A Greedy Approach For Improving Update Processing in Intermittently Synchronized Databases

Wai Gen Yee*, Ed Omiecinski†, Sham Navathe‡, Mostafa Ammar§, Jeff Donahoo¶, Sanjoy Malik
{waigen, edwardo, sham, ammar}@cc.gatech.edu, Jeff_Donahoo@baylor.edu

June 16, 1999

Abstract

Replication of data on portable computers is a new DBMS technology aimed at catering to a growing population of mobile database users. Clients can download data items such as email, or sales data from a server onto these machines, peruse it during commutes, and return any modifications to the server at the end of the day. In this paper, we describe how the servers in these systems generally process update information for clients and reveal a scalability problem–server processing increases quadratically with respect to increasing numbers of clients. We develop a cost model, and propose a solution based on heuristics. By aggregating client interests into datagroups, based on notions such as interest overlap, we can reduce server cost. These techniques are attractive because they are simple and computationally cheap. Simulations show that even simple techniques may yield significant performance improvements.

1 Introduction

The benefits of replication in distributed databases are well known, and include availability, performance, and reliability. The disadvantages include, most prominently, consistency management. However, in certain applications, strict consistency is not required, and new technology make replication much more attractive. For example, a salesperson may have to go on a sales trip, and would like be able to take along portions of sales data in his palm-sized, or laptop computer, and be able to make modifications on them "off-line," which would ultimately be propagated back to the server. An example of a modification would be an on-site recording of a sale. In the evening, he

*Contact Author: College of Computing, Georgia Institutute of Technology, Atlanta, Georgia, 30332-0280, waigen@cc.gatech.edu

†College of Computing, Georgia Institute of Technology

‡College of Computing, Georgia Institute of Technology

§College of Computing, Georgia Institute of Technology

¶Baylor School of Engineering and Computer Science, Waco, TX 76798-7356

can connect his portable to the server and "synchronize" their data by uploading his modifications and downloading others'.

As another common example, someone may wish to download his email, or certain sections of the day's newspaper onto his portable computer for perusal during a commute. Portable computers are becoming as common as cell-phones, and strict consistency in the sales data or the daily news is not as important as that of, say, stock prices or flight data.

As evidence to the importance of this technology, most commercial database software vendors, including Oracle, Microsoft, and Sybase, support replication for "remote" or mobile client databases. Besides database products, a class of applications exist whose sole purpose is to manage the "docking" of portable computers with their "base" computers.

With this technology, clients can work in isolation of the server, and defer the exchange of updates for a more convenient time. While disconnected, the clients and server states lose synchronization and become inconsistent. During intermittent reconnections with the server, both the client state is communicated to the server, and a synchronizing update file is generated for it. Accordingly, we define an *intermittently synchronized database system* (ISDB) as a database system which supports data replication for mobile clients.

Unfortunately, the basic ISDB architecture results in possible scalability problems with respect to increasing the number of clients. The server becomes a bottleneck as the number of clients it manages increases, because it is the means by which clients communicate with each other, and also simplifies consistency management by centralizing it. Hence, the ISDB processing architecture logically forms a star. Natural to star architectures, the server at the hub can be a source of congestion.

In particular, consider an ISDB with c clients. As the number of clients increases, the amount of update data from client to server can be expected to increase by $O(c)$. To handle replication of a change, the server must perform $O(c)$, one for each of the C clients. This cost increases *quadratically* with respect to the number of clients in the system. For example, assume that each of c clients generates t transactions, for a total of nt transactions. The server must check whether, for each of the c clients, whether each of these transactions apply to the client's interests, for a total of $c \times nt = c^2t$ work.

Some ISDB vendors have tried various approaches in reducing server cost. One possible method is distributing the work amongst multiple servers, so that each server is in charge of managing a subset of the clients. Either a hierarchical or a two-tiered structure is formed, and the servers as well as clients must be synchronized. This solution adds another level of complexity to ISDB operations, does not scale well, and does not reduce the asymptotic costs of server update processing.

Without modification, the server deals individually with *physical clients*. We call this *client-centric* server operation. An alternative approach, introduced by [MDNM98], called *data-centric*, takes advantage of the overlap in client subscription to reduce replication of work. Instead of

catering to the needs of physical client individually, interests are aggregated, and *datagroups* are managed instead. For example, given two clients that subscribe to the same parts of the server database, such as northern sales data, the server can just maintain one update file for both of them, instead of maintaining one for each. In this way, the server manages one logical client (in the form of an interest in northern sales data) where there are two physical ones, and, hence, does less work.

Actual details of ISDB operation, such as consistency management and conflict resolution are implementation dependent, and not the topic of this paper. Instead, we develop a conceptual model of ISDB operation, abstracting away all but the relevant details. We isolate each of its components and determine its operating cost. This model is general enough to be applicable to most well-known ISDBs.

In an attempt to solve the scalability problem, we frame it as a specialized database allocation problem, and propose greedy heuristics applicable to solve it. We like the greedy approach because its formulation is intuitive, simpler to implement and manipulate, and easy to compute. Other well-known allocation techniques, such as linear programming or dynamic programming may yield better results, but these heavyweight approaches tend to be too complex, which often inhibits their adoption into products. We use simulation to compare our results against theoretical best cases, and "boundary" solutions. We end this report by analyzing the effectiveness and applicability of our approach, and proposing future work.

2 Related Work

Work related to ISDBs has already been done in other contexts. [IB94][AK93] discuss issues related to mobile wireless computing, focusing on the field known as "mobile computing" "mobile" databases or "nomadic" databases. Much of the research in this field deals issues such as cell-handoff and resource location. These issues are not directly related to our problem. Of more relevance, it mentions concepts such as "client profiles" and server load. The former term refers to the notion that each client may require different information from the server. Server load becomes an issue as more clients are managed by that server. [IB94] suggests that to reduce server load, data may be broadcast, then filtered at the client. The client may also choose to cache data which it accesses often, then "reintegrate" it at a later time. These concepts are involved in this paper, and other work related to mobile systems.

CODA [Sat89] is a distributed file system that deals with varying or non-existant bandwidth between a client and file server. A special case of this work is total disconnection from the server, during which cached copies of frequently referenced data are used. The concept of a client cache is important, because the possibility of client disconnection is real, but, in CODA, cache contents are only a passively maintained.

Recent work addresses the advantages of active maintenance of client data interests. [LCD+99]

states that consideration of receiver interests in server objects can yield benefits by reducing network and client load. This is achieved by grouping "data flows" into "communication groups." [IB94] and [MDNM98], on the other hand, are concerned with server cost, in the face of an increasing number of clients. [IB94] and others [AFZ96] propose an extreme solution by having the server broadcast data, whereas [MDNM98] takes the middle-ground: proposing to aggregate data into datagroups, as in [LCD+99], but in order to save on server costs.

The work presented in this paper is an extension of [MDNM98]. There, the basic architecture of ISDBs are introduced. The major operations involved in such a system, such as update file processing and client side file merging filtration are described.

Although all of the above work mention that grouping data items intelligently has cost benefits, none define exactly what intelligent grouping is, nor do they precisely explain the benefits.

Distributed database design issues are directly related to this work. Fragmentation techniques are important considering the means by which ISDBs periodically transmit aggregated updates to clients, based on the fragments to which they subscribe. The needs for efficient fragmentation do not change, so, we do not consider it in this paper. Techniques for fragment allocation, however, must be specialized, considering that clients are not typically connected to the server.

Our work is also related to view maintenance [SJ96][GM95] and broadcast databases [DVCK98][AFZ97]. View maintenance is related to server processing, but ISDBs clients typically act without contact with the server, so view maintenance must be done in isolation and is the responsibility of the client. Broadcast updates deviate from ISDB operation because ISDB connections with the server are assumed to be connect initiated.

3 ISDB Architecture and Operation

There ISDB architecture involves a single server, multiple client architecture. Its distinguishing characteristic is that the clients generally work while not connected to the server. Hence, the clients maintain individual databases, which have a non-empty intersection with a global database located and managed at the server.

This intersection is typically called a *subscription*, and is defined in terms of predicates, which describes relations, or horizontal and/or vertical partitions of relations. The union of all clients' subscriptions is called the *publication* of the server database. Committed updates to the portions of the database defined by the publication are accumulated in transaction logs where they occur.

ISDBs are centrally administered at the server, so, accumulated modifications to shared data on the client are converted into *server update files*, that are sent to the server upon client connection. The server attempts to incorporate these modifications into the global database. After determining the new consistent state of the database, the server then generates *client update files* for each client. Each contains update operations that synchronize the client's view to that of the server, including

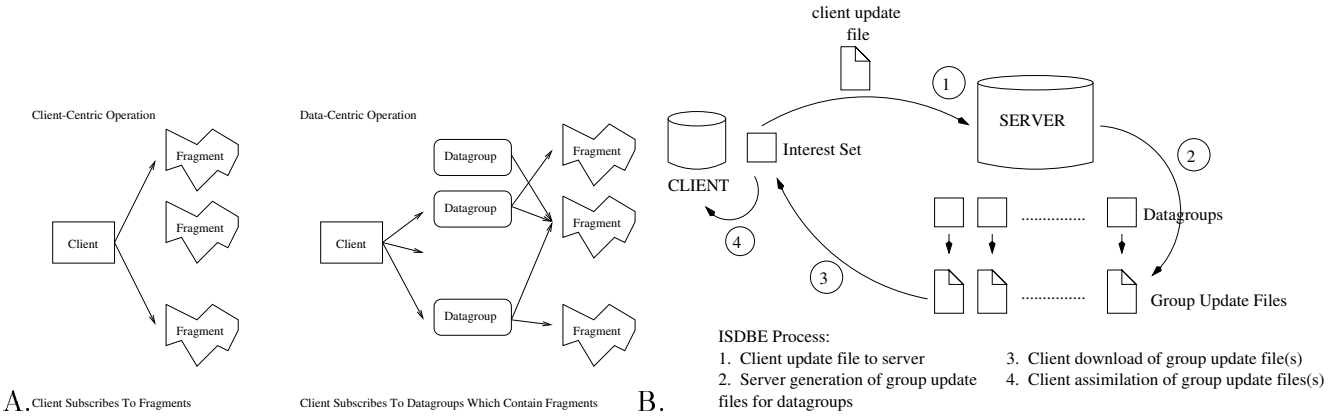


Figure 1: A. The difference between client-centric and data-centric operation from the client's perspective. B. The operation of an ISDB

operations which may undo client-initiated transactions. These files are generated based the client's subscription. When the client next connects to the server, it can download its unique client update file, and and incorporate it into its local database.

Conflicting updates are handled in various ways and different ISDB implementations have different methods of resolving this. This may result in one client's update overwriting that of another. For instance, one method which Sybase implements in SQL Remote allows update conflicts to be resolved thorugh the use of conflict triggers[Syb99]. This is a very important problem that all ISDBs address, but, we consider it orthogonal to the goals of this paper. However, the ideas proposed in current ISDB implementations are easily adaptable to our system.

In this paper, we abstract ISDB architecture. We assume that the database is divided into fragments based on the application. We refer to subscriptions as *datagroups* to reflect the notion that we are focusing on aggregate data subscription patterns rather than on physical client. Datagroups contain listings of fragments which the server uses to generate update files. The server contains a set of datagroups which constitute its "view" of the world.

Each client, on the other hand, has an *interest set*, which lists the fragments to which it subscribes. In client-centric operation, there is one datagroup per interest set, and they are equivalent to each other. In data-centric operation, however, this is not necessarily the case. See Fig. 1. For example, there may be no single group update file which *covers* its interest set, so, it may need to *subscribe* to additional datagroups to *satisfy* its interests. See Fig. 1. There may also be datagroups which over-satisfy its interests, in which it must filter out unnecessary data.

The union of the datagroups to which a client subscribes may form a strict superset of that client's interest set. In this case, the client has to *filter* superfluous data as well as redundant data (which came from itself) from the resultant set of group update files. These details are explained in [MDNM98].

For example, consider a university database with professor, registrar, graduate and undergraduate type users. In the interest set of the professor is student information and grade information. In the interest set of the registrar is student information and enrollment information. In the interest set of the student is student information.

Assuming there exists one of each type of user, under client-centric operation, three update files would be generated. One would contain student information, and grade information. Another would contain student information and enrollment information. The last would contain just student information. These three datagroups correspond exactly to the interest sets of each client. Notice that the contents of the update files overlap.

Under data-centric operation, one possible grouping of data that minimizes disk usage would be to generate one group with student information, one with enrollment information, and one with grade information. The professor could then subscribe to the student and grade information datagroups, the registrar would subscribe to the student and enrollment information datagroups, and finally, the student would subscribe to just student information datagroup. Each would then download the respective update file(s). In this case, the union of the datagroups to which each client subscribes satisfies the respective client's interest set exactly, so they do not need to perform any filtration of data from downloaded group update files. Also, the contents of the datagroups do not overlap, yielding disk usage savings.

3.1 Cost Model

We structure ISDB data distribution cost based on the activities shown in Fig. 1. We ignore the cost of getting the client's updates to the server because we are interested in server processing assuming that it already contains a transaction log full of updates. (This cost is therefore equal to that of the client-centric case.) The costs described below are all normalized into units of time.

Server generation of group update files for datagroups - One common ISDB approach is for the server to scan the log for each datagroup, and, if an element in the log "applies" to a datagroup, it gets saved in the corresponding update file. Hence, there is a *scan* cost and a *disk* cost.

$$\text{Scan Cost} = \frac{\text{\# Datagroups} \times \text{Cardinality of Transaction Log}}{\text{Server Disk Scan Rate}},$$

$$\text{Disk Cost} = \frac{\sum_{datagroup_i} size(datagroup_i)}{\text{Server Disk Save Rate}}$$

where $size(datagroup_i)$ is:

$$\text{Size of Datagroup i} = \sum_{fragment_j \in datagroup_i} weight_j size(log)$$

$weight_j$ is:

$$\text{Weight of Fragment j} = \frac{\#\text{ clients subscribing to } fragment_j \times size(fragment_i)}{\sum_{fragment_j} \#\text{ clients subscribing to } fragment_j \times size(fragment_j)}$$

and $size(log)$ is the size of the transaction log at the server in terms of bytes.

Client download of group update files - Each client must download a subset of update files corresponding to a subset of datagroups that covers its interest set. This cost is proportional to the time it takes for the server to load and all clients to download all necessary files. We assume transmission time amortizes other costs such as network protocol hand-shaking.

$$\begin{aligned}\text{Transmission Cost} &= \sum_{client_i} \sum_{datagroup_j \in subscription(client_i)} size(datagroup_j) \\ &\times \left(\frac{1}{\text{Network Transmission Rate}} + \frac{1}{\text{Disk Scan Rate}}\right)\end{aligned}$$

Client assimilation of group update files - The client receives the update files, and stores them into secondary memory. When all files have been downloaded, it merges them (assumed to be free) and scans the result to determine what is relevant. We use this simplified model for the sake of brevity.

$$\begin{aligned}\text{Client Processing Cost} &= \sum_{client_i} \sum_{datagroup_j \in subscription(client_i)} size(datagroup_j) \\ &\times \left(\frac{1}{\text{Client Disk Save Rate}} + \frac{1}{\text{Client Disk Scan Rate}}\right)\end{aligned}$$

We refer to the scan and disk costs together as *upstream costs* and the transmission and client processing costs together as *downstream costs.*

We now have a cost function of the form:

$$\begin{aligned}\text{Total Cost} &= \alpha(\text{Scan Cost}) \\ &+ \beta(\text{Disk Cost})\end{aligned}$$

$$+ \quad \gamma(\text{Transmission Cost})$$
$$+ \quad \delta(\text{Client Processing Cost})$$

It is this function that we are attempting to reduce. The coefficients, α, β, γ and δ are user definable weights for each cost component.

4 Heuristics

Our plan to reduce ISDB system cost is realized by modifying the *allocation of fragments to datagroups* while considering that *clients subscribe to a subset of the resultant datagroups.* The solution to the "university" example above is done "by hand," with high level knowledge of the application domain. Although the solution is not necessarily optimal, it does yield obvious savings. Our goal is to automate such grouping techniques using just the knowlege of client subscription, some database statistics (such as distributions of data), and of the capabilities of physical devices of the ISDB (such as disk drive speeds and network bandwidth).

Since our intention is to optimize pre-existing client-centric ISDBs, we assume that fragments are already defined–a reasonable assumption, considering that client subscriptions are defined in terms of fragments–and that we know to which fragments each client subscribes.

We structure our discussion by drawing a few basic conclusions on ISDB cost, and developing rough means by which the cost can be reduced. We then iteratively refine the cost considerations, and the means of cost reduction. Based on the cost equations, and our knowledge of the architecture of an ISDB, we conclude a few things:

- Reducing the number of datagroups reduces the scan cost required. In the extreme case, a server would maintain a single datagroup composed of the union of all subscriptions.
- If there exists overlap in datagroup contents, then some of the contents of the transaction log may be written to multiple update files–i.e., disk cost is not minimized.
- We cannot improve downstream (transmission and client) costs over those of client-centric operation, because, without modification, an ISDB sends the minimum amount of data to each client. Consequently, improvements in upstream (scan and disk) costs can be at the expense of increased cownstream costs.

As we mentioned in the introduction, there may exist an ISDB installation in which multiple clients share interest sets. In this case, the server can reduce the amount of work that it does by just maintaining a single datagroup for all these clients. Scan and disk costs are reduced, while downstream costs remain unaffected.

However, even if two datagroups are not identical, they can be *merged* to yield a net cost reduction. For example, if their contents *overlap* then, depending on the degree of overlap, the

savings in both scan and and disk cost can outweigh the cost increase in downstream costs. Even if their contents do not overlap, in some circumstances, it is still possible to reduce the net cost by merging as we will explain later. We can now describe three types of merging. All three naturally reduce scan cost, but have different side effects.

Merge-bytes - This type of merging focuses on minimizing the disk cost by merging the two datagroups whose common fragments' total sizes (in terms of bytes each fragment consumes in the database) are maximal. In this case, when the group update file for the resultant datagroup is generated, the reduction in disk usage is proportional to the total sizes of their common fragments. In other words, we merge the two datagroups with the maximum bytes their common constituent fragments have in common, or *byte overlap*, where byte overlap between datagroups i and j is computed as:

$$\text{Byte Overlap} = \sum_{fragment_k \in (datagroup_i \cap datagroup_j)} size(fragment_k)$$

Merge-degree - This type of merging attempts to reduce scan and disk costs, but also has the lowest degree of impact on downstream costs. The ratio of the size of the fragments they have in common and the size of the unique datagroups they contain yield the *degree overlap*:

$$\text{Degree Overlap} = \frac{size(datagroup_i \cap datagroup_j)}{size(datagroup_i \cup datagroup_j)}$$

Merge-degree merges the two datagroups with the greatest *percentage* of common contents. This can be different than the effect of merge-bytes because the two datagroups with the greatest degree overlap may be small, while there may exist two datagroups with low degree overlap, but with a larger intersection in terms of bytes.

Merge-degree has a lower impact on downstream costs, since by maximizing the degree of overlap of the merged datagroups, we are minimizing the degree of difference, resulting in less superfluous data being sent to each client.

Merge-small - Merge small is used when scan cost still must be reduced regardless of overlap. The two smallest datagroups are merged because, without regard to the effect on disk cost, scan cost is reduced, and, because the datagroups are of minimum size, the impact on downstream costs is minimal.

Considering merging, we can draw more conclusions about the allocation of the datagroups.

- Merging a client's datagroup with that of another does not necessarily result in a datagroup that is equal to that (or any other) client's interest set. The result is generally a superset

of its interest set. Hence, its corresponding update file will contain superfluous data, and increase downstream costs.

- Smaller datagroups can be allocated more easily because their small size has a smaller impact on downstream costs.

The main reason to merge datagroups is to reduce scan cost. However, depending on the hardware involved and on the values of the weights of the cost components, it may be best to reduce the other costs. We propose the processes of *splitting* and *contracting*. Effectively, splitting, is the inverse of merging, and results in datagroup generation. Conversely, contracting gets rid of certain redundancies (explained below) in datagroups.

Split - The main goal of splitting is to reduce disk cost. A fragment that is contained in multiple datagroups is a candidate to be split out of all those datagroups, and placed into a new datagroup by itself. This operation necessarily has an adverse effect on scan cost, but another side effect may reduce downstream costs in special cases. Consider a case where a client only needs fragment A, but not fragment B. If the smallest datagroup in which fragment A is contained also contains fragment B, then the client must receive superfluous data intended for fragment B when downloading this datagroup. However, if A is split into a separate datagroup, the client can receive exactly what it needs.

In terms of disk usage saving, it is best to perform a split on a fragment that maximizes:

$$\text{Excess Fragment Disk Usage} = sizefragment \times (\text{degree of replication} - 1)$$

Excess Fragment Disk Usage is a measure of how much space a fragment's replicas consume.

Contract - It is possible that one datagroup is a subset of another. If datagroup A $\subseteq$ datagroup B, contracting replaces B with B-A, and, force all clients which once subscribed to B subscribe to A as well. Contract reduces the average size of datagroups, and saves disk space. There are no negative side effects to performing this operation. See Fig. 2 for examples of each operation.

4.1 Developing a Heuristic

We have now described a set of concepts which can be used to compare pairs of datagroups and fragments, namely byte overlap, degree overlap, intersection and degree of replication, and how they are related to deciding which of a set of operations to perform on an ISDB. We have also described five operations to perform on the ISDB objects, each with varying effects on cost components.

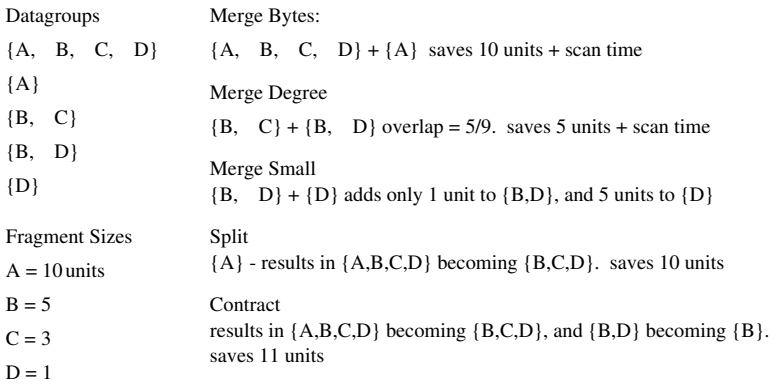


Figure 2: Examples of each operation

	scan cost	disk cost	transmission cost	client processing cost
merge-byte	$\downarrow$	$\Downarrow$	$\Uparrow$	$\Uparrow$
merge-degree	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$
merge-small	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$
split	$\uparrow$	$\downarrow$	$\cdot$ or $\downarrow$	$\cdot$ or $\downarrow$
contract	$\cdot$	$\downarrow$	$\cdot$	$\cdot$

Table 1: Various operations on ISDB objects and their likely effects on cost components. Arrows up and down indicate increased and decreased cost, respectively. Double arrows indicate greater effects relative to single arrows. Dots indicate no effect.

The first heuristic attempts to minimize the total cost by minimizing the highest individual component with the appropriate operation. For instance, if disk cost were at a premium, and transmission cost were unimportant, the appropriate operation is merge-byte. Conversely, if we want to conservatively reduce server costs while minimizing the impact on downstream costs, merge-degree would be appropriate.

This heuristic orders the operations to perform, based on the values of the cost components, seeking to minimize the largest one. If the first operation in the ordering succeeds, the costs are recomputed, the operations are reordered, and another one is performed. If the operation fails, then, the others are tried in order until one succeeds. If none succeeds, then the heuristic ends. We call this the *directed* heuristic.

An alternative heuristic is to check the effect of each operation and apply the one with the greatest negative effect on the cost at the time, and repeating until no operations reduce the cost. We call this the *exhaustive* heuristic.

The advantage of the directed heuristic over the exhaustive one is reduced execution time. The exhaustive heuristic actually applies and undoes all operations before it decides to perform one, so the directed heuristic can conceivably be on the order of n times faster, where n is the number of operations available.

Intuitively, it seems that the exhaustive algorithm should achieve a better end result, but one

cannot be so sure with greedy algorithms. Empirical evidence will be used to determine which is the best.

4.2 Weights

In previous sections, α, β, γ, and δ were introduced. The database administrator can assign any values to these. However, they were conceived as a way of giving him flexibility in prioritizing one component of ISDB cost over another, or tuning the heuristic results. No other input is designed to be as user-dependent.

Here is an example of how one may assign values to α, β, γ, and δ. Consider a case where the accounting department of some company just took a cost inventory, and determined that it costs \$.04, \$.02, \$.01, \$0.2 to scan data, save data, transmit data over the network, and have the client machines process data, respectively. Perhaps some high speed fiber cables have been installed, or there is a worldwide shortage of disk controllers affecting these prices. Perhaps the company president is often an ISDB client, and wants the quickest client service. In this situation, reasonable values of α, β, γ, and δ are .04, .02, .01, .20, respectively.

4.3 Thrashing

Whenever there is a heuristic where one operation has one effect, and another has the opposite, or nearly the opposite effect, we must be wary of the possibility of thrashing–having the two functions alternatively cancel each other out. In particular, split operations increase the number of datagroups, whereas merge operations decrease them, and each have different side effects.

Thrashing does not happen with the given operations. A precondition for splitting a fragment is that the fragment must be replicated. Since merging does not replicate fragments, a fragment can be split at most once, and only if it is replicated, hence eliminating the possibility of thrashing.

5 Experiments

We describe the results of three of the experiments we conducted. The first varies the number of clients. The second varies the degree of fragments subscription by a client, and the last varies the values of α, β, γ, and δ. These tests were performed on both uniformly distributed data and data distributed in the 80%/20% proportion–20% of the fragments are subscribed to by 80% of the clients. Both the directed and the exhaustive heuristics were used.

In the experiments, where applicable, we compare four different approaches: the single datagroup approach (**One Datagroup**); the client-centric approach (**Initial**); the exhaustive approach (**Exhaustive**); and the directed approach (**Directed**). (The labels in the parentheses refer to the labels used in the plots below.) The single datagroup approach is a boundary solution that

attempts to minimize upstream costs without regard to downstream costs.[1] The other approaches have already been described above.

In addition to computing these cost for these approaches, we compute a *minimum* (**Minimal**) cost. This minimal cost is a theoretical minimum that may not be achievable by any approach. It is the sum of all the cost components simulataneously minimized, which is not generally achievable. As we do not know the actual minimal solution this value is included as lower bound for cost.

5.1 Description of Experimental Setup

Tests are simulated. Inputs into the simulation include:

Name	Description
α	Weight of server scan cost component
β	Weight of server disk cost component
γ	Weight of transmission cost component
δ	Weight of the client processing cost component
Number of Clients	Number of Clients
Min, Max Fragment Size	Fragment sizes are random, between min and max
Min, Max Number of Fragments	Number of fragments are random, between min and max
Min, Max Log Size	Transaction log size is random, between min and max
Cscan	Server disk reading rate
Cdisk	Server disk writing rate
Ctransmit	Network transmission rate at the server
Cclscan	Client disk reading rate
Ccldisk	Client disk writing rate
Minimum Fragment Subscription	Minimum number of fragments to which a client must subscribe
Max Fragment Subscription Percentage	Maximum percentage of fragments to which a user may subscribe. The actual number of fragments to which a user subscribes is between min and this percentage of the total number of fragments

The log size is the cardinality of the transaction log at the server. The composition of the log, in terms of the percentage of operations contained within it referring to a particular fragment, is proportional to the weight of that fragment. In the following tests, unless otherwise noted values for the variables are:

[1] The single datagroup approach is similar in spirit to update broadcast approach.

Name	Value
α	1
β	1
γ	1
δ	1
Number of Clients	50
Min, Max Fragment Size	1000, 5000 (KB)
Min, Max Number of Fragments	100, 500
Min, Max Log Size	1000, 5000
Cscan	1
Cdisk	1
Ctransmit	1.25
Cclscan	1
Ccldisk	1
Minimum Fragment Subscription	1
Max Fragment Subscription Percentage	50%

These values correspond to an ISDB operating on a LAN connected via a 10 megabit Ethernet. The server and client hardware are similar, thus yielding similar performance numbers. This may be the case in a small enterprise, that cannot afford a high performance mainframe as a server, but opt for a cheaper high-end desktop PC.

The client's connection to the server may be via modem and some intermediate network, but the server is connected to the to the network via a high-speed LAN. It can thus communicate with as many clients as possible by multiplexing its entire bandwidth.

Replication may be high, as clients may subscribe to as much as 50% of the database. This number may seem high, but in not inconceivable in the case where the global database is small. For example, in a sales database of a startup company, a supervisor may subscribe to all the fragments of his supervisees–a high percentage of the total database.

In this first cut approach, the database administrator has chosen to weight each cost component equally. For each test, ten trials are performed, and their sums averaged.

5.2 Varying Number Of Clients

In these tests, we vary the number of clients from 5 to 95. The effect of increasing the number of clients turns out to actually be marginal on the effectiveness of the heuristic versus the client-centric solution. The effectiveness of the heuristics remains constant as the number of clients increase. After applying either heuristic, the values returned by the cost function are consistently reduced by about 1% under uniform distribution, and 8% under 80/20 distribution when compared to the values of the client-centric cost. One explanation is that, although the absolute number of client pairs with highly overlapping interests increases as the number of clients increases, the percentage of clients with highly overlapping client pairs remains the same, hence, the percentage

of savings remains the same.

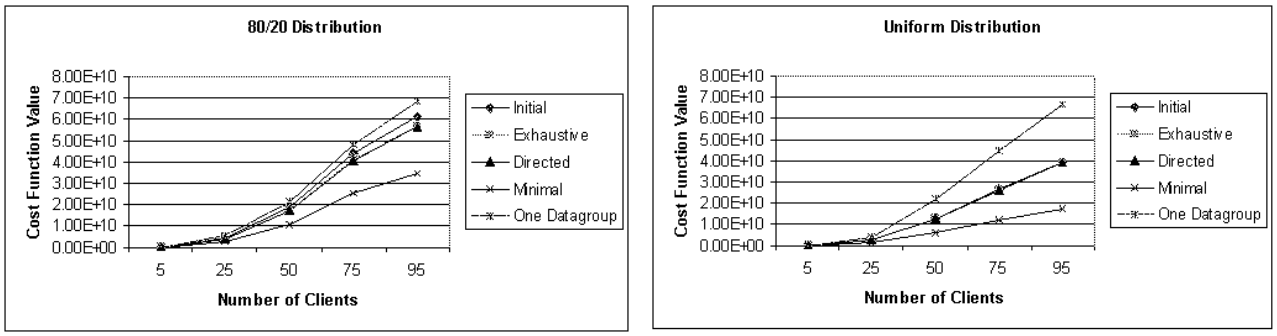


Figure 3: Increasing Number of Clients

Savings is greater under the 80/20 distribution, because expected overlap is increased by the increased probability of clients to subscribe to a fixed subset of fragments. The savings are in the area of 8%. Notice also how close the single datagroup cost is to the initial and post-heuristic solutions in the 80/20 distribution. This is the result of having restricted subscription patterns and a limited percentage of subscriptions to fragments. Since most clients have the same interests, it is more likely that their interests can be aggregated without penalty in terms of downstream costs.

Notice also that although post-heuristic savings compared to the those of the client-centric solution are small, but, savings compared to the single datagroup solution are significant–about 40% with uniform distribution, but much lower with an 80/20 distribution (about 14%). (See Fig. 3.) Again, as access to fragments follows a stronger pattern, more interest overlap occurs, and solutions to the different approaches converge. Under uniform distribution, compared to the minimum values, the client-centric and heuristic approaches have twice the cost, whereas the single datagroup solution has three and a half times the cost.

5.3 Varying Degree Of Client-Fragment Subscription

In these tests, we vary the maximum percentage of fragments to which each client subscribes from 5% to 95%. As the number of fragments to which clients subscribe increases, the effectiveness of the heuristic should also increase, because the average overlap between clients should increase. This diminishes the negative effects merging has on transmission costs.

As we see in Fig. 4, the effectiveness of the heuristics do increase. In the 80/20 distribution case, the rate of savings increases from 3% at subscription rates bounded by 20% to 12% when the bound is at 95% over cost values from the client-centric case. We can indirectly infer that merging is happening, because the costs are converging toward the single datagroup case as subscription rate increases. These costs are about 50% higher than the minimal costs, which is high, but this is a great improvement over relationship amongst these costs as compared to those described in the increasing client case above.

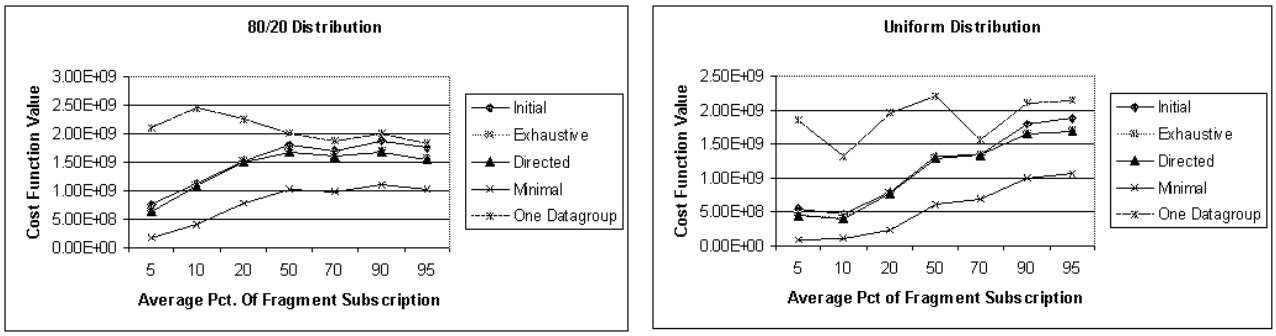


Figure 4: Increasing Degree of Client-Fragment Subscription

5.4 Varying α, β, γ, and δ

In these tests, we aggregate the downstream costs, because the operations do not distinguish them– transmission and client processing costs are both are affected in the same way by them.

These tests simulate the actions of a DBA who is manipulating the weights based on *external* factors such as cost to save bits versus cost to send bits over the network. We show how different weight allocations affect the effectiveness of the heuristics.

Balanced weighting The heuristic does very little in this case. The increasing client tests (See Fig. 3.) use balanced weighting. The benefit of merging datagroups of low overlap is outweighed by its downstream side-effects. Splitting saves on disk cost, and does not affect downstream costs, but its benefit is counterbalanced by the increased scan cost. The operation that dominates in this case is contracting, which seldom happens. Increasing the priority in downstream costs beyond this has little effect.

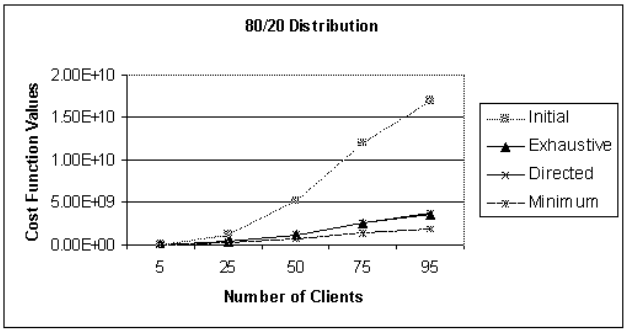
High priority on disk cost Although merging saves on scan cost, it is also a viable operation to save on disk cost. In this case, the benefit of merging is limited by the degree of subscription of clients to fragments, because, this limits the potential overlap between pairs of datagroups. Heuristic savings are similar to those of scan cost optimization (described below), except that savings over client-centric costs are about 45%. Because of the similarity in effect, we combine their discussions.

High priority on scan cost Many merge operations happen in this case, resulting in very few datagroups. In this case, the heuristics are very effective, since all merge operations reduce scan cost and possible disk cost. We yield savings of around 80% over client-centric costs, given α=.8. (See Fig. 5A.)

Notice cost values using the directed and exhaustive heuristics overlap with those of the single datagroup heuristics. When α is high, very few datagroups are formed. In this case, these two approaches yield solutions that are still about 50% more than the minimal, but, the client-centric approach costs are much worse–about 800% more.

Although, weighing either upstream costs make the heuristics active, in our experiments, high α values yield the more percentage savings than do high β values. This is the case because of the limited possible degree of client subscription to fragments. The degree overlap, and hence possible disk cost savings is limited by the degree of subscription of clients to fragments, especially is subscription is uniformly distributed.

A.



B.

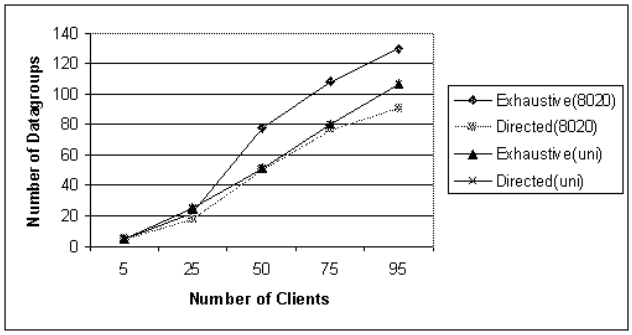


Figure 5: A: Varying the weights, where $\alpha = .8$, $\beta = \gamma + \delta = .1$ B: Comparing the number of datagroups formed versus the number of clients when $\alpha = .1$, $\beta = \gamma + \delta = .45$

Something interesting happens when both downstream and disk cost are given high priority. The number of resultant datagroups is regularly greater than the number of clients. (See Fig. 5B.) This is a result of prioritizing saving disk space, without disturbing downstream costs, and without regarding scan cost. Even more interesting is that, in the 80/20 distribution case, the number of datagroups is different in the exhaustive case than in the directed case, yet, experimental data shows that their savings is practically the same. This is evidence that the same savings can be achieved in different ways with these two heurisitics.

6 Discussion of Heuristics

The heuristics proposed in this paper have some nice properties. They are simple and intuitive, simple to implement, computationally tractable, and most importantly, they provide important benefits to their users. The concepts of client interest and their organization is not new, nor are the processes of merging and splitting, however, their applicability to the unique needs of ISDBs is.

In our tests, our heuristic solution never equaled the theoretical minimal, but always beat the single datagroup solution and client centric solutions, by up to 400%. However, as we have shown, the results vary depending on the context in which these heuristics are used.

6.1 Exhaustive Versus Directed Heuristic

The effectiveness of the heuristics is independent of the order of the inputs. It also seems to be independent of the order in which operations are applied. In our experiments, the savings, over using the client-centric approach, of using the exhaustive heuristic beat the directed ones by at most a couple of percent. However, the exhaustive heuristic ran up to 4 times longer. Depending on the scenario, one may choose the directed one without sacrificing too much performance.

6.2 Relationship To Variability In Client Subscription

As the pattern of data subscription in an ISDB becomes more regular and more constrained, these heuristics also tend to improve. As we saw, in all tests, the 80/20 distribution scenario improved heuristic results, because these patterns increase the chance of overlap, which increases the applicability of the cost saving methods.

6.3 Security Issues

As described in [MDNM98], the client merges update files it downloads from the server, serializing the contents, then filters out superfluous data–such as redundant updates from itself, repeated updates to the same data item, or data from unsubscribed fragments–and finally integrates the remaining data into its local database.

Security is a major problem in generating aggregated groups of data, because some of that data may be restricted. Although this problem is not strictly in the realm of allocation, we can mention a straightforward solution. Each transaction can have appended to it a fragment identifier. All transactions related to that fragment contained in the update file are encrypted with a unique key. The client can then only process it if it can decipher it with the proper key.

6.4 Applicability Of This Work

As we saw, these heuristics offer only marginal advantages in some situations. In client-centric ISDB operation, downstream costs are already optimized. If these costs are the major priority, then only contractions can definitely be performed. However, since these heuristics are cheap to perform, even attempting marginal improvements is feasible.

However, in some situations, major cost reductions can be yielded. As mentioned above, if client access patters are constrained to fixed subsets of data, or, if a DBA wishes to reduce his server load, then, these heuristics are very handy.

7 Conclusion and Future Work

In this paper, we have attacked a problem that is particular to ISDBs, namely the unscaling update processing cost by intelligent fragment allocation to datagroups. This allocation problem is caused by the disconnected environment in which clients exist, which forces the server and clients to communicate via log file transfer. The scalability in server update processing is described and heuristic solutions are proposed. We have shown that, by exploiting certain aspects of data subscription, such as overlap in datagroups, paying consideration to hardware device capabilities, we are able to improve the performance of ISDBs. The two heuristics proposed are shown to be equally effective in realistic cases.

This problem's treatment is very qualitative, aiming to just introduce concepts that will be explained in more detail in future work. Our aim was to introduce some concepts and promote discussion on the most salient concepts of ISDBs that will be explained in more detail in future work. The heuristics described in the paper by no means constitute a finished work. Rather, the work is more general and form a first cut conceptual approach, to which others can add or remove features based on their particular needs.

We will continue work to determine the best and worst case functioning of these heuristics and how robust their solutions in the face of changing variables, such as pattern of fragment subscription.

We will also gather information on real world usage of ISDBs to attack particular problems that arise. One common practice of ISDB users is to *realign* their subscriptions–change their interest sets. For example, a northern salesperson may suddenly need sales data from another region. Although it is not contained in his interest set, he expects to be able to download it from the server upon the next connection. This translates to a client desiring to add fragments to his interest set. Although realignment has been studied in the general distributed database setting, we are currently developing a way of doing this with our heuristics.

Another common situation for ISDBs is that different users may try to synchronize with the server at different frequencies. How access frequencies can affect these heuristics should be studied.

The effectiveness of the allocation of fragments is also limited by the fragmentation scheme. Applicability and consequences of methods such as vertical and horizontal [NKR96][CNW93] must be studied.

The heuristics given in this paper aim at optimizing upstream costs. We are also exploring ways of improving downstream costs as well. Currently, a popular method of update transmission is message-based, using store-and-forward techniques to offload data dissemination. With aggregation, as datagroups become more and more common to an increasing number of clients, reliable multicast may be a viable dissemination method.

We are currently working on incorporating all these improvements in a commercial product.

References

[AFZ96] S. Acharya, M. Franklin, and S. Zdonik. Disseminating updates on broadcast disks. *22nd International Conference on Very Large Databases*, September 1996.

[AFZ97] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proc. ACM SIGMOD*, Tucson, Arizona, May 1997.

[AK93] R. Alonso and H. Korth. Database systems issues in nomadic computing. *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1993.

[CNW93] S. Ceri, S.B. Navathe, and G. Wiederhold. Distribution design of logical database schemas. *IEEE Transactions on Software Engineering*, 9(3), July 1993.

[DVCK98] A. Datta, D. VanderMeer, A. Celik, and V. Kumar. Adaptive broadcast protocols to support efficient and energy conserving retrieval from databases in mobile computing environments. Accepted for publication in ACM TODS,, 1998.

[GM95] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering*, 18(2), June 1995.

[IB94] T. Imielinski and B.R. Badrinath. Wireless computing: Challenges in data management. *Communications of the ACM*, October 1994.

[LCD+99] 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*, January 1999.

[MDNM98] S. Mahajan, M. J. Donahoo, S.B. Navathe, and M. Ammar S. Malik. Grouping techniques for update propagation in intermittently connected databases. *Proceedings of the IEEE 14th Int. Conf. on Data Engineering*, February 1998.

[NKR96] 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.

[Sat89] M. Satyanarayanan. Coda: A highly available file system for a distributed worstation environment. *Proceedings of the Second IEEE Workshop on Worstation Operating Systems*, September 1989.

[SJ96] M. Staudt and M. Jarke. Incremental maintenance of externally materialized views. *22nd VLDB*, 1996.

[Syb99] Inc. Sybase. Sql remote: Replication anywhere. *www.sybase.com:80/products/system11/remote2.html*, 1999.