Minimizing Redundant Work in Lazily Updated Replicated Databases

Wai Gen Yee*, Edward Omiecinski*, Shamkant B. Navathe*

Abstract

Modern databases which manage lazy (or deferred updates) to clients which subscribe to replicated data do so on a client-by-client basis. They ignore any redundant work done during update processing caused by the commonality in client subscriptions to replicas. This paper proposes a new way to process updates which minimizes this redundancy and results in a reduction of update processing cost at the server in terms of disk space and time consumed in this phase. Ultimately, updates are available quicker, and duration during which clients must enduring stale data is reduced. Results of studies involving, iMobile, a currently available system, are reported, and are extremely encouraging.

1 Introduction

The goal of this paper is to describe a technique which dramatically improves the scalability of update processing in databases which use "lazy" replication–databases which lazily propagate updates to replicas. Scalability in this case refers to the minimization of disk space consumed and the time needed by the server to generate each client's set of updates. Ultimately, clients receive updates quicker. Currently, database servers which support lazy updates to replicas–in particular, those that support mobile clients, such as Sybase Remote, and Synchrologic iMobile–naively distribute updates to replicated data on a per-client basis. It turns out that this mode of operation is wasteful because the server does not consider cases in which multiple clients access common replicas of data. This leads to redundant work because server must generate and transmit updates to these replicas multiple times.

The importance of server operation efficiency is increased by the fact that update processing must be centralized. The architecture of a DBMS which supports replication is similar to that of the federated database system as described in [5], and logically forms a star topology with the server at the center, and the clients at the boundaries. This topology is necessary because simultaneous updates from various clients to shared data must be serialized and conflicts must be resolved[8]. Hence, the server is a bottleneck, and its performance degrades while updates are being processed.

1.1 Benefits and Pitfalls of Lazy Replication

Lazy replication is a means by which databases asynchronously propagate local updates to replicated data to other *subscribers* of these data. This technology offers many advantages, including:

*College of Computing, Georgia Institute of Technology - {waigen, edwardo, sham}@cc.gatech.edu

- Applications can be mobilized. Replication frees a client from being tethered to a central server.
- Regardless of connectivity (either mobile or connected), response times are improved because transactions are performed on local copies of data.
- Overall work can be increased. Clients can simultaneously access and manipulate shared data instead of waiting for them to become available[2].

However, there are negative aspects to using lazy replication. First, data can become stale as update propagation is delayed. Second, as data becomes stale, conflicts can occur as multiple independent transactions attempt to update the same data[7]. Because of these disadvantages, guidelines have been developed to determine the applicability of this technology[20].

There are two common applications with which lazy replication is typically associated. One is in mobile computing, where mobile clients *cannot* feasibly maintain–for economic or technical reasons–a constant connection to a server. Automated salesforces typically carry around laptops which periodically exchange data with servers as dial-up or LAN connections become available. Another application is in an office with clients interconnected by a high-speed LAN. To reduce network contention and increase response time during busy periods, clients cache replicas of data as well as updates to that data. Updates are propagated and installed at convenient times.

1.2 Synchronization Methods

The process by which updates are propagated and replicated data converge to a consistent state is called "synchronization." Real-time, on-line synchronization, (described in [8, 10], and used in many products, including Microsoft's SQL Server 7) guarantees that the client simultaneously has the same state as the server for some instant in time. However, we consider this method highly unscalable, as it reduces the availability of the server and results in longer connection times.

The synchronization model that we focus on in this report is based on the one used in two industry leading products for mobile database support, namely Remote[6] and iMobile[21]. In these products, the server generates updates for clients based on pre-defined definitions of the clients' data interests. The updates are stored in files (which we call update logs) and transmitted via a "store-and-forward" protocol. The advantage of this method is that update log generation is decoupled from its transfer and installation, reducing the connection time between client and server and allowing the server to delay update processing until a cost-efficient time (e.g., during periods of low-activity, such as the middle of the night).

We call databases which perform such synchronization **intermittently synchronized databases** (**ISDBs**), referring to the fact that updates to replicas are client-initiated and thus occur only intermittently. ISDBs operate like ordinary databases, but with the addition of replication services, which are generally transparent to users, except during synchronization phases, i.e., applications interact with the databases in the normal fashion. A *replication server* runs in the background, and logs any changes made to its local database. During a *synchronization phase*, the replication server generates a log (or

logs) containing updates made to the database since the last synchronization phase. This log (or logs) is transported to another replication server (from the client to the server, or vice-versa). The replication server on the receiving end installs the updates contained in the log(s) into its local database. Special processing happens on the side of the server.[1] The update logs from multiple clients are serialized, and their conflicts are resolved by the replication server. Accepted updates are installed into the database and logged.[2] During the server's next synchronization phase, the replication server uses the database's catalog information to distribute the accepted updates to update logs which are ultimately sent to clients.

Transfer of updates logs through a central server is the means by which clients communicate. Server-mediated update management for each client is done individually. Previous work shows that update processing "per-client" basis, which is done in all known commercial ISDB software is inherently unscalable with an increasing client population. This is caused in part by a proportional increase in redundant work done by the server [8, 14]. We call this processing approach **client-centric**, because it caters to clients on a one-on-one basis.

Our approach to reducing redundant work and increasing scalability is to generate update logs based on commonality in the way clients subscribe to shared data. All clients determined to have a high degree of overlapping data interests have the overlap reduced by redesign of their subscriptions. We call this processing approach **data-centric**, because it focuses on client data interest patterns. Ultimately, our redesign of client subscriptions results in update processing scalability with an increasing client population in the following cost factors:

- Update processing time - The time the server takes to process update logs is a function of the amount of work it must do. By reducing redundant work, the server availability increases.
- Disk storage space - Rearranging update logs can reduce storage cost as the redundancy in contents among update logs is reduced.
- Synchronization time - Because the server can generate update logs quicker, clients can access and install them quicker.
- Transmission time - As the number of clients interested in particular data-centric update logs increases, it becomes advantageous to multicast updates to them, instead of sending them on a unicast basis.

1.3 Outline

The contributions of this paper include a model of update processing in ISDB servers (Section 2), the identification of the unscalability problem characteristic of all currently known ISDBs (Section 2.3), and a means to alleviate this problem (Section 3). We develop a model for the costs of update processing (Section 2.2) and use a probabilistic analysis to show the increase in processing cost as the client population grows. The problem is to reorganize (or regroup) client subscription patterns to shared data.

[1] The server, in this case, includes the database server as well as the its replication server.

[2] The updates are logged in the transaction log by *Remote* and in a database table in *iMobile*.

We call this the "grouping" problem, and its solution is based on the heuristic application of special operators which incrementally perform the reorganization. The use of these operators is shown to drastically improve ISDB costs (Section 4). Loose ends, such as usability issues involved in our solutions, are discussed at the end of this report (Section 5). Related work and conclusing remarks are discussed at the end of the paper (Sections 6 and 7). The technical details of data-centric extensions to ISDB software, including conflict resolution, serializability, and security issues, have already been described in [14], and implemented in our prototype. They are not repeated here.

2 Model

Formally, we specify the grouping problem as follows. Let $F = \{F_1, \ldots, F_T\}$, where F_i is a predefined fragment over a set of tables on the server. A fragment is a horizontal and/or vertical partition of a table as described in [17]. F defines the ISDB's shared data. Each fragment i has an associated weight, W_i, which provides an estimation of the fraction of updates which apply to it. Fragment weights can be determined by either using database statistics, or, we can estimate them as a function of the amount of data a fragment spans (i.e., the fragment's size) and the number of clients that subscribe to it (i.e., the fragment's replication level):

$$W_i = \frac{(\text{size} \times \text{replication}) \text{ of fragment } i}{(\text{size} \times \text{replication}) \text{ of all fragments}} = \frac{\sum_{j|F_i \in C_j} \text{size}(F_i)}{\sum_{k,l|F_k \in C_l} \text{size}(F_k)} \quad (1)$$

Let $C = \{C_1, \ldots, C_N\}$ where $C_i \subseteq F$ is the set of fragments describing client i's subscription or interests in shared data. Both F and C are given as inputs to the problem[3].

Let $G = \{G_1, \ldots, G_M\}$ where each $G_j \subseteq F$ is a set of fragments which we call datagroup j, and let $\Phi(i) \rightarrow A$, $(A \subseteq G)$, where $\Phi(i)$ is the subscription function which maps a client i to a set of datagroups from G. In other words, Φ returns the subscription of a given client to a set of datagroups.

A grouping scheme (or "grouping" for short) is defined as:

1. **Datagroup Definition:** Definition of G.
2. **Mapping Clients to Datagroups:** Definition of Φ.

Under client-centric grouping, $G = C$ and $\Phi(i) \rightarrow \{G_j | j = i, 1 \leq i \leq N\}$. In other words, there is a one-to-one relationship between client interests and datagroup definitions, and the mapping from clients to datagroups is trivial.

Data-centric grouping aims at reducing overall processing cost by optimally defining G and Φ. See the example in Figure 1.

[3]Typical commercial ISDB software includes design tools that aid users in the initial definitions of F and C. For instance, Sybase Remote allows users to define *publications* and *subscriptions*, which correspond to fragments and subscriptions as defined above.

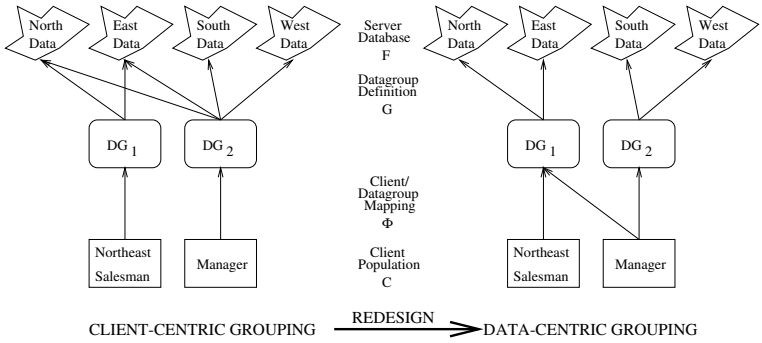


Figure 1: A small hypothetical example of client-centric to data-centric redesign. Notice that the datagroup definition, G, and the mapping of clients to datagroups, Φ, change, and the resultant datagroups are disjoint.

On any grouping scheme, we impose a single constraint, called the *correctness constraint*: the union of all data represented by a client's datagroup subscription is a superset of the client's data interests:

$$\{F_j | F_j \in G_k, \forall G_k \in \Phi(i)\} \supseteq C_i, \forall i, 0 \leq i \leq N$$

The correctness constraint only guarantees that the subset of datagroups assigned to a client contains at least all of the data to which the client subscribes. It is trivially satisfied by client-centric grouping.

2.1 ISDB Operation

Each client i subscribes to and manipulates data in its scope including those defined in C_i, and any private data, subject to the constraints of the application. During client-side synchronization, client i's replication service generates an update log containing local updates to C_i. It also contacts the server, downloads the update logs depending on $\Phi(i)$, and preprocesses and installs the updates pertaining to C_i. Reconciliation[8] of any conflicting updates is application-dependent.

The server maintains primary copies of all data in F[8]. Its replication server accepts and installs updates to them. Again, serialization and reconciliation are application-dependent. During server-side synchronization, the replication server generates update logs for each datagroup defined in G. It neither considers C, nor Φ in doing this.

The main difference between client-centric and data-centric operation, besides the grouping scheme, is in the preprocessing of data-centric update logs received by the client. Data-centric grouping requires the client to receive potentially multiple update logs depending on that client's interest profile, filter out superfluous updates (those outside the scope of C_i), and merge them, as described in [14]. After this process, the resultant update log is identical to the single update log it would receive under client-centric grouping. We will discuss the overhead of this process later. The rest of ISDB operation is identical and not discussed here.

2.2 Cost Model

The cost model uses the I/O model of computation. This model assumes that I/O dominates processing time, so its estimation is a good approximation of true processing time. The model also relies heavily on database statistics to predict cost parameters.

Three activities of update processing are timed in evaluating the performance of a particular grouping scheme: update mapping; update log storage; and update log propagation. The update mapping cost is the amount of time required to map an update to its datagroup[4]. The server storage cost is the time needed to store all datagroup update logs onto disk. The update propagation cost is the time it takes to load update logs into memory, then transmit them over a network to clients.

The variables used in the cost model and the cost functions are described below. Note that the term "record" refers to the data structure–i.e., row–that contains the respective information:

Variable Name	Description (units, if appropriate)
C_S	server disk seek time (secs)
C_L	server disk latency time (secs)
C_D	$C_S + C_L$, defined for convenience
C_T	server disk transmission rate (secs/byte)
C_N^k	transmission rate between client k and server (secs/byte)
V_B	buffer size for each update log file (bytes)
V_D	update operation record size (bytes)
V_P	fragment definition record size (bytes)
V_T	temporary table record size (bytes)
V_G	datagroup definition record size (bytes)
V_S	number of operations in the update file
V_F	$V_D V_S$, defined for convenience
W_i	weight of fragment i
M	number of datagroups
N	number of clients

$$\text{Total Cost} = \text{Update Mapping Cost} + \text{Update Storage Cost} + \text{Update Propagation Cost} \quad (2)$$

Update (to datagroup) mapping cost is the time required to map updates to datagroups. We assume that the log of updates to be distributed and an update-to-fragment mapping table (the publication information) are sequentially read into memory, and the results of their join are saved into a temporary file. The temporary file and the fragment-to-datagroup mapping table (the subscription information) are then read and joined to produce the final result.

$$\text{Update Mapping Cost} = 5C_D + C_T(V_F + |F|V_P + 2V_S V_T + V_G \sum_{G_i \in G} |G_i|) \quad (3)$$

Update storage (to disk I/O) cost measures the time required to store all the update logs onto disk.

[4] An update operation is one of three data manipulating commands, namely `INSERT`, `UPDATE` and `DELETE`. Combinations of these operations constitute the transactions contained in update logs.

We assume that a main-memory buffer is maintained for each update log, and whenever a buffer is filled, its contents are written to disk. This happens until all update logs are written.

$$\text{Update Storage Cost} = \sum_{j=1}^{M}[C_D\lceil\frac{V_F}{V_B}\sum_{i|F_i\in G_j}W_i\rceil + C_T V_F \sum_{i|F_i\in G_j}W_i] \quad (4)$$

Again, the weight of fragment i, W_i, estimates the proportion of operations that are applied to that fragment (see above).

(Server to client) update propagation cost measures the time required to load into memory and then transmit the appropriate update logs to all clients, assuming unicast communication between the server and each client. Each log the client requires is sequentially read into memory, then transmitted over the network at the client's bandwidth. To simplify the model, we assume that the file server has enough bandwidth to handle multiple (perhaps all) clients simultaneously, and that their connection is reliable.

$$\text{Update Propagation Cost} = \sum_{k=1}^{N}[C_D|\Phi(k)| + (C_T + C_N^k)(V_F)(\sum_{G_j\in\Phi(k)}\sum_{i|F_i\in G_j}W_i)] \quad (5)$$

2.3 Problems with ISDB Operation

ISDB update processing is not scalable with an increasing numbers of clients. In a general setting, in which all clients access and modify various parts of the database, as the client population grows, so does the size of the database, and the number of updates and datagroups that must be managed. Data-centric grouping can provide benefits in terms of update mapping and update storage costs, with a limited impact on update propagation costs.

Update mapping cost increases slightly with the number of clients under client-centric grouping. Data-centric grouping only marginally improves this cost by reducing the amount of fragment-datagroup mapping information, i.e., subscription data, that must be maintained. (See the last term of Equation 3.)

Update log storage cost is proportional to the number of copies of each fragment's update operations written to disk, and the number of datagroups. Designing disjoint datagroups reduces writing time, and limiting the number of datagroups limits the disk latency times. In this section, we focus the impact of replicating fragments in datagroups because redundant update storage is generally the greater cost.

For the sake of analysis, assume that L fragments are uniformly and independently distributed amongst N datagroups with a probability of P ($P = Pr\{F_i \in C_j\}, \forall i, j$)[5]. Hence, the expected number of fragments in an arbitrary datagroup is PL. The probability that any fragment F_k is in both C_i and C_j is P^2, and, over all L fragments, the expected number of fragments that any 2 datagroups have

[5]This is admittedly an unrealistic distribution, and used only for analysis. The value of P, for our purposes, can be approximated by $P = \frac{\sum_i |G_i|}{|G||F|}$, which is the ratio of the number of fragments subscribed to by all clients and the number of total possible subscriptions.

in common is therefore P^3L. Therefore, $O(i,j)$, the percentage of fragments that datagroup i has in common with datagroup j is expected to be:

$$E[O(i,j)] = \frac{E[|\text{fragments in } C_i \text{ and } C_j|]}{\text{average number of fragments in } C_i \text{ and } C_j} = \frac{P^3L}{\frac{1}{2}(PL+PL)} = P^2 \tag{6}$$

Equation 6 states that, on average, $(P^2 \times 100)\%$ of the fragments in any datagroup are in another arbitrary datagroup.

Another revealing value is the proportion of disk space used to store update logs which is redundant. Let R be the storage redundancy factor, where $R = 0$ means that there is only one copy of each update over all update logs, and $R \to 1$ means that nearly all of the update logs are redundant. R can be defined as follows. The expected number of copies of fragment F_i maintained by all N clients (or datagroups) is NP, and, over all L fragments, the number of fragments maintained is NPL. Hence, the expected proportion of redundant storage (R) can be computed as the number of *replicas* of fragments over the total number of fragments.

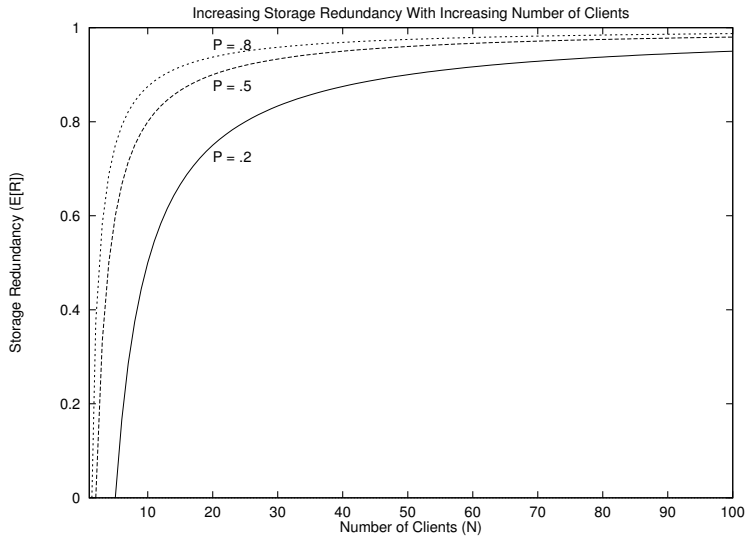


Figure 2: $E[R] = \frac{L(NP-1)}{LNP} = 1 - \frac{1}{NP}, (NP \geq 1)$ Update log storage redundancy (R) increases with the numbers of clients (N) and subscription probability (P). (Ideal = 0, Max $\to$ 1)

The amount of storage redundancy is directly related to both the number of clients supported, and the probability that a client subscribes to a fragment. As the number of clients grows, the amount of redundant storage quickly reaches unreasonable levels. (See Figure 2.) For illustration, assume $F = \{a,b,c,d,e,f\}$, and $G = \{\{a,b,c,d\},\{c,d,e,f\}\}$. If $P = \frac{2}{3}$, then, the overlap between datagroups 1 and 2 is $O(1,2) = \frac{|\{c,d\}|}{4} = \frac{1}{2}$, and $R = 1 - \frac{1}{2 \times \frac{2}{3}} = \frac{1}{4}$. In other words, each datagroup shares half of its fragments with the other, and overall, one-fourth of the current set of datagroups consists of replicated fragments.

Update propagation cost represents the time it takes to read and unicast all the relevant update logs from the server to the clients. Under client-centric operation, each client receives only one copy of all

updates intended for its exact subscription, and nothing more, because $C_i = G_i$, and $\Phi(i) \rightarrow \{G_i\}, 1 \leq i \leq N$. Hence, data-centric grouping can, at best, leave this cost unchanged.

From this brief cost analysis, one can see that superfluous work is done in update processing in terms of storage. Data-centric grouping generally alleviates this problem at the expense of (unicast) update propagation cost. The tradeoff, described below, is beneficial when either datagroups have high degrees of overlap or client transmission bandwidth is high.

2.4 Multiple Replication Servers as a Common Scalability Solution

Vendors acknowledge the processing burden that an ISDB server faces. In order to improve ISDB performance, vendors suggest implementing the "Clustered Server Architecture"[22], in which there are dedicated machines for the database server, multiple dedicated replication servers, and a file server. In this way, replication services can be divided among multiple nodes, and overall update processing time can be reduced. However, in our experience, the database server cannot be fully decoupled from the replication process, because the database server acts as the central repository for updates and catalog information. Replication servers must actively interact with the database during synchronization, the server becomes a bottleneck, and the speedup is sublinear. Moreover, individual replication servers still must in general manage multiple clients, and judicious assignments of clients to replication servers can increase the benefits of data-centric grouping, and speed up the generation of update logs. We very briefly describe one such client-replication server assignment method.

We can model the "closeness" between pairs of clients, and assign the management of their updates to replication servers as follows. Let $\mathbf{F}$ be the *client-fragment* matrix, where $\mathbf{F}_{ij} = 1$ iff client i subscribes to fragment j, and 0 otherwise. Let $\mathbf{W}$ be the diagonal weight matrix where $\mathbf{W}_{ii} = W_i$, $1 \leq i \leq |F|$. Hence, we can define a *client-affinity* matrix $\mathbf{FWF}^T$, where $\mathbf{FWF}^T_{ij}$ expresses the total weight that clients i and j have in common. With this information, we can apply popular clustering techniques to determine how to distribute clients to replication servers. The technique that we propose is similar to the fragment-affinity algorithm described in [16] which iteratively generates pairs of load-balanced attribute clusters in quadratic time with respect to the number of fragments. The exact application of this algorithm to client clusters is not described here, but with judicious assignment of clients to replication servers, the clustered server architecture can yield even higher degrees of overlap, making grouping schemes even more effective. Hence, without loss of generality, in the following operational description, we consider a case in which there is a single replication server interacting with the database server. We do not further investigate this technique in this paper. The file server is not considered as we do not discuss transport mechanisms in this paper.

3 A Heuristic Approach to Grouping Design

We begin this section by discussing ISDB update processing costs, and introduce means by which they can be reduced. Three redesign operators, applicable to a set of client subscriptions are introduced, and their effects are formalized with a cost/benefit analysis. Finally, we introduce a heuristic for operator application.

Based on the cost equations introduced in Section 2.2, certain conclusions can be drawn about the way to manipulate the grouping in order to reduce update processing costs. Namely, we can manipulate the number of datagroups, the composition of datagroups, and the subscription of clients to datagroups in order to change update processing costs.

Intuitive ways of regrouping, although with different side effects, include merging, splitting, and subtracting overlapping datagroups. The definitions, side-effects, and applicability of these operators are described below.

3.1 Operators for Redesigning Datagroups

Merging two datagroups involves replacing two datagroups with their union. Clients subscribing to at least one of the merged datagroups instead subscribe to their union, preserving the correctness constraint (See Section 2). If there is overlap between the merged datagroups, then storage cost is reduced in proportion to the size of the overlap. In the case that a client originally subscribed to only one of the merged datagroups, the client must receive superfluous updates for fragments contained in the "other" merged datagroup, resulting in increased update log transmission costs.

Splitting involves finding two non-totally overlapping datagroups (i.e., if $G_k = G_i \cap G_j$, then $G_k \neq \emptyset$ and $G_k \neq G_i$ and $G_k \neq G_j$) and splitting off their intersection to form a third datagroup. Subscribers to either datagroup must also subscribe to the third datagroup. Splitting reduces overlap in datagroups, but the amount of data that is transmitted to the respective subscribers does not increase. However, there is overhead in terms of disk latencies for each addition datagroup generated.

Subtracting two datagroups applies only if one is a subset (either proper or not) of the other. The smaller of the two is subtracted from the larger one, eliminating their overlap. If the datagroup subtracted from becomes empty (in the case where the subset relationship is not proper), then it is discarded. Subscribers to the larger datagroup must also subscribe to the smaller one (if a smaller one exists), and overhead, in terms of disk latencies, is incurred by clients having to subscribe to additional datagroups. The number of datagroups is not increased by this operation, and savings are proportional to the degree of overlap between the subtracted datagroups. Subscribers to these datagroups do not need to receive extra data.

3.2 Cost Analysis and Group Construction Strategies

In this section, we describe the effects of the operators defined above in terms of the cost model. We then develop application guidelines for the operators.

The amount of catalog information required to map updates to datagroups is directly related to the sum of the cardinalities of datagroups. Applying any of the operators reduces the amount of catalog information and hence the amount of data that must be read to perform the mapping.

$$\Delta\text{Update Mapping Cost}_{ij} = -C_T V_G \sum_{G_k \in G} |G_k|$$

Storage cost, in terms of writing data onto disk as well as seek and latency times, generally decreases in proportion to the degree of intersection (S^I_{ij}, defined below) of the two datagroups affected (i and j) whenever any of the three operators are applied.

$$\Delta\text{Update Storage Cost}_{ij} = -S^I_{ij} V_F [C_T + \frac{C_D}{V_B}]$$

Update propagation cost changes depending on the operation applied to datagroups i and j. In merging datagroups i and j, the server must in general read and transmit more data (measured by S^W_{ij}) to its subscribers at an aggregate throughput (measured by C^W_{ij}). However, overall disk seek and latency time decreases if two datagroups to which a client already subscribes are merged. When datagroup j is subtracted from datagroup i, disk seek and latency penalties are incurred if the overlap between the two datagroups is not total (when their difference is not zero, i.e., $\lceil S^D_{ij} + S^D_{ji} \rceil = 1$) for all clients which do not already subscribe to both. Finally, when two datagroups are split, disk seek and latency time is incurred for all the datagroups which subscribe to at least one of the split groups.

$$\Delta\text{Update Propagation Cost}^{Op}_{ij} = \begin{cases} (V_F)(S^W_{ij} C_T + C^W_{ij}) - \sum_{k|G_i, G_j \in \Phi(k)} C_D & \text{Op. is merge} \\ C_D \sum_{k|G_i \in \Phi(k) || G_j \in \Phi(k)} & \text{Op. is split} \\ \lceil S^D_{ij} + S^D_{ji} \rceil C_D (\sum_{k|G_i \in \Phi(k), G_j \notin \Phi(k)}) & \text{Op. is subtract} \end{cases}$$

Where S^I_{ij} and S^W_{ij} are the *size of the intersection* and the *difference in weights*, respectively, of datagroups i and j. The size of the intersection of two datagroups is the sum of the weights of the fragments in the intersection. The difference in the weights between two datagroups is the sum of the weight of superfluous fragments added to all clients' subscriptions when two datagroups are merged. This definition uses a value known as *size of difference* between datagroups i and j, S^D_{ij}, which measures the weight of the fragments in their difference:

$$S^D_{ij} = \sum_{k|F_k \in (G_i - G_j)} W_k \tag{7}$$

$$S^I_{ij} = \sum_{k|F_k \in (G_i \cap G_j)} W_k \tag{8}$$

$$S_{ij}^W = S_{ji}^W = \sum_{k|G_i \in \Phi(k), G_j \notin \Phi(k)} S_{ji}^D + \sum_{k|G_i \notin \Phi(k), G_j \in \Phi(k)} S_{ij}^D \quad (9)$$

C_{ij}^W is defined as the aggregated time it takes to transmit a unit of superfluous data for clients subscribing to only one of merged datagroups i or j.

$$C_{ij}^W = C_{ji}^W = \sum_{k|G_i \in \Phi(k), G_j \notin \Phi(k)} S_{ji}^D C_N^k + \sum_{k|G_i \notin \Phi(k), G_j \in \Phi(k)} S_{ij}^D C_N^k \quad (10)$$

Since all of the operators save time in terms of writing update logs, we know that they are applicable to datagroups i and j under the following conditions:

- The intersection between i and j is high.
- The server's update log is large.
- The disk drive throughput is low.
- The buffers in memory are small.
- The number of datagroups definitions is low.

The additional conditions which increase the applicability of each particlar operator is shown below:

Operation	Condition for benefit	Description
merge$_{ij}$	$V_F(S_{ij}^W C_T + C_{ij}^W - S_{ij}^I[C_T + \frac{C_D}{V_B}]) <$ $\sum_{k\|G_i,G_j \in \Phi(k)} C_D + C_T V_G \sum_{G_i \in G} \|G_i\|$	•Network bandwidth is high •Many clients subscribe to both
split$_{ij}$	$\sum_{k\|G_i \in \Phi(k) \| G_j \in \Phi(k)} C_D <$ $S_{ij}^I V_F[C_T + \frac{C_D}{V_B}] + C_T V_G \sum_{G_i \in G} \|G_i\|$	•Few clients want either datagroup
subtract$_{ij}$	$\sum_{k\|G_i \in \Phi(k), G_j \notin \Phi(k)} \lceil S_{ij}^D + S_{ji}^D \rceil C_D <$ $S_{ij}^I V_F[C_T + \frac{C_D}{V_B}] + C_T V_G \sum_{G_i \in G} \|G_i\|$	•Difference between datagroups is small •Few subscribe to i without j

3.3 A Greedy Heuristic Approach To Group Formation

The application strategy proposed here reduces the search space by using a greedy approach. Local cost minima are pursued until no cost reduction can be made with the above operators. The greedy algorithm applies all possible subtraction operations on the set of datagroups until no cost-reduction is possible. Subtraction is applied in this way because it can be viewed as a special case of either merging or splitting, and usually has the least side-effects in terms of cost penalties as well as affecting limiting the search space for the other two operators. After the subtraction phase, the single most effective merge or split, based on the Δ equations above, is chosen and applied. These two phases loop until no beneficial merge or split operation is possible.

The reason subtractions are attempted after merge and split operations is that after each merge or split, the possibility of finding candidates for subtraction increases. If, after a merge, $G_k = G_i \cup G_j$, and G_k is a factor of c larger than an average datagroup ($E[|G_k|] = cPL$), then, the probability that an average datagroup, G_l is a subset of G_k increases from $P^{|G_l|}$ to $(cP)^{|G_l|}$. On the other hand, after a split, for a group $G_k = G_i \cap G_j$, the probability that G_k is the subset of an average datagroup increases from $P^{|G_i|}$ to $P^{|G_k|}$. Further details of the algorithm are left out for brevity.

4 Experiments

The following experiments are meant to model the application of lazy replication in both the mobile and office environments described in Section 1. We show the scalability of data-centric grouping as the number of clients increases using various client-fragment subscription rates. In most cases studied, data-centric update processing time and disk space used becomes *constant* as the client population increases. We also show that the average time it takes to synchronize clients is consistently better using data-centric grouping.

Experiments were conducted using a dedicated database server and a dedicated replication server connected by a 10Mbps Ethernet LAN. The database server is Sybase's Adaptive Server Anywhere, version 6.03, running on a 200MHz Pentium PC with Windows NT, 128MB of RAM and a 4GB hard drive. The replication server ran on a 266MHz Pentium PC running Windows NT with 64MB of RAM and a 4GB hard drive and consists of Synchrologic's iMobile software, version 2.41.

Experimental parameters include those described in Section 2.2. The values we choose for these parameters reflect the hardware (e.g., $\tilde{4}$MB/sec disk transfer rate) and software (e.g., 4KB page size) we use. The other parameters in this experiment include N, the number of clients, and P, introduced in Section 2.3. which describes the probability that a fragment is subscribed to by a client, for all clients and fragments.

For the data-centric grouping tests, the replication server was enhanced with the data-centric extensions described in [14]. One of the consequences of the data-centric extensions is that an additional meta-data file must be sent to each client,[6] as well as any possible extraneous updates. We will discuss this overhead where relevant.

4.1 Experiment 1: The Low Bandwidth Scenario

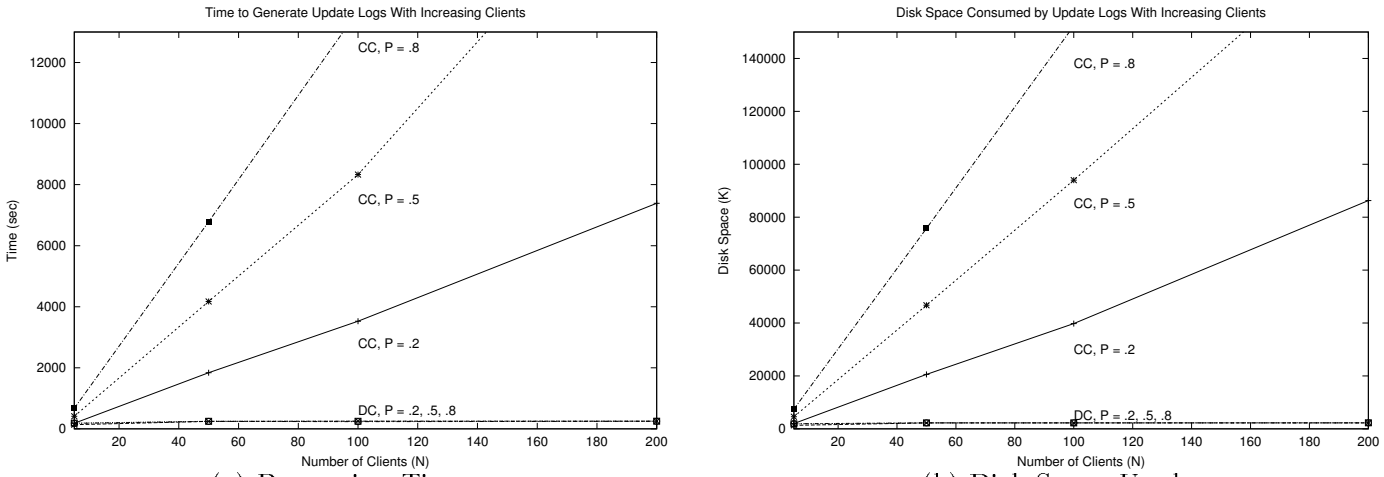
A popular application of ISDBs is in the support of mobile salesforces. In such an applications, a mobile DBMS user manages local replicas, and periodically dials in to the server in order to synchronize its local database. In this experiment, we assume that each user connects to the server at 57600bps, typical of modern modems. Each client is defined as having a fixed subscription to any of 100 equally sized fragments based on P. For simplicity, the database consists of a single universal relation divided into 100 equally sized fragments to which 100,000 updates are uniformly distributed.

In comparing the performance of client-centric to data-centric update processing, we focus on three values which roughly correspond to the cost factors mentioned in Section 2: the total time spent generating all update logs; the amount of disk space consumed by the update logs; and the extra data that needs to be transmitted to clients as a result of data-centric grouping.

We vary the number of clients ($N \in \{5, 50, 100, 200\}$) and the fragments subscription probability ($P \in \{.2, .5, .8\}$). With a fixed P, we expect that increasing the number of clients will increase the client-centric update processing time as well as disk space consumed by update logs linearly. With

[6] In degenerate cases, such as a single-client database, data-centric grouping does not apply.

increasing values of P, the slope of the linear relationship increases. On the other hand, data-centric update processing should result in an asymptotic update processing time as well as drive usage because, with the given configuration, the greedy algorithm sees advantage in generating disjoint datagroups that eventually may correspond to individual fragments. Hence, the amount of time taken to generate update logs and the amount of disk space used reaches a fixed maximum as the client population increases. Experimental results confirm these predictions (See Figure 3). Notice that higher values of P converge to asymptotic costs quicker.



(a) Processing Time (b) Disk Space Used

Figure 3: Time and space consumed during server-side update log generation using client-centric (CC) and data-centric (DC) grouping with increasing clients and various levels of P.

Moreover, the data-centric solutions presented here do not require any client to receive superfluous updates because given the volume of data delivered, it is always advantageous to split datagroups rather than merge them, minimizing the negative impact on update propagation. However, a meta-data file (mentioned above) must be sent to each client, and data-centric preprocessing of the update logs must be performed at the clients.

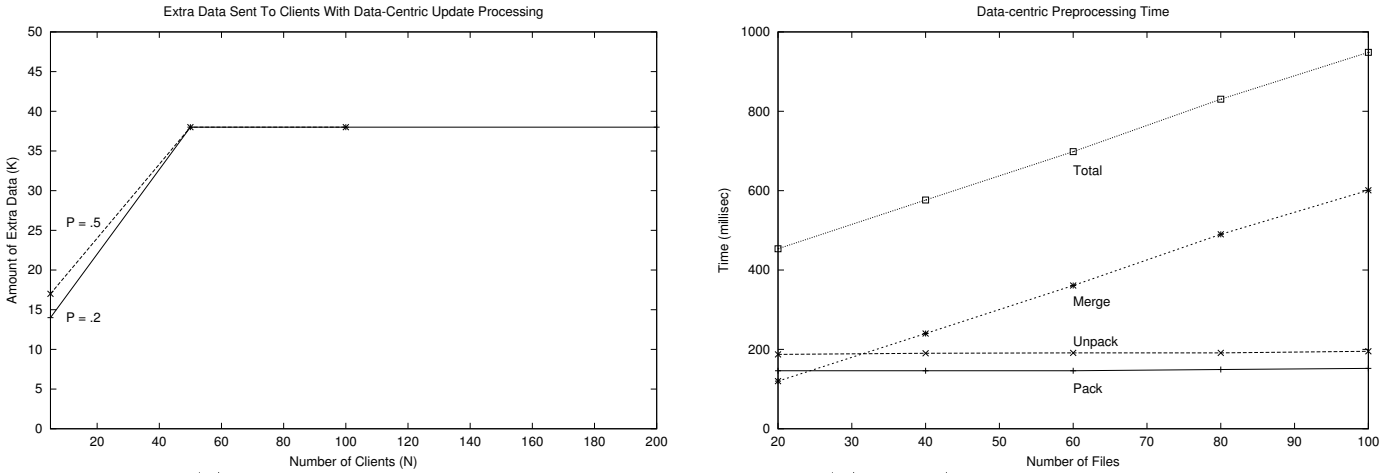
Meta-data transmission and data-centric preprocessing are relatively inexpensive. The volume of meta-data sent to each client depends on the number of groups generated, and *in our particular implementation* is approximately $(\frac{24}{82}|G| + 9)$KB in size.[7] At a transmission rate of 57600bps, the cost of sending this file to the client is relatively low. Also, the number of datagroups generated–and hence the size of the meta-data file–reaches asymptotic levels, so, the volume of extra meta-data sent to clients becomes bounded (See Figure 4a). Other preprocessing includes merging the update files and filtering their contents of superfluous or redundant updates. This operation is very cheap, and using the I/O model again, consists of sequentially reading all the update logs from disk, and writing out the relevant

[7] By empirical evidence.

contents.

$$\text{Update log merging cost}_i = C_T V_D \Big(\sum_{F_k \in G_j \in \Phi(i)} W_k + \sum_{F_k \in C_i} W_k \Big) + |\Phi(i) + 1| C_D \text{[8]}$$

In addition, for the sake of usability, we assume that multiple data-centric update logs are automatically packed at the server (using UNIX's *tar* utility, for example) and unpacked at the client as part of the data-centric preprocessing. We time the costs of packing, unpacking, and merging update logs on a 200MHz PC and find that these operations do not constitute much overhead (See Figure 4b).



(a) Meta-data File Size (b) Pack/Unpack and Merge Times

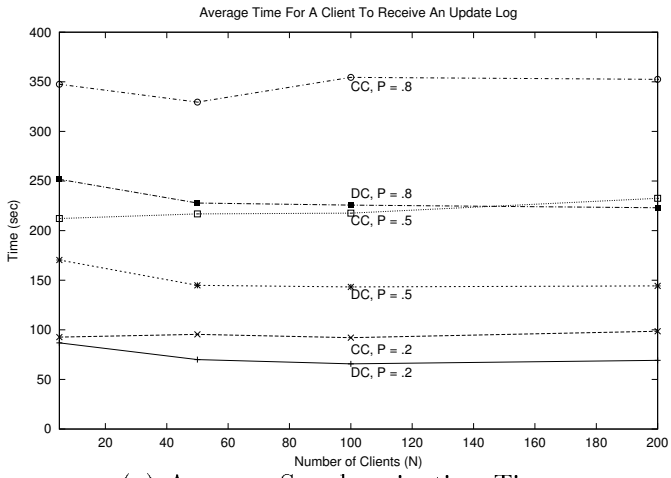
Figure 4: Two cost factors of data-centric update log preprocessing. (a) The meta-data file size reaches a plateau, because the number of datagroups generated reaches asymptotic levels. (b) The files used were the data-centric update logs generated by the $P = 50, N = 50$ test.

With this information, we can calculate the expected minimum time necessary for a client to receive a custom update log from the receiver starting at the beginning of the server's synchronization phase. We compare the average time it takes for a server to generate the necessary update logs, transmit them (at 57600bps), and, in the data-centric case, preprocess them and transmit the additional meta-data for each client. We do not include the cost of installing the updates, because this cost is independent of the grouping scheme. Because of the high savings in generating update logs, the average time it takes for a client to receive an update log is still drastically lower using data-centric grouping (See Figure 5).

4.2 Experiment 2: High Bandwidth, Small Update Volume

Another application of ISDB technology is in the office environment, where workstations are interconnected via high-speed LAN. Updates are propagated in low volumes, based on the application, such as the tracking of a set of orders for a product. Updates are deferred in order to lighten the network load

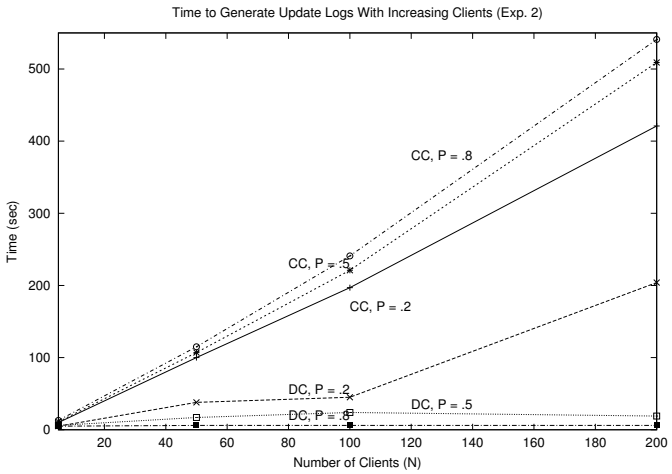
[8] This equation caually borrows server-side variables from Section 2, which, in this case, refer to the client.



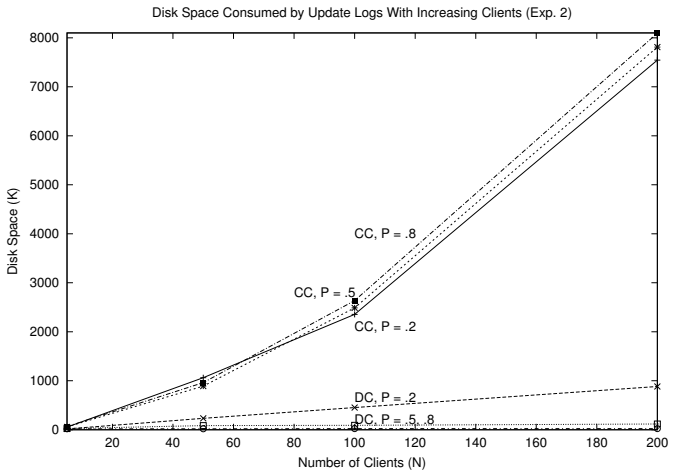
(a) Average Synchronization Time

Figure 5: Synchronization Time = time to generate update log(s) + time to transmit update log(s) + time to do extra processing in the data-centric case. (Note that update installation time is not included.)

in case real-time networked transactions must be made. In these tests, we assume that clients connect to the server at 10Mbps and a total of 250 updates are distributed.



(a) Processing Time

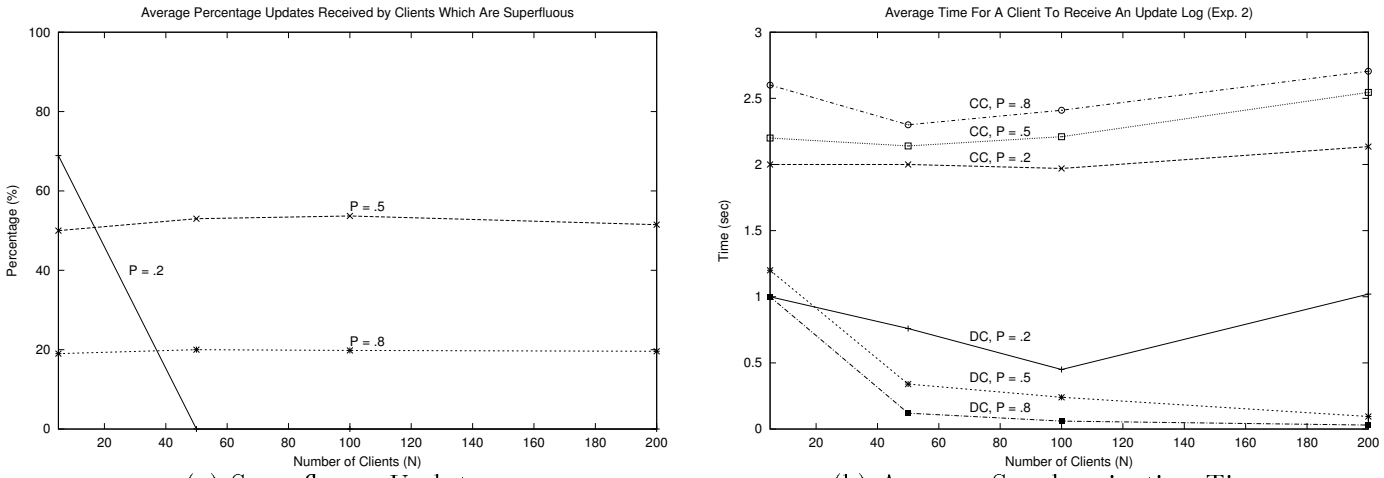


(b) Disk Space Used

Figure 6: Time and space consumption for experiment 2.

Again, data-centric processing is superior in update processing time, disk space consumed, and average time to synchronize a client. (See Figures 6, 7b.) However, cost is minimized differently in this case. In the first experiment, the resultant datagroups were disjoint, and no client received superfluous updates, minimizing transmission time. In this experiment, however, the network is not the bottleneck, and overall synchronization time is actually decreased when there are fewer datagroups, even if extra updates must be sent. (See Figure 7a.)

Two seemingly anomalous behaviors are worth noting. First, data-centric costs do not converge to an



(a) Superfluous Updates (b) Average Synchronization Time

Figure 7: Notice the anomalous behavior when $P = .2$.

asymptotic level when $P = .2$. This happens because the degree of overlap between pairs of datagroups and their associated update logs is so small, that the operators have limited applicability. For $P = .2$, we are witnessing the phenomenon that the application of one operator reduces the applicability of another. For $P = .2$, the clients do not receive superfluous updates once $N = 50$, because, as the number of clients increases, there is more chance for beneficial splitting, which is done instead of merging. Below that threshold, merges are performed, reducing the applicability of splits, resulting in redundant updates (See Figure 7a). The gradual increase in costs exhibited in Figure 6 shows the steady state of $P = .2$. This is a result of an increasing number of datagroups. In this case, as the number of clients increases, the proportion of datagroups which cannot have operators applied to them remains the same. The full analysis of this case is left out for brevity.

Second, the average synchronization time when $P = .5$ or $P = .8$ approaches 0 because average datagroup generation time per client diminishes rapidly and transmission costs are near 0 (See Figure 7b).

5 Discussion

There are two outstanding issues that should be addressed. The first relates to the commercial viability of this technology, and the second to the initial design of ISDBs. Given that the performance of data-centric update processing is so clearly superior, why do commercial ISDB vendors not implement it? We present two of many possible reasons. The effectiveness of data-centric grouping requires much apriori knowledge of user behavior. In the event that a sudden drastic shift in client behavior happens, such as changes in the expected volume or scope of updates, or changes in client-fragment subscription, the data-centric grouping scheme may perform worse than the client-centric one. A model of user behavior must be constructed and performance studies based on it must be done. Our exposure to the user base

of an ISDB vendor suggests that different industries like shipping, pharmaceuticals, and retail, exhibit different usage patterns.

Another reason may be that client-centric design is simpler to implement, and most intuitive to the database designer. ISDB technology, especially in the mobile environment, is still relatively new, and the bottlenecks in performance have yet to be discovered. Ostensibly, there is also an over-developed focus on minimizing transmission costs to clients, given the cost and speed of modern remote communication relative to disk and CPU speeds. Also, given the new market, vendors are still trying to assess and meet the usability demands of customers, which are often independent of performance.

In fact, we have been conducting technology transfers with an ISDB vendor for the past several years. The proposal we have for them is a simple one. Allow the users to define fragments, and have users subscribe to them. During update processing, in the low bandwidth scenario, the server should generate update logs for each of those fragments. In the high bandwidth, low update volume scenario, the number of update logs should be reduced. After update log generation, the server uses the subscription information to pack the appropriate files for each client. Our experiments in the last section show that this is a more cost-effective means of update processing.

The initial design of the ISDB naturally brings us to the second issue. In our analysis, we assume the existence of a fixed set of disjoint fragments. Although experience shows that during the initial design phase, ISDB users generally define fixed subsets of the database (fragments), and then have clients subscribe to them, it is not necessarily the case that these fragments are disjoint, or can be determined to be disjoint. Furthermore, although the fragments may be disjoint in one table, data subscriptions which are dependent on that table may not be. For example, one client subscribes to northern sales data plus associated customer data, and another to southern sales data plus associated customer data. Although the sales data are disjoint, the customer data may not be. In the case where multiple clients subscribe to common fragments, our basic analysis and solution still applies but may be weakened. In the case where the number of predicates is small, we can generate minterm fragments[3]. This is feasible, as update processing performance is not greatly affected by the number of fragments available, and experience shows that very few predicates are defined in many commercial applications.

6 Related Work

Intermittently synchronized databases (ISDBs) naturally fall into a class of applications related to both view (and replica) maintenance and mobile database systems. The server of an ISDB is in charge of maintaining the views of its clients. As in classical view materialization work, updates are propagated to views via refresh processes. Refreshment of views may either be immediate[9]–within the transaction that updates the base table–or, as in ISDBs, deferred[4]. View maintenance work, however, typically deals with issues of consistency and speed of view refresh. One early work deals with replica maintenance on mobile computers[8], but focuses on reduction of deadlock and reconciliation of *lazy replication*. It proposes the master copy replication scheme that exists in many commercial ISDBs today. However, it

does not deal with the scalability of update log generation, the topic of this paper.

Mobile database research takes a more system-wide view and considers the costs incurred by the components in a distributed and even disconnected architecture[12]. Issues such as cell-handoff, power consumption of components and resource location in a wide area are considered. Although an ISDB is an instance of a mobile database system, these issues do not play a role in the update propagation problem we consider. However, ISDBs do share the issues of centralized processing and update propagation with mobile databases and broadcast databases[1]. An ISDB imposes a high workload on the server, as it must collect updates from clients, maintain their views, and return update logs to them. However, an ISDB has different applications because of different means of communication with the server. Unlike much other mobile database work, communication is not assumed to be wireless nor asymmetric.

We identify server-side processing to be a problem, and frame it as a database design problem, in which predefined fragments (horizontal and/or vertical partitions[3, 17, 18] of the database) are allocated to nodes (which, in the ISDB case, are called update logs). Allocation research typically assumes a fixed network of varying reliability and capacity, in which resources (e.g. files or fragments) are assigned to nodes with tradeoffs in reliability, read/update costs, and storage costs[15], but does not consider the ISDB case in which clients can naturally work while disconnected from the server, and communication is mediated via update log transfers.

Some work suggests client caching as a remedy for disconnection or high bandwidth costs[19], and others[11] study dynamic allocation techniques to reduce communication costs. These works focus on availability and communication costs, assume high participation between the client and the server during synchronization phases, which we consider unscalable as the number of clients increases.

Our reference architecture is actually taken from existing commercial DBMSs which include a feature known as "replication." Such an architecture assumes a more passive role by the server, in which it pre-packages update "objects" in order to disseminate updates. Related works [13, 14] recognize possible savings in server or network costs by considering receiver interests in the delivery of updates, but these works leave out the problem of designing the update objects. To our knowledge, ours is the first work that analyzes the cost components of this architecture.

Finally, just as patterns in client subscription can be exploited to reduce update log generation costs, we also see an opportunity in exploiting these patterns to reduce update log dissemination costs. Effective use of bandwidth becomes an issue as the volume of update data increases. Grouping data can have the side-effect that many clients share interests in common update objects, and multicasting or broadcasting them becomes more feasible.

7 Conclusion

We have modeled ISDBs, and have analytically shown the problems with client-centric update processing which limit their scalability. An industry-standard solution to speeding up this process–decoupling and multiplying replication servers–is mentioned, and we suggest that although such schemes may be

beneficial, they work to improve the effectiveness of data-centric group rather than worsen it.

Data-centric grouping reduces redundant work done at the server. This means that we yield reductions in the time needed to synchronize the clients and space required on disk. In fact, our experiments show that, in most cases, for a fixed number of updates, the cost in terms of time and disk space of generating update logs is constant, and independent from the number of clients, and their subscription pattern. In all cases, data-centric synchronization time is lower than that of client-centric grouping.

The revelation of such benefits begs the question as to why ISDB vendors do not perform client-centric processing. We know of no technical reasons for this, but consider that there is good potential for transfering the results of investigations such as our research to these products.

Our work on ISDBs is ongoing. Future work includes an analysis of the benefits of multiplying replication services, how they improve server scalability, and the benefits of judicious assignment of clients to them (See Section 2.4). We are also studying ways to model clients more realistically. For instance, some clients may require more frequent updates than others, and some other clients may frequently change their subscriptions. Capturing these behaviors make the model more realistic and robust, but also more complicated. Finally, as each update log is desired by potentially multiple clients, it may be cost-effective to explore next-generation transport mechanisms, such as multicast, to simultaneously transmit them to multiple clients[23].

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1995.

[2] Y. Breitbart and H. F. Korth. Replication and consistency: Being lazy helps sometimes. *Proceedings of the ACM SIGMOD Principles of Database Systems*, 1997.

[3] S. Ceri, S. B. Navathe, and G. Wiederhold. Distribution design of logical database schemas. *IEEE Transactions on Software Engineering*, 9(3), July 1983.

[4] L. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for deferred view maintenance. *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 1996.

[5] A. R. Diener, R. P. Bragger, A. Dudler, and C. A. Zehnder. Replicating and allocation data in a distributed database system for workstations. *ACM Symposium on Small Systems*, 1985.

[6] Sybase Workplace Database Division. Sql remote: Replication anywhere. Technical report, Sybase Corp., http://www.sybase.com/products/system11/workplace/remote2.html, 1999.

[7] R. Goldring. Things every update replication customer should know. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1995.

[8] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993.

[9] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Proceedings of the IEEE International Conference on Data Engineering*, 18(2), June 1995.

[10] B. Hammond. Merge replication in microsoft's sql server 7.0. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1999.

[11] Y. Huang, P. Sistla, and O. Wolfson. Data replication for mobile computers. *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1994.

[12] T. Imielinski and B. R. Badrinath. Wireless computing: Challenges in data management. *Communications of the ACM*, October 1994.

[13] B. Levine, J. Crowcroft, C. Diot, J. Garcia-Luna-Aceves, and J. Kurose. Consideration of receiver interest in content for IP delivery. *Submitted for publication*, July 1999.

[14] S. Mahajan, M. J. Donahoo, S. B. Navathe, M. Ammar, and S. Malik. Grouping techniques for update propagation in intermittently connected databases. *Proceedings of the IEEE International Conference on Data Engineering*, February 1998.

[15] S. Mahmoud and J. S. Riordion. Optimal allocation of resources in distributed information networks. *ACM Transactions on Database Systems*, 1(1):66–78, March 1976.

[16] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems*, 9(4), 1984.

[17] S. B. Navathe, K. Karlapalem, and M. Y. Ra. A mixed fragmentation methodology for the initial distributed database design. *Journal of Computers and Software Engineering*, 3(4), 1996.

[18] S. B. Navathe and M. Ra. Vertical partitioning for database design: A graphical algorithm. *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1989.

[19] M. Satyanarayanan. Coda: A highly available file system for a distributed workstation environment. *Proceedings of the IEEE Workshop on Workstation Operating Systems*, September 1989.

[20] Synchrologic, Inc. Decision criteria for synchronization and replication tools. Web Document, 1999. www.synchrologic.com/images/whitepapers/decision_criteria.html.

[21] Synchrologic, Inc. Synchrologic imobile data synchronization: Database synchronization for mobile/remote users. Web Document, 1999. www.synchrologic.com/images/whitepapers/imobile_white_paper.html.

[22] Synchrologic, Inc. Synchrologic imobile user guide. Included in version 2.41, 1999.

[23] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated data placement. *ACM Transactions on Database Systems*, 16(1):181–205, 1991.