Replicated Document Management in a Group Communication System

Leonard Kawell Jr., Steven Beckhardt, Timothy Halvorsen, Raymond Ozzie
Iris Associates, Inc.

Irene Greif
Lotus Development Corp.

Abstract

This paper is about the design and implementation of a replicated database that forms the basis for the Notes group communication system. The system supports groups of people working on shared sets of documents and is intended for use in a personal computer network environment in which the database servers are "rarely connected". Most algorithms for guaranteeing consistency across replicas require more reliable network connections between servers for adequate performance. Analysis of many group communication applications, however, revealed relatively weak consistency requirements across copies of the database. These requirements can be met by a simple replication algorithm that works well in rarely connected environments. This kind of replication has been used previously for a limited set of applications such as name directory replication; we have applied this technique to a much larger class of applications. Our characterization of this class of applications suggests that this technique generalizes to support distributed database implementations of other group work systems, including computer conferencing and bulletin board systems.*

**Notes is an internal project name*

Presented at the **Second Conference on Computer-Supported Cooperative Work**, Portland, Oregon, September 26-28, 1988

1. Introduction

Notes is a group communication system that is used by people to share textual, numeric, and graphical information. The system operates on personal computers in local-area and wide-area networks, and provides end-users the ability to design and create document databases for specific applications. This paper focuses on the Notes document manager, which supports replicated databases with a number of interesting characteristics, particularly when compared with replication technology typical in transaction-oriented databases.

In transaction-processing applications for record-oriented database management systems, replication algorithms must meet strict consistency criteria, usually defined in terms of serializability of transactions [Gray]. Most implementations of replication algorithms that provide strict consistency depend on high likelihood of continuous connection between database server machines. Notes has a strong requirement to support workgroups that cannot afford continuously available inter-network connections. We refer to such networks as *rarely-connected* networks. This level of connectivity is typical for PC users. Local area networks that connect small workgroups who share printers often are not interconnected to support cross-group collaboration. Dial-up line connections that cross organizational boundaries are expensive and have low bandwidth.

In rarely connected environments, replication that guarantees serializability would at best be possible only at enormous cost in performance. However, Notes applications do not require this level of consistency across replicas. As a result we are able to rely on a simple replication algorithm. It provides the replication that is crucial to the viability of the product in the PC environment, at a cost that is acceptable in that environment. The replication capability has had several additional

benefits for the product, since it also provides a means of doing static load balancing on a single network, automatic backup of databases, and support of home or portable computers that operate in a standalone mode.

Similar approaches to replicated databases have been previously used for distributed directory services [Oppen] [Smith]. Our work extends this approach to a broader class of applications, including document sharing, electronic mail, and conferencing. The implementation is new as well, because it is optimized to work well over low bandwidth, dial up lines.

Section 2 briefly outlines the characteristics of typical applications. In section 3 we review the design goals for the document manager and replication process. Section 4 presents the algorithms and implementation. Section 5 provides some data on current usage of replication. The concluding section reviews our approach to bringing shared document databases to the PC user. This approach meets a wide-range of user needs, not least of which is the need to collaborate despite the reality of poor network connectivity for PC users in *ad hoc* workgroups.

2. Application Characteristics

Notes is based on a shared document database system that can be tailored to the needs of specific workgroups. A user might participate in a number of workgroup activities each supported by a different shared database. A database is a collection of related forms or semi-structured [Malone] documents, organized through views that sort or categorize information.

Users build specialized applications by tailoring the database to store, organize and present specific kinds of information. For example, a group managing a software development project would want a variety of different documents in the database: bug reports, bug fix notices, comments and suggestions, progress reports, etc. The documents would be organized to make it easy for a reader to find new items, and so that comments related to particular topics were grouped together.

The format of documents and appearance of information can vary from application to application: graphs, images, pictures and numerical information can be intermixed with textual information; layout and use of color can give individual applications very distinctive looks. Databases have been designed to support group applications such as:

- Software Development Project Management
- Document Library Management
- Strategic Business Opportunities
- Market Research Consolidation
- Personnel Department Job Openings
- Group Status Reporting
- Customer Contact Tracking

The way that a group uses one of these databases is reminiscent of "computer conferencing" systems such as EIES [Hiltz] and electronic bulletin boards such as University of Illinois Plato Group Notes and VAX Notes [Gilbert]. As in these systems, the emphasis is on *many-to-many* communication, as opposed to the person-to-person communication of electronic mail. New documents are composed and entered in the database to be read by a group of individuals. Some group members act in special roles – e.g. moderator – and are therefore granted special privileges for accessing and modifying items in the database.

The primary differences between Notes and these kinds of systems have to do with the distribution and availability of information, the high quality screen presentation, multi-media documents and tailorability of applications. Most conferencing systems have been mainframe or mini-based with information viewed only through terminals and with user interfaces based on character displays. These conferences may have different purposes, but they always have the same message-like appearance. Tailorability is just beginning to be addressed in research prototypes of extensions to conventional conferencing sytems [e.g. the "Tailorable EIES" project].

Traditional conferencing systems make different guarantees about availability of data. A mainframe-based conferencing system is available only when the machine is accessible and operating. A networked system such as VAX Notes similarly depends on access to critical machines: although there may be a large number of geographically distributed servers in a network, each "conference" is on a single server in a system. This approach is viable for VAX Notes only because it is intended for use on a DECnet network, a network with continuous connections. In a rarely connected environ-

ment, dependence on a single machine as the sole repository for a group's database would be unacceptable. If the group is widely dispersed, some group members would have to reach the server over expensive phone lines. A local replica gives a low cost, highly available alternative.

Despite the differences in approach, both specifically designed Notes applications and general computer conference discussions share the following properties:

(1) Group communication is accomplished primarily through adding documents to a database.

(2) The applications do not rely heavily on modifying documents once placed in the database. (An exception is a situation in which one user can be identified from the outset as the sole editor of a document.)

(3) The applications do not call for transaction processing that crosses document boundaries (that is, there are no updates to sets of documents that must be done to all or to none of the documents in the set).

(4) Group members do not need to see "up-to-the-minute" data at all times.

Applications with these characteristics can be fully supported by replicated databases that meet weak consistency requirements. As we explain in Section 4, our replication algorithm assures "eventual consistency", a form of weak consistency.

A feature of this design is that a user need only communicate with a single server to modify a database. Propagation of database changes occurs between servers as a background activity. This ensures good response time as well as eventual convergence of the databases. Properties 2 and 3 are particularly important because exclusive locks to documents and two-phase commit protocols cannot be implemented under these conditions.

These application characteristics are shared by many asynchronous meeting support systems. Computer conferencing systems support generic electronic meetings that allow groups to accomplish many of the same goals of face-to-face meetings without co-locating, and without even attempting to all work at the same time. Tailored applications for specific kinds of group meetings and information sharing also fit these characteristics. The growing number of Notes applications forms a large class as well.

Sometimes special consideration has to be made for replication during application design. If, for example, the software development application were based on large documents intended to include bug report and repair information, many individuals might all need to update a single document (violating property 3 above). The bug reporter would write the initial report, but people working on fixing the bug would later have to update the document to indicate that they were working on it and the status of the fix. If instead a set of documents is used, each participant in the process can simply add comments in a new, briefer document that will be stored with the original document.

There are group work applications that do not fit within this class. If a group of co-authors requires immediate notification when more than one person is working on the same document (even if the co-authors are connected to separate servers) [Greif], they will have difficulties in a rarely connected environment. Either the database cannot be replicated, or it must be replicated by a more complex algorithm that will require more continuous connections between servers and stricter consistency across copies.

3. Document Manager

The document manager implements the basic operations for data storage and retrieval. These same operations are used in the replication process. This section outlines the design goals of the document manager, with particular attention paid to replication, concurrency and access control.

3.1 Document Manager Design Goals

The primary goal of the document manager is to provide permanent storage for free-form and semi-structured objects of arbitrary size. The document manager can handle many different types of data (e.g. floating point numbers, times and dates), and is exceptional in its ability to handle large blocks of text and graphics. The document manager is not a generalized database system; instead it is tuned for documents that are 2-3Kbytes in size and can handle documents of approximately 100-200Kbytes. (See [Smith] for comparison of relational databases and databases tuned for more general kinds of object storage.)

Conventional databases tend to work with small records or tables, so they can typically buffer large portions of the database in memory. Since we assumed that our "records" would be large, the Notes system maintains most of the database on-disk and only caches the most recently used parts

of the database in memory. For this reason, the document manager does not provide a well-tuned way to store thousands of small records.

The document manager provides a relatively fast way to search for documents by any small field and very fast access to many, sorted views of the documents in the database. The searching capability is provided by storing the small fields in documents separated from the large fields. The view capability is provided by maintaining up-to-date indices containing sorted copies of fields in the selected documents. View indices can be hierarchical for use in tree-structured conferences and for multi-level "categorization" of documents (especially mail).

The document manager has programmatic interfaces for the following functions:

o Databases can be created, opened, closed, and deleted.

o Documents can be created, opened, updated, closed, and deleted.

o Documents can be retrieved based upon their modification time and/or the contents of fields within the documents.

o Search criteria are specified using a very comprehensive algebraic query language. Indexed views can be created, opened, closed, and deleted.

o Groups of index entries in a view can be read. The desired entries can be specified by either index position, using an infinite precision number (e.g. 2.1.3.4), or by key values.

The document manager protects the confidentiality of documents through access control lists. An access control list (ACL) is a system-managed object contained in the database. An ACL contains a list of the names of all the users and groups who are authorized to access the database. Associated with each name in the list is the access level that the user has been granted.

3.2 Replication Design Goals

Notes replication ensures eventual consistency of the documents in all replicas: changes made to one copy eventually migrate to all. If all update activity stops, after a period of time all replicas of the database will converge to be logically equivalent: each copy of the database will contain, in a predictable order, the same documents; replicas of each document will contain the same fields. In other words, a program cannot detect the difference between replicas through the document manager programmatic interface.

The goals of the replication design in Notes are as follows:

(1) Background Replication. Automatic replication of additions (new documents), deletions, and modifications (edits) is a background activity that does not interfere with normal database access.

(2) Rarely Connected Networks. Support replication of documents on servers that are only seldomly in communication with each other. Because Notes is designed to operate in an environment where servers are rarely in communication with each other, replication does not require any immediate server-to-server message exchanges when a document is added or updated. (See [Stonebraker] for a replication algorithm that does depend on immediate communication.) Instead, a goal of replication is that it be capable of being deferred until a user-defined time (e.g. once a day, once an hour).

(3) Convergence. Over time, all replica databases must converge to contain the same versions of all documents.

(4) No Master Copy of a Database. No database replica is known as the "master copy". Likewise, no instance of a document is viewed as the "master copy". Unless purposefully set up otherwise (via access control), all copies are viewed as peers. This means replicating with any copy of the database is sufficient to obtain a correct copy of all the documents.

(5) No Database Replica Directory. There is no clearinghouse that enumerates the locations of the replicas of a particular database. There may be as many replicas as desired and each site is responsible for maintaining communication with at least one other replica of the same database. Creation of a new replica is a localized event. No centralized notification is required to alert other servers of the existence of a new replica.

(6) Optimized Dial-up Connections. When replication occurs across slow and costly communication links (2400 baud is typical) it is essential that the replication process perform its work quickly. Communication line bandwidth is clearly the limiting factor in moving a large document from one server to another. However, communication costs can be minimized by ensuring that new or updated documents can be efficiently identified, so that only those documents are moved across the link.

Some additional considerations have to do with network administration dependencies and end-user visibility of replication. Unlike some replicated database systems (e.g. [Gifford]), the Notes system does not keep a separate transaction journal or log that causes the same transactions to be performed on each copy of the database. Instead, the Notes system uses time stamps indicating the time of the last update to determine how to produce a consistent copy of the data. With this approach, the only information that must be maintained in each database is the time of the last replication with any of the other replicas of the database. Although databases with rollback capability provide automatic recovery from failures, network administrator intervention of considerable sophistication is often enough required to rule this approach out for our environment.

We also wanted to shield end-users from the details of replication. We wanted it to appear as if each copy of the file magically became consistent with all other replicas. In reality, we've discovered that things like network propagation delays, access control (especially), propagated deletions, and ID-based database equivalency sometimes require at least the database administrators to understand replication. We are currently using our early customer sites to determine the extent to which they find they have to understand these issues. We are also investigating the extent to which we can educate the users using application-specific terminology as opposed to teaching them the details of the technology.

3.3 Concurrency

The goals of concurrency control in the document manager and Replicator process are to:

(1) Detect and signal occurrences of multiple updates to a single version of a document.

(2) Converge to a single version of an updated document.

We do not try to support multi-document update transactions or automatic merging of concurrent updates to a single document. This contrasts with the goals of other replicated database systems for continuously connected networks [Stonebraker] which provide transaction concurrency control by requiring agreement among the servers. These other systems tend to treat the inability to communicate with other servers as an exception condition. Notes treats this type of communication as the norm.

Interactive detection of multiple updates is handled by the document manager for a single database using optimistic concurrency control [Kung]. When a document is opened, its modification time and version number are recorded in memory. When the user chooses to store an updated version of the document, the in-memory modification time and version are compared with the copy in the database on disk. If they differ, then another user has modified the document since the original user opened his or her copy. When this occurs, the user is notified on the workstation and can choose to overwrite the other user's version.

4. Replication

The replication algorithm is given in the first part of this section and is followed by discussion of a number of interesting design and implementation issues.

4.1 Replication Algorithm

The replication algorithm is a one-way "pull model" [Demers] algorithm. It is so named because the algorithm is executed on the local computer and only pulls documents from a remote computer. However, the algorithm is executed simultaneously by the remote computer so the overall effect is that replication occurs in both directions at once. The replication algorithm is as follows:

(1) Create a list of databases requiring replication. Find the databases common to both local and remote computers, and for each database, verify that the remote database has been modified since the last replication with the local computer.

(2) Create document indexes.

a) Open the remote database and create a list of all the documents that have been modified since the last replication. For each document include its document IDs (see section 4.2) and its last modification time.

b) Open the local database and create a list of all the documents.

(3) Replicate. For each entry in the local list, find the corresponding entry in the remote list.

a) If the document in the remote database is a newer version than the version in the local

database then copy the document to the local database.

b) If the document in the remote database is marked as "deleted" then delete the document in the local database.

c) If the document in the remote database is older than the version in local database or the document is not in the remote database then do nothing since the remote Replicator will copy the document from the local database.

d) For the remaining documents in the remote database list, copy them to local database since these are new documents.

4.2 Identifying Documents, Versions and Instances

At a given time, a document can exist in any number of replica databases. Each copy of the document is an *instance*. At most one instance of a document can appear in a single database at any time. An instance in one replica can be edited to produce a new *version* of the document. In that replica the new version replaces the old version. Two replicas of a database may have different versions of the same document if the databases have not gone through the replication process.

Thus at any given time across a set of replicas of a database, there can be many instances of a document, each of which can be a different version of that document. Immediately after replication, the pair of databases that have just replicated have exactly the same set of document instances and document versions.

Notes uses several identifiers to describe documents, document versions and instances. The *DocumentPositionID* denotes a particular version of a document in a particular database. It is specific to a database because it contains information about the relative storage position of the document in the database. As a result, it affords very fast access to the document (in a particular database), but it is not the same identifier for other instances of the document.

The *DocumentVersionID* is used to describe a particular version of a document. It changes as new versions are created. It consists of four components: a database identifier, the time of creation of the document, the time of last modification, and a sequence number. The use of the sequence number will be explained later.

The *DocumentID* is the unique identifier of a document across all replicas. In other words, it is invariant for all versions and all instances of the document. The DocumentID is actually the DocumentVersionID without the version information.

4.3 Identifying Replicas

The first step in replication involves determining what databases should be replicated. Since there is no master catalog or directory of databases that must be replicated, it is up to the server, at the moment of replication to determine what databases must be replicated. The server determines this by obtaining a complete directory listing of all databases on the remote system and comparing this with a directory of all local databases.

The server then determines which remote databases are replicas of local databases. The server must determine this either from file names or from internal identifiers. File names are not used for two reasons:

(1) to allow for replication across heterogeneous operating systems with dissimilar file name syntaxes, and

(2) to allow local control of file and directory names (a database may be given any name and placed in any directory and replication will still take place correctly).

Therefore replicas are determined by an identifier termed *ReplicaID*. The ReplicaID is assigned at creation of the first instance of the database, and is the same for all replicas of the database. The directory listing protocol includes for each database the ReplicaID and the name of the file in which it is stored. In addition, the time of last modification of each database is also provided. The modification time allows an important optimization because it allows the Replicator to ignore those databases that have not been modified since this server's last replication with the remote server. For servers that call each other frequently, this is the normal case for most databases. In our environment, servers typically call each other several times a day and find it necessary to replicate three or four files out of a possible two or three dozen.

4.4 Incremental Replication

In the above algorithm, the remote database must send a list of all documents added or modified since the last replication. This *incremental replica-*

tion is based on information stored in a small replication history table in each database. This table contains an entry for every other database that this database has ever replicated with. Each entry contains the remote database's name and the time of the last successful replication. This time is provided as an input to the procedure that builds the list of documents in the remote database. Only documents modified (added, edited, or deleted) since the last replication are added to the document list. Therefore, each replication only deals with documents modified since the last replication. This greatly reduces the time it takes to transfer the list and build an index from the list. In practice, each replication normally finds only a few new or modified documents, so it usually takes under a minute (of connect time) to determine which documents must be replicated. For new databases (and occasionally in problem situations) there is no useful time of last replication. In that case a full replication is done by sending a list of all documents in the remote database.

4.5 Concurrent Updating of Documents

Traditional locking cannot be used to prevent concurrent updates to a replicated document because copies of the document can be on servers that are not in communication with each other. Notes can detect concurrent updates only when they are updates to the same instance of a document (in the same database). For concurrent updates to instances in replica databases, Notes simply guarantees that after replication, both databases will have the same version of the document.

Notes uses two identifiers for each document to guarantee convergence to a single version. It might be expected that the time of last modification would be sufficient. Each server would simply choose the document with the latest modification time. However, we were concerned that in the haphazardly managed PC environment that a server, with its time incorrectly (or maliciously) set into the future, could store a document that simply could not be superseded (at least until time caught up with it). On the other hand, an increasing integer sequence number has the disadvantage that two concurrent updates could look like no update at all. As a result, Notes uses a sequence number and a modification time. The sequence number is the primary arbitrator between two documents, with the modification time used as a tie-breaker. This has the effect that a document edited and saved many times is chosen over a document updated only once.

This approach does not detect concurrent updates. Clearly, two documents with the same sequence number but different modification times are the result of a concurrent update. But if the sequence numbers are different, it is impossible to tell if one was a direct ancestor of the other or if they took divergent paths. This problem may be solved in a future version of Notes by storing the complete modification history of a document as a series of modification times. Thus, when presented with two versions of the same document, the document manager can determine if one was a direct ancestor of the other or whether a concurrent update resulted in a divergence.

4.6 Access Control as Concurrent Update Protection

Access control in Notes is based on an ordered set of *levels* and a set of user tailorable *privileges* that allow fine tuning of access to specific documents. The access levels correspond to conventional user roles such as Reader, Author, and Editor.

Access Control may be used to control concurrent editing of documents in a replicated database. The access control design provides for the following types of replicated databases:

(1) Single master copy and all others read-only (e.g. Corporate Policies and Procedures Handbook, Newsletter).

(2) Peer databases but only original authors may edit documents (e.g. Design Discussion, Status Reports).

(3) Peer databases and any user with the capability can edit any document (concurrency-control is just a matter of convention).

To set up a "master copy" database, all users are granted read/write access in the master copy and Reader access in all other copies. This ensures that changes only occur in the the master copy and prevents simultaneous updates from ever occurring.

The "master copy" style of access control can be relaxed by granting users Author access in the non-master copies of the database. This enables users to add and edit documents to any replica of the database but only the original author of a document can edit the document. This approach avoids simultaneous updates as long as each user works on a single instance of the database (typically on the user's "home" server).

4.7 Database Administration

Replication is initiated by a call schedule that instructs one server to call another for the purpose of replication (as opposed to calling to route mail, for example). Typical call schedules have one server call another once or several times a day. The frequency of calling depends on the desired group interaction and the type of database. For example, a conferencing database might replicate several times a day to promote a high degree of interaction while once a day is more than sufficient for a product catalog database.

5. Replication in Use

The main benefit of Notes replication is that it performs well in rarely connected environments. As long as a client workstation can communicate with a local server, the Notes user can complete database operations. Any changes made to a database on the local server can be propagated to other servers in the background. Communication between servers can be scheduled to optimize the use of the available communication lines, whether over dial-up, wide-area or local area network.

Replication also provides a number of other benefits. This section describes some of these additional benefits and the ways that replication is being used at one site.

5.1 Benefits of Replication

Originally conceived as a way to provide internetwork communication, the inexpensive form of replication provided by Notes has proved to be valuable for other reasons:

o Static load balancing on single networks - The servers are typically microcomputers of fairly limited capacity. Additional users can be supported by simply adding another server and creating replicas of the databases on the original server.

o Automatic backups - Replication can be used to provide unattended database backups. This is particularly important in the PC environment rife with low reliability PC file systems, disk system crashes, and inexperienced PC operators who make "user errors" managing files. If a server or database goes down, users can easily change their access to another server that has a replica of the affected databases.

o Off-line standalone use - One-person offices, portable computers, and home users have found that having a personal, but more or less up-to-date copy of shared documents is very handy. The isolated PC can periodically dial up and connect for replication, but a permanent connection is not necessary. Once the shared documents have been replicated to the standalone PC, quick access to documents is no longer limited by the slow speed of the dialup connection. In fact the dialup connection is no longer needed.

o Management domain isolation - The combination of Notes security and replication provides a way to ensure that information can be distributed to untrusted sites without danger of bad information flowing back from the sites. Also, workgroups can maintain control over the usage and administration of their own servers.

5.2 Observed Usage

At Lotus and Iris (the authors' companies), there are a large number of databases in use on a regular basis. We have collected information at Lotus, Iris, and our customer test sites (approximately 30 sites in all) to be compiled and analyzed at a later date.

Between Lotus and Iris there are a total of 275 different databases (with replicas counting as a single database) stored on 26 different servers. Of these 275 databases, 110 are either personal mail or inactive databases. Of the remaining 165 active databases, 82 databases have more than one instance and 15 databases have more than 5 instances. The databases with the most instances are:

Database	Number of Instances
Name & Address Book	14
Software Problem Reports	14
Notes User Notes	12

The Name & Address Book is a database used by the Notes distributed directory service. The Software Problems Reports database is a "tracking" application that contains bug reports and their resolution for the Notes project. The Notes User Notes database is a "discussion" database that is used to discuss features and suggestions (as well as complaints) for the Notes project. It is interesting to note that these three databases represent three completely different applications built using Notes. This shows that replication is indeed useful across

a range of Notes applications. What these three applications have in common is their usefulness across the entire organization.

The Software Problems Reports database illustrates a number of the points made earlier in this paper. It is a medium sized database (approximately 950 documents) that combines very structured documents (e.g. Problem Reports) with relatively unstructured documents (e.g. Development's Responses). Users can enter a problem report on any of the 14 different servers containing a replica of the database. Generally, a new report appears in all replicas within 24 hours of its being entered. Although new problem reports and responses can be entered anywhere, by any user, certain edits made to existing reports (such as marking a problem as fixed) are made by a single individual (or a small group) with editor access.

We are also using replication as a way of moving problem reports and suggestions between the customer sites and our office. Since these sites are separate companies, the ability to isolate networks and servers into management domains and still replicate has proven to be an essential capability.

Conclusion

In this paper we have characterized the kinds of applications that are well-suited to implementation in a rarely connected environment and described an underlying database replication technology that can support these applications. Clearly, there are applications which cannot be supported with this kind of replication. These include applications that require exclusive locking or transaction serializability. Distributed database technology which supports these applications also requires more continuous connections between servers.

Just as groups have different working goals, *they also have differing computer environments.* The assumption that full connectivity and consistency is required can make systems very expensive and difficult to implement in some environments. A closer examination of workgroup needs reveals a large class of information sharing and collaborative applications which have much less stringent consistency requirements. These applications can be solved today with a relatively inexpensive rarely-connected network solution.

Additional applications can be accomodated through application design and through conventions about individuals' roles – author vs editor vs commentator – that make it possible for them to work in the rarely connected environment. One of the most interesting things about early tests of Notes has been the observation that workgroups are willing to analyze and adjust their working relationships in order to be in communication with each other today. The benefits of a rarely connected environment may indicate that future systems should allow workgroups to choose between a rarely connected solution and a continuously connected solution depending on their specific application needs.

Acknowledgments

We are grateful to Nancy Enright, David Reiner, Pito Salas, Eric Sall, Sunil Sarin, Steve Sneddon and Richard Wolf for their comments on early drafts of this paper.

References

[Demers] A. Demers, D. Greene, C. Hause, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, D. Terry. Epidemic Algorithms for Replicated Database Maintenance. *ACM Operating Systems Review;* January, 1988

[Gifford] D. K. Gifford. Weighted Voting for Replicated Data. *Proceedings of the Seventh Symposium on Operating Systems Principles.* ACM SIGOPS 1979, Pages 150-159

[Gilbert] P. D. Gilbert. Development of the VAX NOTES System. *Digital Technical Journal* . No. 6 February 1988.

[Gray] Gray, J. N. et al. Granularity of Locks and Degrees of Consistency in a Shared Data Base, in Modeling in Data Base Management Systems, North Holland Publishing, 1976, pp. 365-394.

[Greif] I. Greif, R. Seliger and W Wiehl. Atomic Data Abstractions in a Distributed Collaborative Editing System. *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Programming Languages,* St. Petersburg, Florida, January 1986.

[Hiltz] S. Hiltz and M. Turoff. The Network Nation: Human Communication via Computer. Addison-Wesley, 1978.

[Kung] H. T. Kung and J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, Vol. 6, No. 2, June 1981, Pages 213-226.

[Malone] T. W. Malone, K. R. Grant, Kum-Yew Lai, R. Rao, and D. Rosenblitt. Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination. *ACM Transactions on Office Information Systems*, Vol. 5, No. 2, April 1987, Pages 115-131.

[Oppen] D. C. Oppen and Y. K. Dalal. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment. Xerox Technical Report: OPD-T8103, 1981

[Smith] K. E. Smith and S. B. Zdonik. Intermedia: A Case Study of the Differences Between Relational and Object-Oriented Database Systems. OOPSLA '87 Proceedings

[Stonebraker] M. Stonebraker. Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES. Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Network. San Francisco, August 1978.