Continuously Available SMB
Observations and Lessons Learned

David Kruse
Mathew George
Microsoft
A Very Good Place to Start


- **Key Improvements**
  - Remove limitations within the existing protocol
  - Relieve the burden of backward compatibility
  - Protocol is designed with expandability in mind, allowing for cleaner and faster feature improvements in subsequent releases.
  - Better security options
  - Eventually would allow up-level shops to disable down-level compatibility to lower security attack surface.
Looking Forward


- **Summary**
  - SMB 2.0 will support a method for asking for a “resume key” that can be used to bind to an open on a different connection. This methodology will be extended to allow binding to the original handle even after the connection has gone away and been re-established.

- **Key Improvements**
  - Much of the work needed for persistent handles.
  - Server-side consistency guarantees across intermittent disconnects
The Road Map

Durability

Resiliency

Persistence (CA)
Durability Overview

- Rule #1 – Don’t break anything
  - Concerns of introducing new sharing violation errors and obstructing existing applications were dominant
  - Design focused on minimizing this risk (through reliance on oplocks)
Durability – Design

- Base principals for Continuous Availability (CA) laid out
  - Handle-based recovery (not session based)
  - Operation-level replay includes client responsibilities
Durability – Lessons Learned

- Concerns over app failures with sharing violation were unfounded
- In hindsight, a more aggressive stance on handle reservations could have been targeted
- Positive feedback in wireless file copies, cell modem scenarios, etc.
Resiliency Overview

- Goal: Increase guarantees to application to engage enterprise applications
- Design required application to request resiliency and provide timeout for handles
- Added lock replay logic for only-once semantics
- Removed reliance on oplocks/leases
Resiliency – Lessons Learned

- Requiring app changes greatly slows adoption and deployment.
  - App writers would prefer it just work

- Final solution should be simpler to administer, but allow applications who wish to be aware of CA to integrate
Continuous Availability (CA)

- Added support for create replays, object epochs, application instances, CA for directories.
- Changes to core protocol are incremental upon durability/resiliency.
- Replay logic for most operations solves both server failure and multichannel failover.
- Much more work invested in end-to-end solution including peripheral protocols (VSS, Witness, etc.)
Coming Full Circle

- Durability V2 provides:
  - Create replay
  - Lock sequencing, only-once execution
  - Object version/epoch support

Even in non-clustered, non-CA scenarios!

- What to do with Resiliency?
Diving Deeper into Replay and Recovery

Mathew George
A Recap from SDC 2011

- Introduced the SMB 2.2 (now 3.0) protocol family
  - Multichannel (MC)
    - Replay detection and sequencing
    - Session “binding” to multiple channels.
    - Framework for alternate transports (RDMA)
  - Continuously Available Shares (CA)
    - Core protocol enhancements
    - State preservation and restoration, semantics.
  - Auxiliary protocols (Witness, Remote VSS)
- SDC 2012 – Focus on operation replay, I/O ordering and scale out shares.
Client Retry - To try or not to (re)try

- A surprisingly non trivial problem.

- Factors influencing the client’s retry behavior
  - Responsiveness for client apps
  - Resiliency for server apps (CA file handles.)
  - Semantic correctness.

- Client retry logic is driven by -
  - Errors reported from the network stack.
  - Errors reported from the server.
  - Capabilities of server/share and the handle type.
  - Configured I/O timeouts.
## Client Retry Semantics in a Nutshell

<table>
<thead>
<tr>
<th>Handle Type</th>
<th>Client I/O retry behavior</th>
<th>Default Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent Handles</td>
<td>I/O retried until persistent handle times out OR handle reconnect fails with a fatal error.</td>
<td>60 seconds (Configurable)</td>
</tr>
<tr>
<td>“Regular” handles (on CA share)</td>
<td>Handles re-opened and I/O retried until a configured timeout.</td>
<td>60 seconds. (Same as SessTimeout)</td>
</tr>
<tr>
<td>Resilient Handles</td>
<td>I/O retried until resilient handle times out OR Handle reconnect fails with a fatal error.</td>
<td>Application specified.</td>
</tr>
<tr>
<td>Durable Handles</td>
<td>A fixed number of retries.</td>
<td>3 retries.</td>
</tr>
<tr>
<td>“Regular” handles (on non-CA share)</td>
<td>Single attempt.</td>
<td></td>
</tr>
</tbody>
</table>
Retry based on “error conditions”

- Errors returned by the transport stack.
  - Transport level disconnects.
  - Transport level retransmit timeouts.
- Special error codes returned by the server

<table>
<thead>
<tr>
<th>Class of error</th>
<th>Error codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit retry errors</td>
<td>STATUS_SERVER_UNAVAILABLE</td>
</tr>
<tr>
<td></td>
<td>STATUS_FILE_NOT_AVAILABLE</td>
</tr>
<tr>
<td></td>
<td>STATUS_SHARE_UNAVAILABLE</td>
</tr>
<tr>
<td>Share or session connectivity errors</td>
<td>STATUS_USER_SESSION_DELETED</td>
</tr>
<tr>
<td></td>
<td>STATUS_NETWORK_SESSION_EXPIRED</td>
</tr>
<tr>
<td></td>
<td>SEC_E_WRONG_PRINCIPAL (*)</td>
</tr>
<tr>
<td></td>
<td>STATUS_BAD_NETWORK_NAME</td>
</tr>
<tr>
<td></td>
<td>STATUS_NETWORK_NAME_DELETED</td>
</tr>
<tr>
<td>Volume or file level errors</td>
<td>STATUS_VOLUME_DISMOUNTED</td>
</tr>
<tr>
<td></td>
<td>STATUS_FILE_INVALID</td>
</tr>
</tbody>
</table>
Special Retry Considerations for Multichannel

- The client must retry I/O on all available channels before giving up.
  - In general, all I/O should be retried until there are no more active channels to the server.
  - Guarantees correctness for state creation/destruction operations like CREATE and CLOSE.
- Avoid cascading TCP timeouts using keep-alives or other liveness checks.
- Sequencing of replayed operations.
  - Explicit “replay ID” for CREATE and (UN)LOCK.
  - Channel epoch numbers to handle write-write conflicts.
First time connect to a server

- Client may not be able to accurately determine whether the server/share is CA capable.
  - How long should the client retry?
  - Should the client re-establish another connection to a different IP address? (for a scaleout configuration.)
- If nothing is known about the server, use default behavior based on client side settings.
- If negotiated dialect < SMB 3.0, client can safely assume non-CA & limit retries.
  - Client “remembers” negotiated server/share capabilities.
- Post tree-connect, client can use share capabilities
Keepalives for fast failure detection

- Client enables transport level keepalives (if available) on all channels.
  - On RDMA a lightweight “echo” mechanism is used.
- Different keepalive thresholds based on state.
  - Smallest chosen value is set on all underlying connections to the server.

<table>
<thead>
<tr>
<th>“Reason” for keepalive</th>
<th>Default value (seconds.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outstanding I/O operations</td>
<td>30 seconds. (on non-CA shares.) 10 seconds. (on CA shares.)</td>
</tr>
<tr>
<td>Open handles (non-persistent)</td>
<td>30 seconds. (half the default SessTimeout)</td>
</tr>
<tr>
<td>Open handles (persistent)</td>
<td>10 seconds</td>
</tr>
</tbody>
</table>
Scale Out File Server 101

- Scaleout at the NETNAME level.
  - Not at the share level!!
  - All (disk) shares exposed under a scaleout name MUST be scaleout shares.
  - All nodes must expose the same set of shares.
  - By default, pipes are not exposed under a scaleout name. (except for specific pipes used by MS-SRVS MS-WKSTA etc.)

- Only SMB 2.0 or higher clients can connect.
- Windows clients only connect to a single node at any given time. (for a given NETNAME).

DNS
- smbsrv.test.com
  - 10.1.1.5
  - 10.1.1.6
  - 10.1.1.7

\ SMBSRV.test.com\A
\ SMBSRV.test.com\B
Connecting to Scale Out Shares

- Clients typically use DNS round-robin to resolve a path to a node hosting the scaleout share.
  - DNS records could be stale.
  - Client attempts to connect to one (first) IP address.
  - Alternate IP addresses are used if connect attempt fails.
- Client MUST switch nodes if initial connection was made to a “bad” node.
  - Authentication fails with “incorrect target”.
  - Tree-Connect fails due to bad / unavailable share.
- Client tries to maintain affinity to a node.
  - Switching between nodes is expensive.
## Error handling for Scale Out Shares

<table>
<thead>
<tr>
<th>Error category</th>
<th>Recommended retry logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectivity errors at the transport layer.</td>
<td>Re-attempt another connection to the same IP address and then try other IP addresses.</td>
</tr>
<tr>
<td>Session setup errors. SEC_E_WRONG_PRINCIPAL</td>
<td>(WB) Retry 2 more times and switch to a different server.</td>
</tr>
<tr>
<td>Tree connect errors. STATUS_BAD_NETWORK_NAME</td>
<td>(WB) Retry 2 more times and switch to a different server.</td>
</tr>
</tbody>
</table>

(WB) Other errors in the “retryable” list will not result in the client switching nodes.
IO Ordering

- IO ordering MUST be enforced for Multichannel & Continuous Availability scenarios.
  - Non-state changing (safely replayable)
  - Exactly once (byte-range-locks, creates)
  - State changing operations prone to Write-Write conflicts

- For CA/failover scenarios, servers MUST ensure that all in-progress IOs are drained.
  - when client re-establishes its session.
  - when client reconnects to its handle.
IO Ordering – Channel Epoch
(Write-Write conflicts)

WRITE replay using channel epoch

- Lightweight compared to full replay detection.
- Guarantees that all previous “instances” of an I/O are drained before the replay is executed.
- Client maintains 16-bit channel epoch number.
  - Incremented on a network failure.
  - Sent to server via unused Status field.
  - Wraparound is expected after $2^{16}$ failures.
- Server fails “state changing” “non-replay” requests with stale epoch numbers.
- Server fails “state changing” “replay” requests when there are outstanding operations with older epoch numbers.
- Server returns STATUS_FILE_NOT_AVAILABLE to client. Client retries the operation (possibly with updated epoch numbers).
- Server validates the channel epoch at “handle” granularity.

Diagram:
- Client detects channel failure
- Server detects channel failure
- WRITE[1] (epoch = 5)
- READ[2] (epoch = 6)
- STATUS_SUCCESS
- WRITE[1] (replay, epoch = 6)
- STATUS_FILE_NOT_AVAILABLE
- WRITE[1] (replay, epoch = 6)
- STATUS_SUCCESS
Key Takeaways

- SMB 3.0 server implementations should carefully control the error codes they return during failover.
  - Exploit the retry-logic already built into the client.
  - Avoid blocking on the server side.
- Special client handling for scaleout shares.
- Exploit keepalives to detect server/network failures and avoid cascading timeouts.
  - Server-Client keepalives for lease breaks !!
- Pay attention to I/O ordering issues – especially on the server.
Questions?