Repeating History Beyond ARIES

C. Mohan

IBM Almaden Research Center
650 Harry Road, K01/B1
San Jose, CA 95120, USA
mohan@almaden.ibm.com
www.almaden.ibm.com/u/mohan/

Abstract

In this paper, I describe first the background behind the development of the original ARIES recovery method, and its significant impact on the commercial world and the research community. Next, I provide a brief introduction to the various concurrency control and recovery methods in the ARIES family of algorithms. Subsequently, I discuss some of the recent developments affecting the transaction management area and what these mean for the future. In ARIES, the concept of *repeating history* turned out to be an important paradigm. As I examine where transaction management is headed in the world of the internet, I observe history repeating itself in the sense of requirements that used to be considered significant in the mainframe world (e.g., performance, availability and reliability) now becoming important requirements of the broader information technology community as well.

1. Introduction

Transaction management is one of the most important functionalities provided by a database management system (DBMS). Over the years, several techniques have been developed to deal with the two most important aspects of transaction management, namely, concurrency control and recovery (CC&R). The transaction abstraction with its ACID (atomicity, consistency, isolation and durability) properties [HaRe83] has been supported by DBMSs to let users designate the scope of their atomic and isolated database interactions. Over the last two decades, several CC&R techniques have been developed and a small subset of them have been implemented in products and prototypes [BeHG87, BeNe97, Elmag92, GrRe93, JaKe97, KuHs98, Kumar95].

In this paper, I describe the background behind the invention of the ARIES (*Algorithms for Recovery and Isolation Exploiting Semantics*) family of CC&R algorithms, and the significant impact that these algorithms have had on the research community and the commercial world (section 2). In section 3, I briefly summarize some of the ARIES algorithms and discuss their implementation status. This presentation also gives a roadmap across the numerous papers that focus on ARIES and related work. The rest of this paper discusses some of the recent developments in transaction processing and distributed computing in general (section 4). Based on these observations, I speculate on what is likely to happen in the next few years (section 5).

2. Background and Impact

In the Starburst project [HCLMW90], which was started in the mid-80s at IBM's Almaden Research Center as the R* distributed DBMS project [MoLO86] was ending, we focussed on building a brand new relational DBMS with extensibility as the primary objective. Some of us treated this new project as a golden opportunity and decided to revisit many of the assumptions and conclusions of IBM's very influential System R RDBMS project [CABGK81] in the area of transaction management. We also decided to learn from the accumulated experiences with the IBM RDBMS products DB2/MVS [HaJa84] and SQL/DS [ChGY81], which, by then, had undergone customer usage for a few years. In particular, we decided to examine more closely the different approaches to database recovery adopted by these products – write-ahead logging (WAL) in DB2/MVS and shadow paging in SQL/DS.

While the System R researchers concluded [CABGK81] that WAL is better than shadow paging, they did not succeed in producing a recovery method that supported fine-granularity (e.g., record level) locking while still allowing the flexible management of variable length records as in System R. This was the primary reason behind DB2/MVS being released in 1984 with only a page as the smallest granularity of locking. Even though IBM's hierarchical DBMS IMS supported record level locking

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland. 1999.

[GaKi85, Ober80, Ober98a, Ober98b, PeSt83], it was unsatisfactory since it was doing very physical (e.g., byte-range) logging and locking that resulted in inflexible storage management and the need for frequent offline data reorganizations. We decided to aim for the best of both worlds! Since most of us were not IBMers during the System R days, we were able to look at the problems with a fresh perspective.

At a time when the research community had wrongly come to the conclusion that everything about CC&R was well understood and that there was no need for any additional work to be done, we had to be courageous to try to justify spending time on examining not only the paltry documentation but also the code of System R, SQL/DS, DB2/MVS and even IMS to try to understand everything about how their CC&R worked. Finding significant design bugs in System R ten years after the release of the product version of it (SQL/DS) was not considered, by some colleagues, to be a worthwhile exercise! In retrospect, such investigations proved to be extremely educative, insightful and invaluable. Almost all of the knowledge that we gained through these investigations have been documented extensively in the ARIES collection of papers cited here.

The original ARIES work, which was done in the mid-80s and publicly documented in a research report form in 1989 [MHLPS92], led to the establishment of the IBM Data Base Technology Institute (DBTI). This umbrella organization, which encompassed many database research and product groups within IBM, gave us, the IBM researchers, numerous opportunities to interact with IBM's DBMS customers and product developers. We were able to learn from them about important unsolved problems, and drawbacks of solutions and features implemented in different DBMS products and prototypes.

The basic ARIES algorithm summarized in Section 3.2 has been extended by us and others in numerous ways. We describe some of the extensions in the following subsections. Because of its generality and its extensive flexibility, ARIES has been implemented not only in DBMSs but also in persistent object systems, recoverable file systems, messaging and queuing systems, and transaction-based operating systems. Various approaches have been taken to formalize subsets of ARIES [Kuo96, LoTu95, MaRa97]. Extensions to ARIES have been proposed to exploit operation semantics [Billa96], provide high availability [BGHJ92] and support the client-server context [FZTCD92]. Simulation and analytical studies of ARIES's performance have been done [JhKh92, VuDo90].

Many of the algorithms summarized in this paper have been implemented to varying degrees in numerous products and research prototypes like Starburst extensible DBMS [HCLMW90], OS/2 Extended Edition Database Manager [ChMy88], DB2/390 [JMNT97], DB2 UDB for Unix, Windows and OS/2, Encina transaction processing monitor and recoverable file system, Microsoft SQL Server and NT file system, Gamma database machine [DGSBH90], EXODUS extensible DBMS [FZTCD92], Shore persistent object system [CDFHM94], Paradise GIS system, PREDATOR object-relational DBMS, MQSeries transactional messaging and queuing product [MoDi94], SQL/DS [ChGY81], ADSTAR Distributed Storage Manager (ADSM) [CaRH95], Lotus Domino/Notes R5 [Mohan99], QuickSilver distributed operating system [CMSW93], and VM Shared File System [StNC91]. There are likely to be many other implementations about which I am unaware.

ARIES has been covered in many database textbooks and tutorials (e.g., [RaCh96, Ramak98, Weihl95]). A cursory search on the web reveals that ARIES is being taught in university courses at Austin, Ben Gurion, Berkeley, Cornell, Duke, Ioannina, Maryland, Pittsburgh, Rensselaer, Seoul, Stanford, Trier and Wisconsin. At Cornell, based on earlier work at Wisconsin, a system called Mars has been developed for educational purposes. It is a recovery simulator that is used to explain and explore ARIES. It includes visualization features.

IBM obtained European patents [HLMPS94] on the basic ARIES recovery method. However, due to some fumbling by lawyers, after many years it gave up on trying to get the corresponding US patent! Subsequently, it let the European patents also lapse. IBM did obtain in the US and elsewhere patents on most of the other ARIES-related locking and recovery methods (for details, see www.almaden.ibm.com/u/mohan/aries_papers.html).

3. The ARIES Family of Algorithms

This section summarizes some of the CC&R algorithms that belong to the ARIES (*Algorithms for Recovery and Isolation Exploiting Semantics*) family. These algorithms support very high concurrency via fine-granularity locking, operation logging, efficient recovery, and flexible storage and buffer management. They relate to nested transactions, index management, hashing, cheap techniques for reducing or eliminating locking while guaranteeing consistency, fast restart recovery and interactions between query processing and concurrency control. While the recovery techniques are based on write-ahead logging, many of the concurrency control techniques that have been developed are applicable to systems using other recovery methods also (e.g., shadow paging).

3.1 Recovery Methods

There are two general approaches to recovery: the *write-ahead logging* (***WAL***) approach [Gray78, MHLPS92] and

the *shadow-page technique* [GMBLL81, MHLPS92]. WAL is the recovery method of choice in most systems, even though the shadow-page technique of System R is used in some systems, possibly in a limited form (e.g., for managing long fields or BLOBs). In WAL systems, an updated page is written back to the same disk location from which it was read. That is, *in-place updating* is done on disk. The *WAL protocol* asserts that the log records representing changes to some data must already be on stable storage *before* the changed data is allowed to replace the previous version of that data on disk.

Each log record is assigned, by the log manager, a unique *log sequence number (LSN)* at the time the record is written to the log. The LSNs are assigned in ascending sequence. Typically, they are the logical addresses of the corresponding log records [Crus84]. At times, version numbers or timestamps are also used as LSNs [Borr84, MoNP90]. On finishing the logging of an update to a page, in many systems whose recovery is based on WAL, the LSN of the log record corresponding to the *latest* update to the page is placed in a field in the page header. Hence, knowing the LSN of a page allows the system to correlate the state of the page with respect to those logged updates relating to that page. That is, at the time of recovery, given a log record, the LSN of the database page referenced in the log record and the LSN of the log record can be compared to determine unambiguously whether or not that log record's update is already reflected in that page. The buffer manager, in order to enforce the WAL protocol, uses the LSN associated with a modified page to ensure that the log has been forced to disk up to that LSN *before* it writes that page to disk.

With the shadow-page technique, as it is implemented in System R and SQL/DS, the *first* time a (logical) page is modified after a checkpoint, a new *physical* page is associated with it on disk. Later, when the page (the *current* version) is written to disk, it is written to the new location. The old physical page (the ***shadow*** version) associated with the (logical) page is not discarded until the next checkpoint. Restart recovery occurs from the *shadow* version of the page if a system failure should occur. With shadow paging, checkpoints tend to be very expensive and disruptive. This is because a checkpoint is taken only when all activities in the data manager have been quiesced to an *action-consistent* state. After quiescing, all the modified pages in the buffer pool and the log are written to disk. Then, the shadow version is discarded and the current version is also made the new shadow version. As a result of all these synchronous actions by the checkpointing process, restart recovery always happens from the internally consistent, *shadow* version of the database.

Even when the shadow-page technique is used for recovery, logging of updates is still performed. Commercially, the WAL approach has been much more widely adopted than the shadow-page technique. Very detailed comparisons between the two methods are given in [MHLPS92]. In this paper, whenever I discuss recovery methods, I assume that it is based on WAL. The concurrency protocols that I discuss are applicable also to systems that use the shadow-page technique.

In the following, I summarize the original ARIES algorithm and its variants ARIES-RRH and ARIES/NT. I also discuss the adaptation of ARIES for the shared disks and client-server environments, and for the management of semi-structured data in Lotus Domino/Notes.

3.2 ARIES

The aim of this section is to provide a brief overview of the original ARIES recovery method which was developed for the flat (i.e., unnested) transaction model [MHLPS92].

3.2.1 Logging

Like other recovery methods, ARIES also guarantees the atomicity and durability properties of transactions [HaRe83]. In order to provide these guarantees, ARIES keeps track of the changes made to the database by using a log. It implements the WAL protocol. All updates to all pages are logged. Changes to each page may be logged in a logical fashion. That is, not every byte that was changed on the page needs to be logged. ARIES uses an LSN on every database page to track the page's state. Every time a page is updated and a log record is written, the LSN of the log record is placed in the ***page_LSN*** field of the updated page. Tagging every page with an LSN allows ARIES to precisely track, for restart/media recovery purposes, the state of a page with respect to logged updates for that page.

In addition to logging, on a per-affected-page basis, update activities performed during forward (i.e., normal) processing of transactions, ARIES also logs, typically using ***compensation log records*** (***CLRs***), updates performed during partial or total rollbacks of transactions during both normal and restart *undo* processing. For example, if the original log record (***nonCLR***) described the deletion of record R10 on page P1, the CLR written during the undo of that log record would describe the insertion of R10 on P1. As a result of writing CLRs and updating the page_LSN field with the LSNs of the CLRs also, as far as recovery is concerned, the state of a page is always viewed as evolving forward, even when some original updates are being undone.

ARIES allows the support of even semantically-rich lock modes like *increment/decrement* [BaRa87] that permit multiple transactions to update the same data concurrently.

This is the kind of feature that requires a recovery method to (1) support *operation* logging (i.e., logging the quantity by which a field's value was decremented or incremented, rather than logging the before and after values of the field as in IMS), (2) avoid erroneous attempts to undo or redo some actions unnecessarily by precisely tracking the state of a page using the LSN concept, and (3) write CLRs.

Unlike in earlier recovery methods, in ARIES, CLRs have the property that they are redo-only log records. By appropriate *chaining* of the CLRs to log records written during forward processing, a bounded amount of logging is ensured during rollbacks, even in the face of repeated failures during restart recovery or of nested rollbacks.[1] This is to be contrasted with what happens in IMS [PeSt83], which may undo the same nonCLR multiple times, and in AS/400 [ClCo89], DB2/MVS V1 and NonStop SQL, which, in addition to undoing the same nonCLR multiple times, may also undo CLRs one or more times (see [MHLPS92] for examples). In the past, these have caused severe problems in real-life situations.

When the undo of a log record causes a CLR to be written, the CLR is made to point, via the ***UndoNxtLSN*** field of the CLR, to the *predecessor* of the log record being undone. The latter information is readily available since every log record, including a CLR, contains a pointer (***PrevLSN***) to the most recent preceding log record written by the same transaction. Thus, during rollback, the UndoNxtLSN field of the *most recently written CLR* keeps track of the progress of rollback. It tells the system from where to continue the rollback of the transaction, if a system failure were to interrupt the completion of the rollback or if a nested rollback were to be performed. It lets the system bypass those log records that had already been undone.

Since CLRs can describe what actions are actually performed during the undo of an original action, the undo action need not be, in terms of which page(s) is affected, the exact inverse of the action that is being compensated (i.e., ***logical undo*** is made possible). This allows very high concurrency to be supported. For example, in a B^+-tree, a key inserted on page 10 by one transaction may be moved to page 20 by another transaction *before* the key insertion is committed, as we permit in ARIES/IM [Mohan95b, MoLe92] (see [Mohan93a] for the description of ARIES/LHS which also exploits this feature). Now, if the first transaction were to roll back, then the key will be located on page 20 by retraversing the tree and deleted from there. A CLR will be written to describe the key deletion on page 20. This enables ***page-oriented redo***, which is very efficient, during restart and media recovery [MHLPS92].

3.2.2 Restart Recovery

When restarting the transaction system after an abnormal termination, recovery processing in ARIES involves making three passes (*analysis*, *redo* and *undo*) over the log. In order to make this processing efficient, periodically during normal processing, ARIES takes checkpoints. The checkpoint log records identify the transactions that are active, their states, and the addresses of their most recently written log records, and also the modified data (***dirty*** data) that is in the buffer pool. During restart recovery, ARIES first scans the log from the last checkpoint to the end of the log. During this ***analysis pass***, information about dirty data and transactions that were in progress at the time of the checkpoint is brought up to date as of the end of the log. The analysis pass, using the dirty data information, determines the starting point (***RedoLSN***) for the log scan of the immediately following redo pass. The analysis pass also determines the list of transactions to be rolled back in the undo pass. For each in-progress transaction, the LSN of the most recently written log record will also be determined.

Next, during the ***redo pass***, ARIES ***repeats history*** with respect to those updates logged on stable storage but whose effects on the database pages did not get reflected on disk before the system failure. This is done for the updates of ALL transactions, *including the updates of those transactions that had neither committed nor reached the in-doubt state of two-phase commit by the time of the crash* (i.e., even the missing updates of the so-called ***loser*** transactions are redone).

The process of repeating history essentially reestablishes the state of the database as of the time of the failure. A log record's update is redone if the affected page's page_LSN is *less than* the log record's LSN. The redo pass also obtains the locks needed to protect the uncommitted updates of those distributed transactions which will remain in the *in-doubt* (*prepared*) state [MoLO86] at the end of restart recovery. In contrast, in the recovery methods of System R [GMBLL81] and DB2 V1 [Crus84], only the missing updates of terminated and in-doubt transactions (the ***nonloser*** transactions) are redone during the redo pass. This is called the ***selective redo*** paradigm. In [MHLPS92], we show why this paradigm leads to problems when fine-granularity (i.e., smaller than page-granularity) locking is to be supported with WAL.

The next pass is the ***undo pass*** during which all loser transactions' updates are rolled back, in reverse

[1] A *nested rollback* is said to have occurred if a partial rollback were to be later followed by a total rollback or another partial rollback whose point of termination is an *earlier* point in the transaction than the point of termination of the first rollback.

chronological order, in a single sweep of the log. This is done by continually taking the maximum of the LSNs of the next log record to be processed for each of the yet-to-be-completely-undone loser transactions, until no loser transaction remains to be undone. Unlike during the redo pass, during the undo pass (as well as during normal undo), performing undos is not a conditional operation. That is, ARIES does *not* compare the page_LSN of the affected page to the LSN of the log record to decide whether or not to undo the update. Once a log record is processed for a transaction, the next record to process for that transaction is determined by looking at the PrevLSN or the UndoNxtLSN field of the record, depending on whether it is a nonCLR or a CLR, respectively.

3.2.3 Nested Top Actions

There are times when we would like some changes of a transaction to be committed irrespective of whether later on the transaction as a whole commits or not. We do need the *atomicity* property for these changes themselves. A few of the many situations where this is very useful are: for performing page splits and page deletes in indexes [Mohan95b, MoLe92], for relocating records in a hash-based storage method [Mohan93a], and for allowing out-of-current-transaction PUTs and GETs in a transactional messaging system [MoDi94]. ARIES supports this via the concept of ***nested top actions (NTAs)***. The desired effect is accomplished by writing a ***dummy CLR*** at the end of the NTA. The dummy CLR has as its UndoNxtLSN the LSN of the most recent log record written by the current transaction just before it started the NTA.. Thus, the dummy CLR lets ARIES bypass the log records of the NTA if the transaction were to be rolled back *after* the completion of the NTA..

ARIES's repeating history feature ensures that the NTA's changes would be redone, if necessary, after a system failure even though they may be changes performed by a loser transaction. If a system failure were to occur before the dummy CLR is written, then the NTA will be undone since the NTA's log records are written as undo-redo (as opposed to redo-only) log records. This provides the desired atomicity property for the NTA itself.

3.2.4 Concurrency Control

While locks are acquired on data at the desired granularity to assure *logical* consistency of the accessed data, latches[2] on pages are acquired both during forward and undo processing to assure *physical* consistency of the data, when a page is being examined. Deadlocks involving latches alone, or latches and locks are avoided by ensuring that the following rules are obeyed:

1. Restricting the number of page latches held simultaneously to 2 [MoHa94].
2. Ordering the latches hierarchically and if they are requested unconditionally then ordering the requests to obey the hierarchy restriction.
3. Avoiding requesting a lock unconditionally while holding a latch.

No locks have to be acquired during transaction rollback, thereby preventing rolling back transactions from ever getting involved in deadlocks (contrast this with what happens in System R and R* [GMBLL81, MoLO86]).

ARIES supports selective and deferred restart [Mohan93c], fuzzy image copies (archive dumps) and efficient media recovery [MoNa93], and high-concurrency lock modes (e.g., increment/decrement), which exploit the semantics of the operations and which require the ability to do operation logging. It is flexible with respect to the kinds of buffer management policies (e.g., *steal*, *no-force*, etc. [HaRe83]) that can be implemented and the characteristics of the stored data. Efficient storage management can be done for varying length objects [MoHa94]. In the interest of efficiency, page-oriented redos and, in the interest of high concurrency, logical undos are supported. Opportunities also exist for exploiting parallelism during restart recovery. Algorithms for supporting the above features are summarized in [MHLPS92] and detailed in the other cited papers. Algorithms for creating remote site backups for recovering from disasters are presented in [MoTO93].

Even though CLRs have been written by many systems for a long time, [MHLPS92] was the first paper to explain the rationale behind writing them, and to point out the numerous advantages of writing them and not undoing their updates. In [MHLPS92], besides presenting a new recovery method, by way of motivation for our work, we also describe some previously unpublished aspects of recovery in System R (e.g., how partial rollbacks are handled). That paper also shows why the following System R paradigms for logging and recovery, which were based on the shadow page technique, had to be changed in the context of WAL.

- Selective redo
- Undo pass preceding redo pass
- No logging of updates performed during transaction rollback (i.e., no writing of CLRs)
- No logging of index updates and space management information changes

[2] A latch is like a semaphore. Compared to a lock, acquiring and releasing a latch is very cheap in terms of instructions executed [MHLPS92, Mohan90a, Mohan90b]. Readers of a page acquire a share (S) latch on the page before reading it, while updaters acquire an exclusive (X) latch.

- No tracking of page state on page itself to relate it to logged updates (i.e., no LSNs on pages)

With our ARIES work, we also showed why it is very important to consider concurrency control, recovery and storage management together to produce high concurrency and high performance CC&R methods.

3.3 ARIES for Shared Disks

With multiple computer systems, there are two approaches to providing scalability in DBMSs. One is the shared disks (SD) architecture and the other is the shared nothing (SN) architecture. SN has been implemented in Tandem's NonStop SQL, NCR's Teradata DBMS and IBM's DB2 Parallel Edition. SD has been implemented in IBM's IMS, DEC's Rdb, Oracle and, more recently, in IBM's DB2/390. The introduction of record level locking and support for SD was done in the same release of DB2/390. This necessitated enhancements to ARIES to deal with the fact that multiple instances of DB2, each with its own buffer pool, had concurrent read and write accesses to the same set of data on the shared disks. To make matters even more interesting, in addition to the shared disks, as in previous systems, in the S/390 Parallel Sysplex environment, we also had to deal with a page-addressable store (called *Coupling Facility*) that is shared by the S/390 machines running DB2 [IBM97].

As we designed for the SD environment, the hardware and software environment that we had to deal with kept changing: from centralized lock manager to distributed lock manager, from special-purpose hardware to general-purpose hardware running specialized software, from a software-only global lock manager to a hybrid lock manager, and so on. Our papers [JMNT97, MoNa91, MoNa92a, MoNa92b, MoNP90] document some of the alternatives in this environment for locking, logging, recovery, etc. What was implemented is described in [JMNT97]. Each DB2 instance writes its log records to its own local log, but the local logs are asynchronously merged for media recovery purposes [MoNa93]. The failure possibilities here are much more complex than in a single system environment.

3.4 ARIES/CSA

In the typical client-server environment, as exemplified by the object-oriented DBMSs, the client DBMS software directly operates on the database pages even though the disks containing the database are managed by only the server. The server ships the database pages to the clients and handles global locking across clients. Clients might cache pages across transaction commits. ARIES/CSA (*ARIES for the Client Server Architecture*) [MoNa94] supports such an environment. Here, the clients produce log records when they perform their updates and send them to the server at appropriate times. They generate LSNs locally rather than letting the log manager assign them. The server manages the log disk. It writes into a single log the log records received from the different clients. In many such ways, the CS environment differs from the SD environment of the last section in which the sharing systems have a peer-peer relationship. While we did not implement ARIES/CSA, a different version of ARIES designed for the client server environment has been implemented in EXODUS [FZTCD92].

3.5 ARIES for Semi-Structured Data

Since its first release in 1989, long before the topic became fashionable in the database and web research communities, Lotus Notes had been targeted for the management of semi-structured data. A few years ago, Notes was enabled for the internet. At that time, the product name was changed so that *Domino* represents the server and *Notes* the client. Database functionality is almost identical in Domino and Notes. Through the joint efforts of Lotus's subsidiary Iris Associates and IBM Almaden's Dominotes project, one of the major features implemented in the latest release (R5) of Lotus Domino/Notes is a traditional DBMS-style, log-based recovery scheme [Mohan99]. Since Notes had not been designed originally with this type of recovery in mind, accomplishing this required significant design work. Enhancements had to be made to ARIES to deal with the fact that storage management in Notes is done in an unconventional way, as described below.

A Notes database in its entirety is stored in a single operating system file in a location and machine architecture independent format. Some of the data structures in the file are paginated while others are just byte-streams. Over time, these data structures might also be moved around in arbitrary ways. Since some of the data structures might contain attachments like audio, video, etc., logging had to be made optional at the data structure level also. At the granularity of a database, logging can be turned on or off by the user. Notes users also frequently move or replicate databases by doing file copying via the operating system. This can cause a logged version of a database to be overlaid with an older or newer version of that database from another system. Accommodating all these complications has required changes to the analysis and redo passes of ARIES. For example, the modified analysis pass gathers some extra information that is used during the redo pass to skip processing some log records whose LSNs might have normally been compared with LSNs on corresponding database pages. In the future, we will write a paper describing the resulting variant of ARIES called ARIES/SSD (ARIES for Semi-Structured Data).

3.6 ARIES-RRH

ARIES-RRH (*Algorithm for Recovery and Isolation Exploiting Semantics with Restricted Repeating of History*) [MoPi91] is an enhanced version of the original ARIES recovery method. The ARIES-RRH enhancements relate to the amount of redo of updates that needs to be performed at the time of system restart in order to bring the database to a consistent state. They try to minimize the extent of repeating of history that needs to be performed.

As described earlier, the *repeating history* paradigm of ARIES includes redoing the missing updates of even those transactions that are to be rolled back later in the undo pass of restart. The latter may lead to some wasted work being done. We illustrated in the ARIES paper why repeating history was required to support fine-granularity (e.g., record) locking. [MoPi91] further analyzed this paradigm and proposed more efficient handling of redos, especially when the smallest granularity of locking is *not* less than a page, by combining the paradigm of *selective redo* from DB2/MVS V1 [Crus84] with the original ARIES algorithm. Even for data for which fine-granularity locking is being done, it is not always the case that all the unapplied but logged changes need to be redone. ARIES-RRH, which incorporates these changes, still retains all the good properties of ARIES - avoiding undo of CLRs, single pass media recovery, NTAs, etc. The ARIES-RRH enhancements should result in a reduction in the number of I/Os and in the amount of CPU processing during the redo and undo passes of restart. This should improve the availability of the system by allowing processing of new transactions to begin earlier than with the original ARIES algorithm [Mohan93c].

ARIES-RRH requires that, for each page, *all* updates logged at least up to the point of the *most recent* committed[3] or *in-doubt* update for that page be redone, if those updates are not already present in the page. The latter is as usual determined by comparing the LSN of the page with the LSNs of the relevant log records. For data for which page or coarser granularity of locking is being used, this rule implies that all loser transactions' logged but missing updates need not be redone, as was the case with DB2 V1 [Crus84]. It turns out that following this rule alone is not sufficient since some loser transaction might have already been rolling back when a system failure happened and as a result some CLRs might have been written which survived the system failure. Some of the pages affected by those CLRs' updates might not have been written back to disk after those undos were performed. Since those pages might have been written to disk after the *original* updates (i.e., the ones which the CLRs compensated) were performed on them, we need to ensure that the corresponding CLRs' updates are also redone, even though they belong to a loser transaction and they may not be followed by any nonloser transactions' updates for the affected pages. To be able to figure out when such a condition is true, given a CLR and the page affected by it, if the page's LSN is *less than* the CLR's LSN, then we need to know if the page contains the original log record's (nonCLR's) update. Comparing the UndoNxtLSN of the CLR with the LSN of the page is not sufficient for this purpose since the page LSN being greater than UndoNxtLSN does not necessarily mean that the original update is present in the page (see [MoPi91] for an example). What is needed is the LSN of the original (nonCLR) log record. So, the contents of a CLR are enhanced to also include a field called ***UndoneLSN*** which is the LSN of the log record which the CLR compensated.

Now, the rule for handling a CLR can be stated as follows: The update of a CLR must be redone if the LSN of the affected page is *greater than or equal to* the UndoneLSN of the CLR and is *less than* the LSN of the CLR. It should be noted that a rule like this was not needed in DB2 V1 since (1) DB2's recovery method performed the undo of CLRs' updates and (2) CLRs did not have the UndoNxtLSN pointer and hence DB2 did not bypass processing of already undone nonCLR log records.

Since, with ARIES-RRH, history is not being completely repeated, the handling of undos also needs to be changed to be a *conditional* one like in DB2 V1. That is, during undo, when a nonCLR is encountered, the undo of that log record's update should be performed only if the page's LSN is *greater than or equal to* the log record's LSN. A surprising requirement is that, irrespective of whether the undo has to performed or not, a CLR must always be written as if the undo was performed (see [MoPi91] for the explanation of why this is the case).

With the flexibility offered via operation logging and the support for semantically-rich modes of locking by ARIES and ARIES-RRH, this is the best that can be done in terms of reducing the extent of repeating of history for loser transactions' updates. Of course, if only physical logging and locking are supported (as in IMS), then the missing updates of loser transactions for a given page that even *precede* the updates of nonloser transactions for the same page need not be redone. ARIES-RRH does not compromise on the original ARIES algorithm's properties of never undoing a CLR's updates and never undoing the same nonCLR's updates more than once. [MoPi91] also explains the fundamental reasons behind why certain existing recovery algorithms work correctly in the face of

[3] Conceptually, we treat a transaction which terminated after rolling back completely as a transaction which performed a partial rollback to its beginning and then committed.

failures during restart recovery or during media recovery. The ARIES-RRH work has led to a better understanding of the fundamental interactions between concurrency control and recovery methods. So far ARIES-RRH has not been implemented.

3.7 ARIES/NT

ARIES/NT (*Algorithm for Recovery and Isolation Exploiting Semantics for Nested Transactions*) [RoMo89] is an extension of the ARIES algorithm which was originally designed for the single-level transaction model. ARIES/NT applies to a very general model of nested transactions [HaRo87, HaRo93], which includes partial rollbacks of subtransactions, upward and downward inheritance of locks, and concurrent execution of ancestor and descendent subtransactions. The adopted system architecture encompasses aspects of distributed database management also.

We will briefly summarize here the extensions that were made to the original ARIES recovery method to obtain ARIES/NT. In both ARIES and ARIES/NT, all log records written by the same transaction are linked via a so-called ***backward chain (BW-chain)*** using the PrevLSN pointers. In addition, in ARIES/NT, the BW-chains of *committed* subtransactions are linked to the BW-chains of their parents to reflect the transaction trees on the log. When a subtransaction T commits, a *c-committed* log record, which contains a pointer to the last record of T's BW-chain, is written to the BW-chain of T's parent. Consequently, the BW-chain of an *in-progress* transaction together with the chains of its *committed* inferiors form a tree structure, which is called the transaction's ***backward chain tree (BWC-tree)***. Since the parent/child relationships of committed subtransactions are stored on the log, subtransactions can be forgotten after their commit. The analysis pass need not collect data about committed subtransactions, thereby simplifying recovery.

Because our very general model of nested transactions allows upward and downward inheritance of locks, and concurrent execution of ancestor and descendent subtransactions, when a (sub)transaction is to be rolled back, the actions of that (sub)transaction and its (committed or active) inferiors must be rolled back in reverse chronological order. Like ARIES, ARIES/NT logs updates performed during rollback by means of CLRs. A CLR is also used to keep track how much of a (sub)transaction *and* its committed inferiors has already been rolled back, and how much more remains to be undone. This is achieved by recording in a CLR a *set of pointers*, each of which points to the next log record to be processed in the BW-chain of the (sub)transaction or a committed inferior during undo.

As in ARIES, in ARIES/NT also, restart processing starts with an analysis pass, continues with a redo pass and ends with an undo pass. Redo processing of ARIES/NT works in exactly the same way as in ARIES, while the algorithms of the analysis and undo passes have been modified to support tree-structured log contents. In ARIES/NT, the UndoNxtLSN field of a CLR contains a set of log addresses rather than a single LSN as in the original ARIES algorithm.

So far ARIES/NT has not been implemented. Basic features of ARIES/NT have been adapted in [Lomet92] to support recovery in multi-level systems. [Dombr95] presents modifications to ARIES/NT to support advanced transactions.

3.8 Index Management

Even though concurrency in search structures (e.g., B^+-tree indexes) had been discussed frequently in the literature, the problem of providing recovery from transaction and system failures when transactions consist of multiple search structure operations received very little attention until the late 80s. [MoLe92], in its original research report form, was the first paper to provide a comprehensive treatment of concurrency control and recovery for index management in transaction systems. [Mohan90a] was the first paper to document in detail the System R key-value locking algorithms and to explain the rationale behind their design features. That paper also enhanced those algorithms to vastly improve their concurrency and performance characteristics. In spite of these efforts and publications by a few others (e.g., [LoSa92]), index CC&R are not well understood by the research community. They are not taught sufficiently in database courses or discussed enough in database textbooks.

In this section, I summarize the two algorithms, ARIES/KVL [Mohan90a] and ARIES/IM [MoLe92], that we developed. A transaction may perform any number of nonindex and index operations, including range scans. Both serializable (*repeatable read*) and, optionally, nonserializable (*cursor stability*) executions of transactions are supported. To present them, I assume a tree architecture in which all the indexes on the data (e.g., a relational table) contain only the key values and record identifiers (RIDs) of records containing those key values. The ***RID*** of a record identifies the record's location in a set of data pages. All the leaf pages of an index contain ***index entries*** in the form of ***key-value,RID*** pairs. In most systems, when a nonunique index contains duplicate instances of a key value, the key value is stored *only once in each leaf page* where it appears. The single value is followed by as many RIDs as would fit on that page.

In ARIES/KVL, the object of locking is a key value, whereas in ARIES/IM, it is the individual index entry. This should make a difference only in the case of nonunique indexes. Apart from that difference, ARIES/IM does what is called ***data-only locking***. That is, an index entry is locked by locking the underlying data whose key is the one in the index entry to be locked. This means that if record locking is being done, then the lock will be on the RID; with page locking, it will be on the pageID part of the RID. In contrast, in ARIES/KVL, the index locks are different from the data locks. There are some performance and concurrency tradeoffs involved in choosing between these two approaches (see [Mohan95b, MoLe92] for detailed comparisons). ARIES/KVL's ***index-specific key value locking*** would be necessary where the records are stored in the index itself and an index entry contains the corresponding record, instead of a RID, as in NonStop SQL. It could also potentially lead to higher concurrency compared to the data-only locking feature of ARIES/IM, but with an increase in the locking overhead. It is possible to retain ARIES/IM's idea of locking individual index entries and still perform index-specific locking by taking the lock name as obtained in the case of data-only locking and prefixing that lock name with the index ID to make it specific to this index entry, as explained in [Mohan95b, MoLe92].

There are many problems involved in supporting recoverable, concurrent modifications to an index tree. Some of the questions to be answered are:

1. How to log the changes to the index so that, during recovery after a system failure, the missing updates can be reapplied efficiently?
2. If an ***SMO*** (**structure modification operation** - page split/deletion operation) were to be in progress at the time of a system failure and some of the effects of that SMO had already been reflected in the disk version of the database, how to ensure the restoration of the structural consistency of the tree during restart?
3. How to update index pages with minimal interference to concurrent accessors of the tree?
4. If a transaction were to roll back after successfully completing an SMO, how to ensure that it does not undo the SMO, since doing so might result in the loss of some updates performed by other transactions in the intervening period to the pages affected by the SMO?
5. How to detect that a key that had been inserted by a transaction T1 in page P1 had been moved, by a subsequent SMO by T2, to P2 so that if T1 were to roll back, then P2 is accessed and the key is deleted?
6. How to detect that a key that had been deleted by T1 from P1 no longer belongs on P1 but only on P2 due to subsequent SMOs by other transactions, so that if T1 were to roll back, then P2 is accessed and the key is inserted in it?
7. How to avoid a deadlock involving a transaction that is rolling back so that no special logic is needed to handle a deadlock involving only rolling back transactions?
8. How to support different granularities of locking and what to designate as the objects of locking?
9. How to lock the *not found* condition efficiently to guarantee repeatable read (i.e., the *phantom problem* - see [EGLT76])?
10. How to guarantee that in a unique index if a key value were to be deleted by one transaction, then no other transaction is permitted to insert the same key value before the former transaction commits?
11. How to let tree traversals go on even as an SMO is in progress and still ensure that the traversing transactions are able to recover if they run into the effects of the SMO that is still in progress?

3.8.1 ARIES/KVL

ARIES/KVL (*Algorithm for Recovery and Isolation Exploiting Semantics using Key-Value Locking*) [Mohan90a] is a method for concurrency control in B^+-tree indexes. The concurrent executions permitted by the locking protocols are correct logging and recovery are made possible. ARIES/KVL supports very high concurrency during tree traversals, structure modifications, and other operations. Unlike in System R, in ARIES/KVL, when one transaction is waiting for a lock on a key value in a given index page, reads and modifications of that page by other transactions are allowed. Further, transactions that are rolling back will never get into deadlocks. ARIES/KVL's locking rules differ depending on whether the index is a unique index or a nonunique index. Compared to System R, ARIES/KVL, by also using for key value locking the IX and SIX lock modes which were intended originally for table level locking, is able to exploit the semantics of the operations to improve concurrency. These techniques are also applicable to the concurrency control of links-based storage and access structures.

During a key lookup (Fetch) call, even if the requested key value is not found, the next key value is locked to make sure that the requested key does not suddenly appear (due to an insert by another transaction) before the current transaction *terminates* and prevent repeatable read from being possible. For the protection of the reader, if the value being inserted in not already present in the index, then an inserting transaction has to *check*, via an *instant* lock call, the lock on the next higher key value. Thus, a lock on a key value is really a ***range lock*** on the range of keys spanning the values from the *preceding* key value that is

currently present in the index to the locked key value. For this range-locking protocol to work, the inserting transaction must check the lock on the *next key value*, *before* it does the insert of a given key value. The mode of Insert's next key value lock request must be such that it is incompatible with the S lock acquired by Fetch.

Since the deletion of the only instance of a certain key value would result in the key value disappearing from the index, a way is needed to communicate to readers (and inserters in the case of a unique index) the existence of an uncommitted deletion of that key value. By convention, under these conditions, the deleting transaction acquires a lock on the next key value. This is another reason why a reader needs to check the next key value lock when Fetch does not find the requested key value. The mode of Delete's next key value lock request must be such that it is incompatible with the S lock acquired by Fetch. Whereas the next key lock during an insert is only a check (instant lock), the one during deletion must be a lock which is held until commit.

Some of our ARIES/KVL enhancements over the original System R index concurrency control algorithms have been implemented in SQL/DS and the VM Shared File System.

3.8.2 ARIES/IM

ARIES/IM (*Algorithm for Recovery and Isolation Exploiting Semantics for Index Management*) [MoLe92] is a method for controlling concurrency and logging changes to index data stored in B^+-trees. ARIES/IM's recovery features are based on ARIES. ARIES/IM supports very high concurrency by

1. not locking the index data per se (i.e., keys),
2. locking the underlying record data in data pages only (e.g., at the record level),
3. not acquiring commit duration locks on index pages even during index structure modification operations (SMOs) like page splits and page deletions,
4. allowing retrievals, inserts, and deletes to go on concurrently with even an SMO, and
5. optionally, supporting degree 2 consistency of locking (cursor stability).

Even if a transaction which performed an SMO were to roll back, if all the effects of the SMO had been propagated successfully up the tree before the rollback is initiated, then the SMO is not undone. This is accomplished by doing the following:

1. Performing the SMO as an NTA.
2. If an insert requires a page split, all the actions relating to that split (the leaf-level actions, the propagation up the tree and the writing of the dummy CLR) are completed *before* the insert which necessitated the split is performed.
3. If the deletion of a key necessitates a page deletion (because the page became empty), the key deletion is first performed and logged and then all the actions relating to that page deletion are completed. The dummy CLR will point to the key deletion log record.

If the transaction were to rollback after completing the SMO, the dummy CLR lets it bypass the log records relating to the SMO. At the same time, it is ensured that the insert/delete operation causing the SMO is undone, on a rollback.

To restore the structural consistency of the tree, partially completed SMOs are undone in a page-oriented fashion. At the time of restart recovery, no special processing is performed to determine which indexes are structurally inconsistent. There is no special handling of such indexes.

During restart, any necessary redos of the index changes are always performed in a page-oriented fashion (i.e., without traversing the index tree) and, during normal processing and restart, undos are performed in a page-oriented fashion whenever possible. The protocols used during normal processing are such that if a system failure were to occur any time, then, during the subsequent restart, any incomplete SMO would be undone and thereby the structural consistency of the tree would be restored, *before* any necessary logical undo is attempted. This is done without resorting to any special restart processing.

Most of the ARIES/IM features were first implemented in the OS/2 Extended Edition Database Manager [ChMy88], which in its far enhanced form is now called DB2/UDB for Windows, Unix and OS/2. It was for that product that ARIES/IM was designed originally. Since the concurrency control techniques of ARIES/IM have general applicability, some of those techniques have also been incorporated in SQL/DS and the VM Shared File System even though those systems are based on System R which uses the shadow-page technique for recovery. ARIES/IM supports page-oriented media recovery for indexes - i.e., dumps of indexes can be taken and when there is a problem in reading a page (because, e.g., a crash had occurred when that page was being written [Mohan95a]), the page can be loaded from the last dump and then, by rolling forward using the log, the page can be brought up-to-date. Details concerning media recovery, deferred restart, etc. are presented in [MHLPS92].

Since ARIES/IM is able to handle deletion of empty pages, performing the merge of partially filled leaf pages requires only simple extensions to our method. ARIES/IM has been extended and implemented to handle the shared disks (*data sharing*, in the IMS terminology [PeSt83])

environment [JMNT97, MoNa91, MoNa92a, MoNa92b, MoNP90], in which multiple instances of DB2/390 access and modify the same database. We have developed ways to improve concurrency even further by reducing the negative implications of next index entry locking. This was done for DB2/390 by doing logical, rather than physical, deletion of keys (see [Mohan90b] for an outline of our solution). ARIES/IM and KVL ideas in conjunction with logical key deletions have been adapted in [KoMH97] for use with a generalized search tree (GiST). Adaptation of the ARIES/IM and KVL ideas to a hierarchy of indexes in a distributed database context is presented in [ChMo96]. The concurrency control implications of using multiple indexes in accessing a single table's records are presented in [MHWC90, Mohan90b, Mohan92a].

3.9 ARIES/LHS

Even though extendible hashing has been studied for a long time, very little has been reported in the literature on the concurrency control of multiple transactions simultaneously accessing such structures. Whatever little has appeared is usually based on a very simplified notion of a transaction. Generally, each transaction is assumed to consist of only one action (insert, delete, or retrieval) against the search structure.

The problems associated with guaranteeing serializability become much more complicated when one considers transactions consisting of multiple actions. Some papers deal with only extendible hashing, rather than the more complicated linear hashing. In any case, none of the papers deals with the problem of providing recovery from transaction and system failures for a general model of transactions with fine-granularity locking. Some of the concurrent activities permitted by the algorithms in the literature will cause inconsistencies when one considers failures and recovery. The interactions between concurrency control and logging (and recovery) with multiaction transactions are quite subtle. ARIES/LHS (*ARIES for Linear Hashing with Separators*) [Mohan93a] deals with the concurrency control and recovery aspects of multiaction transactions accessing dynamic hashing-based storage structures.

Larson proposed a dynamic hashing algorithm called *Linear Hashing with Separators* (LHS) that, given a unique primary key value, uses a table in memory to allow the retrieval of the corresponding record in the file in one page access to secondary storage [Lars88]. Larson considers LHS to be the first practical method offering one-access retrieval for large dynamic files. He did not discuss the impact of concurrent operations by different users, some of whom are reading the file while others are performing operations like inserts, deletes, updates, file expansions or file contractions which can cause relocations of records. ARIES/LHS is a method for controlling such concurrent operations with fine-granularity (e.g., record) locking, while guaranteeing serializability. ARIES/LHS prevents rolling back transactions from getting involved in deadlocks. It also includes recovery techniques for handling transaction and system failures, while allowing multiple operations in each transaction. To provide high concurrency and efficient recovery using write-ahead logging, ARIES/LHS exploits the power of the ARIES recovery method (e.g., the concept of NTAs and the ability to support logical undos). The impact of the LHS storage method on range queries and prefetching of data is discussed in [Mohan93a]. ARIES/LHS handles varying length records and updates of records also. So far ARIES/LHS has not been implemented.

3.10 Commit_LSN

Fine-granularity (e.g., record) locking is very helpful in increasing the level of concurrency that can be supported by reducing contention amongst transactions for access to data. The drawback of fine-granularity locking is that for those transactions that access large number of records, the number of locks that need to be acquired may increase dramatically compared to the situation with, for example, page locking. If, for those transactions which only need to determine that some piece of data is in the committed state the system could somehow avoid locking, then we can have the benefits of fine-granularity locking for transactions which access few records and at the same time avoid the drawbacks of such a locking granularity for transactions that access numerous records for reading. A method for avoiding locking is expected to be useful very often since in most databases, at any given time, most of the data is in the committed state.

The Commit_LSN method proposed in [Mohan90b] is one such idea. It is a novel and simple method for determining if a piece of data is in the committed state in a transaction processing system. This method is a much cheaper alternative to the locking approach used in the past for this purpose. The method takes advantage of the LSN concept. As described before, in transaction systems using WAL, an LSN is recorded in each page of the database to relate the state of the page to the log of update actions for that page.

The crux of the Commit_LSN method is to use this LSN information and information about the currently active update transactions to come to some conclusions about whether or not all the data on a given page is in the committed state, *without resorting to locking*. This is done by comparing the page's LSN with the information about the *oldest* update transaction still executing in the system. The crucial fact that makes our method accomplish its objectives is that no page with an LSN value that is *less than* the LSN (call it *Commit_LSN*) of the

Begin_Transaction log record of the *oldest* executing update transaction could have any uncommitted data. The Commit_LSN method applies whether the lowest granularity of locking is a page or something finer than that (e.g., record).

This simple new method reduces locking and latching. In addition, the method may also increase the level of concurrency that could be supported. It also benefits update transactions by reducing the cost of fine-granularity locking when contention is not present for data on a page. Many non-trivial applications of this method are discussed in detail in [Mohan90b]. In order to apply the Commit_LSN method, extensions have been proposed for those systems in which (1) LSNs are not associated with pages (AS/400, SQL/DS, System R), (2) LSNs are used only partially (IMS), and/or (3) not all objects' changes are logged (AS/400, SQL/DS, System R).

The Commit_LSN method's steps at the time of a page access are:

1. Find out Commit_LSN from the recovery manager or access it in shared storage. Note that it is not *necessary* for the transaction to obtain the latest value of Commit_LSN before every page access, as long as it is done at least once before the first page access. While an out of date Commit_LSN does not cause any inconsistencies, it may increase the number of times locks have to be obtained.
2. Latch the page in share (S) mode.
3. If page_LSN < Commit_LSN, then conclude that all data on the page is in the committed state; otherwise, do locking as usual and determine whether data of interest is committed or not.

Instead of having one **global** Commit_LSN that covers all objects, transactions can benefit further by computing an **object-specific Commit_LSN** that is specific to the object (e.g., file or table) to be accessed. In this way, a long-running update transaction that accesses some other objects and keeps the global Commit_LSN quite a bit in the past will not unduly restrict the applicability of the Commit_LSN method to the object of interest.

The Commit_LSN concept has turned out to be very useful in practice. It has been fully implemented in DB2/390 and partially in DB2/UDB for Windows, Unix and OS/2. Its performance advantages have been especially beneficial in the shared disks context of DB2/390. It has been exploited for providing fast restart capabilities in [Mohan93c]. Processing of new transactions can be done even while the redo and undo passes of restart recovery are in progress.

3.11 Query Processing and Locking Concerns

Traditionally, starting from the System R days, work on query processing has generally ignored considerations relating to concurrency control in making query execution choices during query optimization. Typically, concurrency control related actions are taken by the data manager (the RSS component in the case of System R) and the query optimization related actions are taken by the upper parts of the system (RDS component in the case of System R). In [Mohan92a], I have given numerous examples to illustrate why it is important to consider locking related issues while planning query executions. While it is sometimes merely a performance advantage to take such an integrated view, at other times even the correctness of query executions depends on such an approach. Some of the issues to consider are: isolation levels (repeatable read, dirty read, cursor stability), access path selection (table scan, index scan, index AND/ORing [MHWC90]), Commit_LSN optimization [Mohan90b], locking granularity (record, page, table), and high concurrency as a query optimization criterion. Our ideas are implemented in the DB2 family.

4. Transactions in the Internet Age[4]

In the last few years, there have been many significant developments in the transaction processing (TP) and distributed computing (DC) areas. Many areas of computing have influenced the recent trends in TP and DC: client-server computing, database management, object-oriented programming, groupware, internet and processor architectures, to name a few. The emergence of the worldwide web and Java has also had a dramatic influence on TP and DC.

Opening up the information resources of enterprises to customers and business partners for internet (web) access has changed the data access patterns of the DBMSs and file systems storing such information. This change has dramatically increased the requirements on TP systems with regard to attributes like availability, reliability, performance and ease of use. In this sense, history is repeating itself! What used to be considered high-end requirements in the context of mainframe computing by large enterprises are now becoming the requirements of small and medium enterprises also when they choose to web-enable their TP applications. Globalization and mergers of enterprises are also important driving factors.

Permitting data access from heterogeneous hardware and software environments has become a necessity. Legacy TP systems like CICS and IMS have been internet enabled.

[4] The slides of a long talk that expands on this section can be found at www.almaden.ibm.com/u/mohan/tp_dc.pdf

Windows NT has had to provide support for IBM SNA network protocols for distributed program to program communication and two-phase commit of distributed transactions involving mainframe and PC environments. With web enablement, enterprises are now able to provide better customer service and also reduce costs by eliminating certain intermediaries (e.g., distributors, call center operators) who are necessary in traditional ways of doing business. Network-centric computing is now a reality and, to remain competitive, organizations have to adapt their information systems to support it. When an enterprise's customers directly interface to that enterprise's TP systems their performance and usability expectations are more demanding than when they go through other trained people (e.g., customer service representatives) in that enterprise to get some services. Web-enabling a TP application is not merely an issue of purchasing the appropriate web gateway software. Some basic aspects of the TP application might have to be redesigned.

Electronic commerce (e-commerce) is taking off, especially with respect to business to business (B2B) interactions more than business to consumer (B2C) transactions. Of course, opening up traditional TP systems to doing full blown e-commerce, which would potentially involve performing multiple database updates originating from browsers as part of a single distributed transaction, as opposed to only information retrieval style accesses across the web, requires addressing a number of issues relating to security, client failures, payment systems, etc. Advanced transaction models like sagas, flex transactions, etc. [Elmag92] will have a key role to play. It is my belief that product-level support for such concepts will finally appear in the next few years in workflow management systems, rather than as extensions to traditional TP monitors.

While 2 tier distributed computing (client-server) was quite popular a few years ago, of late, 3 tier computing (caused by the addition of some middleware software running in a mid-tier machine) is being embraced more widely. Enterprises have become disillusioned with the difficulties and costs involved in realizing the often-trumpeted major benefits (e.g., cost reductions) of client-server computing with exclusively (non-mainframe) Unix/PC-based servers. This has resulted in the resurgence of the mainframe and the emergence of the concept *of server consolidations*. The latter refers to the replacement of a large collection of Unix/PC servers with a cluster consisting of a small number of CMOS-based, air-cooled mainframes like, for example, the IBM S/390. IMS, CICS and DB2 have been enhanced to support the shared disks S/390 cluster environment with valuable features like workload balancing and single system image [IBM97]. Modifying those systems to support the clustered environment, with a coupling facility (an intelligent shared store with sophisticated capabilities) in the midst of the sharing systems, has required the development of several innovative solutions for problems relating to global lock management, buffer coherency, logging, recovery and performance [JMNT97].

Asynchronous program to program communication in the form of transaction-based persistent messaging has become quite popular. IBM's MQSeries, for example, is a very successful product in this arena. MQSeries includes its own non-DBMS-based persistent storage mechanism, using an extended version of ARIES, for managing the messages [MoDi94]. Oracle, on the other hand, has recently introduced some messaging functionality directly in its RDBMS itself so that the messages are also managed by the Oracle DBMS. Such an approach requires enhancing the concurrency control protocols and isolation level support of the DBMS to meet the different consistency and performance requirements of a transactional messaging system. This is an area to which the research community has not paid enough attention. Concepts like publish-subscribe are currently very popular in the commercial world and they deserve to be researched.

In general, TP and DC standardization activities have become more widespread. The belief (and hope!) is that object technology is the right approach for improving software productivity and for reusing/integrating existing legacy TP applications by adding OO wrappers to those applications. Recently, many commercial implementations of OMG's Object Transaction Services (OTS) have become available. It is not yet clear how widely and quickly such products will be used in production TP applications and what their performance characteristics would be in comparison with applications built using traditional, procedural TP technologies. More recently, OTS has been extended to the Java world via Java Transaction Services (JTS). Enterprise Java Beans have also been proposed as a way of exploiting Java on the server side for building component-based TP applications. Performance and industrial-strength attributes like robustness are some of the major concerns with respect to such technologies.

In the last few years there have many debates in the TP and DC communities on the appropriate paradigms for program to program communications, and the role of TP monitors in the world of web servers and feature-rich RDBMSs. Many performance-enhancing features like support for threads which used to be present for a long time only in TP monitors like CICS and IMS/DC have now become widely available in RDBMSs like Oracle. TP monitors have traditionally been deeply involved in application and data management. RDBMSs have gone from managing only data to managing programs also (via triggers, stored procedures, etc.). Web servers and some

CORBA Object Request Brokers (ORBs) have also taken on many of the attributes of TP monitors. We are seeing the emergence of the so-called *Application Servers*, which are not too different from classical TP monitors enhanced with support for the web and, possibly, object technology.

Emergence of specialized Online Analytical Processing (OLAP) DBMSs like RedBrick and Arbor's ESSBASE has allowed huge data warehouses to be built and queried efficiently. Many algorithms and tools have been developed by vendors for extracting warehouse data from operational TP systems. Specialized indexes and massive data handling features have been developed for use in managing warehouse data. Only now researchers are beginning to address these issues. In contrast to what the research community focussed on with respect to replicated data algorithms, log-based asynchronous replication has been the more favored choice for implementation in RDBMS products.

One of the significant developments of the last few years is the widespread trend towards outsourcing of information technology (IT) operations by organizations whose core business is not computer related (e.g., Kodak). This is the result of (1) difficulties encountered by such organizations in managing networks of heterogeneous systems and (2) those organizations' desire to reduce their IT costs by letting computing professionals of companies like EDS and IBM Global Services do the job. A related trend is that many organizations have stopped developing their own TP application software. Instead, they buy packaged applications (e.g., for enterprise resource planning (ERP)) from vendors like SAP and PeopleSoft. The ERP vendors have been causing some significant enhancements to be made in the functionality of DBMSs that they rely on for storing their data.

Many businesses are also reengineering their supply chains by integrating their applications with those of their partners, suppliers and customers in order to improve their operational costs and time to market. Much of this integration is expected to happen in the near future using the internet rather than via private networks as was the case in the past with electronic data interchange (EDI). Workflow management systems are expected to play a big role in this transformation of way of doing business.

The industrial and academic research communities working on TP and DC have not always focussed on the problems that are of great interest in the commercial world. This has led to some very innovative technology being developed directly by product developers themselves.

5. Conclusions

In this paper, I repeated first the history of the evolution of the popular ARIES family of concurrency control and recovery (CC&R) algorithms. I also discussed the significant impact that those algorithms have had on the research and commercial worlds. With a view towards providing a roadmap across the numerous related papers, I provided a brief summary of most of the ARIES CC&R algorithms. While some researchers might have ignored certain aspects of CC&R as being engineering work, rather than science, by focussing on the details and deciding to pay attention to the practical experiences from the past, we were able to make some fundamental contributions to the area of transaction management. We were lucky to be working in an environment where this was possible.

I expect DBMSs to be enhanced in the future with features that allow higher concurrency and improved data availability to accommodate the demanding requirements of the internet world. Parallelism will be exploited to reduce the time taken to perform operations like data backups, index build, etc. Systems will be designed to be self tuning and manageable by less-qualified people. Transactions will be everywhere, in the least expected places in our daily lives. Designing systems with industrial-strength attributes like performance, reliability and availability in mind from the beginning will be crucial for such systems to be successful in real-life usage. Most of the related problems discussed in [Mohan93d] have not been addressed sufficiently so far by database researchers.

In this paper, I did not intend to do an exhaustive survey of CC&R work in the research literature. Many papers have been written in the last few years on application recovery, semantics-based CC&R protocols (especially in the object-oriented context), theories on an integrated view of recovery and concurrency control, sophisticated indexing protocols using numerous lock modes, etc. As it is usually the case, most of the proposed algorithms have not been implemented. Nor have the designs been spelt out in most cases in enough detail for others to implement them without the need for substantial additional design work.

Acknowledgements I would like to acknowledge the contributions of the following (past/present) colleagues of mine who have collaborated with me on some of the work that I have summarized here: Ron Barber, Dick Dievendorff, Don Haderle, Dave Herbert, Russ Holden, Jeff Josten, Tina Lee, Frank Levine, Bruce Lindsay, Bob Lyle, Inderpal Narang, Andrew Peterson, Hamid Pirahesh, Kurt Rothermel, Peter Schwarz, Amit Somani, Jim Teng, Kent Treiber, Julie Watts, Steve Watts and Markos Zaharioudakis. I would also like thank the people in the different product and research groups everywhere that have adopted our research results, and have brought out products or prototypes incorporating them. I highly appreciate the strong encouragement that I received from non-IBMers like David DeWitt, Theo Haerder, Mike Carey (now an IBMer), Paris Kanellakis, Andreas Reuter, Dave Lomet and Betty Salzberg. I would also like to acknowledge the cooperation of managers like Pat Selinger, Bob Yost, Laura Haas and Irv Traiger for tolerating my

unconventional way of working at IBM Research and for letting me publish. My wife Kalpana Mohan deserves special thanks for her acceptance of my long working hours and frequent travels.

6. References

[BaRa87] Badrinath, B., Ramamritham, K. *Semantics-Based Concurrency Control: Beyond Commutativity*, **Proc. 3rd International Conference on Data Engineering**, February 1987.

[BeHG87] Bernstein, P., Hadzilacos, V., Goodman, N. **Concurrency Control and Recovery in Database Systems**, Addison-Wesley, 1987.

[BeNe97] Bernstein, P., Newcomer, E. **Principles of Transaction Processing for the Systems Professional**, Morgan Kaufmann, 1997.

[BGHJ92] Bhide, A., Goyal, A., Hsiao, H., Jhingran, A. *An Efficient Scheme for Providing High Availability*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992.

[Billa96] Billard, D. *Recovery of Transactions Exploiting Operation Semantics: Beyond ARIES*, **Proc. International Workshop on Advanced Transaction Models and Architectures (ATMA)**, Goa, August 1996.

[Borr84] Borr, A. *Robustness to Crash in a Distributed Database: A Non Shared-Memory Multi-Processor Approach*, **Proc. 10th International Conference on Very Large Data Bases**, Singapore, August 1984.

[CABGK81] Chamberlin, D., Astrahan, M., Blasgen, M., Gray, J., King, F., Lindsay, B., Lorie, R., Mehl, J., Price, T., Putzolu, F., Selinger, P., Schkolnick, M., Slutz, D., Traiger, I., Wade, B., Yost, R. *A History and Evaluation of System R*, **Communications of the ACM**, Vol. 24, No. 10, October 1981.

[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.

[CDFHM94] Carey, M., DeWitt, D., Franklin, M., Hall, N., McAuliffe, M., Naughton, J., Schuh, D., Solomon, M., Tan, C., Tsatalos, O., White, S., Zwilling, M. *Shoring Up Persistent Applications*, **Proc. ACM SIGMOD International Conference on Management of Data**, Minneapolis, May 1994.

[ChGY81] Chamberlin, D., Gilbert, A., Yost, R. *A History of System R and SQL/Data System*, **Proc. 7th International Conference on Very Large Data Bases**, Cannes, September 1981.

[ChMo96] Choy, D., Mohan, C. *Locking Protocols for Two-Tier Indexing of Partitioned Data*, **Proc. International Workshop on Advanced Transaction Models and Architectures**, Goa, August 1996.

[ChMy88] Chang, P.Y., Myre, W.W. *OS/2 EE Database Manager Overview and Technical Highlights*, **IBM Systems Journal**, Vol. 27, No. 2, 1988.

[ClCo89] Clark, B.E., Corrigan, M.J. *Application System/400 Performance Characteristics*, **IBM Systems Journal**, Vol. 28, No. 3, 1989.

[CMSW93] Cabrera, L.-F., McPherson, J., Schwarz, P., Wyllie, J. *Implementing Atomicity in Two Systems: Techniques, Tradeoffs*, and Experience, **IEEE Transactions on Software Engineering**, Vol. 19, No. 10, October 1993.

[Crus84] Crus, R. *Data Recovery in IBM Database 2*, **IBM Systems Journal**, Vol. 23, No. 2, 1984.

[DGSBH90] DeWitt, D., Ghandeharizadeh, S., Schneider, D., Bricker, A., Hsiao, H.-I, Rasmussen, R. *The Gamma Database Machine Project*, **IEEE Transactions on Knowledge and Data Engineering**, Vol. 2, No. 1, March 1990.

[Dombr95] Dombrowska, H. *ARIES/NT Modification for Advanced Transactions Support*, **Proc. International Symposium on Advances in Databases and Information Systems (ADBIS'95)**, Moscow, 1995.

[EGLT76] Eswaran, K.P., Gray, J., Lorie, R., Traiger, I. *The Notion of Consistency and Predicate Locks in a Database System*, **Communications of the ACM**, Vol. 19, No. 11, November 1976.

[Elmag92] Elmagarmid, A. (Ed.), **Database Transaction Models for Advanced Applications**, Morgan Kaufmann, 1992.

[FZTCD92] Franklin, M., Zwilling, M., Tan, C.K., Carey, M., DeWitt, D. *Crash Recovery in Client-Server EXODUS*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992.

[GaKi85] Gawlick, D., Kinkade, D. *Varieties of Concurrency Control in IMS/VS Fast Path*, **Database Engineering**, Vol. 8, No. 2, June 1985.

[GMBLL81] Gray, J., McJones, P., Blasgen, M., Lindsay, B., Lorie, R., Price, T., Putzolu, F., Traiger, I. *The Recovery Manager of the System R Database Manager*, **ACM Computing Surveys**, Vol. 13, No. 2, June 1981.

[GrRe93] Gray, J., Reuter, A. **Transaction Processing: Concepts and Techniques**, Morgan Kaufmann, 1993.

[HaJa84] Haderle, D., Jackson, R. *IBM Database 2 Overview*, **IBM Systems Journal**, Vol. 23, No. 2, 1984.

[HaRe83] Haerder, T., Reuter, A. *Principles of Transaction Oriented Database Recovery - A Taxonomy*, **Computing Surveys**, Vol. 15, No. 4, December 1983.

[HaRo87] Haerder, T., Rothermel, K. *Concepts for Transaction Recovery in Nested Transactions*, **Proc. ACM-SIGMOD International Conference on Management of Data**, San Francisco, May 1987.

[HaRo93] Haerder, T., Rothermel, K. *Concurrency Control Issues in Nested Transactions*, **VLDB Journal**, Vol. 2, No. 1, 1993.

[HCLMW90] Haas, L., Chang, W., Lohman, G., McPherson, J., Wilms, P., Lapis, G., Lindsay, B., Pirahesh, H., Carey, M., Shekita, E. *Starburst Mid-Flight: As the Dust Clears*, **IEEE Transactions on Knowledge and Data Engineering**, Vol. 2, No. 1, March 1990.

[HLMPS94] Haderle, D., Lindsay, B., Mohan, C., Pirahesh, H., Schwarz, P. *Method for Managing Subpage Concurrency Control and Partial Transaction Rollback in a Transaction-Oriented System of the Write-Ahead Logging Type*, **United**

Kingdom and France Patent 0,295,424, Germany Patent 3,889,254,508, IBM, April 1994.

[IBM97] Special Issue on IBM's S/390 Parallel Sysplex Cluster, **IBM Systems Journal**, Vol. 36, No. 2, 1997.

[JaKe97] Jajodia, S., Kerschberg, L. (Eds.) **Advanced Transaction Models and Architectures**, Kluwer Academic Publishers, 1997.

[JhKh92] Jhingran, A., Khedkar, P. *Analysis of Recovery in a Database System Using a Write-Ahead Log Protocol*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992.

[JMNT97] Josten, J., Mohan, C., Narang, I., Teng, J. *DB2's Use of the Coupling Facility for Data Sharing*, **IBM Systems Journal**, Vol. 36, No. 2, 1997.

[KoMH97] Kornacker, M., Mohan, C., Hellerstein, J. *Concurrency and Recovery in Generalized Search Trees*, **Proc. ACM SIGMOD International Conference on Management of Data**, Tucson, May 1997.

[KuHs98] Kumar, V., Hsu, M. (Eds.) **Recovery Mechanisms in Database Systems**, Prentice Hall, 1998.

[Kumar95] Kumar, V. (Ed.) **Performance of Concurrency Control Mechanisms in Centralized Database Systems**, Prentice Hall, 1995.

[Kuo96] Kuo, D. *Model and Verification of a Data Manager Based on ARIES*, **ACM Transactions on Database Systems**, Vol. 21, No. 4, December 1996.

[Lars88] Larson, P.-A. *Linear Hashing with Separators - A Dynamic Hashing Scheme Achieving One-Access Retrieval*, **ACM Transactions on Database Systems**, Vol. 13, No. 3, September 1988.

[Lomet92] Lomet, D. *MLR: A Recovery Method for Multi-Level Systems*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992.

[LoSa92] Lomet, D., Salzberg, B. *Access Method Concurrency with Recovery*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992.

[LoTu95] Lomet, D., Tuttle, M. *Redo Recovery after System Crashes*, **Proc. 21st International Conference on Very Large Data Bases**, Zurich, September 1995.

[MaRa97] Martin, C., Ramamritham, K. *Toward Formalizing Recovery of (Advanced) Transactions*, Chapter 8 in [JaKe97].

[MHLPS92] 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. Reprinted in **Readings in Database Systems**, 3rd Edition, M. Stonebraker, J. Hellerstein (Eds.), Morgan Kaufmann Publishers, 1998. Reprinted in **Recovery Mechanisms In Database Systems**, V. Kumar, M. Hsu (Eds.), Prentice Hall, 1998. Also available as IBM Research Report RJ6649, IBM Almaden Research Center, January 1989; Revised November 1990.

[MHWC90] Mohan, C., Haderle, D., Wang, Y., Cheng, J. *Single Table Access Using Multiple Indexes: Optimization, Execution and Concurrency Control Techniques*, **Proc. International Conference on Extending Data Base Technology**, Venice, March 1990. A longer version of this paper is available as IBM Research Report RJ7341, IBM Almaden Research Center, March 1990.

[MoDi94] Mohan, C., Dievendorff, R. *Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing*, **Data Engineering**, Vol. 17, No. 1, March 1994.

[MoHa94] Mohan, C., Haderle, D. *Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking*, **Proc. 4th International Conference on Extending Database Technology**, Cambridge, March 1994. A longer version of this paper is available as IBM Research Report RJ9732, IBM Almaden Research Center, March 1994.

[Mohan90a] Mohan, C. *ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes*, **Proc. 16th International Conference on Very Large Data Bases**, Brisbane, August 1990. Another version of this paper is available as IBM Research Report RJ7008, IBM Almaden Research Center, September 1989.

[Mohan90b] Mohan, C. *Commit_LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems*, **Proc. 16th International Conference on Very Large Data Bases**, Brisbane, August 1990. A slightly revised version is reprinted in **Performance of Concurrency Control Mechanisms in Centralized Database Systems**, V. Kumar (Ed.), Prentice Hall, 1995.

[Mohan92a] Mohan, C. *Interactions Between Query Optimization and Concurrency Control*, **Proc. 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing**, Tempe, February 1992.

[Mohan92b] Mohan, C. *Less Optimism About Optimistic Concurrency Control*, **Proc. 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing**, Tempe, February 1992.

[Mohan93a] Mohan, C. *ARIES/LHS: A Concurrency Control and Recovery Method Using Write-Ahead Logging for Linear Hashing with Separators*, **Proc. 9th International Conference on Data Engineering**, Vienna, April 1993. A longer version of this paper is available as IBM Research Report RJ8682, IBM Almaden Research Center, March 1992.

[Mohan93b] Mohan, C. *IBM's Relational DBMS Products: Features and Technologies*, **Proc. ACM SIGMOD International Conference on Management of Data**, Washington, D.C., May 1993.

[Mohan93c] Mohan, C. *A Cost-Effective Method for Providing Improved Data Availability* During *DBMS Restart Recovery After a Failure*, **Proc. 19th International Conference on Very Large Data Bases**, Dublin, August 1993.

[Mohan93d] Mohan, C. *A Survey of DBMS Research Issues in Supporting Very Large Tables*, Invited Paper, **Proc. 4th International Conference on Foundations of Data Organization and Algorithms**, Evanston, October 1993. LNCS Volume 730, D. Lomet (Ed.), Springer-Verlag, 1993.

[Mohan95a] Mohan, C. *Disk Read-Write Optimizations and Data Integrity in Transaction Systems Using Write-Ahead Logging*, **Proc. 11th International Conference on Data Engineering, Taipei, March 1995.**

[Mohan95b] Mohan, C. *Concurrency Control and Recovery Methods for B^+-Tree Indexes: ARIES/KVL and ARIES/IM*, In **Performance of Concurrency Control Mechanisms in Centralized Database Systems**, V. Kumar (Ed.), Prentice Hall, 1995.

[Mohan99] Mohan, C. *A Database Perspective on Lotus Domino/Notes*, **Proc. ACM SIGMOD International Conference on Management of Data**, Philadelphia, June 1999. Slides at www.almaden.ibm.com/u/mohan/domino_sigmod99.pdf

[MoLe92] Mohan, C., Levine, F. *ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992. A longer version is available as IBM Research Report RJ6846, IBM Almaden Research Center, August 1989.

[MoLO86] Mohan, C., Lindsay, B., Obermarck, R. *Transaction Management in the R* Distributed Data Base Management System*, **ACM Transactions on Database Systems**, Vol. 11, No. 4, December 1986.

[MoNa91] Mohan, C., Narang, I. *Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment*, **Proc. 17th International Conference on Very Large Data Bases**, Barcelona, September 1991. A longer version is available as IBM Research Report RJ8017, IBM Almaden Research Center, March 1991.

[MoNa92a] Mohan, C., Narang, I. *Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment*, **Proc. 3rd International Conference on Extending Database Technology**, Vienna, March 1992.

[MoNa92b] Mohan, C., Narang, I. *Data Base Recovery in Shared Disks and Client-Server Architectures*, **Proc. 12th International Conference on Distributed Computing Systems**, Yokohama, June 1992.

[MoNa92c] Mohan, C., Narang, I. *Algorithms for Creating Indexes for Very Large Tables Without Quiescing Updates*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 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 1993.

[MoNa94] Mohan, C., Narang, I. *ARIES/CSA: A Method for Database Recovery in Client-Server Architectures*, **Proc. ACM SIGMOD International Conference on Management of Data**, Minneapolis, May 1994.

[MoNP90] Mohan, C., Narang, I., Palmer, J. *A Case Study of Problems in Migrating to Distributed Computing: Page Recovery Using Multiple Logs in the Shared Disks Environment*, IBM Research Report RJ7343, IBM Almaden Research Center, March 1990.

[MoPi91] Mohan, C., Pirahesh, H. *ARIES-RRH: Restricted Repeating of History in the ARIES Transaction Recovery Method*, **Proc. 7th International Conference on Data Engineering**, Kobe, April 1991.

[MoPL92] Mohan, C., Pirahesh, H., Lorie, R. *Efficient and Flexible Methods for Transient Versioning of Records to Avoid Locking by Read-Only Transactions*, **Proc. ACM SIGMOD International Conference on Management of Data**, San Diego, June 1992.

[MoTO93] Mohan, C., Treiber, K., Obermarck, R. *Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery*, **Proc. 9th International Conference on Data Engineering**, Vienna, April 1993. A longer version of this paper is available as IBM Research Report RJ7885, IBM Almaden Research Center, December 1990; Revised June 1991.

[Ober80] Obermarck, R. *IMS/VS Program Isolation Feature*, **IBM Research Report RJ2879**, IBM San Jose Research Laboratory, July 1980.

[Ober98a] Obermarck, R. *IMS/360 and IMS/VS Recovery: Historical Recollections*, In [KuHs98].

[Ober98b] Obermarck, R. *Logging and Recovery in Commercial Systems*, In [KuHs98].

[PeSt83] Peterson, R., Strickland, J.P. *Log Write-Ahead Protocols and IMS/VS Logging*, **Proc. 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems**, Atlanta, March 1983.

[RaCh96] Ramamritham, K., Chrysanthis, P. **Advances in Concurrency Control and Transaction Processing - An Executive Briefing**, Computer Society Press, 1996.

[Ramak98] Ramakrishnan, R. **Database Management Systems**, WCB/McGraw-Hill, 1998.

[RoMo89] Rothermel, K., Mohan, C. *ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions*, **Proc. 15th International Conference on Very Large Data Bases**, Amsterdam, August 1989. A longer version of this paper is available as IBM Research Report RJ6650, IBM Almaden Research Center, January 1989.

[StNC91] Stone, R., Nettleship, T., Curtiss, J. *VM/ESA CMS Shared File System*, **IBM Systems Journal**, Vol. 30, No. 1, 1991.

[VuDo90] Vural, S., Dogac, A. *A Performance Analysis of the ARIES Recovery Method Through Simulation*, **Proc. 5th International Conference on Computer and Information Sciences**, Cappadocia, November 1990.

[Weihl95] Weihl, W. *Transaction Processing Techniques*, Chapter 13 in **Distributed Systems**, S. Mullender (Ed.), 2nd Edition, ACM Press, 1995.