Level 1 Opportunistic Locks

Si intuisce che level 1 corrisponde concettualmente allo stato M di un dato in cache nel protocollo MESI

A level 1 opportunistic lock on a file allows a client to read ahead in the file and cache both read-ahead and write data from the file locally. As long as the client has sole access to a file, there is no danger to data coherency in providing a level 1 opportunistic lock.

The client can request a level 1 opportunistic lock after opening a file. If no other client (or no other thread on the same client) has the file open, the server may grant the opportunistic lock. The client can then cache read and write data from the file locally. If another client has opened the file, then the server refuses the opportunistic lock and the client does no local caching. (The server may refuse the opportunistic lock for other reasons as well, such as not supporting opportunistic locks.)

Seguono alcune considerazioni abbastanza ovvie finalizzate al rispetto della coerenza dei dati

When the server opens a file that already has a level 1 opportunistic lock on it, the server examines the sharing state of the file before it breaks the level 1 opportunistic lock. If the server breaks the lock after this examination, the time the client with the former lock spends flushing its write cache is time the client requesting the file must wait. This time expenditure means that level 1 opportunistic locks should be avoided on files that many clients open.

However, because the server checks the sharing state before it breaks the lock, in the case where the server would deny an open request due to a sharing conflict the server does not break the lock. For example, if you have opened a file, denied sharing for write operations, and obtained a level 1 lock, the server denies another client's request to open the file for writing before it even examines your lock on the file. In this instance, your opportunistic lock is not broken.

For an example of how a level 1 opportunistic lock works, see Level 1 Opportunistic Lock Example. For more information on breaking opportunistic locks, see Breaking Opportunistic Locks.

Si intuisce che il termine “breaking a lock” indica un cambio di stato di un dato in cache (p.e. da M a S o da M a I nel caso MESI) con le relative azioni

Level 2 Opportunistic Locks

Si intuisce che level 2 corrisponde concettualmente allo stato S di un dato in cache nel protocollo MESI

A level 2 opportunistic lock informs a client that there are multiple concurrent clients of a file and that none has yet modified it. This lock allows the client to perform read operations and obtain file attributes using cached or read-ahead local information, but the client must send all other requests (such as for write operations) to the server. Your application should use the level 2 opportunistic lock when you expect other applications to write to the file at random or read the file at random or sequentially.

A level 2 opportunistic lock is very similar to a filter opportunistic lock. In most situations, your application should use the level 2 opportunistic lock. Only use the filter lock if you do not want open operations for reading to cause sharing-mode violations in the time span between your application's opening the file and receiving the lock. For more information, see Filter Opportunistic Locks and Server Response to Open Requests on Locked Files.

For more information on breaking opportunistic locks, see Breaking Opportunistic Locks.

Batch Opportunistic Locks

Si intuisce che di tratta di un’ottimizzazione di prestazioni ortogonale al concetto di stato del dato in cache

A batch opportunistic lock manipulates file openings and closings. For example, in the execution of a batch file, the batch file may be opened and closed once for each line of the file. A batch opportunistic lock opens the batch file on the server and keeps it open. As the command processor "opens" and "closes" the batch file, the network redirector intercepts the open and close commands. All the server receives are the seek and read commands. If the client is also reading ahead, the server receives a particular read request at most one time.

When opening a file that already has a batch opportunistic lock, the server checks the sharing state of the file after breaking the lock. This check gives the holder of the lock a chance to complete flushing its cache and to close the file handle. An open operation attempted during the sharing check does not cause the sharing check to fail if the lock holder releases the lock.

For an example of how a batch opportunistic lock works, see Batch Opportunistic Lock Example. For more information on breaking opportunistic locks, see Breaking Opportunistic Locks.

Si intuisce che di tratta di una variante di oplock di livello 2 (stato S del MESI) con una ottimizzazione di prestazioni nel transitorio di scambio messaggi tra client e server

Filter Opportunistic Locks

A filter opportunistic lock locks a file so that it cannot be opened for either write or delete access. All clients must be able to share the file. Filter locks allow applications to perform nonintrusive filtering operations on file data (for example, a compiler opening source code or a cataloging program). A filter opportunistic lock differs from a level 2 opportunistic lock in that it allows open operations for reading to occur without sharing-mode violations in the time span between your application's opening the file and receiving the lock. The filter opportunistic lock is the best lock to use when it is important to allow other clients reading access. In other cases, your application should use a level 2 opportunistic lock. A filter opportunistic lock differs from a batch opportunistic lock in that it does not allow multiple openings and closings to be handled by the network redirector the way a batch opportunistic lock does.

Your application should request a filter opportunistic lock on a file in three steps:

1. Use the CreateFile function to open a handle to the file with the DesiredAccess parameter set to zero, indicating no access, and the dwShareMode parameter set to the FILE_SHARE_READ flag to allow sharing for reading. The handle obtained at this point is called the locking handle.
2. Request a lock on this handle with the FSCTL_REQUEST_FILTER_OPLOCK control code.
3. When the lock is granted, use CreateFile to open the file again with DesiredAccess set to the GENERIC_READ flag. Set dwShareMode to the FILE_SHARE_READ flag to allow others to read the file while you have it open, the FILE_SHARE_DELETE flag to allow others to mark the file for deletion while you have it open, or both. The handle obtained at this point is called the read handle.

Use the read handle to read from or write to the contents of the file.

When opening a file that already has a filter opportunistic lock, the server breaks the lock and then checks the sharing state of the file. This check gives the client that held the former opportunistic lock a chance to abandon any cached data and acknowledge the break. An open operation attempted during this sharing check does not cause the sharing check to fail if the former lock holder releases the lock. The file system holds in abeyance the open operation until the lock's owner closes both the read handle and then the locking handle. After this is done, the other client's open request can proceed.

Breaking Opportunistic Locks

Si intuisce forse un po' meglio di prima che il termine "breaking a lock" indica un cambio di stato di un dato in cache con le relative azioni

Breaking an opportunistic lock is the process of degrading the lock that one client has on a file so that another client can open the file, with or without an opportunistic lock. When the other client requests the open operation, the server delays the open operation and notifies the client holding the opportunistic lock.

The client holding the lock then takes actions appropriate to the type of lock, for example abandoning read buffers, closing the file, and so on. Only when the client holding the opportunistic lock notifies the server that it is done does the server open the file for the client requesting the open operation. However, when a level 2 lock is broken, the server reports to the client that it has been broken but does not wait for any acknowledgment, as there is no cached data to be flushed to the server.

Quando si parte dallo stato S non serve aspettare la risposta del client per garantire la coerenza

In acknowledging a break of any exclusive lock (filter, level 1, or batch), the holder of a broken lock cannot request another exclusive lock. It can degrade an exclusive lock to a level 2 lock or no lock at all. The holder typically releases the lock and closes the file when it is about to close the file anyway.

Quando si parte dallo stato M si può passare a S o I

Applications are notified that an opportunistic lock is broken by using the hEvent member of the OVERLAPPED structure associated with the file on which the lock is broken. Applications may also use functions such as GetOverlappedResult and HasOverlappedIoCompleted.

Server Message Block 2.0

L’introduzione di azioni composte e’ una tipica soluzione di ottimizzazione vista anche nell’evoluzione di NFS

A new version of the Server Message Block (SMB) protocol has been introduced with Windows Vista. A significant improvement over SMB support in prior versions of Windows is the ability to compound multiple actions into a single request, which significantly reduces the number of round-trips the client needs to make to the server, improving performance as a result. SMB1 also has a compounding mechanism (known as AndX) to compound multiple actions, but is rarely used by Microsoft clients. Larger buffer sizes are supported, also increasing performance with large file transfers. The notion of "durable file handles" is introduced, which allow a connection to an SMB server to survive brief network outages, such as with a wireless network, without having to construct a new session. Support for symbolic links is included as well. In SMB 1, various sizes in the protocol are 16 bits. Many have been changed to 32 or 64 bit, and in the case of file handles to 16 bytes.

Questa nozione richiama una caratteristica delle prime versioni di NFS legata alla scelta “stateless”

SMB2 reduces the 'chattiness' of the protocol by reducing the number of commands and subcommands to 19 from over 100. It has mechanisms for pipelining, that is, sending additional requests before the response to a previous request arrives. Other improvements include caching of file properties, improved message signing with HMAC SHA-256 hashing algorithm and better scalability by increasing number of users, shares and open files per server among others.

Il pipelining è un’altra tipica soluzione di ottimizzazione riscontrabile in molti protocolli (non solo per file system)

Introduction to DFS Replication

DFS Replication, the successor to the File Replication service (FRS) introduced in Windows 2000 Server operating systems, is a new, state-based, multimaster replication engine that supports replication scheduling and bandwidth throttling. DFS Replication uses a new compression algorithm known as remote differential compression (RDC). RDC is a "diff-over-the wire" client-server protocol that can be used to efficiently update files over a limited-bandwidth network. RDC detects insertions, removals, and re-arrangements of data in files, enabling DFS Replication to replicate only the changed file blocks when files are updated. DFS Replication uses many sophisticated processes to keep data synchronized on multiple servers. Before you begin using DFS Replication, it is helpful to understand the following concepts.

Si dichiara la scelta stateful

Corrisponde alla politica “Write-one” con propagazione “gossip” delle modifiche

Per la replicazione non è necessario duplicare tutti i dati

Introduction to DFS Replication

- DFS Replication is a multimaster replication engine. Any change that occurs on one member is replicated to all other members of the replication group. **Corrisponde alla politica “Write-one” con propagazione “gossip” delle modifiche**

Usa la tecnica generale del numero di versione vista p.e. anche nel caso DNS

- DFS Replication detects changes on the volume by monitoring the update sequence number (USN) journal, and DFS Replication replicates changes only after the file is closed.

Usa la semantica di condivisione detta “Session Semantics”

- DFS Replication uses a staging folder to stage a file before sending or receiving it. For more information about staging folders, see Staging folders and Conflict and Deleted folders.

- DFS Replication uses a version vector exchange protocol to determine which files need to be synchronized. The protocol sends less than 1 kilobyte (KB) per file across the network to synchronize the metadata associated with changed files on the sending and receiving members.

- When a file is changed, only the changed blocks are replicated, not the entire file. The RDC protocol determines the changed file blocks. Using default settings, RDC works for any type of file larger than 64 KB, transferring only a fraction of the file over the network.

Per la replicazione non è necessario duplicare tutti i dati

- DFS Replication uses a conflict resolution heuristic of last writer wins for files that are in conflict (that is, a file that is updated at multiple servers simultaneously) and earliest creator wins for name conflicts. Files and folders that lose the conflict resolution are moved to a folder known as the Conflict and Deleted folder. You can also configure the service to move deleted files to the Conflict and Deleted folder for retrieval should the file or folder be deleted. For more information, see Staging folders and Conflict and Deleted folders.

I problemi di coerenza, ove necessario, devono essere gestiti a mano controllando la cartella delle inconsistenze

- DFS Replication is self-healing and can automatically recover from USN journal wraps, USN journal loss, or loss of the DFS Replication database.

- DFS Replication uses a Windows Management Instrumentation (WMI) provider that provides interfaces to obtain configuration and monitoring information from the DFS Replication service.