Analyzing Metadata Caching in the Windows SMB2 Client

David Kruse, Mathew George
Microsoft Corporation

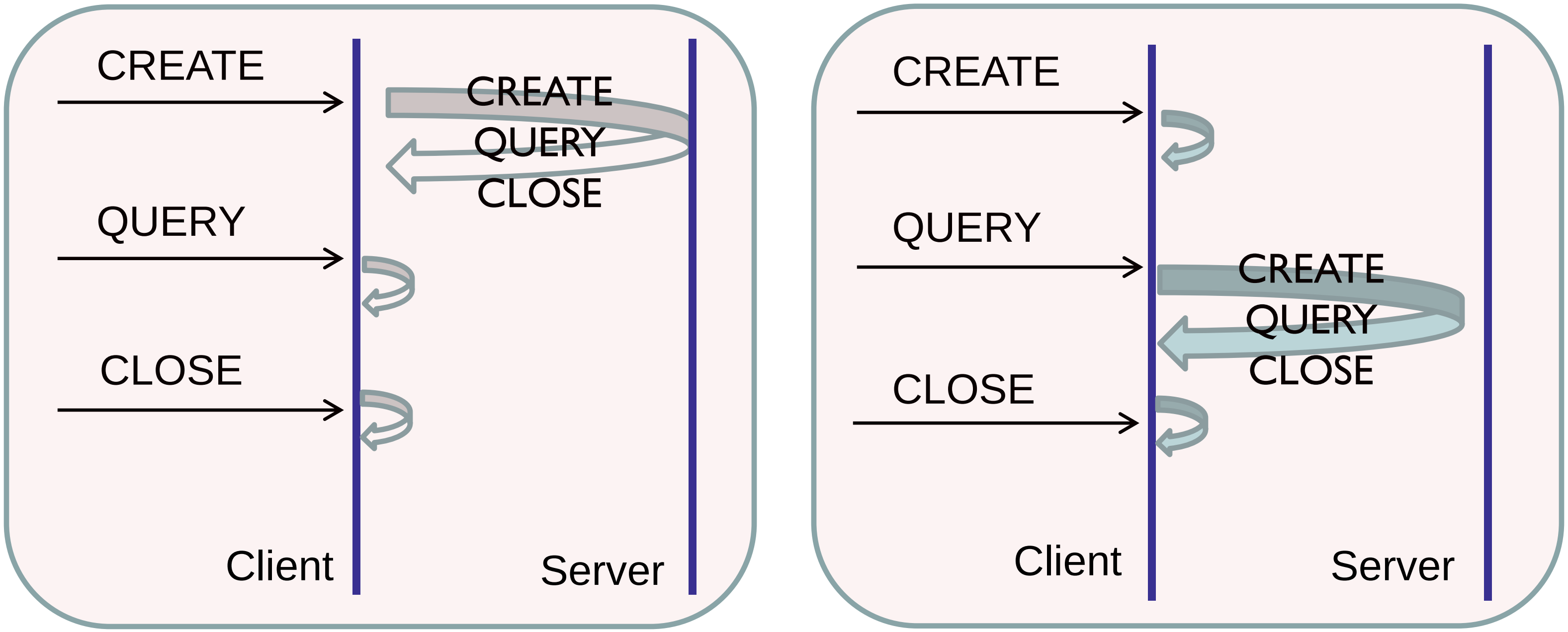
Agenda

- Understand what kind of metadata is cached by existing Windows SMB2 clients
- Discuss the coherency guarantees provided for these metadata caches.
- Analyzing protocol performance under different metadata intensive workloads.
- Outline possible protocol extensions to improve metadata caching in SMB2 through directory leases.

Why is metadata caching important?

SDC STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2010

- Handle based I/O semantics & simplified SMB2 command set => more round trips to achieve a given operation.
  - CREATE + QUERY/SET + CLOSE
  - Compounding (predictive or lazy) can be to reduce round trips.



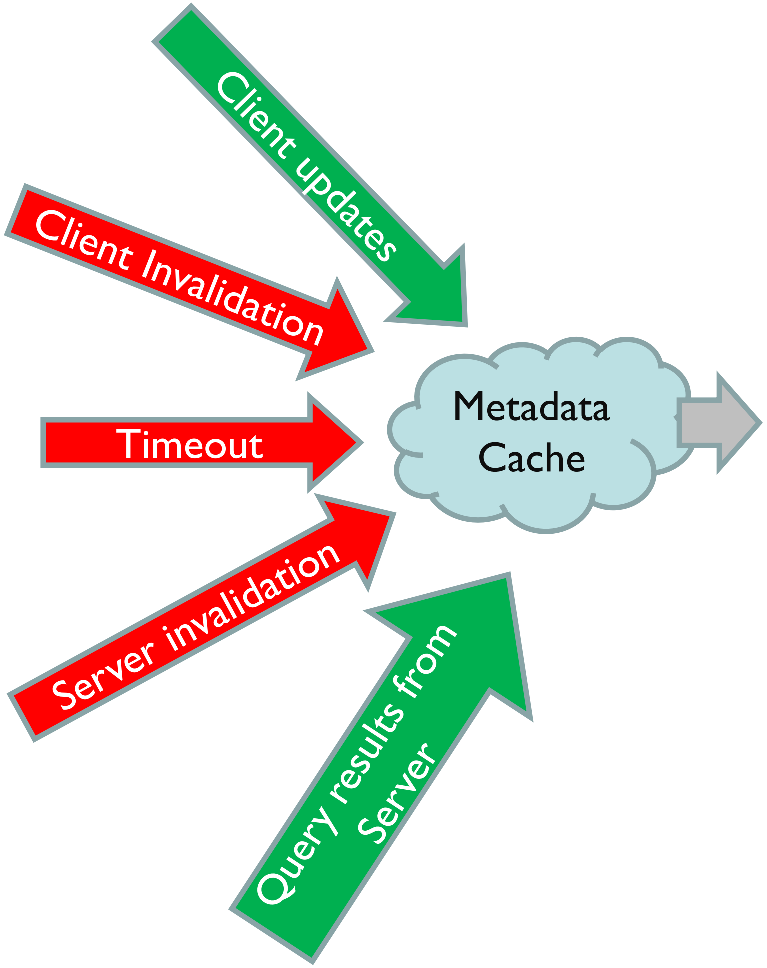
Why is metadata caching important?

- Metadata intensive "home folders" workload characterized by FSCT.
  - Significant fraction (60%) of the network round trips involve metadata queries.
  - Significant reduction in server load (improved server scalability.)
  - Reduction in network traffic.
  - Overall improvement in user perceived response times (especially over high latency links.)

What is "best effort" caching?

- Maintaining metadata cache on the client without associated state on the server.
  - No explicit metadata cache coherency support in the SMB2 protocol.
  - Leverage existing protocol to achieve "near consistent" view of metadata.
- Strike a balance between correctness and performance by caching metadata for the shortest possible time.

What is a "near coherent" cache?



- Present a consistent view of the cache to single client apps.
  - Update or invalidate cache on I/O (writes, deletes, renames, set-attributes)
- Short cache lifetimes.
- Leverage protocol exchanges as much as possible.
- Avoid name aliasing issues as much as possible.

Scaling cache timeouts

- Scaling based on network characteristics
  - Windows 7 client extends cache lifetimes on high latency links
- Scaling based on protocol exchanges.
  - Windows clients extend cache lifetimes if there are outstanding change notifications posted.

Network Latency	Cache Timeout
< 50 msec	10 sec (default)
< 200 msec	120 sec
>= 200 msec	300 sec

The 3 metadata caches : File Information Cache

SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2010

- Caches file metadata (attributes, size, timestamps)
  - *FileNetworkOpenInformation* structure.
- One entry per file identified by a128-bit ID.
  - ID returned via the QFid create context.
- Default lifetime is 10 sec.
- Cache updated via CREATE, CLOSE, QUERY responses
- Cache is invalidated when
  - File is locally modified by client app.
  - Oplock/Lease break notification from the server.
  - Directory change notification from the server.
- Application queries for *FileNetworkOpenInformation, FileBasicInformation and FileStandardInformation* satisfied from cache.
- Serves as a file existence cache.

The 3 metadata caches : File Not Found Cache

- Caches names of files which are known not to exist on the server.
  - Allows the client to locally short circuit repeated opens to the same name.
- Populated on a CREATE which fails with a "object not found" error.
- Very short lifetime (5 sec.)
- Invalidated when the client creates a new file or when the server notifies a directory change.
- Very useful in a "compile over network" workload.
  - ~ 2000 / 6000 creates failed !

The 3 metadata caches: Directory Cache

- Each directory cache entry caches an entire directory enumeration.
  - *FileIdBothDirectoryInformation* is cached.
  - Populated when application enumerates directory.
- Can satisfy information queries for individual files.
- Individual files within a directory cache entry can be updated / deleted when the client fetches updated file information from the server.
- Default lifetime is 10 secs.
- Can serve as a file existence / non-existence cache.
- Invalidation is similar to the file information cache.

Pitfalls to watch out for!

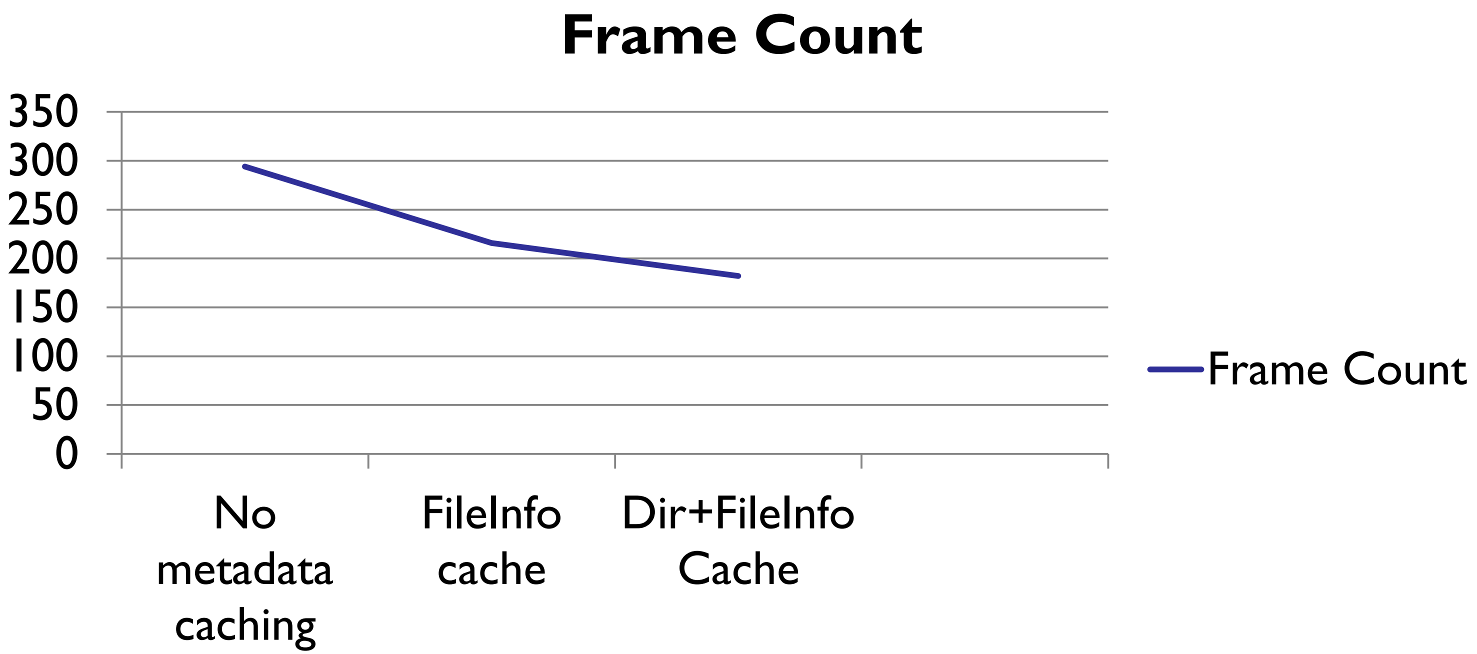
- Distributed Applications
  - Very typical in HPC scenarios which require data sharing between concurrently running tasks on multiple nodes.
  - Producer / consumer type of scenario
  - Workaround is to force apps (or system) to use mechanisms like change notifications.
- Legacy Applications
  - Applications which mix long and short names.
  - Applications coded for specific behavior patterns.
- Access based enumeration (ABE)
  - Different views of a directory based on user.

Caching Parameters

- Cache size
  - Maximum number of entries
  - Elastic thresholds to handle bursts of activity
  - Dynamic trimming of cache over time.
- Cache lifetime
  - Fixed vs. dynamic
  - Tradeoff between performance and correctness.
- Cache eviction policy

Scenario I : File browsing

The following chart shows the impact (in terms of the # of network frames exchanged) of the file-info and directory caches for a simple file browsing scenario of a directory with 60 files.



Scenario 2 : Engineering workload

- The data (frame counts) in the below table was provided by Marc Ullman and Ken Harris from MathWorks to highlight the effect of the metadata cache sizes for a "compile over the network" workload typical of large engineering environments.

	Windows 7 **Default Settings** FileInfoCacheEntries = 64 DirectoryCacheEntries = 16 FileNotFoundCacheEntries=128 Cache lifetime = 10s, 10s, 5s DormantFileLimit = 1024 CacheFileTimeout = 10 sec	**Windows 7** FileInfoCacheEntries = 32K DirectoryCacheEntries = 32K FileNotFoundCacheEntries=32K Cache lifetime = 60s, 60s, 60s DormantFileLimit = 4096 CacheFileTimeout = 10 sec	**Windows 7** **(Large MTU enabled)** FileInfoCacheEntries = 32K DirectoryCacheEntries = 32K FileNotFoundCacheEntries=32K Cache lifetime = 60s, 60s, 60s DormantFileLimit = 4096 CacheFileTimeout = 600 sec	**Windows 7** **(Large MTU enabled)** FileInfoCacheEntries = 32K DirectoryCacheEntries = 32K FileNotFoundCacheEntries=32K Cache lifetime = 600s, 600s, 600s DormantFileLimit = 16384 CacheFileTimeout = 3600 sec
CREATE	180302	53860	52920	48396
CLOSE	173078	47030	46631	42359
QUERY_DIR	68580	21103	19536	13495
READ	19340	16474	16448	15976
QUERY_INFO	763	225	217	198
Total Ops	449575	146126	143208	127858
% Improvement over default	N/A	67%	68%	72%

Looking Forward…

Caching Inefficiency

- A timeout based caching approach forces us to release a potentially correct cache entry due to uncertainty of its validity
- A timeout also results in us maintain (and return) an incorrect cache entry for a period of time, resulting in application or user confusion
- Both of these issues would exist with file data as well, but coherency is reinforced with opportunistic locking and leasing.

Directory Leasing

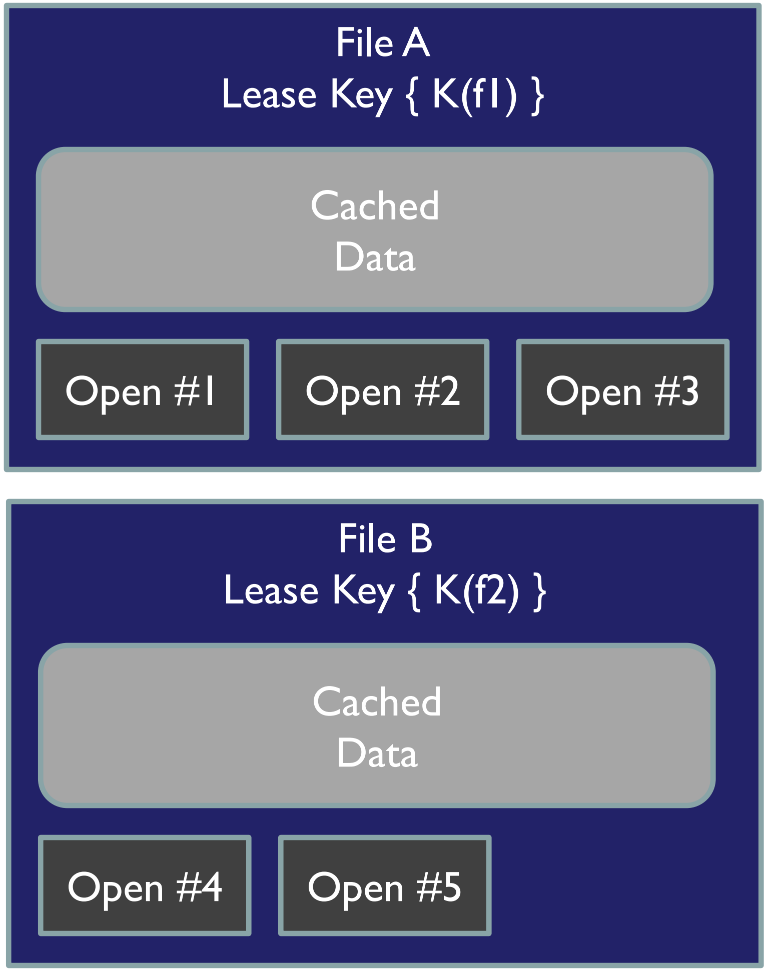
	File Leasing	Directory Leasing
Handle Caching	The client is permitted to cache a handle to the file. Revoked if server receives a conflicting open, etc.	Same
Read Caching	The client is permitted to cache data read from the file. Revoked if a write or byte-range lock is taken on the file by another client.	The client is permitted to cache directory enumeration results from the directory. Revoked if a meta-data modifying operation is received for this directory from another client.
Write Caching	The client is granted exclusive access to the directory, and permitted to cache writes and byte range locks. Revoked if another client attempts to access the file.	

Directory Leasing

- Leasing model allows multiple clients to obtain RH cache in collaboration or publication scenarios
- In scenarios with a single client (MyDocuments), the directory cache lifetime will often be infinite
- When a modification is made by another client or a local user, notification is sent immediately to improve coherency of clients

File Lease Key

- Client associates a lease key for a file with a cache
- Client provides that key for all opens to the same file
- Server ensures access on an open do not break leases or oplocks with a matching key
- Coherency is preserved even if the client is not aware that File A is the same as File B.

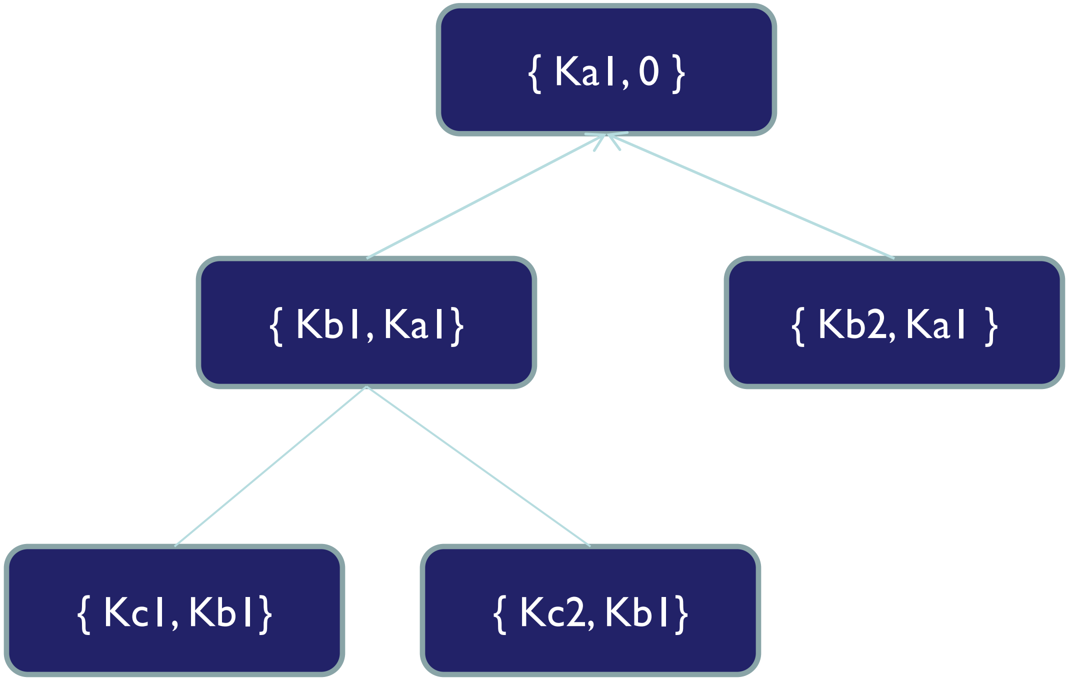


Directory Leasing Keys

- For directories, operations on the children modify the metadata of the parent (create, delete, modify)
- The children may themselves be a directory which also has children
- Read caching is revoked by a change in my children
- Handle caching is revoked by a conflicting open to myself
- Client selects lease keys to match their caching structure (as seen previously)

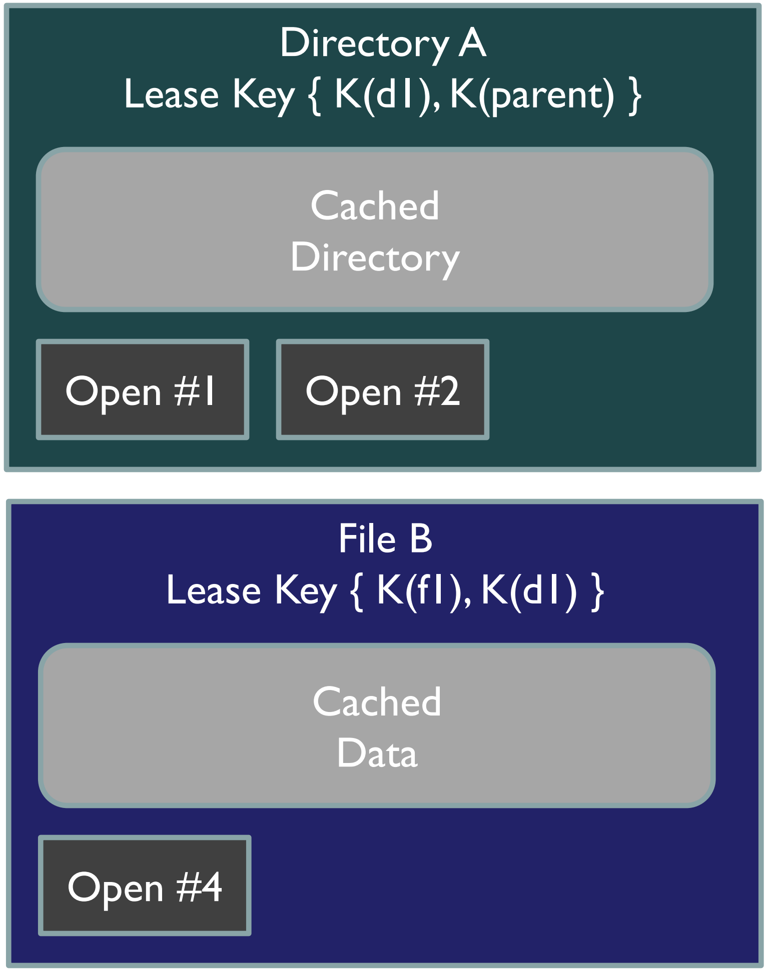
Directory Leasing Keys

- Client provides lease key pair $\{ K_{self}, K_{parent} \}$



Directory Leasing Key

- Client maintains directory cache on directory object, associates key
- Opens to children provide key of parent directory as K_{parent}
- Modification of child will not revoke read lease on parent directory if K_{parent} of modified open is equal to K_{self} of parent directory



Lease Break Algorithm

- Handle Caching
  - Handle caching is revoked if K_{self} of new open is not equal to K_{self} of conflicting open at create time. (Same as specified for opens to files.)
  - Delete and rename of folders is the most common issue that handle caching helps resolve.
- Read Caching
  - Read caching is revoked on directory metadata update if K_{parent} of the child who triggered the modification is not equal to K_{self} of the open on which the least was obtained.
  - Create new, modification of metadata, deletion of a file, or updating timestamps on close would be most common.

Modeled Performance Results

- Modeled reduction of frames in FSCT scenarios assuming directory oplock is held and cache is synchronized before running scenario

Directory Oplocks	Change Notify	Close	Create	Query Directory
CmdLineFileDelete		-2	-3	-2
CmdLineFileDownload		-2	-2	-2
CmdLineFileUpload		-1	-2	
CmdLineNavigate		-3	-4	-4
ExplorerDragDropFileDownload		-4	-7	-2
ExplorerDragDropFileUpload		-6	-8	-4
ExplorerFileDelete	-2	-13	-14	-10
ExplorerNavigate		-8	-11	-4
ExplorerSelect		-1	-1	-2
WordEditAndSave		-3	-6	
WordFileClose	-2	-17	-19	-8
WordFileOpen		-21	-25	-14

Extrapolated Perf Results

Relative FSCT Latency	
SMB 2.1	SMB vNext - DirOplock
100.0%	63.7%

- 36% client latency improvement compared to SMB 2.1 with directory caching

Key Takeaways

- If you are implementing an SMB2 client (or a SMB2 protocol accelerator), you could employ some of the strategies we use to ensure "best effort" consistency of the cached metadata.
- Depending on the application workload, the reduction in network traffic / server load can be very significant.
- The metadata caching parameters can be tweaked on the Windows SMB2 clients.
- Be aware that there will always be applications that require very strict metadata consistency guarantees and "best effort" caching may not work in those cases.
- Directory leasing could solve many issues relating to cache coherency and helps clients to cache metadata for longer periods of time, resulting in significant meta-data traffic reductions.

Questions?

Thanks!