Continuously Available SMB Observations and Lessons Learned

David Kruse
Mathew George
Microsoft

A Very Good Place to Start

From SMB2 Protocol Proposal Overview (2003)

- **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.

Make it Simpler, Stupid

SMB_COM_CREATE_DIRECTORY
SMB_COM_DELETE_DIRECTORY
SMB_COM_OPEN
SMB_COM_CREATE
SMB_COM_CLOSE
SMB_COM_FLUSH
SMB_COM_DELETE
SMB_COM_RENAME
SMB_COM_QUERY_INFORMATION
SMB_COM_SET_INFORMATION
SMB_COM_READ
SMB_COM_WRITE
SMB_COM_LOCK_BYTE_RANGE

SMB_COM_UNLOCK_BYTE_RANGE
SMB_COM_CREATE_TEMPORARY
SMB_COM_CREATE_NEW
SMB_COM_CHECK_DIRECTORY

SMB_COM_PROCESS_EXIT
SMB_COM_SEEK
SMB_COM_LOCK_AND_READ
SMB_COM_WRITE_AND_UNLOCK
SMB_COM_READ_RAW
SMB_COM_READ_MPX
SMB_COM_READ_MPX_SECONDARY
SMB_COM_WRITE_RAW
SMB_COM_WRITE_MPX
SMB_COM_WRITE_MPX_SECONDARY
SMB_COM_WRITE_COMPLETE

SMB_COM_QUERY_INFORMATION_SRV
SMB_COM_SET_INFORMATION2
SMB_COM_QUERY_INFORMATION2
SMB_COM_LOCKING_ANDX
SMB_COM_TRANSACTION
SMB_COM_TRANSACTION_SECONDARY
SMB_COM_IOCTL
SMB_COM_IOCTL_SECONDARY
SMB_COM_COPY
SMB_COM_MOVE
SMB_COM_ECHO
SMB_COM_WRITE_AND_CLOSE
SMB_COM_OPEN_ANDX

SMB_COM_READ_ANDX
SMB_COM_WRITE_ANDX
SMB_COM_CLOSE_AND_TREE_DISC
SMB_COM_TRANSACTION2

SMB_COM_TRANSACTION2_SECONDARY
SMB_COM_FIND_CLOSE2
SMB_COM_FIND_NOTIFY_CLOSE
SMB_COM_TREE_CONNECT
SMB_COM_TREE_DISCONNECT
SMB_COM_NEGOTIATE
SMB_COM_SESSION_SETUP_ANDX
SMB_COM_LOGOFF_ANDX
SMB_COM_TREE_CONNECT_ANDX
SMB_COM_QUERY_INFORMATION_DISK
SMB_COM_SEARCH

SMB_COM_FIND
SMB_COM_FIND_UNIQUE
SMB_COM_FIND_CLOSE
SMB_COM_NT_TRANSACT
SMB_COM_NT_TRANSACT_SECONDARY
SMB_COM_NT_CREATE_ANDX
SMB_COM_NT_CANCEL
SMB_COM_NT_RENAME
SMB_COM_OPEN_PRINT_FILE
SMB_COM_WRITE_PRINT_FILE
SMB_COM_CLOSE_PRINT_FILE
SMB_COM_GET_PRINT_QUEUE
SMB_COM_SEND_MESSAGE

SMB_COM_SEND_BROADCAST_MESSAGE
SMB_COM_FORWARD_USER_NAME
SMB_COM_CANCEL_FORWARD
SMB_COM_GET_MACHINE_NAME

SMB_COM_SEND_START_MB_MESSAGE
SMB_COM_SEND_END_MB_MESSAGE
SMB_COM_SEND_TEXT_MB_MESSAGE
TRANS_SET_NMPIPE_STATE
TRANS_RAW_READ_NMPIPE
TRANS_QUERY_NMPIPE_STATE
TRANS_QUERY_NMPIPE_INFO
TRANS_PEEK_NMPIPE
TRANS_TRANSACT_NMPIPE
TRANS_RAW_WRITE_NMPIPE
TRANS_READ_NMPIPE

TRANS_WRITE_NMPIPE
TRANS_WAIT_NMPIPE
TRANS_CALL_NMPIPE
TRANS2_OPEN2
TRANS2_FIND_FIRST2
TRANS2_FIND_NEXT2
TRANS2_QUERY_FS_INFORMATION
TRANS2_SET_FS_INFORMATION
TRANS2_QUERY_PATH_INFORMATION
TRANS2_SET_PATH_INFORMATION
TRANS2_QUERY_FILE_INFORMATION
TRANS2_SET_FILE_INFORMATION
TRANS2_FSCTL

TRANS2_IOCTL2
TRANS2_FIND_NOTIFY_FIRST
TRANS2_FIND_NOTIFY_NEXT
TRANS2_CREATE_DIRECTORY

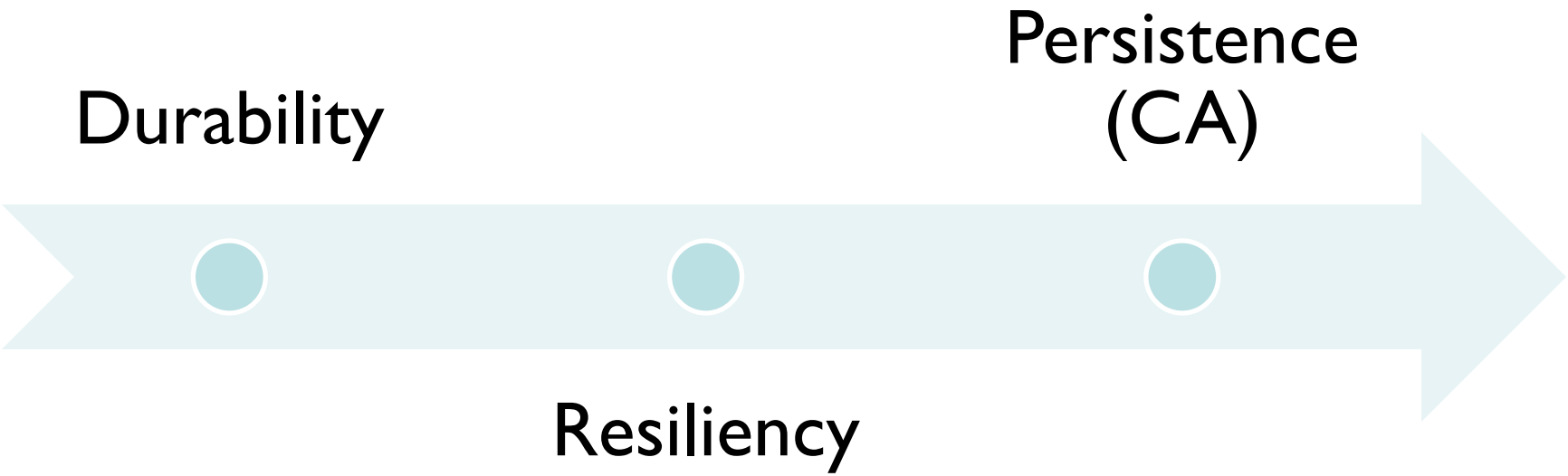
TRANS2_SESSION_SETUP
TRANS2_QUERY_FS_INFORMATION_FID
TRANS2_GET_DFS_REFERRAL
TRANS2_REPORT_DFS_INCONSISTENCY
NT_TRANSACT_CREATE
NT_TRANSACT_IOCTL
NT_TRANSACT_SET_SECURITY_DESC
NT_TRANSACT_NOTIFY_CHANGE
NT_TRANSACT_RENAME
NT_TRANSACT_QUERY_SECURITY_DESC
NT_TRANSACT_QUERY_QUOTA
NT_TRANSACT_SET_QUOTA

Looking Forward

From SMB2 Protocol Proposal Overview (2003)

- **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 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 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

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

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

Class of error	Error codes
Explicit retry errors	STATUS_SERVER_UNAVAILABLE STATUS_FILE_NOT_AVAILABLE STATUS_SHARE_UNAVAILABLE
Share or session connectivity errors	STATUS_USER_SESSION_DELETED STATUS_NETWORK_SESSION_EXPIRED SEC_E_WRONG_PRINCIPAL (*) STATUS_BAD_NETWORK_NAME STATUS_NETWORK_NAME_DELETED
Volume or file level errors	STATUS_VOLUME_DISMOUNTED STATUS_FILE_INVALID

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

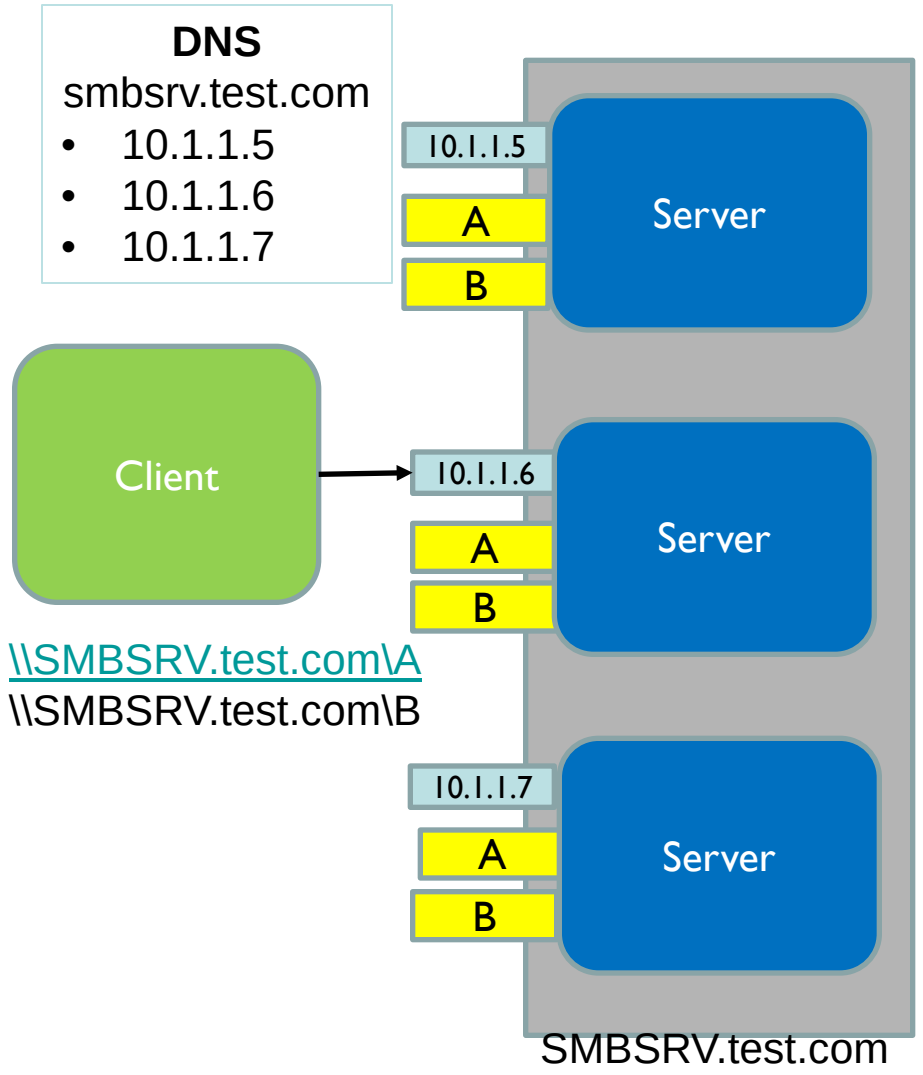
Keepalives for fast failure detection

- 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.

“Reason” for keepalive	Default value (seconds.)
Outstanding I/O operations	30 seconds. (on non-CA shares.) 10 seconds. (on CA shares.)
Open handles (non-persistent)	30 seconds. (half the default *SessTimeout*)
Open handles (persistent)	10 seconds

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).



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

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

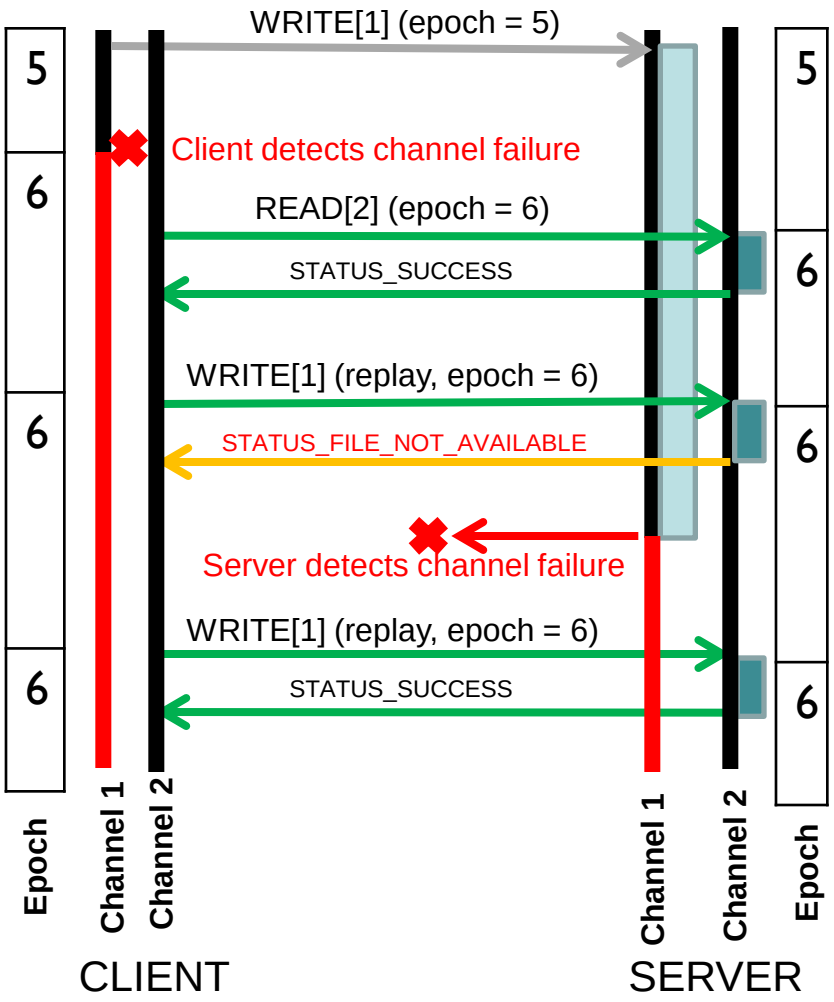
(WB) Other errors in the “retryable” list will not result in the client switching nodes.

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.

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?