Open access • Proceedings Article • DOI:10.1145/168750.168761

Concurrency control in collaborative hypertext systems — Source link

Uffe Kock Wiil, John J. Leggett

Institutions: Aalborg University, Texas A&M University

Published on: 01 Dec 1993 - ACM Conference on Hypertext

Topics: Hypertext, Non-lock concurrency control, Multiversion concurrency control, Concurrency control and Collaboration

Concurrency Control in Collaborative Hypertext Systems

Uffe Kock Wiil
Programming Systems Laboratory
Department of Computer Science
Aalborg University
Fr. Bajers Vej 7E, 9220 Aalborg Ø,
Denmark
Telephone: +4598154211-5055
Fax: +4598158129
Email: kock@iesd.auc.dk

John J. Leggett
Hypermedia Research Laboratory
Department of Computer Science
Texas A&M University
College Station, Texas USA 77843-3112
Telephone: 409-845-0298
Fax: 409-847-8578
Email: leggett@bush.cs.tamu.edu

ABSTRACT
Traditional concurrency control techniques for database systems (transaction management based on locking protocols) have been successful in many multiuser settings, but these techniques are inadequate in open, extensible and distributed hypertext systems supporting multiple collaborating users. The term "multiple collaborating users" covers a group setting in which two or more users are engaged in a shared task. Group members can work *simultaneously* in the same computing environment, use the same set of tools and *share* a network of hypertext objects. Hyperbase (hypertext database) systems must provide special support for collaborative work, requiring adjustments and extensions to normal concurrency control techniques. Based on the experiences of two collaborative hypertext authoring systems, this paper identifies and discusses six concurrency control requirements that distinguish collaborative hypertext systems from multiuser hypertext systems. Approaches to the major issues (locking, notification control and transaction management) are examined from a supporting technologies point of view. Finally, we discuss how existing hyperbase systems fare with respect to the identified set of requirements. Many of the issues discussed in the paper are not limited to hypertext systems and apply to other collaborative systems as well.

KEYWORDS
Collaborative work, distributed hypertext systems, concurrency control, hyperbases, open architectures, extensibility, supporting technologies, user-controlled locking, events, transaction management, version control.

1. INTRODUCTION
Computing support for a large engineering enterprise [12] provides an example of the need for an advanced, collaborative, hypertext-based information management system capable of integrating several different application domains such as CAD, CASE, programming environments, office information systems and digital libraries. These advanced systems require management of a very rich and complex set of data types as well as dynamic management of the structure of the data itself (metadata). To make hypertext the integrating factor among diverse data-intensive application domains, hyperbase (hypertext database) systems must provide a wide variety of features not found in the current generation of database systems [1,5,9,11,19]. In addition to the usual database requirements of controlled sharing, integrity, backup and recovery of data, hyperbase systems must be capable of modeling complex interrelationships and providing support for novel data types. Hyperbase systems must also handle enormous amounts of data, long transactions, extensibility, notification control and versioning of both the data and the structure of hypertext. As a result of these requirements, hyperbase systems are emerging as an important new research direction [2,14,17,25,27].

Both the Programming Systems Laboratory (PSL) at Aalborg University and the Hypermedia Research Laboratory (HRL) at Texas A&M University have conducted research on hyperbase systems and distributed hypertext system architectures over the past six years (see Tables 1 and 2). Projects at the PSL have mainly focused on providing support for collaborative work [24] and lately also on open, extensible architectures [22]. Research at the HRL has mainly focused on open, extensible architectures [10] and recently also on support for multiple users [14]. These projects involve numerous distributed hypertext applications.

Table 1. Hyperbase Research at the PSL

Project Period	Hyperbase System	Refs.
1987 - 1988	HyperBase #0	
1989 - 1990	HyperBase #1	[24,26]
1990 - 1991	HyperBase #2	
1990 - 1991	Fenris	
1990 -	Hyperform	[22,25]

Table 2. Hyperbase Research at the HRL

Project Period	Hyperbase System	Refs.
1987 - 1990	PROXHY	[10]
1990 - 1991	HB1	[15,16]
1991 - 1992	HB2	[14]
1992 -	HB3	

Typically, distributed systems have taken a systems-oriented approach to distribution control [13]. The systems-oriented approach deals with the problem of distribution by masking problems (such as geographical location, sharing, concurrency, consistency and reliability) from applications *(distribution transparency)*. Hence sharing is transparent, with each user unaware of the activity of others. This clearly contradicts the needs of collaborative systems [13,23]. To coordinate and collaborate, group members must be aware of the activities of other members. This paper focuses on the special requirements for concurrency control in hyperbase systems posed by open, extensible and distributed hypertext systems supporting multiple collaborating users. The term "multiple collaborating users" covers a group setting in which two or more users are engaged in a shared task. Group members can work *simultaneously* in the same computing environment, use the same set of tools and *share* a network of hypertext objects. Based on the experiences of two collaborative hypertext authoring systems, we enumerate and discuss concurrency control requirements that distinguish collaborative hypertext systems from multiuser hypertext systems. Finally, we examine the extent to which existing hyperbase systems provide support for the identified requirements.

2. COLLABORATIVE HYPERTEXT SYSTEMS

It is useful to distinguish collaborative systems from other multiuser systems. Both database systems and timesharing systems support multiple users. However, neither of these are collaborative since they provide little notification. If one user performs some action, other users are not normally notified of the action and may only learn of it by explicitly querying the system. There are three broad categories of collaborative systems: *real-time*, *non-real-time* and *mixed* systems [4,13]. Real-time (synchronous) systems require the simultaneous presence of all users and one user's actions must be quickly propagated to the other users. In non-real-time (asynchronous) systems, users typically work in isolation at their own pace. As a result, non-real-time sessions are less focused and often longer in duration. Mixed systems contain elements of support for both synchronous and asynchronous collaboration. Collaborative hypertext authoring systems are examples of mixed systems. Several research projects have revealed shortcomings of existing database technology in managing concurrency control for collaborative hypertext systems [8,9,24]. Based on the experiences of EHTS [23] and SEPIA [20], we identify concurrency control requirements that distinguish collaborative hypertext systems from multiuser hypertext systems.

2.1 EHTS

EHTS [23] is a collaborative hypertext authoring system based on Aalborg University's HyperBase [26]. HyperBase is designed to support collaboration among its users and was the first hyperbase system to include a general event mechanism. EHTS consists of two tools: a multiple window text editor and a graphical browser. EHTS enables a group to collaborate on a shared task. Changes made on shared data by one user are immediately visible to all other members of the group. Group members can communicate in real-time and send asynchronous messages within the EHTS environment, enabling collaboration among members separated by time as well as space.

Trigg et al. [21] describe three fundamental issues in simultaneous sharing: access contention, real-time monitoring and real-time communication. EHTS uses events and user-controlled locks to deal with these issues and to provide controlled data sharing among collaborating group members [24]. EHTS also provides contention resolution at the

level of attributes in nodes and links and allows any number of users to simultaneously read and display the data field of a given node in a window on the screen. HyperBase allows locked objects to be read by any number of users, however permission to make modifications to the data field are restricted to one user at a time. Locks are allocated when the user invokes the editor lock command indicating a change in mode from browse to edit. Locks are deallocated when either the editor unlock command is invoked or the window is closed. All readers are notified as soon as possible that a data field they are accessing may be changed. Readers are provided with four types of modification notices:

1. *Intention.* All readers are notified when one person signals intention to modify the data field of the node by obtaining a lock. The node icon in the browser is shown **bold faced**. The readers also get the name of the person, enabling contact through use of the internal talk mechanism (real-time communication). Readers can then subscribe to the event corresponding to when the writer unlocks the data field.

2. *Update.* When the writer actually writes the modified data field of the node onto the shared database, all readers of the data field automatically get the contents in the data field display updated with modifications made by the writer (real-time monitoring).

3. *Completion.* When the writer is finished modifying the data field of a node, users having subscribed to this event get notified that the data field of the node has been unlocked and is write accessible.

4. *Deletion.* When a node is deleted, the display of the node is removed from the screens of all readers.

2.2 SEPIA

The latest version of SEPIA has been characterized as a cooperative hypermedia authoring environment [20]. SEPIA captures three different collaboration modes that arise naturally during group work processes. In *individual* mode, collaborators work on separate parts of the information base (in parallel or at different times), or the way in which documents are processed ensures that at most one person at a time works on a document. At other times several collaborators may concurrently access some part of the information base but still wish to work as independently as possible. In this *loosely-coupled* mode awareness of co-workers' presence and activities is necessary to detect possible conflicts and coordination needs. Finally, if several collaborators wish to work in a synchronous way, they engage in *tightly-coupled* work where they cooperate and coordinate their activities in a conference-like meeting. In tightly-coupled mode co-workers are provided with direct communication channels and a shared environment that includes a common view of the information base.

The earlier version of SEPIA was a single user system and hence did not support collaboration. The move from a single user authoring environment to a collaborative authoring environment required changes in the underlying hyperbase system [8]. GMD-IPSI's HyperBase [18] was extended and became the Cooperative Hypermedia Server (CHS) [17]. CHS is built on a relational database system that provides shared access to hypermedia documents, transaction facilities, crash recovery, recovery from deadlock and livelock situations and database consistency. In order to support collaboration, the data model of HyperBase was extended with user-controlled locks (or activity markers) and a notification mechanism. CHS also maintains collaboration information such as event subscriptions.

2.3 Concurrency Control Requirements

Collaborative work poses the following six special requirements on hyperbase systems that may be unnecessary in other multiuser settings (these requirements will be explored further in the next section):

1. *Event notification.* Both EHTS and SEPIA use events to provide real-time monitoring of changes made by group members to hypertext objects.

2. *Fine-grained notification.* It is important to be able to distinguish among operations on different attributes in an object. To provide real-time monitoring, the event handler of the application must be able to distinguish between write operations on the data field and on other attributes of nodes.

3. *User-controlled locking.* Normally, locking is done implicitly within the scope and control of transactions. Collaborative hypertext systems must support computer protocols required for traditional short transactions and social protocols inherent in collaborative work. User-controlled locking is used to provide support for long updating sessions that may outlast several short transactions.

4. *Shared locking.* Locks should be shared, since exclusive locking does not allow real-time monitoring. In order to support a common view of the information space, applications may need to read locked objects to get the latest version.

5. *Fine-grained locking.* Fine-grained locking enables a fine granularity of sharing between group members in collaborative working sessions. Locking at the object-level only allows one person to operate on an object at a time. If one group member is updating the data field of a node, object-level locking will not allow other group members to annotate or create new attributes in the node.

6. *Persistent collaboration information.* If collaboration information (e.g. records of users, event subscriptions and user-controlled locks) is only stored in the server process, clients will not be able to recover from server crashes. Collaboration information must be stored persistently.

3. CONCURRENCY CONTROL ISSUES

This section contains a discussion of the three most important cornerstones for systems supporting collaborative work: locking, notification control and transaction management. Following this, we briefly discuss the use of version control as an extension to traditional concurrency control techniques.

3.1 Locking

Four of the six requirements (3,4,5,6) listed in Section 2.3 directly imply the need for locking in hyperbase systems. Hyperbase systems must provide an additional feature not found in traditional database systems: user-controlled locking. User-controlled locking should be *shared, fine-grained* and *persistent.*

The rationale behind user-controlled locking is the same as the rationale behind long transactions. Operations on objects in hypertext environments (such as compiling source code or circuit layout) are often long-lived. If a hypertext network contains large documents, a user may require anywhere from a few minutes to several days to make updates to a single object. Variants and extensions to the following simple collaboration example will be used throughout the paper to illustrate the need for various new concurrency control constructs. We have chosen an authoring session since this type of collaboration requires support for both real-time and non-real-time sharing.

> *Two researchers, Dave and Pete, are collaborating on a paper in a hypertext authoring system. The deadline for the paper is approaching rapidly, so they decide to work in parallel on different parts of the paper. Dave must finish the introduction and Pete the conclusion. Dave locks the introduction and Pete locks the conclusion, and both start working. The update process takes several hours. Both Dave and Pete are cautious people, so they save their documents several times during the session to avoid loss of data from system crashes.*

This scenario shows the need for combining user-controlled locking with short transactions. User-controlled locking is used to provide support for long updating sessions that may outlast several short transactions. All operations (such as frequent saves) on locked objects will be performed within the scope of short transactions. Since user-controlled locking is persistent and short transactions provide traditional means of crash recovery, this type of system can recover from both client and server crashes. The only loss in case of system failure is the unsaved data in the client applications (all systems have this problem). Thus, user-controlled locking is not an alternative to short transactions, merely a necessary supplement. The use of user-controlled locking versus long transactions to provide support for long duration updates will be further examined in Section 3.3.

3.1.1 Shared Locking

The basic information access metaphor in hypertext is navigation and many systems provide (graphical) browsers to help users navigate in the hypertext network. Allowing locked objects to be read enables users to navigate through a hypertext network containing locked objects. Exclusive locking is too restrictive in collaborative systems, since applications need to read updated parts of objects to support real-time monitoring. Shared locking enables group members to collaborate and makes the complete hypertext network accessible. Thus, we are only interested in avoiding write-write conflicts, since read-write (and read-read) situations are not considered conflicts.

> *During the writing of the conclusion, Pete needs to read the introduction to see how Dave has defined a certain term important to the paper. Dave is still working on the introduction and holds a lock on it. Since the system supports shared locking, Pete is able to open the introduction and read the definition.*

Separation of collaboration mechanisms from security mechanisms is also very appropriate. When the user (or group) wants to protect data from unauthorized access, a separate access control mechanism must be used. In collaborative systems, the access control mechanism should support at least the notions of user, group and others. In this way access can be limited to members of the project group.

3.1.2 Fine-grained Locking

Locking is a well-known way of dealing with access contentions. Experiences with EHTS show that locking at the attribute level allows a finer grained concurrency and hence collaboration. In HyperBase [26] and Hyperform [25], users can lock whole objects in one operation or single attributes one at a time. The fine grained locking mechanism changes the nature of access contentions, moving the contention from the level of objects to the level of single attributes in objects. In this way several group members can share attributes of an object instead of sharing a network of hypertext objects with only one member operating on an object at a time. In EHTS, one member could be updating the data field of a node, while another member adds a new attribute and yet another member creates a link from/to the node. Access contentions cannot be avoided, but by moving the contention to a finer grained level, conflicts will occur less frequently.

> *While writing the conclusion, Pete needs to reference some of the material Dave is writing in the introduction. Pete opens the introduction, finds the particular passage and creates a reference to it. In this particular authoring system cross-references are implemented with a special link type. Thus, what Pete actually did, was to create a link from a node in the conclusion to a node in the introduction which has the data field locked by Dave.*

Many outstanding issues in the area of locking require further research. Queuing of lock requests in collaborative systems must be considered. When two users try to lock the same object, only one will be granted the lock. Should we queue the lock request of the other user? Should we provide a notification event when nodes of interest become unlocked? Should the system provide a mechanism for negotiating about locks? An additional issue related to locking in collaborative systems is that of killing locks. Should it be possible for a group member to take over a lock from another member (in case he has forgotten to unlock and has gone home)?

3.1.3 Persistent Locking

If locks are only stored in the server process, clients will not be able to recover from server crashes. Such information must be stored persistently.

> *Suddenly, during the collaborative writing session the server machine crashes and reboots. When the hyperbase server is restarted, the transaction log is used to put the hyperbase in a consistent state and information on locks are restored into the server process. Both Dave and Pete can continue their update process exactly where they were before the server machine went down. Since only the server machine crashed, they did not even lose the unsaved data on the client machines.*

The opposite case (a client machine crashes) is also supported, since the user does not lose granted locks. When the client application is restarted, persistent locks can be retrieved from the server.

3.2 Notification Control

Notification control allows group members to be notified of important actions on the shared network of hypertext objects performed by other members of the group. Three of the six requirements listed in Section 2.3 are related to event notification (1,2,6). It has been well established that a notification mechanism is necessary to support collaboration [3,8,9,24]. Notification control should be *fine-grained* and *persistent.*

Garrett et al. [6] discuss four possible notification strategies: (1) immediate update; (2) immediate notification; (3) passive notification and (4) no notification. Collaborative hypertext systems will typically use the first strategy since this allows applications to provide real-time monitoring of changes to the shared hyperbase. Real-time monitoring is a crucial part of collaboration.

> *In order to follow up in the conclusion on all issues mentioned in the introduction, Pete needs to follow the progress of Dave's work. Pete opens the introduction and gets notified that the document is locked and hence cannot be updated by him. He keeps the document open in a window on his display. Each time Dave performs a save operation, the window containing the introduction automatically gets updated with the latest changes performed by Dave. In this way Pete can follow the progress of Dave's work.*

It is important to notice that event notification should be asynchronous. Synchronous notification requires that applications poll either the hyperbase server or the client library for event messages. This polling is very inefficient. In asynchronous notification, client applications are interrupted when event messages arrive. The event handler reads the arriving message and, based on the message, performs some action. In the example, the event handler of Pete's application reads the updated introduction from the hyperbase and updates the window. This operation is performed automatically and immediately.

An additional issue related to notification control is that of individual versus on/off event subscriptions. Should the notification mechanism allow applications to only subscribe to specific events of interest or should applications receive all (or no) events from the server? The first solution places the selection overhead with the server. For every operation the server must check which client applications have subscribed to that specific event. The second solution places the overhead with the application. For each incoming event, the event handler of the application must determine if the event should trigger some action. The first solution will typically be implemented with inter-process communication, while the second may use a broadcast protocol.

3.2.1 Fine-grained Notification

The authoring example combined with the description of EHTS illustrates four important applications of a notification mechanism: notifying about lock, unlock, delete and write operations. In general, notification of all operations on objects and attributes in objects should be supported. As mentioned, it is important to be able to distinguish between operations on different attributes in an object. In order to provide real-time monitoring, the event handler of the application must be able to distinguish between write operations on the data field and on other attributes of nodes. Thus, fine-grained collaboration requires both fine-grained locking and fine-grained event notification. If the locking mechanism only operates at the object level, an attribute-level event mechanism cannot (reliably) be implemented.

3.2.2 Persistent Event Subscription

Event information should be kept persistently in the hyperbase system. The rationale behind this is the same as for persistent locking. If the server crashes, clients will not be able to recover if event information is only stored in the server process. If the client machine crashes, the server still holds event subscriptions for the client.

3.3 Transaction Management

In traditional database systems, transactions serve three distinct purposes [1]: (1) they are logical units that group operations comprising a complete task; (2) they are atomicity units whose execution preserves the consistency of the database; and (3) they are recovery units that ensure that either all the steps enclosed within them are executed or none. This section will discuss three issues important to collaborative hypertext systems: short transactions, long transactions and data model extensibility in hyperbase systems as an alternative to short transactions.

3.3.1 Short Transactions

The authoring example establishes the need for short transactions in collaborative hypertext systems. In database systems, locking is typically done implicitly within the scope and control of short transactions. Locks are used to avoid both read-write and write-write conflicts on shared data by multiple concurrently accessing users (processes). Data is locked in different granularities and in different modes to prevent other users from performing conflicting operations on the same set of data.

Several hyperbase systems are built on top of existing database systems. A typical problem with this approach is that database systems do not support user-controlled locking and often provide transaction management that is inadequate for collaborative systems. HB2 [14], for instance, is built on top of an extended relational database that currently only provides locking implicitly within short transactions at the level of classes. Obviously, this locking granularity significantly reduces the level of concurrency (and hence collaboration) in the system, compromising performance in collaborative settings.

3.3.2 Long Transactions

Long transactions have been identified as crucial in hyperbase and other database systems for CAD, CASE and software development applications [1,5,9,11,19]. The rationale behind long transactions is that operations on objects in hypertext environments are often long-lived. User-controlled locking combined with short transactions is in many respects a better solution to long updating sessions than long transactions. The major problem with long transactions is that they prevent other users from gaining access to resources for an extended period of time. This causes serious performance problems if these transactions are allowed to lock resources until they commit [1]. Other

transactions (long or short) wanting to access the same resources are forced to wait, even though the long transaction might have finished using the resources. Also, there is a problem with aborting and rolling back long transactions since much time will be spent (lost) undoing and redoing the long transaction. The differences between long transactions and user-controlled locking are three-fold:

1. *The logical, atomicity and recovery units are much smaller in user-controlled locking.* Therefore, sharing and exchange of information can take place at much smaller intervals.

2. *User-controlled locking does not completely prevent other users from getting to resources over long periods of time.* User-controlled locking only avoids write-write conflicts, not read-write conflicts, which are also avoided in long database transactions.

3. *In user-controlled locking, users or applications must explicitly lock needed resources.* Locking is done implicitly in long transactions, sometimes at a granularity inadequate for multiuser and collaborative settings.

In a system designed for collaboration, we want the users to be aware of the multiuser situation. This is not possible with long transactions because of the atomicity feature of these transactions. Long transactions do not support sharing within long updating sessions and hence real-time monitoring, which is crucial to collaboration. User-controlled locking combined with short transactions preserves all three transaction purposes as well as providing support for sharing.

3.3.3 Extensibility as an Alternative to Short Transactions

Experiences with Hyperform [25] indicate that data model extensibility in hyperbase systems might provide an alternative to short transactions. Instead of grouping operations at the application level with a transaction mechanism, it is possible to group operations at the hyperbase level with the object-oriented data modeling facility. Thus, atomic operations involving more than one Hyperform operation can also be supported by moving the operation from the application to the hyperbase. This solution actually speeds the operation since internal operations are much faster than operations over the network. Another advantage of using extensibility instead of short transactions is that users (applications) can decide to only receive one event notifying of the complex operation instead of several events notifying about a series of basic operations (leaving it up to the user to interpret what complex operation was performed). Extensibility in Hyperform directly deals with the grouping and atomicity aspects of transactions. Hyperform can also easily be extended to deal with the recovery aspect (e.g., using a log).

3.4 Version Control as an Extension to Concurrency Control

The HB3 project at the HRL is exploring the use of version control as an extension to concurrency control in hyperbase systems (personal communication). Their impression is that effective solutions to multiuser concurrency control inevitably involve versioning. We briefly describe how version control can assist in managing concurrent access to shared objects.

Each object is considered to be a collection of different versions. Each version represents the state of the object at some time in the history of its development. The versions are usually stored in the form of a compact representation that allows the full reconstruction of any version. Once the original version of the object has been created, it becomes immutable. A new version can be created after explicitly reserving the object. This reservation makes a "copy" of the original version (or the latest version thereafter) and gives the owner exclusive access to the copy to modify and later deposit as a new version. Other users needing access to the same object must either wait until the new version is deposited or reserve another version. Thus, two or more users can modify the same object only by working on two parallel versions, creating branches in the version history. Branching ensures write serializability by guaranteeing that only one writer per version of an object exists. The result of consecutive reserves, deposits and branches is a version tree that records the full history of development of the object. These branches may be merged *at some point in time* to reflect the updates performed by all users. The time interval between merges is a parameter to the version control mechanism. Synchronous collaborative systems require merges at short intervals of time (if modification has occurred), while asynchronous collaborative systems require merges at longer intervals. When branches of the version tree are merged (for example, by merging the latest version of each branch into one version), the tree becomes a directed acyclic graph.

To summarize, version control provides the following extensions to concurrency control:

1. The existence of multiple versions of objects eliminates the need for write-write synchronization since each write operation occurs in the context of one version and thus cannot conflict with other operations.

2. A version control mechanism preserves the object identity of different versions of collaborative artifacts.

3. A version control mechanism allows several long updating sessions on the same object to run in parallel with the restriction that different branches of the object are used. These branches must be merged at some point in time to reflect the updates performed by all users.

4. The integrity of individual contributions to an object is maintained since the complete history of the object is recorded by the version control mechanism.

4. EXISTING HYPERBASE SYSTEMS

Table 3 shows how eight major hyperbase systems fare with respect to the basic requirements for concurrency control put forth in this paper. Four of the eight systems directly provide features necessary for supporting collaborative work. Aalborg University's HyperBase [26] provides a minimal set of concurrency control features but lacks support for transactions and persistent collaboration information. Although it has proven valuable as a hyperbase system for EHTS [23], these missing features make EHTS vulnerable to many kinds of system failures. CHS [17] fulfills all six requirements and has been successfully used to support SEPIA [20]. Since CHS only provides locking and notification at the object-level, SEPIA is limited to object-level sharing. HB2 [14] falls short on the requirement for shared locking, and (like CHS) only provides support for object-level sharing. HB3 (the successor of HB2) and CHS (in cooperation with its version server CoVer [7]) are envisioned to solve many concurrency control issues through their version control mechanisms. Hyperform [25] directly addresses all requirements except support for persistent collaboration information. Due to Hyperform's extensible nature, the missing feature can easily be incorporated in the system.

We have not seen the first successful commercial hyperbase system. Consequently, it is tempting to use existing database systems to manage storage in hypertext systems, and many have chosen this solution, both as a foundation for development of hyperbase systems and to directly manage storage in hypertext systems. The advantage of having a full-featured database system handle physical storage is that many database features can be inherited directly from the underlying database system. The danger is that the services provided in the underlying database system are usually too specialized or restricted to support the given task, thus requiring inelegant and inefficient workarounds. Existing hyperbase systems are useful vehicles for examining a range of architectural, data storage and data sharing issues.

5. CONCLUSION

We have identified and discussed six basic requirements for concurrency control that distinguish collaborative hypertext systems from multiuser hypertext systems. The requirements can be grouped into two broad categories. Hyperbases must provide support for event notification and user-controlled locking. Event subscription should be fine-grained and persistent. User-controlled locking should be shared, fine-grained and persistent. The requirements were discussed in the light of a simple authoring example since authoring systems require support for both non-real-time and real-time collaboration. Real-time collaborative systems require fast response and notification times in order to support a common view of the information space and in some cases even finer-grained (paragraph, sentence, word or letter) locking and notification to provide support for group editing sessions [3].

To be useful for collaborative work, hypertext transactions should provide for long duration update sessions and sharing and exchange of information within these sessions. We suggest a move from systems-oriented towards user-oriented distribution control in collaborative settings. This move requires adjustments and extensions to traditional concurrency control techniques as discussed throughout the paper. Collaborative systems require a mixture of system and user distribution control. Distribution issues such as geographical location, consistency and reliability are best left to the system, while the user should have some control over concurrency and sharing. Applications should be able to acquire necessary locks based on user actions such as moving from browse to edit mode in a hypertext authoring system (for instance, by hitting a key in a window on the display). Applications should also be able to share data within long updating sessions. In this context, we have argued for the use of user-controlled locking in combination with short transactions, since long transactions do not provide enough support for sharing and exchange of information within sessions.

Table 3. Selected Hyperbase System Features.

System / Feature	HAM [2]	Aalborg University's HyperBase [26]	GMD-IPSI's HyperBase [18]	HB1 [15,16]	HB2 [14]	CHS [17]	Hyperion [27]	Hyperform [22,25]
User-controlled Locking	NO	YES	NO	(single user) NO	YES	YES	NO	YES
Shared Locking	not applicable	YES	not applicable	not applicable	NO	YES	not applicable	YES
Locking Granularity	not applicable	Attribute	not applicable	not applicable	Object	Object	not applicable	Attribute
Persistent Locking	not applicable	NO	not applicable	not applicable	YES	YES	not applicable	Possible
Notification Control	Limited	YES	NO	(single user) NO	YES	YES	NO	YES
Notification Granularity	not applicable	Attribute	not applicable	not applicable	Object	Object	not applicable	Attribute
Notification Persistency	not applicable	NO	not applicable	not applicable	YES	YES	not applicable	Possible
Transaction Management	YES	NO	YES	(single user) NO	YES	YES	NO	YES
System Architecture	Centralized Client-Server	Centralized Client-Server	Centralized Client-Server	Centralized Client-Server	Centralized Client-Server	Centralized Client-Server	Centralized Client-Server	Centralized Client-Server
Storage Facility	File System	File System	Relational DB	Semantic DB	Extended Relational DB	Relational DB	Nested Relational DB	File System
Version Control	Nodes	Nodes	NO	NO	(Provided in HB3) NO	(Provided in CoVer) NO	NO	Objects
Data model Extensibility	NO	NO	NO	NO	NO	NO	NO	YES

Note: Except for the HAM, the information in this table was verified by the developers of each system.

ACKNOWLEDGMENTS

This research was supported in part by the Danish Natural Science Research Council through Program #11-0061 and in part by the Information Technology & Organizations Program and the Database and Expert Systems Program of the IRIS division of the CISE directorate of the National Science Foundation (USA) through grant #IRI-9217185.

REFERENCES

[1] Barghouti, N. S., and Kaiser, G. E. 1991. Concurrency control in advanced database applications. *ACM Computing Surveys,* 23, 3, (September), 269-317.

[2] Campbell, B., and Goodman, J. 1988. HAM: A general-purpose hypertext abstract machine. *Communications of the ACM*, 31,7, (July), 856-861.

[3] Ellis, C. A., Gibbs, S. J., and Rein, G. L. 1991. Groupware: Some issues and experiences. *Communications of the ACM*, 34, 1, (January), 38-58.

[4] Ellis, C. A., and Gibbs, S. J. 1989. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, (Portland, Oregon, May), pp. 399-407.

[5] Fishman, D. H., Beech, D., Cate, H. P., Chow, E. C., Connors, T., Davis, J. W., Derrett, N., Hoch, C. G., Kent, W., Lyngbaek, P., Mahbod, B., Neimat, M. A., Ryan, T. A., and Shan, M. C. 1987. Iris: An object-oriented database management system. *ACM Transactions on Office Information Systems*, 5, 1, (January), 48-69.

[6] Garrett, L., Smith, K., and Meyrowitz, N. 1986. Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system. In *Proceedings of the CSCW '86 Conference*, (Austin, Texas, December), pp. 163-174.

[7] Haake, A. 1992. CoVer: A contextual version server for hypertext applications. In *Proceedings of the Fourth ACM Conference on Hypertext (ECHT '92)*, D. Lucarella, J. Nanard, M. Nanard, P. Paolini, Eds., (Milan, Italy, December), pp. 43-52.

[8] Haake, J. M., and Wilson, B. 1992. Supporting collaborative writing of hyperdocuments in SEPIA. In *Sharing Perspectives, Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work, (CSCW '92)*, (Toronto, Canada, November), pp. 138-146.

[9] Halasz, F. 1988. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31, 7, (July), 836-852.

[10] Kacmar, C. J., and Leggett, J. J. 1991. PROXHY: A process-oriented extensible hypertext architecture. *ACM Transactions on Information Systems*, 9, 4, (October), 399-419.

[11] Leggett, J. J., Schnase, J. L., Smith, J. B., and Fox, E. A., Eds. 1993. Final report of the NSF workshop on hyperbase systems (Washington, DC, October 15-16, 1992). Department of Computer Science Technical Report No. TAMU-HRL 93-002, Texas A&M University, College Station, TX, July.

[12] Malcolm, K. C., Poltrock, S. E., and Schuler, D. 1991. Industrial strength hypermedia: Requirements for a large engineering enterprise. In *Proceedings of the Third ACM Conference on Hypertext (Hypertext '91)*, (San Antonio, Texas, December), pp. 13-24.

[13] Rodden, T., and Blair, G. 1991. CSCW and distributed systems: The problem of control. In *Proceedings of the Second European Conference on CSCW (ECSCW '91)*, L. Bannon, M. Robinson, and K. Schmidt, Eds., (Amsterdam, The Netherlands, September), pp. 49-64.

[14] Schnase, J. L. 1992. HB2: A hyperbase management system for open, distributed hypermedia system architectures. Ph.D. Dissertation. Texas A&M University, College Station, Texas.

[15] Schnase, J. L., Leggett, J. J., Hicks, D. L., Nürnberg, P. J., and Sánchez, J. A. 1993. Design and implementation of the HB1 hyperbase management system. *Electronic Publishing: Origination, Dissemination and Design*, 6, 2, (July).

[16] Schnase, J. L., Leggett, J. J., Hicks, D. L., and Szabo, R. L. 1993. Semantic data modeling of hypermedia associations. *ACM Transactions on Information Systems*, 11, 1, (January), 27-50.

[17] Schütt, H. A., and Haake, J. M. 1993. Server support for cooperative hypermedia systems. In *Hypermedia-Proceedings der Internationalen Hypermedia '93 Konferenz*, H. P. Frei and P. Schauble, Eds., (Zurich, Switcherland, March), Springer Verlag, Serie Informatik Aktuell, pp. 45-56.

[18] Schütt, H. A., and Streitz, N. 1990. *HyperBase:* A hypermedia engine based on a relational database management system. In *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext*, A. Rizk, N. Streitz, and J. Andre, Eds., (Versailles, France, November), Cambridge University Press, pp. 95-108.

[19] Silberschatz, A., Stonebraker, M., and Ullman, J. D., Eds. 1991. Database systems: Achievements and opportunities. *Communications of the ACM*, 34, 10, (October), 110-120.

[20] Streitz, N., Haake, J., Hannemann, J., Lemke, A., Schuler, W., Schütt, H., and Thüring, M. 1992. SEPIA: A cooperative hypermedia authoring environment. In *Proceedings of the Fourth ACM Conference on Hypertext (ECHT '92)*, D. Lucarella, J. Nanard, M. Nanard, P. Paolini, Eds., (Milan, Italy, December), pp. 11-22.

[21] Trigg, R., Suchman, L., and Halasz, F. 1986. Supporting collaboration in NoteCards. In *Proceedings of the CSCW '86 Conference*, (Austin, Texas, December), pp. 153-162.

[22] Wiil, U. K. 1993. Extensibility in open, distributed hypertext systems. Ph.D. Dissertation. Aalborg University, Denmark.

[23] Wiil, U. K. 1992. Issues in the design of EHTS: A multiuser hypertext system for collaboration. In *Proceedings of the 25th Hawaii International Conference on System Sciences (HICSS-25)*, B. Schriver, Ed., (Kauai, Hawaii, January), pp. 629-639.

[24] Wiil, U. K. 1991. Using events as support for data sharing in collaborative work. In *Proceedings of the International Workshop on CSCW*, K. Gorling and C. Sattler, Eds., (Berlin, Germany, April), pp. 162-176.

[25] Wiil, U. K., and Leggett, J. J. 1992. Hyperform: Using extensibility to develop dynamic, open and distributed hypertext systems. In *Proceedings of the Fourth ACM Conference on Hypertext (ECHT '92)*, D. Lucarella, J. Nanard, M. Nanard, P. Paolini, Eds., (Milan, Italy, December), pp. 251-261.

[26] Wiil, U. K., and Østerbye, K. 1990. Experiences with HyperBase - A multi-user back-end for hypertext applications with emphasis on collaboration support. Department of Computer Science Technical Report R 90-38, Aalborg University, Denmark, October.

[27] Zobel, J., Wilkinson, R., Thom, J., Mackie, E., Sacks-Davis, R., Kent, A., and Fuller M. 1991. An architecture for hyperbase systems. In *Proceedings of the First Australian Multi-Media Communications, Applications & Technology Workshop*, pp. 152-161.