthEnabled across all the clusters)

If you are running an older version of Scale and can not upgrade -> request an eFix

A Disk Lease Related Process - DMS Timeouts

- Nodes in a Storage Scale cluster have a requirement for all I/Os to complete within a threshold of time after disk lease expiration.

- To preserve the integrity of the file system, Storage Scale must ensure there are no delayed/unexpected I/Os issued to the disks while log recovery is happening.

- If there are pending I/Os which have not completed after a server node's lease expires, this can trigger Storage Scale to panic (crash) the impacted server node. This panic is to prevent file system corruption.

- When a server node renews its lease, it will schedule a timer event **leaseDMSTimeout** seconds after the point at which a lease timeout will occur.

- When the timer event occurs, if there are still pending local I/Os to the Storage Scale cluster file system storage, and the node in question has still not renewed its lease, the Storage Scale kernel extension will initiate a panic event that crashes the node.

- In this context, 'local I/Os' are defined as I/Os to the storage that is managed/owned by the server, but for which the server no longer has a valid disk lease for this storage.

- This panic mechanism is referred to as the Dead Man Switch (DMS) timeout.

Another Disk Lease Related Process - Cluster Manager Takeover

- The cluster manager can never be expelled from a GPFS cluster, but the cluster manager may decide not to renew its own lease and the quorum nodes can elect a new cluster manager. So the net effect of cluster manager takeover is roughly the same effect as an expel, but it should not be called an expel.

- If a majority of the quorum nodes do not renew their lease back to the cluster manager, then the cluster manager will not renew its own lease. This failure scenario will result in the quorum nodes not being able to renew their leases. After they timeout trying to contact the cluster manager, the quorum nodes will try to talk to each other, and this process may potentially result in the election of one of the candidate manager nodes as a new cluster manager (a PAXOS algorithm is run in this case).

- Note that the quorum nodes use a shorter disk lease so they can efficiently validate the existence of a ‘good’ cluster manager and, if needed, initiate the process of cluster manager takeover. This shorter lease interval employed on the quorum nodes allows for a cluster manager failure to be detected in the same time frame as regular client node failures.

Backup

Details on Storage Scale

Lease configuration parameters

The “Real” Lease Related Variables Used by mmfsd and How to Display Them

```
# mmfsadm saferdump config | egrep “failureDetectionTime|recoveryWait”
```

<will print all of the following on 4 lines of output>

```
failureDetectionTime 35

recoveryWait 35

dmsTimeout 23

leaseDuration 35.0/23.3

renewalInterval 30.0/11.7

renewalTimeout 5

fuzz 3.00/1.17

missedPingTimeout 15x2.0=30.0

totalPingTimeout 60x2.0=120.0
```

Note: Avoid running ‘mmfsadm [safer]dump’ in production, as there is a timing hole exposure that may cause a daemon failure (even with ‘saferdump’). A close match is: `mmdiag --config | egrep \`

`"failureDetectionTime|leaseDMSTimeout|leaseDuration|leaseRecoveryWait|PingTimeout"`

Lease Configuration Parameters As they Apply to Disk Leasing Described (1/6)

failureDetectionTime: The default value used in the Storage Scale (mmfsd) daemon is 35 seconds (mmdiag will report the default as '-1').

- This variable controls how long it takes GPFS to react to a node failure (how quickly the cluster manager may detect a failed node, expel it, and recover the cluster). When disk fencing (Persistent Reserve) is enabled, Storage Scale is capable of initiating faster recovery times, but Persistent Reserve is not a typical configuration so we will instead describe the more common 'disk leasing' mechanism used to manage disk leases.
- When disk leasing is enabled, **failureDetectionTime** can be changed via mmchconfig to set the length of the lease (**leaseDuration**, can be also be set directly via mmchconfig but large clusters have typically tuned **failureDetectionTime** instead, allowing **leaseDuration** to be derived from **failureDetectionTime**, so only this approach will be considered here).

leaseDMSTimeout: The default value is 23 seconds (mmdiag will report the default as -1 and mmlsconfig will report the default as "2/3 leaseDuration").

- When a Storage Scale server node acquires a disk lease, it will schedule a timeout interrupt to occur **leaseDMSTimeout** seconds after the new lease is set to expire. When that timer pops, if the lease associated with the timer had not been renewed, then Storage Scale will cause a kernel panic if there is any local I/O pending to a cluster file system for which the server's lease has expired.

Lease Configuration Parameters As they Apply to Disk Leasing (continued, 2/6)

leaseDuration: The default value for the duration of a Storage Scale disk lease is 35 seconds (mmdiag will report the default as -1).

- It's strongly recommended that this option not be tuned (use **failureDetectionTime** instead).

- The 'mmfsadm saferdump' command (which should be avoided on production systems) displays two values for this configuration option, first the length of non-quorum node leases, and second the length of quorum node leases. Quorum nodes need to renew their disk leases more frequently (2/3rd of the non quorum node lease values) because the quorum nodes' renewal mechanism also verifies the existence of a valid cluster manager, and this shorter lease facilitates the quorum nodes' election of a replacement cluster manager when required.

Lease Configuration Parameters As they Apply to Disk Leasing (continued, 3/6)

leaseRecoveryWait: The default value is 35 seconds.

- To maintain the integrity of the file system, it's recommended this value not be lowered below the default of 35.
- The **leaseRecoveryWait** configuration option is used in two ways by the Storage Scale code when disk leasing is enabled (as opposed to Persistent Reserve which is not covered here).
- First, the default value of **leaseDMSTimeout** is calculated to be two thirds of **leaseRecoveryWait**.
- Second, the value of **leaseRecoveryWait** is used by the GPFS daemon to determine how long to wait before running recovery protocols (also called log recovery) after an expel event occurs. As recovery must always wait at least **leaseRecoveryWait** seconds after a node is expelled, Storage Scale ensures that **missedPingTimeout** (discussed later) can never be smaller than **leaseRecoveryWait**.

Lease Configuration Parameters As they Apply to Disk Leasing (continued, 4/6)

renewalInterval: The default value is 30 seconds; **renewalInterval** is derived by subtracting **renewalTimeout** (which is by default 5 seconds) from **leaseDuration** (which defaults to 35 seconds).

- This tunable defines the maximum number of seconds a node will wait, after it obtains a valid lease, before it attempts to renew that lease. On each new lease renewal, a random 'fuzz factor' (described below under the **fuzz** configuration option description) is subtracted from this **renewalInterval** value, in an attempt to stagger lease renewal times across the nodes.

renewalTimeout: The default value is 5 seconds, and always set to 5, unless the value of **leaseDuration** is set to less than 10 seconds. In the case that **leaseDuration** is set to less than 10 seconds, renewalTimeout is set to one half of **leaseDuration**.

- This value is used in calculating the interval in which the lease is renewed as described above.

fuzz: The default value set to 3 seconds (derived from 1/10th of the **renewalInterval**, which defaults to 30).

- As per the **renewalInterval** description, after each new lease renewal, the number of seconds a node will wait before initiating a new lease renewal is calculated to be **renewalInterval** seconds, minus a random 'fuzz factor', which is defined to be a random number of seconds between zero and this **fuzz** configuration option. So, with the default tuning, a node will attempt to renew its lease on an interval between (**renewalInterval** – **fuzz** = 30-3) 27 and (**renewalInterval**) 30 seconds.

Lease Configuration Parameters As they Apply to Disk Leasing (continued, 5/6)

maxMissedPingTimeout: The default value is 60 seconds.

- When a node fails to renew its lease within the allotted **leaseDuration** window, the cluster manager will begin to ping (send ICMP datagrams to) the node (this is sometimes called the ping timeout window). One of the goals of these pings is to determine if a node still has a good connection back to the cluster manager, and therefore may be given more time to renew its lease before an expel. The **maxMissedPingTimeout** is used to define a maximum value for Storage Scale (mmfsd) daemon's **missedPingTimeout** variable, which defines the length of time the cluster manager will ping before expelling a node, if the pings fail. The value of the internal **missedPingTimeout** variable is calculated as per the details described under **minMissedPingTimeout**

minMissedPingTimeout: The default value of the tunable is 3 seconds.

- The value of **minMissedPingTimeout** defines a minimum value for GPFS daemon's **missedPingTimeout** variable. The actual value of the internal **missedPingTimeout** variable is taken to be the maximum of **minMissedPingTimeout** and **leaseRecoveryWait** (but it's limited to a maximum value according to **maxMissedPingTimeout).** GPFS uses the daemon **missedPingTimeout** value to define the amount of time the cluster manager will wait before expelling a node with an overdue lease in the case that the node does not respond to the cluster manager's pings. Note that, with the default tuning, missedPingTimeout will be defined by **leaseRecoveryWait .**

Lease Configuration Parameters As they Apply to Disk Leasing (continued, 6/6)

totalPingTimeout, The default value is 120 seconds.

- This tunable is used by the GPFS daemon to define the total amount of time the cluster manager will wait before expelling a node with an overdue lease, in the case that the node responds to the pings sent my the cluster manager. As this tunable is already set to 120 seconds by default, it is generally left at the default (IBM has little experience tuning this value on large systems).

- The intent of having separate values for **totalPingTimeout** and **missedPingTimeout** (defined according to the values of **minMissedPingTimeout, maxMissedPingTimeout,** and **leaseRecoveryWait)** is to allow for tuning such that nodes that still appear to have good network connectivity back to the cluster manager are given more tolerance in terms of slow lease responses back to the cluster manager.

Thank You!