A Mathematical Optimization Approach To Improve Server Scalability In Intermittently Synchronized Databases

Wai Gen Yee, Sham Navathe, Anindya Datta, Saby Mitra
{waigen@cc, sham@cc, adatta@cc, saby.mitra@mgt}.gatech.edu

June 11, 1999

Abstract

This paper addresses a scalability problem in the process of synchronizing the states of multiple client databases which only have deferred access to the server. It turns out that the process of client update file generation is not scalable with the number of clients served. In this paper we concentrate on developing an optimization model to address the scalability problem at the server by aiming for an optimal grouping of data fragments at the server given the "interest sets" of the clients - the set of fragments the client deals with for its"local" processing. The objective is to minimize the total cost of server operation which includes processing updates from all clients and transmission cost of sending the right set of updates to each client based on the client's interest set. An integer programming formulation is developed and solved with an illustrative problem, yielding interesting results.

keywords: distributed database, client-server database, database synchronization, database replication

1 Introduction

With the advances in telecommunications and database technologies, information processing is reaching new frontiers. Distributed databases and client-server databases currently provide solutions to a large number of applications where data is either centralized on a server or distributed physically on the nodes of a network and a variety of applications can run at client locations. The mobile database community is considering the problem of databases that move with the user who are users of cellular and satellite communications, and they can be reached from a server and updates or fresh copies of data can be sent to them. World Wide Web databases allow anyone with Internet access to act as a user of a database if appropriately served by the database management system (DBMS) at a web site.

In this paper we are addressing a different and novel scenario which promises to become very commonplace with the work forces in many types of work: all kinds of sales, particularly in pharmaceuticals, consumer goods, industrial parts; insurance and financial consulting and planning; real estate or property management or maintenance activities, etc. In these applications, a server or

a group of servers manages the central database and the clients carry laptops or palmtops with a resident (DBMS) software to perform "local" transaction activity for most of the time. The clients connect via a network or a dial-up connection to the server typically for a half-hour to one-hour session. They send their updates to the server and the server in turn, must apply them to its central database, and, in turn, prepare appropriate copies of the updates applicable to the clients.

Each client receives a batch of updates to be applied to its local database whenever it connects. The primary characteristic of this scenario is that the clients are mostly disconnected. Clients connect for short periods of time and expect to be "refreshed" on demand during the connection. The server is not typically able to contact the client. We feel that this environment draws from previous work in distributed and client-server databases, and some from mobile databases, but is very unique and presents several research problems for investigation.

The term used by industry to describe the process of refreshing is *synchronization*, which basically refers to deferred client-server access. Hence, we will refer to the present environment as *Intermittently Synchronized Database Environment* (ISDBE), and the corresponding databases as *Intermittently Synchronized Databases* (ISDBs). The clients are referred to as *ISDB users*.

Before defining the scope of our research, we identify the salient features of the $ISDB$ with a simple example.

A company selling water purifiers has mobilized a national sales-force. Each member of the sales-force has some degree of autonomy in conducting business with clients, and must record this sales information, such as recording sales, marketing data, etc., on a *client* machine. Because of the itinerant nature of his work, the client machine is typically a laptop containing a subset of the *server* data located at headquarters. This laptop must generally be *disconnected* from the server as well, because on-demand *connection* inhibits his responsiveness, and constant connection is wasteful.

The salesman typically refreshes his copy of the relevant data on the client before he sets off on a sales trip. Then, during the trip, he may run transactions against this data, reflecting sales he has made, or information he has received. These transactions are saved in a local log. When convenient, the salesman connects to the server (e.g., via modem or Internet connection) and sends the local update log to it. In turn, the server prepares update files reflecting these changes to the database, and sends them back to clients upon connection.

This example helps to identify the following characteristics of $ISDB$'s that establish the need for new research in this area: (a) A client connects to the server when it wants to receive updates from a server or send its updates to a server or process transactions that need non-local data; (b) A server cannot connect to a client at will; (c) A client is free to manage its own data and transactions while it is disconnected; (d) A client may have multiple ways of connecting to a server and may choose a particular server to connect to.

Synchronization of data is already recognized be industry as important in any business with replicated data[Als98]. Among major companies doing research and development in there areas are:

Millions of dollars in research and development are being spent by both blue-chip and start-up companies such as:

- Lotus (IBM)
- Starfish Software (Motorola)
- Puma Technology
- Synchrologic
- Planetall (Amazon.com)
- Visto

In this paper, we begin at the server by describing the particular problem of scalability in the design of databases and subscription of clients for parts of the server databases. We introduce some of the terminology and develop an optimization model which is then described and explained. We then report on the results of solving the optimization model for a small illustrative problem. Finally, an interpretation and execution of our work is presented.

1.1 Relationship to previous work

With the ISDB environment as defined above, the goal of this paper may be succinctly stated as:

design the server database in the ISDBE by grouping the data fragments at the server so that the overall cost of dissemination of updates at the server is minimized.

The above addresses the scalability of server processing by making the complexity of it proportional to the number of data groups as opposed to the number of clients. We discussed the rationale behind grouping and the details of the client side and server side processing in [MDNM98].

This paper is an attempt to develop a usable mathematical optimization model that can be converted into a design tool for ISDB environments. Specifically, this paper develops an analytical formulation by which the design of groups may be attempted. The actual design of horizontal (and vertical) fragments may be done using the horizontal [CNW93], and mixed fragmentation methodologies [NKR96] based on the knowledge of most frequently used transactions.

The proposed approach draws from our experience with an original optimization model we developed for horizontal partitioning [CNW93]. In the former we dealt with allocation of horizontal fragments to sites of a network. Here, we deal with assignment of fragments to groups. In the former we considered costs of data access and transmission for processing the most important transactions. Here, we consider the server processing and communication cost of updates originating at the clients. Both models result in a non-linear optimization model and would need further development of heuristics to implement practical design tools. This paper provides a mathematical optimization framework for the grouping problem just as we did in [CNW93] for the horizontal fragmentation problem.

On the surface, the ISDB environment seems to resemble the mobile computing scenario. However, it is important to contrast the former with a variety of problems found in the mobile database and mobile computing literature. Issues for mobile databases are well explained by [AK93], [IB94], [FZ96]. Work by Acharya et al. on broadcast disks has addressed problems related to repetitive broadcasts [AAFZ95], update dissemination [AFZ96] and have recently considered the use of mixing pull with push [AFZ97] to allow clients to pull information. They consider many related problems. Imielinski and Badrinath [IVB97] have discussed a series of related problems under "data on air" approach similar to above. Datta et al. [DVCK98] address protocols for retrievals by mobile clients focusing on power conservation.

While many of the above works have addressed query processing by mobile clients and update propagation from server to mobile clients, there are some significant differences between our context and the mobile database context as follows:

1. In our environment, unlike that of mobile databases, the issue of wireless communication is irrelevant. Typically, the work we have been doing with a commercial prototype uses dial up networks and the Internet.
2. Cell boundaries, energy conservation/battery power etc. are irrelevant. In other words, mobility is not important has no particular impact on the overall problem.
3. Clients freely manage their own data and process transactions while not connected to the server.
4. Most importantly, the notion of "broadcasting," which plays an important role in virtually all mobile database work, does not play a role. While multicasting is identified as an area of future studies, in this problem, we assume an on-demand strategy employed by the clients to get data from the server.

There is also a body of work on the incremental view materialization problem that is related to our work. For example, Concept-Base is a client-server architecture [SJ96] where differential changes are computed by the server and propagated to clients. They deal with this as a monitoring service without assuming knowledge of original materialization and with no database capabilities at the server. Previous works on materialized views–in data warehousing and such other contexts (e.g., [GM95]), mostly consider propagating updates from "base tables" to views. Work related to bandwidth utilization by multitasking has a direct bearing on the ISDB scenario (e.g., [CA97]). We are actually evaluating a demand based unitasking pull with multitasking as a future possibility in our work on ISDBs.

2 Problem Description

In our environment of "intermittently synchronized databases," we assume that servers are always ready to accept client connection, and are continuously operational. We assume no mutual connectivity between clients, as it is often impractical. A client connects to a server (the mode of connectivity

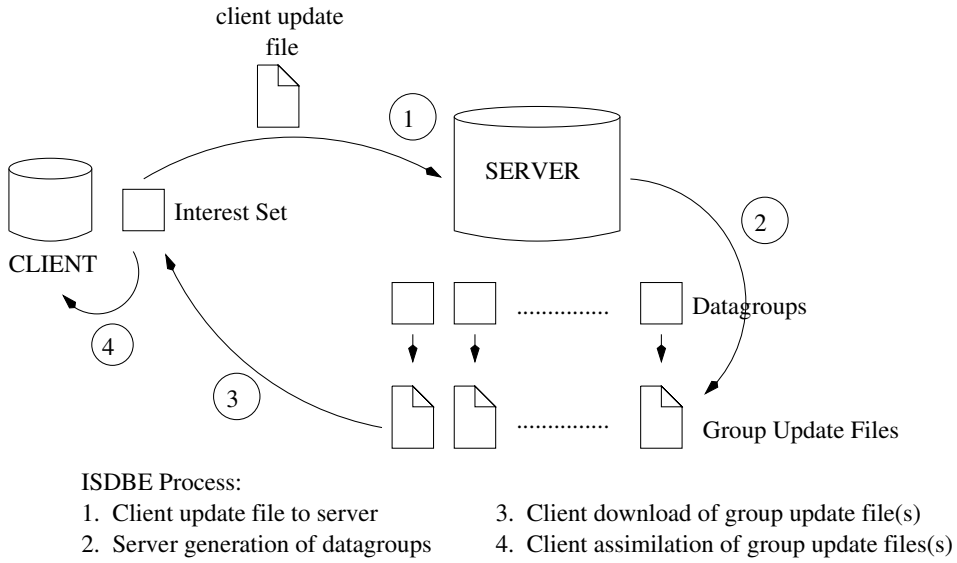


Figure 1: ISDBE process

is irrelevant for the purposes of this paper) and performs some local processes, followed by some operations under complete disconnection from the fixed server network. Whenever a client connects to the server network, it may connect to any node on the network based on factors like availability, location, or convenience.

In this paper, we focus on server scalability. There is a need to accumulate the updates or the effects of these updates performed at the clients during disconnection from the server network. These accumulated updates/results would be shipped to the server network upon connection. As there is no connectivity among clients, the servers maintain consistency by propagating the updates amongst the clients to maintain global data consistency in the system. The workload and disk space requirement at the servers in the fixed network increases quadratically in relation to the number of clients if the servers handle updates on a per-client basis–the server must both collect updates from clients, and, for each client, generate an update file. We call this *client-centric*[MDNM98] processing to reflect the notion that clients are handled on an individual basis.

We propose *data-centric* processing–defining aggregate groups of data–which takes advantage of certain patterns of client data subscription, such as overlap, by combining such data for multiple clients into a single file. By reducing the amount of data that is written and by reducing the number of client update files that need to be maintained, it is intuitively obvious that server update processing cost is reduced. As the number of clients or the volume of overlapping data increases, the savings also increase. Data-centric processing results in the number of data groups (which correspond to files used to update client sub-databases) being system tunable. One of the negative side effects of data grouping is that excess data may exist in a group. For example, if the update files of two clients are combined, then, unless their update files are identical, the combination will contain more data than necessary for at least one client. (We assume that excess data is filtered on the client side.)

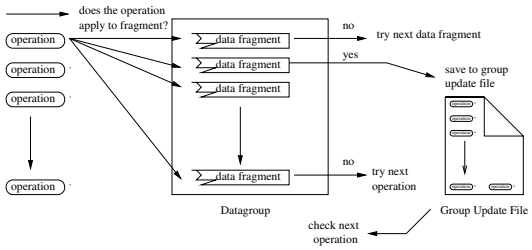


Figure 2: Server Processing Diagram

In terms of server side processing, this side-effect manifests itself in increased file transmission time (relative to client-centric transmission time). Hence, an inherent tradeoff exists between amount of server processing and update file transmission time.

The main components of the ISDBE are the server and the clients. The server database is divided into *data fragments*, which are non-overlapping horizontal and/or vertical partitions of the database. Each client contains a *sub-database*, which can be defined in terms of a subset of the server's data fragments. This set of data fragments is called the client's *interest set*. The client may keep other data for local processing in which the server has no interest. The server contains definitions of *datagroups* (explained below), each of which contains a set of data fragments. See figure 1.

2.1 User-system interaction scenario

System activity starts at the client. Each client runs transactions on its interest set, the net effects of which are saved into a *source update file*. These files are transmitted to the server which stores their contents in an update log.

The server's processing consists of generating a *group update file* for each datagroup of which it has a definition. The server does this by checking the update log, and, for each operation therein, checks each datagroup for "applicability" (e.g., if the operation is an `INSERT` into a certain fragment, check if the datagroup contains that fragment). If the operation is applicable to a given datagroup, then it is stored in that group's group update file.

The client then connects to the server, and downloads the appropriate group update files. *Appropriate* in this case is defined as the set of group update files which correspond to the minimum set of datagroups which, when combined, cover the interest set of the client. The contents of the group update files are applied to the client's local database to update its state.

In the *client-centric approach*, for each client interest set, there is an equivalent datagroup on the server. Hence, the server generates a unique group update file for each client, and each client must download only one group update file. In the *data-centric approach*, there are fewer datagroups than interest sets, reflecting the grouping that has taken place. Also, in this case, the number of datagroups required to cover a client's interest set may be greater than one, depending on the grouping scheme.

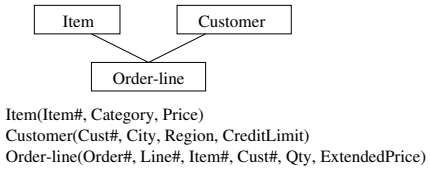


Figure 3: Example

2.2 Example Problem

In this section, we introduce an example which is used for illustrative purposes to define a sample problem. This example will be used in developing our optimization model later.

As shown in figure 3, three tables are used, CUSTOMER, ITEM, and ORDER_LINE, with their respective attributes; the application involves order processing for a company whose salesmen move on the road visiting customers.

For the CUSTOMER and ITEM tables, we assume the following fragments:

C1 : Northern Customer - All rows in the Customer table such that Region=N.

C2 : Eastern Customer - All rows in the Customer table such that Region=E.

C3 : Southern Customer - All rows in the Customer table such that Region=S.

C4 : Western Customer - All rows in the Customer table such that Region=W.

Fragments of the ITEM table are based on product category. They are:

I1 : Appliances Items - All rows in the Item table such that Category=Appliances.

I2 : Electronics - All rows in the Item table such that Category=Electronics

I3 : Clothing - All rows in the Item table such that Category=Clothing.

I4 : Toys - All rows in the Item table such that Category=Toys.

I5 : Furniture - All rows in the Item table such that Category=Furniture.

Clients in this example are salesmen and sales managers. Because of the nature of the schema where ITEM and CUSTOMER are partitioned horizontally into 5 and 4 disjoint fragments respectively, a total of 20 fragments of ORDER_LINE are possible based on the intersection of predicates which result from a derived horizontal partitioning (a term defined in [CNW93]). Thus, fragment OL_{11} is for orderlines that refer to customers in the northern region who order items of type appliance. OL_{12} is for Region=North and Category=Electronics, etc. This yields fragments OL_{11}...OL_{15}, OL_{21}...OL_{25},...OL_{45}: a total of 20 disjoint fragments of ORDER_LINE where the first subscript refers to customer region and the second subscript to item category. In general, based on conjunction of

```
for each datagroup, pi,
  for each operation, Oh, applied to the central database,
    if Oh applies to pi,
      then save Oh into pi's group update file, fi.
```

Figure 4: Server update distribution algorithm. The subscription pattern of given datagroup p_i can be encoded in the server catalog.

predicates, it is possible to define a suitable set of non-overlapping fragments for any such "dependent table" which derives its partitioning from one or more independent tables.

A salesman client with interest set that spans customers in North and East and who sells product categories Appliances and Clothing has:

$$\text{Interest Set} = \{C_1, C_2, I_1, I_3\}$$

If this client is also keeping track of all orderlines in the orders placed by these customers for those items, then his interest set expands to:

$$\text{Interest Set} = \{C_1, C_2, I_1, I_3, OL_{11}, OL_{13}, OL_{21}, OL_{23}\}$$

In our problem formulation of an integer program, we will use a subset of this schema to illustrate the process of grouping, and show that the resultant set of datagroups is less than or equal to the number of clients.

3 The Model

Now that the problem has been introduced, we will refine its description. We assume a pattern of synchronization activity consisting of four steps (see figure 1):

1. Client generation of source update files and their transmission to the server. This happens as a result of clients' "local" transaction activity that involves the fragments in the local interest set.
2. Server generation of group update files
3. Server transmission of each group update file to the client subscription list for that group
4. Client assimilation of received group update files

(We currently have a fully operational testbed, (see [MDNM98]), based on a commercial system, which executes the above scenario based on previously assigned groups.)

Our goal is to minimize the cost of steps 2 and 3, group update file transmission and generation.

In generating a model, certain simplifying assumptions are made.

Decision Variables	
x_{ij}	decides if data fragment i is included in datagroup j
y_j	decides if datagroup j is active
z_{jk}	decides if datagroup j is transferred to client k
Pre-Decision Variables	
n	number of data fragments
s_i	size of data fragment i
b	size of each operation transmitted
m	total number of operations per day
u_{ik}	indicator variable–shows if data fragment i in the in interest set of client k
w_i	% of operations that apply to data fragment i
D	number of clients
d_i	data fragment i
p_j	datagroup j
C_c	cost of checking an operation against a datagroup
C_u	cost of saving an update operation to a group update file
C_t	cost of transmitting an update operation to a client
C_a	cost of assimilating an update operation at the client
α	weight of server-side processing
β	weight of group update file transmission
γ	weight of client side update assimilation
Subscripts	
i	index of data fragment (i=1...n)
j	index of datagroup (j=1...n)
k	index of client (k=1...D)

Table 1: Notation used in objective function formulation

1. The amount of operations applied to a data fragment by a client is proportional to the data fragment's size.
2. Each client performs operations uniformly on the data fragments in its interest set. Operations include traditional SQL operations of *INSERT*, *DELETE*, *UPDATE* on tables.
3. There is low variance in the sizes of the messages containing the operations that applied to each data fragment.
4. The rate, per unit time, at which operations are applied to unit fragments of the database is fixed.
5. Network traffic is reliable and has a guaranteed fixed bit rate.
6. Network traffic is unicast in nature.
7. All clients connect at the same bit rate.
8. Server processing complexity is on the order of number of datagroups times number of update operations. See figure 4.
9. Server processing cost is dominated by disk access, and processing time is not considered.

We now lay out the objective function and its constraints, then proceed to explain each component of the formulation.

4 Formulation of the problem

The data-centric objective function can be expressed as

$$\text{Minimize } (\alpha(\text{server processing cost}) + \beta(\text{transmission cost}) + \gamma(\text{client processing cost})),$$

where

$$\text{server processing cost} = mC_c \sum_j y_j + mbC_u \sum_j \sum_i w_i x_{ij}, \tag{1}$$

$$\text{transmission cost} = mbC_t \sum_k \sum_j \sum_i w_i x_{ij} z_{jk} \tag{2}$$

$$\text{client processing cost} = mbC_a \sum_j \sum_i w_i x_{ij} z_{jk} \tag{3}$$

subject to the following constraints:

- client satisfaction

$$u_{ik} \leq \sum_j z_{jk} x_{ij} \forall i, k \tag{4}$$

- cover constraint

$$\sum_j x_{ij} \geq 1, \forall i \tag{5}$$

- datagroup bounding

$$\sum_i x_{ij} \leq n y_j, \forall j \tag{6}$$

- containment constraint

$$x_{ij} \in \{0,1\}, \forall i,j \tag{7}$$

$$z_{jk} \in \{0,1\}, \forall j,k \tag{8}$$

$$y_j \in \{0,1\}, \forall j \tag{9}$$

The objective function is to minimize the overall cost of processing updates in the ISDB scenario. α and β are system tunable relative weights that determine how processing or transmission is weighed against each other. Such tuning may be desirable, say, if a DBA must put more priority on transmission costs in the event of rising network costs. Although we have included the client processing cost in the total cost, we ignore it in the rest of the formulation since clients work off-line to install the updates and cause no particular hinderance to the overall efficiency of operation.

Formula 1 : Server cost is the cost over generating all group update files. The first term of this the server side cost expression in formula 1 describes the cost of checking each operation for containment in a datagroup (cf. the outer loop of the algorithm in figure 4). m is the total number of operations per day, C_c is the fixed cost of performing a single check, and $\sum_j y_j$ is the cardinality of the set of datagroups. y_j is a binary decision variable that describes "active" datagroups. We consider that the maximum number of possible datagroups is equal to the total number of data fragments, n, which would correspond to one data fragment being contained in each datagroup, hence, the $j \leq n$.[1] A datagroup j is "active," or exists, ($y_j = 1$) if it contains at lease one data fragment, so that $\sum_j y_j \leq n$:

$$y_j = \begin{cases} 1 & y_j \; is \; active \\ 0 & otherwise \end{cases}$$

Note that the product, $m \sum_j y_j$, is proportional to the number of iterations of the algorithm shown in Fig. 4.

The second term describes how much it costs to save the operations pertinent to the datagroups to the group update files (cf. the inner loop of the algorithm in figure 4). w_i describes what

[1] It can be shown that there cannot exist a grouping in which has more datagroups than fragments for the sake of brevity.

proportion of operations are applied to data fragment i, and is defined as the ratio of the product of the size and frequency (over all interest sets) of a data fragment, and the product of the size and frequency of all data fragments:

$$w_i = \frac{s_i \sum_k u_{ik}}{\sum_i s_i \sum_k u_{ik}} \tag{10}$$

The product of w_i and m determines how many total operations are applied to data fragment i.

x_{ij} determines if a data fragment d_i belongs to datagroup p_j:

$$x_{ij} = \begin{cases} 1 & d_i \in p_j \\ 0 & \text{otherwise} \end{cases}$$

With x_{ij}, m, and w_i, we can determine, for each datagroup j, the number of operations applied to it by summing their product over all i. We then multiply this number by a constant C_u and b, which reflects the cost of performing the `SAVE`ing of an update operation to the group update file, then sum this product over all datagroups, and get the total cost of generating the group update files.

Formula 2 : Transmission cost is roughly the cost of tranmitting all necessary group update files to each of the clients. Transmission cost, denoted in formula 2, is similar to the second term of the server cost formula, except that it: has different cost coefficients, reflecting that it is a different operation; is summed over all clients, k; and multiplied by a variable that decides if the datagroup j is subscribed to by client k:

$$z_{jk} = \begin{cases} 1 & \text{client } k \text{ subscribes to datagroup } j \\ 0 & \text{otherwise} \end{cases}$$

Formula 3 : Client processing cost is included for completeness. Client cost increases similarly to transmission cost. As superfluous data is sent to the client, there is an added cost in filtering it. Because we do not assume that client-side processing is a performance bottleneck, it is not actually considered further in the treatment of this model.

Formula 4 : The client satisfaction constraint, formula 4, indicates that, if a client k has a data fragment i in its interest set, then, at least one datagroup to which it subscribes, must contain that fragment.

Formula 5 : The cover constraint indicates that some datagroup j must contain data fragment i for all data fragments.

Formula 6 : The datagroup bounding constraint indicates that data fragments only exist in "active" datagroups, and is limited in number to n for all datagroups.

Formulas 7, 8, 9 : The containment constraints, formulas 7, 8, 9, indicate that either the decision for which these variables are true (1) or false (0). Any other value, either negative, or greater than 1, makes no sense.

4.1 Dynamics of the server cost function

The corresponding client-centric cost function (without grouping) is:

$$\alpha m C_c D + \alpha m b C_u \sum_k \sum_i w_i u_{ik} + \beta m b C_t \sum_k \sum_i w_i u_{ik} \tag{11}$$

(The physical meaning of each of the terms of this formula are similar to those of its data-centric counterpart. For brevity, we leave out a detailed explanation.)

- It is possible to save on server side processing by grouping [MDNM98]. With grouping, there is a definite savings in terms of checking for "applicability" of an operation to a datagroup. There are fewer group update files which must be managed by the server (D versus $\sum_j y_j$, i.e., #clients > #datagroups). Since the server must check each operation for containment within each datagroup, it is clear that:

  $$\alpha m b C_c D > \alpha m b C_c \sum_j y_j$$

  With grouping, it is also possible to reduce cost in terms of the number of `SAVE` operations required to write all group update files. The minimum number of save operations occur when there is no overlap amongst datagroups, because, for each update operation, the data fragment to which it corresponds occurs only once over all datagroups, and hence, the operation must be saved in only one group update file.

  Degree of overlap can be reduced by means of aggregating the datagroups. One may replace overlapping datagroups with their union. In this case, any client which once needed one of the overlapping datagroups can have its interests satisfied by their union. The resultant datagroups size is strictly less than the sum of the sizes of its overlapping constituents, and, hence, it should require fewer total save operations and its corresponding group update file should be smaller than the sum of those if its constituents.

  Note, however, that the degree of overlap depends on minimizing the entire objective function, which also involves considering group update file transmission cost. In general, there is an inverse relationship between server processing cost and transmission cost.

- Transmission costs are minimized with client centric operation, and cannot decrease under grouping. When datagroups are aggregated, they, in general, no longer match the contents of any single client interest set. Hence, when the resultant group update file is downloaded, the client receives superfluous data. To show this relation, we prove that the transmission term of the data-centric cost function, 2 can not be greater than that of the client-centric cost function, 11, term 3:

$$\sum_k \sum_j \sum_i w_i x_{ij} z_{jk} \geq \sum_k \sum_i w_i u_{ik}$$

Proof:

$$\sum_k \sum_j \sum_i w_i x_{ij} z_{jk} = \sum_k \sum_i \sum_j z_{jk} x_{ij} w_i$$

We know from constraint 4 that $u_{ik} \leq \sum_j z_{jk} x_{ij}$, so,

$$\sum_k \sum_i \sum_j z_{jk} x_{ij} w_i \geq \sum_k \sum_i u_{ik} w_i.$$

5 Tests

We test our model by plugging it into a math program solver. We used IBM's Optimization Subroutine Library (OSL), release 3. In order to run it on this software, we relax a part of our model by getting rid of quadratic decision variable expressions. First, we get rid of the quadratic constraint (i.e., the product term $z_{jk} x_{ij}$) in formula 4 and replace it with a linear constraint:

$$n z_{jk} \geq \sum_i x_{ij} u_{ik}, \forall j, k \tag{12}$$

Constraint 12 states that a client subscribes to all datagroups which contain any data fragments in its interest set. Compare this to constraint 4, which states that, given a data fragment in a client's interest set, at least one of the datagroups to which it subscribes must contain it. Clearly the former constraint is a relaxation of the latter.

Given constraint 12, if data fragments which are contained in a client's interest set are also contained in multiple datagroups, the, it is possible that the it will be sent an exorbitant amount of redundant data during synchronization. In order not to overload transmission cost, we modify constraint 5 so that datagroups do not overlap:

$$\sum_j x_{ij} = 1, \forall i \tag{13}$$

The model obviously becomes simpler with non-overlapping datagroups.

We must now get rid of the quadratic expression, $x_{ij} z_{jk}$ in the transmission cost (formula 2).

First, we approximate the transmission cost by breaking it up into two linear terms:

$$\beta mbC_t(\sum_j \sum_k z_{jk} - \sum_j y_j) \tag{14}$$

The first term reflects the notion that, transmission cost increases with the degree of datagroup subscription. As the number of datagroups to which a client subscribes increases, the network traffic also increases. The second term reflects the notion that transmission cost decreases as the number of datagroups increases. Roughly, the more datagroups there are without overlap, the better they are at satisfying a client's interests without including superfluous data.

Now, notice that saving of group update files becomes constant because there is no overlap:

$$\alpha mbC_u \sum_j \sum_i w_i x_{ij} = \alpha mbC_u \sum_i w_i(1) = \alpha mbC_u$$

Since one of our cost terms becomes constant, we drop it from out cost formulation. The result is:

$$\text{Total Cost} = \alpha mbC_c \sum_j y_j + \beta mbC_t(\sum_j \sum_k z_{jk} - \sum_j y_j) \tag{15}$$

This linear approximation of the ISDBE is first plugged into an OSL linear program solver. Branch and bound is then applied to the partial results (where x_{ij}, z_{jk}, and y_j are continuous variables) to get a final solution with integer values for variables.

Although we are quite confident that our original formulation accurately captures the cost of synchronization, these relaxations may cause unexpected side-effects when computing a solution. Our tests will some indication as how viable this relaxed version of the mode is.

5.1 Test description

For our tests, we use a subset of the example schema mentioned above, including only the CUSTOMER and ORDER_LINE tables. There are four derived fragments on the Order-line table based on the four Customer table fragments, for a total of eight fragments.

In this test, we generate a synthetic problem, and show that the math problem solver generates sensible results based on our model.

In our model, there are two client types: managers and salesmen. There is a northeast manager and a southwest manager, who manage the north and east regions and the south and west regions, respectively. There are four salesman, each of whom get a region, north, south, east, or west.

Intuitively, there are four sensible solutions, based on the relative weights of the server processing and transmission costs. Table 3 describes some obvious possible groups.

Generating a single datagroup (strategy 1 in table 3) is an extreme solution and optimizes server cost and disk usage, because only a single datagroup must be checked and saved to a group update file.

Client	Interest Set
Northeast Manager	C_1, C_2, OL_{1x}, OL_{2x}
Southwest Manager	C_3, C_4, OL_{3x}, OL_{4x}
North Salesman	C_1, OL_{1x}
East Salesman	C_2, OL_{2x}
South Salesman	C_3, OL_{3x}
West Salesman	C_4, OL_{4x}

Table 2: Client Interest Sets

	Datagroups	Comment
1	$\{C_1, C_2, C_3, C_4, OL_{1x}, OL_{2x}, OL_{3x}, OL_{4x}\}$	Minimal server cost, disk usage
2	$\{C_1, C_2, OL_{1x}, OL_{2x}\}$, $\{C_3, OL_{3x}, C_4, OL_{4x}\}$	Server cost/network cost compromise, minimal disk usage
3	$\{C_1, OL_{1x}\}$, $\{C_2, OL_{2x}\}$, $\{C_3, OL_{3x}\}$, $\{C_4, OL_{4x}\}$	Minimal network cost, disk usage
4	$\{C_1, C_2, OL_{1x}, OL_{2x}\}$, $\{C_3, C_4, OL_{3x}, OL_{4x}\}$, $\{C_1, OL_{1x}\}$, $\{C_2, OL_{2x}\}$, $\{C_3, OL_{3x}\}$, $\{C_4, OL_{4x}\}$	Minimal network cost. Client-centric solution

Table 3: Sensible grouping strategies

This maximizes transmission costs, however, because all clients receive all data fragments, regardless of interest.

Generating two datagroups (strategy 2 in table 3) is a compromise. Fewer groups are generated, which saves on server processing. However, network costs are still suboptimal, as salesmen clients still receive some superfluous data.

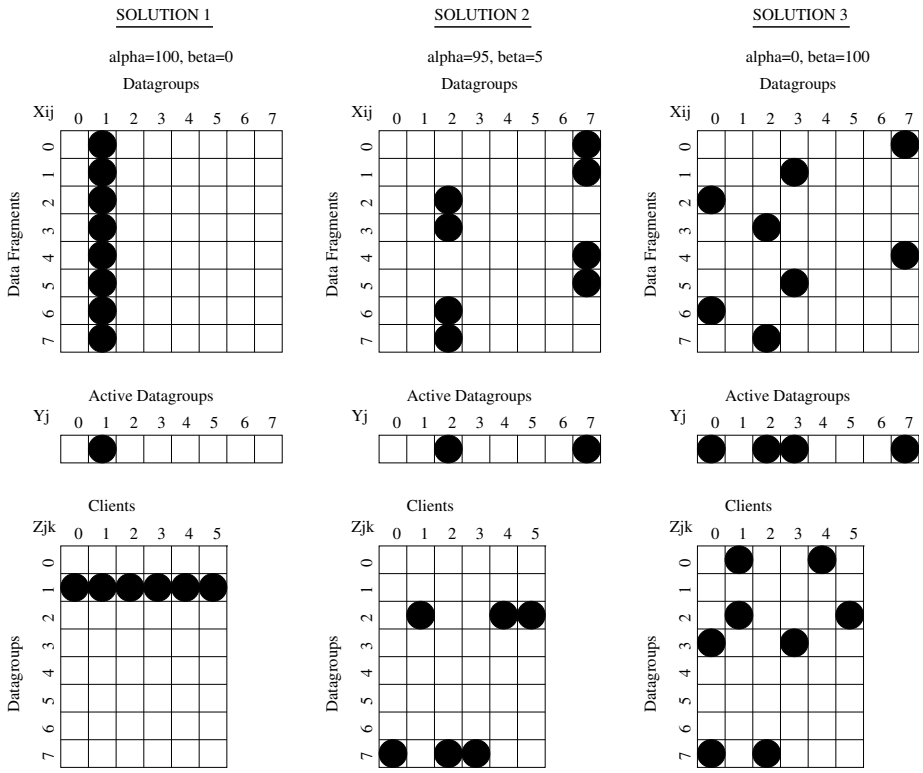
Generating four datagroups (strategy 3 in table 3) optimizes transmission cost, because there all clients' interests can be satisfied without sending any superfluous data. However, more datagroups are generated, increasing server cost.

The baseline solution, (the client-centric solution, strategy 4 in table 3), with six custom groups also minimizes network costs. However, this solution contains the most datagroups, and results placing some data fragments in more than one group, increasing disk usage.

We vary the coefficients of our cost function to see if and to what degree our model responds to perturbations in a predictable manner. High values of α encourage fewer datagroups, while high values of β encourage more datagroups.

5.2 Test results

Despite relaxing our original model, we were able to achieve expected, and, in one case, better than expected results. Representative results are in figure 5, in terms of x_{ij}, y_j, and z_{jk}. By adjusting the values of α and β, we are able to achieve predictable grouping trends: the more priority we put on server-side processing, the fewer datagroups are generated. In solution 1 in table 5, with $\beta = 0$, transmission costs are considered negligible. Hence, server processing costs are given maximum



Input values:

Number of data fragments(n) =	8		Cost of checking (Cc) =	1	
Number of clients (D) =	6		Cost of saving (Cu) =	32	
			Cost of transmission (Ct) =	32	

Client 0: NE Manager	Data Fragment 0: N Customers	(C1)	Data Fragment 4: Derived Orderline Fragment 0		(OL1x)
Client 1: SW Manager	Data Fragment 1: E Customers	(C2)	Data Fragment 5: Derived Orderline Fragment 1		(OL2x)
Client 2: N Salesman	Data Fragment 2: S Customers	(C3)	Data Fragment 6: Derived Orderline Fragment 2		(OL3x)
Client 3: E Salesman	Data Fragment 3: W Customers	(C4)	Data Fragment 7: Derived Orderline Fragment 3		(OL4x)
Client 4: S Salesman					
Client 5: W Salesman					

Figure 5: Test results.

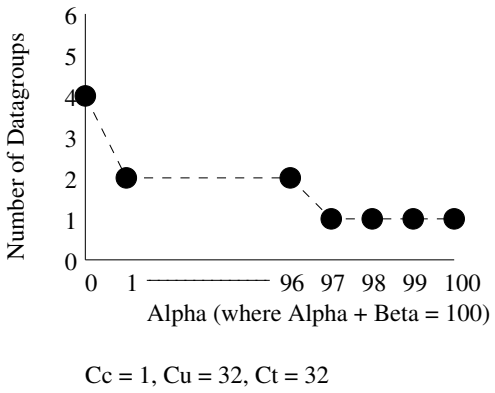


Figure 6: Sensitivity to varying coefficient values.

priority, and only one datagroup containing all fragments results. At the other extreme, in solution 3, with $\beta = 100$, processing cost is negligible, so transmission cost is given maximum priority. In this case, four datagroups are generated, which makes it possible to allocate datagroups to all clients, without allocating superfluous data fragments.

The compromise solution, which occurs in over 90% of the intermediate values of α and β results in two datagroups (solution 2 in table 5). In this case, there is a compromise between server processing cost and transmission cost.

Note that, the client-centric solution (see solution 4 in table 3) optimizes transmission costs, and, arguably, for real-world applications, setting β to 100 results in that solution. However, in our tests, when $\beta = 100$, we pleasantly got solution 3. This is a sensible solution, because, it turns out that solution 3 (four datagroups) has the side effect of having lower server processing costs than does solution 4 (six datagroups), while preserving the minimal transmission requirement. This is also surprising, because there are multiple solutions that minimize transmission cost (e.g., 8 datagroups, with one data fragment in each, qualifies), and our linear model does not discriminate amongst them.

OSL did not generate a solution with 3 datagroups with any of the values of α between 0 and 1 which we chose, which lead us to guess that there is no "sensible" such solution.

In our tests, we chose $C_c = 1$, $C_u = 32$, $C_t = 32$, which should roughly reflect the relative costs of checking an operation against a datagroup, saving an operation to a group update file, and transmitting an operation over a network. The large difference between C_c and C_t has the effect of diminishing the sensitivity of the cost function to changes α and β, because only extreme values of the weighting factors can overcome the cost coefficients. See figure 6 for results for different α β combinations.

6 Conclusion and Future work

In this paper we presented a non-linear mathematical programming model to solve the problem of optimal grouping of data fragments at the server in the intermittently connected synchronized database (ISDB) environment. The model was relaxed by converting a quadratic constraint into linear constraints. It was solved using the OSL Library of math programming routines and the solution procedure was validated using some semantically meaningful groupings. To our knowledge, this is the first attempt at formulating this important design problem analytically.

We wish to point out that the assumptions we made regarding non-overlapping groups and uniform distribution of fragments are reasonable assumptions for the "first-cut," or initial design of the ISDBE server database. Today, DBAs are in need of tools/aids that can help them start with some initial feasible solutions. By allowing them to tweak the C_c, C_u, C_t cost parameters and α and β, a whole range of solutions can be explored.

A number of possible directions can be taken with the model. The optimization model developed here can be extended in several ways. First, the model can be refined to accommodate overlapping groups that have one or more data fragments in common. This would be particularly useful when there are a few data fragments that are used by several clients. Second, for larger sized problems, it is infeasible to solve the formulation through a branch and bound algorithm. The development of a family of inequalities that render the current continuous optimal solution invalid without removing any integer solution from the feasible set, will be beneficial. The integer problem can then be solved by solving the continuous problem repeatedly after incorporating the inequalities. Third, computationally efficient heuristic methods and lower bounding techniques can be developed to compare the heuristic solutions generated with lower bounds.

Currently, we solve it as a linear program and then apply a branch and bound procedure to reach the desired integer solution. We will need to be trying to use some integer program solvers such as AMPL, XPRESS MP or AIMMS. We will also deal with quadratic constraints directly if possible. In order to deal with real-life problems we will work with hundreds of fragments and tens of classes of clients. The above tools allow one to define the model and input the problem easily. There is a need to relax constraints - e.g., non overlapping groups or uniform activity against all fragments. We plan to initially attempt modifying our model to incorporate these. Later, we will develop a simulator that enables us to have a tighter control over a number of parameters and allows a "dynamic model formulation" where the changing nature of clients' data requirements and redesign of fragments at the server can be incorporated. We also plan to develop heuristics to take the result of the static analytical model and move toward a new solution allowing for changes in client interest sets, new data at the server, reorganization of clients among new classes, etc.

The work on multicasting has been proceeding in parallel in our ISDB project with the development of a multitasking protocol by Donahoo [Don98]. We will evaluate the option of multicasting updates for a group in comparison to on demand delivery of updates to clients to achieve a proper

balance of push-pull model for update dissemination with maximum efficiency.

This work is being undertaken in close association with an industrial partner. Issues like security are important in certain applications. Ultimately, we will be working toward a design and administration tool that will benefit from the mathematical model and the heuristics to achieve feasible and acceptable solutions in the ISDB scenario. We expect the ISDB environment to be a dominant mode of client-server applications in the next several years.

References