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?

- 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.

<table>
<thead>
<tr>
<th>Network Latency</th>
<th>Cache Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 50 msec</td>
<td>10 sec (default)</td>
</tr>
<tr>
<td>&lt; 200 msec</td>
<td>120 sec</td>
</tr>
<tr>
<td>&gt;= 200 msec</td>
<td>300 sec</td>
</tr>
</tbody>
</table>
The 3 metadata caches:

File Information Cache

- Caches file metadata (attributes, size, timestamps)
  - FileNetworkOpenInformation structure.
- One entry per file identified by a 128-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 1: 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.

Frame Count

```
0  50  100  150  200  250  300  350
No metadata caching  FileInfo cache  Dir+FileInfo Cache

---

Frame Count
```
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.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Windows 7 Default Settings</th>
<th>Windows 7 (Large MTU enabled)</th>
<th>Windows 7 (Large MTU enabled)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FileInfoCacheEntries = 64</td>
<td>FileInfoCacheEntries = 32K</td>
<td>FileInfoCacheEntries = 32K</td>
</tr>
<tr>
<td></td>
<td>Cache lifetime = 10s, 10s, 5s</td>
<td>Cache lifetime = 60s, 60s, 60s</td>
<td>Cache lifetime = 60s, 60s, 60s</td>
</tr>
<tr>
<td></td>
<td>DormantFileLimit = 1024</td>
<td>DormantFileLimit = 4096</td>
<td>DormantFileLimit = 4096</td>
</tr>
<tr>
<td></td>
<td>CacheFileTimeout = 10 sec</td>
<td>CacheFileTimeout = 60 sec</td>
<td>CacheFileTimeout = 3600 sec</td>
</tr>
<tr>
<td>CREATE</td>
<td>180302</td>
<td>53860</td>
<td>52920</td>
</tr>
<tr>
<td>CLOSE</td>
<td>173078</td>
<td>47030</td>
<td>46631</td>
</tr>
<tr>
<td>QUERY_DIR</td>
<td>68580</td>
<td>21103</td>
<td>19536</td>
</tr>
<tr>
<td>READ</td>
<td>19340</td>
<td>16474</td>
<td>16448</td>
</tr>
<tr>
<td>QUERY_INFO</td>
<td>763</td>
<td>225</td>
<td>217</td>
</tr>
<tr>
<td>Total Ops</td>
<td>449575</td>
<td>146126</td>
<td>143208</td>
</tr>
<tr>
<td>% Improvement over default</td>
<td>N/A</td>
<td>67%</td>
<td>68%</td>
</tr>
</tbody>
</table>

% 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

<table>
<thead>
<tr>
<th></th>
<th>File Leasing</th>
<th>Directory Leasing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle Caching</td>
<td>The client is permitted to cache a handle to the file. Revoked if server receives a conflicting open, etc.</td>
<td>Same</td>
</tr>
<tr>
<td>Read Caching</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Write Caching</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
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)
- Client provides lease key pair \( \{ K_{self}, K_{parent} \} \)

- Diagram:
  - \( \{ K_{a1}, 0 \} \)
  - \( \{ K_{b1}, K_{a1} \} \)
  - \( \{ K_{b2}, K_{a1} \} \)
  - \( \{ K_{c1}, K_{b1} \} \)
  - \( \{ K_{c2}, K_{b1} \} \)
- Client maintains directory cache on directory object, associates key
- Opens to children provide key of parent directory as $K_{\text{parent}}$
- Modification of child will not revoke read lease on parent directory if $K_{\text{parent}}$ of modified open is equal to $K_{\text{self}}$ 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

<table>
<thead>
<tr>
<th>Directory Oplocks</th>
<th>Change Notify</th>
<th>Close</th>
<th>Create</th>
<th>Query Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>CmdLineFileDelete</td>
<td></td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>CmdLineFileDownload</td>
<td></td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>CmdLineFileUpload</td>
<td></td>
<td>-1</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>CmdLineNavigate</td>
<td></td>
<td>-3</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>ExplorerDragDropFileDownload</td>
<td></td>
<td>-4</td>
<td>-7</td>
<td>-2</td>
</tr>
<tr>
<td>ExplorerDragDropFileUpload</td>
<td></td>
<td>-6</td>
<td>-8</td>
<td>-4</td>
</tr>
<tr>
<td>ExplorerFileDelete</td>
<td></td>
<td>-2</td>
<td>-13</td>
<td>-14</td>
</tr>
<tr>
<td>ExplorerNavigate</td>
<td></td>
<td>-8</td>
<td>-11</td>
<td>-4</td>
</tr>
<tr>
<td>ExplorerSelect</td>
<td></td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>WordEditAndSave</td>
<td></td>
<td>-3</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>WordFileClose</td>
<td></td>
<td>-2</td>
<td>-17</td>
<td>-19</td>
</tr>
<tr>
<td>WordFileOpen</td>
<td></td>
<td>-21</td>
<td>-25</td>
<td>-14</td>
</tr>
</tbody>
</table>
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!