Coordinated Backup and Recovery between Database Management Systems and File Systems

Inderpal Narang, C. Mohan, Karen Brannon, Mahadevan Subramanian

650 Harry Road, IBM Almaden Research Center, San Jose, CA 95120, USA
{narang, mohan, kbrannon, maha}@almaden.ibm.com, http://www.almaden.ibm.com/u/mohan

Abstract: We consider a network of computers consisting of file servers and a Database Management System (DBMS) where a linkage is maintained, with referential integrity, between data in the DBMS and files in the file servers which are external to the DBMS. We present algorithms for performing backup and recovery of the DBMS data in a coordinated fashion with the files on the file servers. When a file is associated (*linked*) with a record in the DBMS, certain constraints are applied to support referential integrity, access control, and coordinated backup and recovery *as if* the file is stored in the DBMS. Backup of a referenced file is initiated when the file is linked. The file backup is performed asynchronously to the linking process so that the linking transaction is not delayed. In a typical scenario, when a database backup operation starts, all unfinished file backups are ensured to be completed before the database backup is declared successful. When a database is recovered to a state which includes references to files in one or more file servers, the DBMS ensures that the referenced files are also restored to their correct state in those file servers. However, since database backup and recovery are critical for database availability, the presence of an unavailable file server is tolerated during the database backup and recovery operations. Our algorithms for coordinated backup and recovery have been implemented in the IBM DB2/DataLinks product. The DataLinks concept is also part of the ISO SQL/MED standard [ISO00].

1. Introduction

The motivation for the DataLinks feature of DB2 stems from a number of observations. Most of the world's data lives in files. Most data would continue to live there and the volumes will grow. Most of the world's applications work on files. Corba and the OLE DB framework have been proposed for providing uniform access to database and nondatabase data [Obj95, Blak96]. An example of extending database capabilities to external repositories such as file systems can be found in [BlRS96] which focuses on extending database indexing, partitioning, replication and query processing to data stored external to the database. While much of the world' data resides in files, file systems do not provide the rich data management characteristics found in a DBMS. File systems do not provide sufficient metadata, nor do they provide a general purpose query engine to manage access to and integrity of files. The DBMS has evolved to become a superb manager of data, i.e., it provides referential integrity, access control, and backup and recovery for its data. It allows arbitrarily complex data models and arbitrary queries. But a DBMS may not always be appropriate for the storage of very large objects . Many RDBMSs support the LOB data type for storing such data, however LOBs require the use of a SQL API to operate on them. This presents a problem for preexisting applications written to operate on data in files. The Oracle Internet File System (iFS) [Ora00] addresses this concern by supporting the file system interface. SHORE [CaDNS94] is another system which merges database and file system technologies. SHORE objects can be accessed via the normal UNIX file system interface. In both systems, however the file/object data is actually stored in the DBMS as LOBs. LOBs typically do not have support for hierarchical storage management as provided by products like Tivoli TSM (previously called ADSM [CaRH95]). Such a support is crucial for cost-effective management of data with varying access patterns (frequent to infrequent

usage). In addition, there is a substantial read/write performance penalty to access LOB data compared to accessing the data as normal files in a networked file system [Ora00]. Also, legacy data maintained in files would have to be loaded into such systems. With DataLinks, legacy data can remain in files as is. DataLinks attempts to marry file systems and databases in such a way to extend the DBMS data management capabilities of data-valued based access control, referential integrity and backup/recovery to file systems, while retaining the powerful data management properties of file systems. DB2/DataLinks is an optional feature available for IBM's DB2 on open systems.

The importance of integrating files containing unstructured and semi-structured data with business applications is growing. It is possible to maintain pointers to external files as VARCHAR data, for example in a DBMS, to address this integration problem. While supporting applications written to use the file paradigm, this solution presents a daunting administration task: keeping the file data and DBMS data synchronized, especially over DB backup/recovery and file system backup/recovery. A system such as Oracle iFS or SHORE could be used to store the file data. Since this data is actually stored in the DBMS, there are no issues with coordinating DBMS backup/recovery and file system backup/ recovery. However, there can be a significant performance degradation for reading the data compared with reading directly from a file system. Also, the time to do a DB backup can become excessive when backups include a large amount of data in LOBs. DataLinks extends the DBMS functionality of management of data to files stored in file systems while providing referential integrity, access control, and coordinated backup and recovery AS IF the files are stored in the DBMS [NaRe95]. A key feature of DataLinks is that the data in linked files can be accessed directly from the file system with no need to flow the data through the DBMS. This provides a performance advantage as well as the ability to support existing/new applications based on the file paradigm, but with improved DBMS-style management of the data in files. DataLinks can also be used for the management of a large number of files in a network of computers. The storage model is the usage of a DB as the metadata repository with URLs (Uniform Resource Locators) as pointers to the files residing in multiple file servers, which may belong to heterogeneous computer nodes and operating systems. This is shown below in Figure 1. The semantics of the field containing the URL is that the reference to the object stored in the external file system is consistent with the metadata (relating to that object) which is stored in the database. For example, the object cannot be deleted from the external store as long as there is a reference to it in the database. An application searches the metadata in the database via an SQL query and gets the URLs in the result of the query. Each URL has a web/file server name and a file name. Then, using a web browser or standard file system calls such as fopen, fread, etc., the application accesses the file directly from the appropriate server. Supporting standard file system APIs allows legacy applications and web applications to work without modification.

Typically, a user extracts features of an image or a video and stores them in the database in order to perform searches on the extracted features, with perhaps predicates on other business data. An example of the features which can be extracted for an image are color, shape and texture. IBM's Query By Image Content (QBIC) supports extraction and search on such features [Flic95]. So, searches are done in the DBMS on such features so that applications can still take advantage of the power of SQL and RDBMSs.

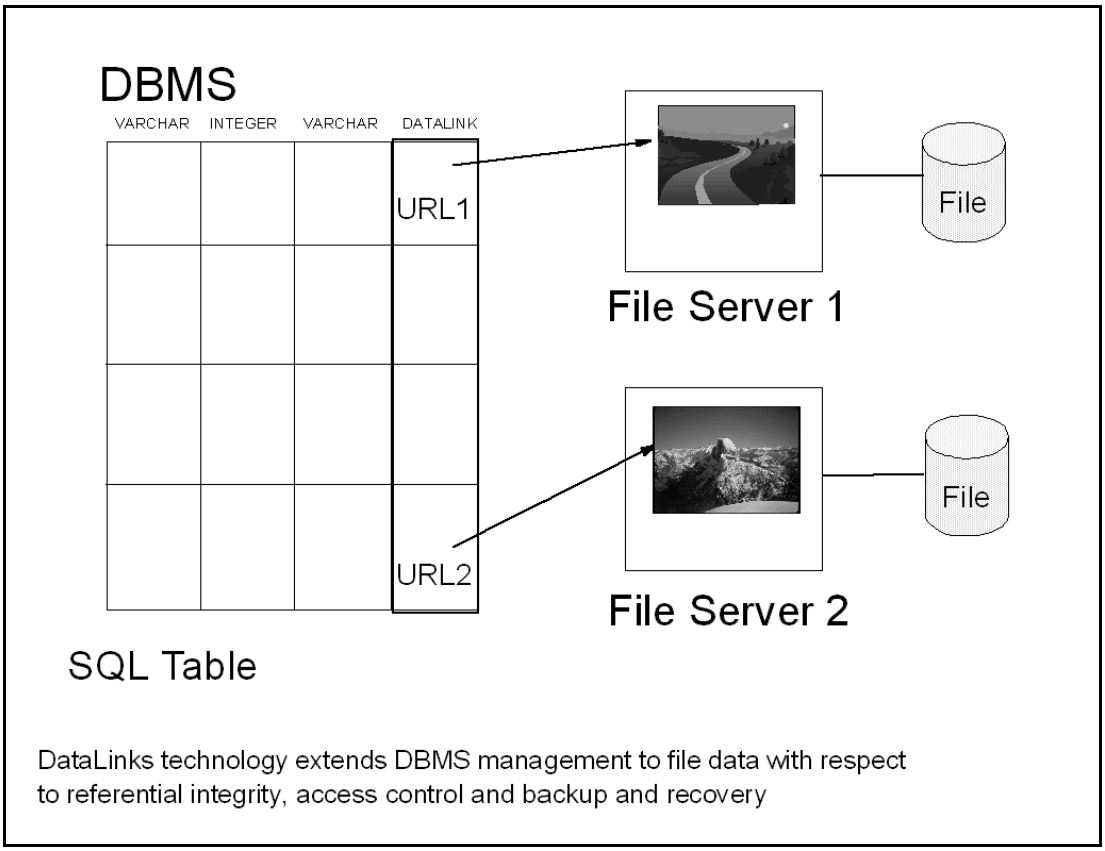

Figure 1. Storage Model of DataLinks

Normal database administration requires that the database be backed up periodically, for example, once a week. This is so that in case of database corruption or device failure the database can be restored from one of the backup copies, typically the most recent one [MoNa93]. Depending on whether or not updates to the database are allowed to take place while the backup operation is in progress, the backup copy may or may not be self consistent. The restoration of the database contents will typically require processing of the log records to bring the database to its most recent state. The latter is typically the desired state (e.g., when a device failure occurs). However, at times, it is desired that the database not be brought to its most recent state, because an application may have corrupted the database. When the database is brought to a state other than the most recent state, it is called point-in-time recovery. In this paper, we present algorithms for performing backup and recovery of the DBMS data in a coordinated fashion with the file servers which have files referenced from the database. From a backup and recovery standpoint, the files are considered to be part of the database but they are actually stored external to the DBMS. In anticipation of a DBMS backup operation, which would occur sometime in the near future, the backup of a referenced file is initiated when the file is associated (*linked*) with a record in the DBMS. The file backup is performed asynchronously to the linking process so that the linking transaction is not delayed. In a typical scenario, when a database backup occurs, all unfinished file backups are ensured to be completed before the database backup is declared successful. When a database is restored to a state which includes references to files in the file system, the DBMS ensures that the

referenced files in the file servers are also restored to their correct state. Note that the files could have been deleted or newer versions may have been created as a result of operations subsequent to the backup from which the database is restored. Our algorithms take into account these possibilities. It should be noted here that the referenced external files themselves are not kept in the same backup file where the database metadata is backed up. Rather, these external files may be backed up individually in a backup server, such as, Tivoli Storage Manager (TSM) [CaRH95]. The remainder of this paper is organized as follows. The normal operations of DataLinks which are relevant to backup and recovery are presented in Section 2. Section 3 presents the actions that take place as part of a database backup operation. Techniques relating to restoration of a database are then discussed in section 4. Section 5 presents extensions to algorithms to ensure that the database backup and recovery operations succeed even if one or more DLFS file servers are unavailable. This is needed since database backup and recovery are critical for availability of the database data. The summary and conclusions are presented in Section 6.

2. Normal Operation of DataLinks

Figure 2. illustrates a typical DataLinks configuration. DataLinks functionality is supported by introducing **DATALINK** as an SQL data type in the DBMS, and by having a piece of cooperating software, called *DataLinks Manager* on a file server. The DATALINK SQL datatype has been introduced as an ISO standard [ISO00]. The Datalinks Manager (DLM) has two distinct software components, DLFM and DLFF, which are described below. The DATALINK column(s) in an SQL table contain the "pointer" to the file stored in a file server. The data structure for the DATALINK data type includes the name of a file server and the name of a file, as in URLs. An application uses an SQL call to query the DBMS to search on the business data and perhaps, the features of unstructured (and/or semi-structured) data stored in the DBMS. The query specifies DATALINK column(s) in the select list and the DBMS returns the URL information to the client application. The client application can then access the file(s) using the normal file system protocols. The DBMS is not in the data path for delivery of file data, although file data can optionally flow through the DBMS in case an application just wants to maintain a single connection to the DB (rather than using connectivity with the file server). Note that we have introduced the new data type in the DBMS, but not any new API for database access or file access. We believe that this is significant for the following reasons.

- Current DBMS applications can be easily extended to incorporate new types of data such as image, video, text etc. using the SQL API to obtain the file server name(s) and file name(s) of interest.
- Applications to capture, edit and deliver the new types of data work on the file paradigm and would not require any change. Typically the file server name and file name obtained from the database can be fed into the application which provides the file access, e.g., web browser, or viewer (or helper) applications on the desktop for image, video, etc.
- Since the file data does not flow through the DBMS, there is negligible impact on the performance of file access compared to native file system access.

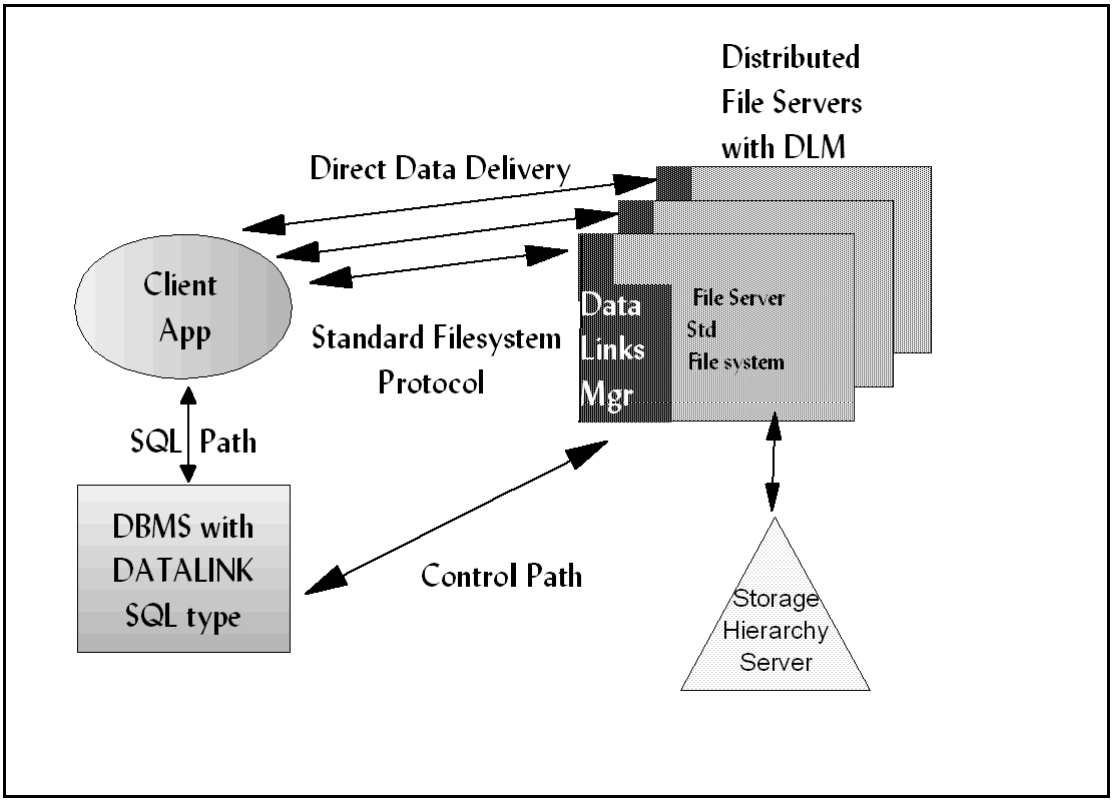

Figure 2. System Configuration for DataLinks

The components of the of the Datalinks Manager are described below.

- **DLFM** (DataLinks File Manager)

  As part of the data manipulative processing for storing the DATALINK value in the database, the DBMS communicates with the appropriate DLFM in the network, based on the server name provided in the DATALINK data value. This communication is the basis for the DBMS coordination for management of files on the corresponding file server. DLFM supports transactional properties to the database data and references to the files, and coordinated backup and recovery between database data and files. The DBMS interacts with DLFM using a two-phase commit protocol as described by Hsiao and Narang [HsNa00] to provide transaction atomicity. For this purpose, the DBMS acts as the coordinator and DLFM as the subordinate. DLFM needs some recoverable (transactional) storage for keeping persistent information about the operations (like LinkFile, UnlinkFile, etc.) that are initiated at DLFM by the DBMS. This functionality may come from a recoverable file system or a DBMS. To support coordinated backup and recovery with the DBMS, DLFM interfaces with a backup/archive server, such as TSM. It would backup files and if required, restore files from the backup server.

- **DLFF** (DataLinks Filesystem Filter)

  DLFF filters certain file system calls, such as file open, rename and delete. For example, the filter logic in the file open call can be used to provide database-centric access control which is rule based [IBM00]. The filter logic for operations like file rename and delete can be used to prevent such actions directly by users, if the file is referenced by the database, to preserve referential integrity, i.e., to avoid the problem of dangling pointers from the database to the file system. DLFF does *not* come in the normal file read/write path of the underlying file system and does not impact performance. The functionality of DLFF is supported without requiring changes to existing file systems in which the files pointed to by the database are stored. This is very important for the case for supporting existing file data without any migration since such data may be terabytes in size. Also, the

underlying file systems may be specialized such as a streaming media server [Hask93]. The added functions of DLFF are implemented in a layered approach using the facilities, like installable file systems, of modern file systems. If database centric access control option has been specified, then when an application interacting with the DBMS retrieves a DATALINK column value, the DBMS returns not only the server name and file name of the referenced file but also a *token* which may be embedded in the file path name. The application passes all of this information during the file system open call. The token signifies that the DBMS has allowed access to the file temporarily and DLFF validates it before allowing the user to access the file.[1] This is the mechanism by which DataLinks supports DBMS-managed access control for files.

Note that the DLFF logic is only exercised for certain file system calls. It does not play any role in coordinated backup and recovery, so we do not discuss it any further in this paper.

In the next two subsections, we describe the APIs that are used by the DBMS to interact with DLFM during normal database processing and that are relevant to the subject matter of this paper. The APIs flow along the "control path" illustrated in Figure 2.

2.1 LinkFile

The LinkFile API *links* a file and places it under the control of the DBMS. The DBMS would issue this API to DLFM when a record is inserted into a table with a DATALINK (type) column or when a record update involving a DATALINK column occurs. The file server name is determined from the server name part of the DATALINK data structure. When DLFM executes the LinkFile API, it applies certain constraints to the file referred to in the DATALINK column. Such constraints can include the following and can be combined as options which are specified on the DATALINK column when the SQL table is defined. The options are described later.

- **DB-Owner Constraint** The DBMS becomes the owner of the linked file. This constraint implies that it is the DBMS access permissions which would determine access to the file. Such an access would be based on a DBMS assigned token, as explained above.

  The user can choose to use the file system permissions rather than the DBMS permissions to support legacy applications if that is more appropriate. When the file system permissions are to be used, the DBMS would not generate a token when it returns the DATALINK column value.

- **Read-only Constraint** The linked file is marked *read-only*. Marking the file read-only guarantees the integrity of indexes that may be created on the contents of the file and stored in the DBMS for search purposes. DataLinks supports coordinated backup and recovery only if the linked file is marked read-only. Then, in order to update the file, either the file has to be in the unlinked state or a file versioning mechanism needs to be supported.[2] If only the file reference in the DATALINK column is kept in the DBMS, i.e. no indexes on the contents of the file, and if coordinated backup and recovery is not required, then the user may not want to use the DATALINK column option which results in linked files being marked read-only. Due to space limitations, support for update-in-place of a file[3] will not be discussed here.

- **Referential Integrity Constraint** metadata is maintained to indicate that the file is referenced by the DBMS. This constraint would prevent renaming or deletion of the file by a file system user once the file is referred to by the DBMS. It is applied to maintain referential integrity of the DBMS reference to the file.

---

1 The token has an expiry time which is configurable.

2 A file versioning mechanism would be a middleware function implemented between the end-user and DataLinks. It would support, for example, checkout/checkin functions for updating a file. It would use DataLinks to support updating of indexes on the file content and the DATALINK column atomically to affect the change from one file version to another.

3 With the support of update-in-place of a file, the constraint of marking the file read-only is relaxed as follows. The updates to the file are disallowed only until the previous version of the file is not backed up.

In order to support the trivial option of no referential integrity, we allow an option called “No_control” which can be specified for the DATALINK column.

Below, we describe the possible combinations of the above constraints which can be specified as options for a DATALINK column during CREATE or ALTER TABLE.

Option	DB-owner constraint	Read-only constraint	Referential Integrity Constraint
DBMS_control	Yes	Yes	Yes
File_control	No	No	Yes
File_control_read_only	No	Yes	Yes
No_control	No	No	No

Note: For No_Control, the DBMS does not issue the LinkFile API.

As part of the LinkFile API, the DBMS passes the constraints which DLFM should apply to the file which is being linked. For DBMS_control, DLFM may also save the original owner and access permissions (or access lists) of the file, so that it is possible to restore them when the file is unlinked from the DBMS. For File_control_read_only, DLFM may save the file's write permission information since it may need to be restored during unlink. Whether to save the original owner and permissions is determined by the Unlink option which can be specified as Unlink(Restore) or Unlink(Delete) for the DATALINK column. Unlink(Restore) implies that the original owner/permissions should be restored when the file is unlinked from the DBMS; Unlink(Delete) implies that the file should be deleted when it is unlinked from the DBMS. For the latter, it is not necessary to save the original owner/permissions at the time of link.

The linking operation is done in a transactional manner. This means that if the transaction performing the link were to fail, then the link operation is undone during transaction rollback. On the other hand, if the transaction were to commit, then the effects of the LinkFile API will persist, even if there are failures. DLFM persistently records the fact that the file is now pointed to by the database. For brevity, we do not describe here the details of change of ownership and permission of a linked file, and their persistence.

As mentioned above, coordinated backup and recovery can be supported for those files which belong to a DATALINK column with options DBMS_control or File_control_read_only, since the file is marked read-only during the LinkFile operation. We provide an option of specifying Recovery(Yes) or Recovery(No) to declare whether the files pointed to by an DATALINK column are important enough to require backup since backing up of files costs storage, CPU and, perhaps, network resources.

An example of a CREATE TABLE for T1 with 2 columns, C1 an integer value and C2 an DATALINK value, is given below. For the DATALINK column, we have selected the options DBMS_control, Recovery(Yes) and Unlink(Delete).

```
CREATE Table T1 (
      C1 integer,
      C2 DATALINK ( DBMS_control,
      Recovery(Yes),
      Unlink(Delete) ) );
```

An example of an SQL INSERT into table T1 which is defined above is given below.

```
hvc1 = 5;
hvurl = "http://server1/foo"; /*Setup host variable for DATALINK value*/
EXEC SQL INSERT INTO T1 (C1, C2) VALUES (:hvc1, DLVALUE(:hvurl));
```

DataLink Manager software should be installed in server1. The DBMS would communicate with DLFM on server1 and request it to link file foo transactionally. The DBMS would accept the data value in the DATALINK column of the row to be inserted only if the file foo exists and the transaction commits.[4]

In the above example, when the transaction commits, DLFM would make the DBMS the owner of the file and the file would be marked read-only. DLFM would not save the original owner and permissions of the file since Unlink(Delete) is specified. DLFM would make a persistent record that the file is linked to a DBMS. In this persistent record, DLFM saves a DBMS provided time-based recovery identifier called *RecoveryID_at_link*. This identifier is also stored in the DATALINK column by the DBMS.

Stored values of relevance here, in the DATALINK column in the DBMS for the above SQL Insert are as follows.

URL scheme	**Server Name**	**File Name**	**RecoveryID_at_link**
http	server1	foo	t1

Stored values of relevance here for the file record for file foo in server1's DLFM are as follows.

File Name	**RecoveryID _at_link**	**Linked State**	**Original Owner**	**Original Group**	**Original Permissions**	**RecoveryID _at_unlink**	**Backup Number**
foo	t1	L (linked)	-	-	-	null	0

Note: `RecoveryID_at_unlink and Backup Number are discussed later in the context of the UnlinkFile API.`

Since Recovery(Yes) is specified for the DATALINK column in our example, DLFM would make a persistent entry in a *copy-table* so that a backup copy of the file would be made soon after this transaction commits. Such a copy may be made in a batch mode by a copy-daemon in DLFM which processes many files. It is not efficient to make a copy of the file in the DBMS transaction scope since the size of the file may be large (several hundred megabytes to gigabytes). However, the persistent entry in the copy-table is made within the DBMS transaction's scope. It is possible to make the copy asynchronously because the file is marked *read-only* at link time. Recall that coordinated backup and recovery is supported for the options DBMS_control and File_control_read_only only. Of course, we have to handle the case where the file is unlinked before copying is completed. This is described in section 3.4,

4 Certain restrictions are applicable to the linking operation. A file can be linked to only one column value in the same or a different DBMS. DLFM ensures that a file is linked at most once. It is an implementation restriction for reasons of simplicity, rather than a fundamental one. If multiple references are needed from one DBMS, the application can assign a logical ID to a physical file name and use the logical ID for multiple references. We are working to support references from multiple DBMSs.

"Impact on LinkFile and UnlinkFile Operations". It is crucial that the backup operations of the copy-daemon operate efficiently and finish the file backups in a timely manner. The database backup will not succeed unless the file backups are completed. If the copy-daemon gets far behind in making the backup copies of the files, then there is an increased window during which backups could fail. These concerns can be addressed by parallelizing the functions of the copy-daemon. Since metadata is available in DLFM about the files, possibly including on which disk a file is stored, multiple copy-daemons could operate efficiently to ensure that file backups are completed as quickly as possible.

In case DLFM fails prior to completion of making a copy of the file, DLFM would start the copy-daemon during its restart recovery so that the copy would be made ultimately. The record entry (of relevance to this paper) in DLFM's copy-table would be as follows.

File Name	RecoveryID_at_link
foo	t1

The copy-daemon would delete the entry from the copy-table after the file is copied.

2.2 UnlinkFile

The UnlinkFile API lets DLFM know that the database no longer references the named file. This API will be issued when a record containing a DATALINK column is deleted or when a record update involving a DATALINK column occurs. If the options specified on the DATALINK column are DBMS_control and Unlink(Restore), then DLFM will restore the ownership of the file to the original user (i.e., the DBMS will no longer be the owner of the file) and also restore the access permissions which were recorded earlier when the LinkFile API was processed. If Unlink(Delete) option was specified, then DLFM would delete the file from the file system. Furthermore, if Recovery(No) is specified, then DLFM would also delete the persistent information it had stored earlier about the file. However, if Recovery(Yes) is specified, then in order to support coordinated, point-in-time recovery with the DBMS, DLFM, rather than deleting its information about the unlinked file, continues to retain that information persistently and notes that the file is in the unlinked state. Note that the backup server still has a copy of the file in case the file needs to be restored as part of a coordinated restore with the DBMS. If the DATALINK column options are File_control_read_only and Unlink(Restore), then the write permissions to the file are restored. The Unlink(Delete), Recovery(Yes) and Recovery(No) discussions are the same as those for DBMS_control as described above.

The UnlinkFile API also includes a parameter called RecoveryID. It is a time-based recovery identifier, assigned by the DBMS, known as RecoveryID_at_unlink. DLFM records this ID in the record for the file when it is updated to note that the file is in the unlinked state. Note that both RecoveryID_at_link and RecoveryID_at_unlink are maintained. These can be used subsequently to support coordinated recovery with the DBMS.

A file foo can be linked at time t1, then unlinked at time t2 and get deleted, then subsequently created again at time t3, and unlinked at time t4 and get deleted. Assuming that there is no garbage collection of unlinked files in between, DLFM would have the following records with their Recovery_IDs.

File Name	RecoveryID _at_link	Linked State	Original Owner	Original Group	Original Permissions	RecoveryID _at_unlink	Backup Number
foo	t1	U (unlinked)	-	-	-	t2	3
foo	t3	U	-	-	-	t4	5

Even though file foo does not exist in the file system because the user chose Unlink(Delete) option, the backup server would have 2 distinct files for foo: foo as it existed at t1 and foo as it existed at t3. Having these versions of file foo in the backup server allows coordinated recovery to be possible between DBMS and DLFM. It is possible for the DBMS to reconcile with DLFM based on the RecoveryID_at_link and/or RecoveryID_at_unlink as described in section 4.0, "Restore Operation".

It is clear that garbage collection of the unlinked files is necessary at some point in time. At the time of unlink, a backup number is assigned to the file record in the Backup Number field. How this backup number is maintained and used in the garbage collection of unlinked files is explained in section 3.2, "Efficient Garbage Collection after a Backup".

3. Backup Operation

Traditionally, database backups have been supported with two flavors [MoNa93]. If a backup is made while there are no concurrent updates, then it is referred to as an offline backup. If the backup is made while concurrent updates are allowed, then it is referred to as an online backup. DataLinks supports both types of backups. For coordinated backup between database metadata and external files, a straightforward approach could be to copy all the files which are referenced by the DATALINK fields at the time the database is backed up. However, the performance of this approach may be unacceptable. This is because of the following reasons:

- A database backup typically copies a database page at a time rather than a record at a time [MoNa93]. Reading each record at backup time to determine the DATALINK field values would cause significant performance degradation to the backup procedure. Uncommitted updates may also cause additional complications.
- There may be many files which may be large in size (e.g., many megabytes). Copying them may take more time than the time it takes to back up the database.

Hence, we need an alternate way of identifying and backing up the newly linked files.

One of the objectives of our design is that performance of the database backup should not suffer in the presence of the DATALINK columns referencing files. This objective is realized by initiating the backup of a file when the transaction referencing that file in a LinkFile operation commits. This is described in section 2.1, "LinkFile". The key point is that the backup of the file is performed asynchronous to the transaction, and it is guaranteed that the backup copy would be made.

3.1 Coordinated Backup

Initiating the asynchronous copying of files poses some challenges with respect to the backup of the database and some mainline functions. At the time the DBMS backs up the database, it needs to ensure that any asynchronous copy operations of the files referenced by this database which were started since the immediately preceding backup are complete. It is sufficient to check completion of only the copy operations initiated since the immediately preceding DBMS backup because of the following reasons:

- A file is marked read-only when it is linked to the database. That is, the file cannot be updated as long as it is linked from the database. To update a file in place[5], one has to unlink the file from the DBMS, update it and then link it again. From DLFM's standpoint, the RecoveryID_at_link makes these two versions of the file temporally unique.
- Completion of asynchronous copy operation is checked at every DBMS backup, so by induction it is sufficient to check completion of the copy operations which were initiated since the immediately preceding backup.

To accomplish the above, the DBMS issues the following APIs to the DLFM. First, we describe the parameters associated with these APIs and then their use.

BackupVerify: The purpose of this API is to verify whether the asynchronous copying of the files, which gets initiated as a result of LinkFile, is complete. This is to ensure that the DBMS's backup of data is synchronized with the file data backup. The information passed via this API is as follows:

```
BackupVerify  (dbid,
               RecoveryId sincetime,
               Int sinceTimeSupplied,
               RecoveryId curtime);

Parameters:
   dbid
      The handle which determines the database for which this API is
      invoked.
   sincetime
      Timestamp which represents the lower bound for the files whose
      LinkFile operations were performed at or after this timestamp and
      resulted in asynchronous copying of the files to be made to the
      backup server.
   SinceTimeSupplied
      This specifies whether sincetime is supplied or not.
   curtime
      Timestamp which represents the upper bound for the files whose
      LinkFile operations were performed at or before this timestamp and
      resulted in asynchronous copying of the files to be made to the
      backup server.
```

BackupEnd: The purpose of this API is to declare that the coordinated backup has completed successfully. This API is transactional and is issued only after verifying that asynchronous copy operations as requested by BackupVerify APIs are complete on all the DLFMs. The information passed via this API is as follows:

---

5 Update-in-place is supported, but is not described in this paper.

```
BackupEnd (dbid,
          TxnId txnId,
          RecoveryId backupendtime);

Parameters:
   dbid
      The handle which determines the database for which this API is
      invoked.
   txnId
      The transaction within which this API is invoked.
   backupendtime
      Timestamp which represents the end of the coordinated backup. All
      subsequent LinkFile operations should have a timestamp which is
      greater than backupendtime.
```

The BackupVerify API is issued by the DBMS to DLFM at the time the DBMS starts to backup the database. Two parameters of this call are of interest here: *SinceTime* and *CurTime*. To verify that all the asynchronous copy operations, which are set in motion as a result of LinkFile calls, which were initiated since the immediately preceding backup are complete, the DLFM uses the backupendtime of the previous backup registered in the backup table as the SinceTime since sinceTimeSupplied is false for the first BackupVerify, and the start-timestamp of the current backup or the end-of-log LSN at the time of the start of the current backup as CurTime. The set of files which satisfy the following condition would be checked to see whether they have been backed up or not:

CurTime >= RecoveryID_at_Link >= SinceTime

If all the files in this set are already backed up, then the BackupVerify call returns *success*. If not, it returns *copying_in_progress*. In the latter case, the DBMS can continue to do its processing to copy the database but when the DBMS completes copying the database, it must issue the BackupVerify API to check whether the files in the above set and any newly linked files have been copied. If it is an offline backup of the database (i.e., no updates to the database are allowed while the backup is in progress), then no new LinkFile calls would be issued while the backup is in progress and so if BackupVerify returns copying_in_progress the first time, most likely DLFM would return success on the second call. If success is not returned, then the DBMS decides on a time interval frequency at which it would continue to check whether the files have been copied or not. In the extreme case, if DLFM were to fail or some large interval of time were to expire with files in the above set still not copied, then the DBMS would fail the backup operation.[6]

For an online database backup, i.e., updates to the database are allowed while the backup is in progress, file backup operations could be initiated as a result of new LinkFile operations while the database backup operation was in progress. The DBMS must ensure that those linked files are copied as well before declaring that the coordinated backup is complete. The DBMS must issue BackupVerify on completing the backup on the database side. This is in contrast to the offline backup where BackupVerify needs to be issued a second time during a given backup only if the first BackupVerify (at the start of the backup) returned copying_in_progress.

6 This is very unlikely, since efficient ways to copy files are used, such as parallel copy operations

The SinceTime and CurTime values for this second call for online backup are set as follows:

SinceTime: For the first BackupVerify, it is set to end-timestamp or end-LSN of the previous backup. For the second BackupVerify, that value is moved forward to the timestamp/LSN taken just *before* the first BackupVerify was issued ONLY if that first BackupVerify returned success.

CurTime: Timestamp or LSN taken at the end of database backup.

BackupEnd tells DLFM that the coordinated backup succeeded. DLFM records the backupendtime and other information related to this coordinated backup within the Backup table. BackupEnd is issued by the DBMS to the various DLFMs in a transactional fashion, using the two phase commit protocol. After the coordinated backup is successful, garbage collection is initiated by DBMS on DLFMs based on the logic mentioned below.

3.2 Efficient Garbage Collection after a Backup

A garbage collection procedure on the DBMS side monitors the number of database backups that are kept valid as well as dictates when to garbage collect files that were unlinked on the various DLFMs based on the database configuration parameter, NUM_DB_BACKUPS. When database backup files are to be garbage collected, the garbage collection procedure will mark the utility history tracking, UHT, file entry for the database backup as expired. The procedure will also notify all DLFMs to garbage collect the associated files which were unlinked before this expired backup. The backup number field associated with the unlinked file entry is used to identify such files. The unlinked files would be deleted from the backup server and the corresponding DLFM metadata would be discarded. Determining which unlinked files should be deleted is performed efficiently, as described below.

DLFM tracks a backup sequence number for a database as backups of that database are taken. As each backup is taken for that database, this number is incremented. An example of DLFM's persistent Backup table is given below. The backupendtime specified as BackupEnd is also tracked in this table. The use of the backupendtime is described in 3.4, “Impact on LinkFile and UnlinkFile Operations”.

dbid	Backup Number	Time of Backup
db1	2	19
db2	3	27
....		

Figure 3. DLFM's Backup Table for DBMS's backups

At the time of an UnlinkFile operation, a backup sequence number is assigned to the referenced file as mentioned in section 2.2, “UnlinkFile”. The backup sequence number assigned with the unlink is equal to the current backup number.

The garbage collection procedure initiated on the DLFMs is asynchronous to the database backup that initiates the garbage collection. Thus it is possible that when the procedure is going on in the DLFMs, another database backup is

taken. Though unlikely, more than one database backup may be performed while garbage collection is in progress on the DLFMs. The DBMS garbage collection procedure makes sure that garbage collection is performed on the DLFMs for each of the database backups that are taken. An example follows:

Assume NUM_DB_BACKUPS is set to 2. The current database backup number is 3. Thus the garbage collection procedure initiated on the DLFMs would garbage collect all unlinked files which were unlinked prior to backup number 1. While this procedure is running, database backups 4 and 5 are complete. Each of these database backups would initiate garbage collection on the DLFMs, which would get queued and processed in sequence.

3.3 Information to be Tracked with DBMS Backups

Recall the topology where a DBMS can reference files on multiple file servers. Additional information which is tracked with a database backup includes the list of DLFMs, and the list of DATALINK columns with data referencing this DLFM. The reason this information needs to be tracked is that during the restore operation we need to validate that those DLFMs still have the meta information about the DATALINK columns of the database. The case in point is that a database may have been deleted but it is necessary to restore it from an earlier version. When a database is deleted, DLFMs do not delete DATALINK meta information immediately but keep it for some period of time which is user configurable. When a database is restored to an earlier version and a DLFM which is in the list of DLFMs which are tracked with that backup no longer has the DATALINK column defined, then the restore procedure has to resort to the reconciliation procedure. The reconcile utility is described later.

Now, the question arises as to how does the DBMS determine which DLFMs are involved since, as mentioned earlier, the database backup does not read any database record as it makes the copy of the database. This is discovered by contacting all known DLFMs as per the configuration file and then recording the ones which respond positively that they have the DATALINK metadata defined. The DATALINK metadata is defined in a DLFM when the first LinkFile (ever) is issued to that DLFM.

3.4 Impact on LinkFile and UnlinkFile Operations

It was mentioned in Section 3.1, “Coordinated Backup” that DLFM needs to verify the completion of copying of only those files which were linked since the last backup. This requires validation of RecoveryID_at_link (a timestamp and/or LSN) so that it is greater than the backupendtime. Otherwise, the verification of whether the asynchronous copy of the file is complete or not may not be performed correctly. This validation is performed by the DLFM since it tracks the BackupEndTime of the last backup, as shown in Figure 3. If the above condition is not met, then DLFM returns an error condition. The DBMS can then reassign a new RecoveryID and reissue the LinkFile operation.

In Section 2.1, “LinkFile”, we mentioned the case where a file is unlinked before it is copied to the backup server. Below, we describe how we handle this. The UnlinkFile operation has to pay attention to whether asynchronous

copying of the referenced file, which would have been initiated when the corresponding LinkFile call was issued, is complete or not. In the rare case where a database update of the DATALINK field causes the file to be unlinked and the copy of the file is not yet made, then the unlink operation is serialized by the DLFM with the backing up of the file.[7] This is necessary since the file could be deleted from the file system as a result of the unlink operation committing before the copy is made. Once the deletion is performed, copying cannot be done and a subsequent database restore operation might require that version of the file be available.

4. Restore Operation

A backup copy of the database can be used to recover the database to one of many possible consistent states:

- The consistent state could be the state when an offline backup was taken. Such a state is referred to as offline backup state, OBS.
- The consistent state could be the state at some point in the log when no update was being allowed to the database. Such a state is referred to as a quiesce point state, QPS.
- The consistent state could be to some arbitrary point-in-time state, PTS.
- The consistent state could be the state at the time of crash (i.e., the database is current) known as the current time state, CTS.

If the database does not have a DATALINK field, i.e., there are no references to external objects, e.g., files, then the restore of a database from an offline backup (OBS recovery) restores the database to a consistent state. For QPS, the restore processing applies log records to the restored version of the database from when the appropriate online copy was started to the point when the update transactions were quiesced. The database is brought to a consistent state up to when the quiesce point was established. However, when DATALINK fields are defined, references to files would be involved. Additional processing is required to synchronize references to files with respect to the restored version of the database. Later, we describe the details of synchronizing files referenced by the database for the cases OBS and QPS.

For recovery up to the current state or CTS, the DBMS applies the log to the last database backup that was restored up to the point when the database was closed after a problem like media failure (e.g., a disk crash). For point-in-time recovery or PTS, the user dictates that the DBMS may apply the log only up to some arbitrary point in the log, for example, in the case of an application corrupting the database. In the case of CTS, applying the log up to the point of closing the database after the media failure implies that the database was not in use after that point, hence the database would be brought to a consistent state. The references to the files would already be synchronized in this case (CTS). However, in the case of PTS, when the log is applied up to some arbitrary log record which is at a point earlier in the log than at the end of the log, the database may not be restored to a consistent state with respect to the

7 The parallel copy operation to copy files helps avoid such situations.

DATALINK data that is referenced by the database. The administrator may have to run the reconciliation procedure to bring the database to a consistent point. Later, we describe the reconciliation procedure.

4.1 Restore of a Database for OBS

Below, we describe the additional processing which is required to synchronize references to the files with respect to the restored version of the database. Efficient synchronization is possible because the DBMS provides DLFM a RecoveryID, at the time of establishing the link of the reference, Recovery_ID_at_Link, and another Recovery_ID at the time of unlinking the reference, Recovery_ID_at_Unlink. As mentioned before, these Recovery_IDs could be in terms of LSNs or timestamps and DLFM tracks them in its persistent storage. Recall that an unlinked file is tracked by DLFM for some number of backups until it is purged as a result of garbage collection. Therefore, when the database is restored to a consistent state, before declaring that the database has been restored, the DBMS would do the following additional processing. The DBMS would provide a Restore_Recovery_ID, which is the timestamp or the LSN of when the backup used for restore happened, to DLFMs which were involved in the backup. A DLFM would do the following processing for the files which it tracks:

- Unlink all files whose RecoveryID_at_link is greater than or equal to the restore_recovery_ID. This would unlink all files which were linked after the backup.
- Link all files whose RecoveryID_at_link is less than Restore_recovery_ID AND RecoveryID_at_unlink is greater than or equal to the restore_recovery_ID. This would relink all files which were in the linked state prior to the backup but were unlinked after the backup.

The above reconciliation between DBMS and its external references is referred to as *reconciliation with respect to RecoveryID*.[8] This is in contrast to the more elaborate reconciliation process which is performed by accessing each database record, determining the file names, and then reconciling with the appropriate DLFMs. Such a detailed reconciliation procedure may be needed when point-in-time recovery (PTS, QPS) is performed for the database, and is described later.

It should be noted that relinking a file after it was unlinked implies restoring the version of the file when it was linked. If required, DLFM would interact with the backup server, e.g., TSM to retrieve that version of the file. If the correct version of the file exists in the filesystem, then that version can be used for relinking. There are possibilities of various errors while restoring the file, such as, duplicate file name in the file system. If there are any such errors, then the database table is put in the datalink-reconcile pending state since detailed analysis of which row/column is linked to the file needs to be reported. This reconciliation process is described later. Though reconciliation with respect to RecoveryID is more efficient, the following point should be noted.

8 This procedure can only be followed if there no ambiguity of log sequence numbers, LSNs. In some DBMS's such as DB2 UDB such ambiguity can exist. Details to explain this is beyond the scope of this paper

If the database backup image used for restore is older than the limit of "NUM_DB_BACKUPS" backups, then the files which were unlinked "NUM_DB_BACKUPS" backups ago may have been garbage collected. Such a check should occur before Reconciliation with respect to RecoveryID procedure is invoked. If the above conditions are not met, then the database tables should be put in the datalink-reconcile pending state and the more detailed reconciliation process to be described next should be performed.

4.2 Restore of a Database for PTS and QPS

After restoring a database from a backup file which was made while concurrent updates were going on, log records have to be applied to the restored copy. This process is called *roll forward* [MHLPS92]. The following possibilities exist with respect to this application of the log:

- Rollforward is performed to the end of the logs

  Processing to the end of the logs ensures transactional consistency between the database references and the files in the file servers. So there is no need for any additional processing when DATALINK columns are defined in the database. This is the CTS case mentioned earlier.

- Rollforward is performed to an arbitrary point in the log, PTS or to some point in the log when no update was being allowed to the database, QPS.

  The database tables with DATALINK columns are put in the datalink-reconcile pending state at the end of database rollforward processing. Such tables are identified through the DBMS catalogs, which are consistent at the end of rollforward processing. In this case, the Reconcile utility which we describe next should be run. For QPS, *reconcile with respect to recoveryID* can be performed when the log sequence numbers, LSNs are unambiguous.[9]

The Reconcile utility performs the processing necessary to ensure that files which are referenced by the database table data are in the linked state and files which are not referenced by the database table data are in the unlinked state. Reconciliation is performed in the transactional context since files may have to be linked and unlinked in this process. The DBMS Reconcile utility examines the DATALINK columns of the restored database's tables records and collects the list of referenced file names, file server names and the (DBMS) record-IDs. The record-IDs are needed to report errors as described below. Then, this list of file names and their respective record-IDs is passed to appropriate DLFMs. DLFMs have to try to restore the corresponding files and put them in the linked state. Any other files (i.e., files not in the list passed by the DBMS) that are linked from that database, as per DLFM metadata, should be unlinked. If a DLFM cannot find in its backup server some of the files needed by the DBMS or the files cannot be restored for any variety of reasons, it returns the corresponding record-IDs for exception processing to the DBMS. The DBMS can set such DATALINK column value to NULL and report them as exceptions. By removing such entries from the database tables, the DBMS can make the table available for use. After determining the fate of the files reported in the exception list, and taking appropriate actions on the file server(s), the DATALINK column of the exception rows may be updated to reference the appropriate files.

9 In DB2 UDB log sequence numbers, LSNs can be ambiguous, and hence reconcile with respect to recoveryID is not performed, instead tables with DATALINK column are placed in the datalink-reconcile pending state.

5. Toleration of unavailability of file servers for Database Backup and Recovery operations

As mentioned earlier, Backup, Restore, Rollforward and Reconcile utilities contact DLFMs for DATALINK metadata validation and synchronization of file references. The goal is to ensure that these database utilities must succeed even though one or more DLFMs may be unavailable for a while. That way we can provide high availability for database data. Based on customer experience with the original design of DataLinks, we had to respond to their requirement of high availability by extending the original design with what we describe below.

The basic design point is that the utility operations are performed on DLFMs that are currently available and are deferred on the DLFMs that are currently unavailable by recording persistently the pending operations within the *pending operations file for utilities, POFU.* There is one POFU which is maintained per DLFM. All these utilities append pending operations to the POFU files. The unavailable DLFMs are marked blocked until the pending operations are performed on the DLFM when it becomes available. The database operation is declared successful even though some DLFMs were unavailable.

A polling process on the DBMS side referred to as the Async daemon periodically polls unavailable DLFMs for its availability. The Async daemon performs the pending operations which are recorded in the POFU file for that DLFM. The Async daemon processes these operations in sequence, deletes the file when all operations are processed successfully and unblocks the DLFM for normal operations such as link file, unlink file, etc.

The operations within the POFU can be restarted. If an operation had already been finished successfully , it will be skipped. This is needed in case the Async daemon fails while processing the file and has to start processing from the beginning of the file again. These POFU files are backed up as part of each database backup and restored (as required) by the restore utility.

Subsequent database backups are allowed to succeed, even though the previous backup operations are still pending on the unavailable DLFMs. The pending operations for the subsequent backups are also recorded in the POFU files corresponding to the unavailable DLFMs. Restore of a database from a backup that has pending operations on unavailable DLFMs is allowed to succeed, since pending operations for restore with respect to unavailable DLFMs are also recorded/appended to the POFU files. Since the operations within the POFU files are processed by the Async daemon in sequence, backup related pending operations get processed before the restore related pending operations.

Since Reconcile processing can be very time consuming, its reprocessing with respect to a DLFM is avoided. This can be achieved by tracking persistently the information about which database tables have reconcile processing pending on a DLFM. This is tracked in a *reconcile processing pending table, RPPT*, file which is maintained per DLFM. These files are processed by the reconcile utility and are deleted when no database tables have reconcile processing pending on its corresponding DLFM.

5.1 Availability Considerations for Backup

As mentioned in section 3.1, initially the Backup utility contacts all DLFMs which are configured to a database. However the DLFMs that respond positively saying that they have DATALINK meta information for this database are the ones that form the list of DLFMs having files referred to by this database. The database backup image includes the list of DLFMs and the list of DATALINK columns with data referencing this DLFM. Backup related operations performed at each of the DLFMs would be BackupVerify, BackupEnd, Garbage Collection as described in sections 3.1 and 3.2.

On the DBMS side, the status of a database backup operation is recorded persistently in the *Utility History Tracking, UHT,* file. The purpose of having UHT file entries is to enable the DBA to know whether the database backup operation is complete or not. The status is incomplete if the backup operations are pending on one or more DLFMs. Completion of pending backup operation on all these DLFMs would result in the database backup being marked as complete. The Async daemon updates the UHT file entry for the database backup after it successfully completes backup processing on a DLFM and after processing for all DLFMs is done, it updates the backup status to complete.

If a DLFM is unavailable for a long period of time based on the fact that the number of pending database backup operations as per the corresponding POFU file exceeds a configurable limit (for example, 2 * NUM_DB_BACKUPS) then the current database backup is declared to have failed.

5.2 Availability Considerations for Restore (OBS case)

Restore of a database without rollforward involves contacting maximal set of (DLFMs that are recorded in the backup image known as the required DLFMs, those configured to the database). Note that the DLFMs which are configured but not in the backup image need to be contacted to delete DATALINK metadata information and unlink files in case these files were linked after the offline backup. Restore related operations performed at each of the required DLFMs would be to verify the validity of the backup used for restore, i.e., whether the backup is older than NUM_DB_BACKUPS, and to perform *reconcile with respect to recoveryID*, to remove extra DATALINK metadata information.

If the backup image which is being restored indicates that one or more DLFMs were unavailable, then *reconcile with respect to recoveryID* is not performed on any of the DLFMs, instead tables with DATALINK data are placed in the datalink-reconcile pending state and pending operations are recorded in the POFU files to delete DATALINK metadata information which were created subsequent to the offline backup on the unavailable DLFMs.

If the backup image which is being restored indicates that all the DLFMs were available and during restore one or more required DLFMs are unavailable to verify validity of the backup or to perform *reconcile with respect to*

recoveryID, then all tables with DATALINK data referring to these DLFMs will be placed in datalink-reconcile pending state, i.e., no *reconcile with respect to recoveryID* are performed on these DLFMs, pending operations are recorded within POFU files to delete DATALINK metadata information which was created subsequent to the offline backup on the DLFMs that are unavailable.

5.3 Availability Considerations for Restore and Rollforward(PTS, QPS, CTS cases)

After restore of a database, rollforward is executed to apply log records to the restored database. As described in section 4.2, for the PTS and QPS case, at the end of rollforward processing, tables with DATALINK column are placed in datalink-reconcile pending state and all DLFMs that are configured to the database are contacted to delete the DATALINK meta information that were created subsequent to the log point for PTS, QPS. If the database backup image used for restore was taken when all the DLFMs were available and during restore one or more required DLFMs were unavailable to verify validity of the backup, then at the end of rollforward tables with DATALINK data are placed in datalink-reconcile pending state. Note that Restore cannot set the table states, since rollforward might reset those states while processing through the log. If DLFM is unavailable, pending operations are recorded within POFU files to delete DATALINK metadata information which were created subsequent to the log point for PTS, QPS.

If rolling forward is performed to the end of the log (CTS case), no DLFMs need to be contacted, since the state of the database with respect to the DLFMs will be exactly the same as it was before the restore/rollforward operations.

5.4 Availability Considerations for Reconcile

Reconcile of a table involves contacting each of the DLFMs that are referenced by the table data (required DLFMs) as well as those configured to the database, but not referenced by the table data (extra DLFMS, to delete the extra DLFM metadata information related to this table). Reconcile related operations performed at each of the required DLFMs would be to link files referred to by table data that are not in linked state on the DLFM and unlink files that are not currently referred to by table data but are in linked state on the DLFM.

If one or more of the required DLFMs are unavailable during reconcile, then reconcile processing is done on DLFMs that are available, table is placed in datalink-reconcile-dlfm-pending (DRDP) state, which implies that the table has been reconciled with respect to one or more DLFMs. This is done to avoid expensive reprocessing of reconcile. The information about the tables incomplete reconcile status gets recorded in the RPPT file for the required DLFMs that are unavailable. If one or more of the extra DLFMs are unavailable, then pending operations are recorded within the corresponding POFU files to delete extra DLFM metadata information for this table.

If reconcile is run on a table and it is in DRDP state, then reconcile processing is done on only those DLFMs that were unavailable during the previous run. The list of DLFMs that were unavailable during the previous run can be known from the RPPT files.

If a table is in the DRP or DRDP states, then no inserts/updates/deletes are allowed on such a table.[10] The table returns to normal state only after reconcile processing is successfully completed on all the required DLFMs.

6. Summary and Conclusions

For an environment where a linkage is maintained, with referential integrity, between data in a DBMS and files in a file server which is external to the DBMS, we present algorithms for performing backup and restore of the DBMS data in a coordinated fashion with the file system. We have introduced a new SQL data type in the DBMS, DATALINK, to enable a merger of DBMS and file system technologies. The object referenced in a DATALINK column can be perceived as being stored in the database for access control, referential integrity, and backup and recovery, but actually it is stored outside in a file system. DataLinks does not require changes to existing file systems which manage the files pointed to by the database. For backup, our approach makes use of a backup/archive server like TSM for storing the file backups. In order to avoid delays in the database backup operation, backup of a referenced file is initiated when the file is associated (*linked*) with a record in the DBMS. The file backup is performed asynchronously to the linking process so that the linking transaction is not delayed. No database locks are held while the backup of the referenced file is in progress. During a database backup operation, we do not require the processing of the copied records to identify what files need to be backed up on the filesystem side. This results in improved performance. It even allows for efficient copying of the database pages (e.g., by avoiding reading in the copied pages into the buffer pool of the DBMS [MoNa93]). When database backup occurs, all unfinished file backups are completed before the database backup is declared successful. When a database is restored to a state which includes references to files in a file system, the DBMS ensures, using its cooperative software on the file server, that the referenced files are also restored to their correct state. We have covered all cases when one or more file servers may be unavailable during Backup, Restore, Rollforward and Reconcile processing. We maintain transactional consistency during normal and recovery operations. The coordinated backup and recovery operations were designed to tolerate unavailable file servers so as not to hinder database availability. We provide referential integrity between the DBMS data and the file system data. Our algorithms have been implemented in the IBM DB2/DataLinks product and the DataLinks concept has become part of the SQL standard. This product is currently used in production environment, for engineering designs in large automotive and aerospace enterprises.

We are not aware of any related prior work in the literature which discusses coordinated backup and recovery between DBMSs and file systems. In the future, we would like to enhance the functionality of DataLinks so that it permits more than one reference to a given file from multiple DBMSs. We have implemented in place update of a file and provide coordinated backup and restore support without the user having to unlink the old version of the file

10 Though it may be possible to allow these operations for available DLFMs, we did not implement it for simplicity reasons.

and link in the new version with a different name. This would be described in a subsequent paper. We would like to build the middleware for checkout/checkin with versioning using DataLinks for document management and control.

7. Acknowledgments

Members of the DataLinks development team are Suparna Bhattcharya, Karen Brannon, Vitthal Gogate, Hui-I Hsiao, Joshua Hui, Inderpal Narang, Ajay Sood, Mahadevan Subramanian and Parag Tijare. Other people who have contributed to the DataLinks project are, Ashok Chandra, Lindsay Hemms, Dale McInnis, Nelson Mattos, C. Mohan, Robert Morris, Frank Pellow, Bob Rees and Stefan Steiner.

8. References

Blak96 Blakeley, J. *Data Access for the Masses through OLE DB*, Proc. ACM SIGMOD International Conference on Management of Data, Montreal, June 1996.

BlRS96 Blott, S., Relly, L., Schek, H. *An Open Abstract-Object Storage System*, **Proc**. ACM SIGMOD International Conference on Management of Data, Montreal, June 1996.

CaDNS94 Carey, M., DeWitt, D., Naughton, J., Solomon, M., et al. *Shoring Up Persistent Applications*, Proc. ACM SIGMOD Conference, Minneapolis, MN, pp. 383-394, May 1994

CaRH95 Cabrera, L.-F., Rees, R., Hineman, W. *Applying Database Technology in the ADSM Mass Storage System*, Proc. 21st International Conference on Very Large Data Bases, Zurich, September 1995.

Flic95 Flickner, M., et al. *Query by Image and Video Content: The QBIC System*, IEEE Computer, Vol. 28, No. 9, September 1995.

Hask93 Haskin, R.L., *The Shark Continuous-Media Server*, CompCon Spring '93, San Francisco, February 1993.

HsNa00 Hsiao, H., Narang, I., *DLFM: A Transactional Resource Manager*, Proc.ACM SIGMOD Conf., p 518-528, May 2000

IBM00 IBM, *DB2 Universal Database V7, Administration Guide: Controlling Access to Database Objects, 2000.*

ISO00 ISO/IEC 9075-9-2000, *Information Technology - Database Languages - SQL - Part 9: Management of External Data (SQL/MED), 2000.*

MHLSP92 Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*, ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992..

MoNa93 Mohan, C., Narang, I. *An Efficient and Flexible Method for Archiving a Data Base*, Proc. ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 1993. A corrected version of this paper is available as IBM Research Report RJ9733, IBM Almaden Research Center, March 1994.

NaRe95 Narang, I., Rees R. *DataLinks - Linkage of Database and FileSystems*, Proc. Sixth International Workshop on High Performance Transaction Systems, Asilomar, September 1995.

Obj95 Object Management Group. CORBA: *The Common Object Request Broker: Architecture and Specifications*, July 1995. Release 2.0

Ora00 Oracle Corporation: *Oracle Internet File System, Features Overview, Oracle Internet File System, Frequently Asked Questions: Technical Questions*, Oct 2000