Scalable Synchronization of Intermittently Connected Database Clients

Wai Gen Yee and Ophir Frieder
Database and Information Retrieval Laboratory
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
yee@iit.edu, ophir@ir.iit.edu

ABSTRACT

Synchronization performance is a major problem with intermittently connected mobile databases. A server periodically generates update files for each client, which are downloaded and applied when convenient. Unfortunately, the time required to synchronize clients in this way increases drastically with client population. We show that this trend could be altered by appropriately modifying the way that update files are designed, resulting in significant performance improvements.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases*

General Terms

Experimentation, Performance

Keywords

Mobile computing, synchronization, performance, scalability

1. INTRODUCTION

Intermittently synchronized database systems (ISDBs) support mobile database activities by allowing clients to work on locally replicated data without a constant connection to the server [3, 4, 6, 7, 8]. ISDBs are widely deployed, with common applications in areas such as customer relationship and supply chain management.

ISDB synchronization is performed asynchronously, via message passing. The server collects updates that clients send to it in *update files*. Periodically, the server reciprocates by generating update files for each and every client containing the relevant subsets of these updates. When convenient, clients download these files and apply the updates to their local replicas, and so on. Although practical, this type of synchronization results in performance that is unscalable with an increasing client population. In particular, the server may become a performance bottleneck as the number of clients grows.

There are benefits to using message-based synchronization versus online (i.e., synchronous) synchronization. For example, the client can control when and how long it spends connected with the server. This allows it to better control its energy usage, which is at a premium for mobile devices.

There are other problems with message-based synchronization as well, including staleness of data and conflict resolution. We do not cover these issues in this work, but they are considered in commercial ISDB product documentation.

The process of generating update files for clients, which we call a *synchronization session*, takes on the order of hours in practice. For example, if it took 30 seconds to generate an update file for each of 500 clients, then the entire synchronization session would last 4 hours. This duration determines the minimum latency for clients getting the most recent updates.

Moreover, the cost of a synchronization session grows quadratically with client population. Assume that there are N_C clients, and each generates an average of R_U updates. Each of these $R_U N_C$ updates are sent to the server and must ultimately be distributed to clients' update files based on their *subscriptions* (i.e., data needs). To do this, the server must compare each update to each client's subscription, a process that takes $O(N_C^2)$ time.

The poor server performance can be attributed to two factors: redundant processing by the server or misuse of available system resources. Redundant server processing occurs because the server may have to examine and write the updates to update files multiple times. By modifying the update files, the server can offload synchronization work onto other system resources. For example, assume the server generates a single update file containing all updates. This minimizes server work, but offloads work onto the network, as the all updates must be transmitted to all clients regardless of their subscriptions. This is a generally poor update file design as the network is often the performance bottleneck.

These solutions capture the idea behind this work. By altering the design of update files, we can control the cost of synchronizing a client. Proper design requires knowledge of the ISDB configuration: the performance of various sys-

MDM 2005 05 Ayia Napa Cyprus

tem components and client subscriptions. In this paper, we elaborate on this idea and propose a way of increasing the scalability of client synchronization with various system configurations.

2. MODEL

The server maintains the *primary database*, which is partitioned into *publications*. By *subscribing* to a publication, the client makes a local *replica* of the published data and periodically downloads updates for it from the server via update files.

Each update file is defined by a *datagroup*. A datagroup is a set of publications, and thereby determines the contents of an update file. Specifically, each update is compared to each publication in each datagroup. If an update *modifies* a datagroup's publication, then the update is written to the datagroup's update file. In general, there is no constraint to the number of contents of datagroups. We overload the term *subscribe* by stating that clients subscribe to datagroups (in addition to publications). By subscribing to a datagroup, the client will download the update files based on it.

In commercial ISDB systems, a datagroup is created for each client. The publications in each datagroup are exactly those that the client desires, so each client downloads the minimal necessary updates. We call this datagroup design *client-centric*. We used client-centric design in the example given in Section 1.

We are primarily concerned with the speed with which the average client can become synchronized with the server. Because the server and network are the shared components of the architecture, they are most likely to be performance bottlenecks. The costs, therefore, are in terms of generating and then transmitting the update files to the clients.

Update file generation cost components:

• Number of update files - This cost includes disk latencies encountered by switching between files and the maintenance of associated metadata.

• Bytes written to disk - This cost describes the amount of time the server spends writing to disk.

Update file transmission cost components:

• Number of update files downloaded by a client - This cost describes the overhead of establishing a network connection to download an update file.

• Bytes downloaded - This cost describes the time required to download the "contents" of the update files.

From these costs, we see that our goal is to design the smallest and fewest datagroups that cover, but do not oversatisfy the data needs of clients. These goals are in conflict. For example, minimizing the number of bytes written to disk (i.e., by generating a single update file containing all updates, as in the Example in Section 1) significantly increases the number of bytes downloaded by each client.

We do not consider the cost of client filtration of unwanted updates: the client is not a shared resource (i.e., system bottleneck) and bandwidth is generally a greater limiting factor than is processing power.

3. DATAGROUP DESIGN

As suggested in the previous section, we can affect the synchronization performance of an ISDB by manipulating the design of the datagroups and the subscription of the clients to datagroups.

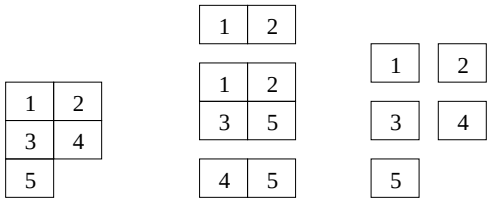

Figure 1: Example of Alternate Datagroup Designs.

The design of datagroups is combinatorial, and making it difficult to find an optimal solution. We therefore consider two alternatives to the traditional client-centric (**cc**) strategy. These alternatives will help us better understand the subtleties of datagroup design:

•**One-per publication** (**op**) - The number of datagroups equals the number of publications, and there is a unique publication in each.

•**One-giant datagroup** (**og**) - There is a single datagroup containing all publications.

With both *op* and *og* design, the subscription to datagroups by clients is straightforward. With *op*, if the datagroup contains a publication that a client needs, the client subscribes to that datagroup. With *og*, there is no subscription decision as there is but one datagroup.

Example Consider an ISDB with three clients and five publications (1,2,3,4,5). The three clients' data needs are {1,2}, {1,2,3,5}, and {4,5}, respectively. The three ways in which datagroups can be designed using *cc*, *og*, and *op* are shown in Figure 1. Any one of these designs is capable of satisfying the data needs of the clients, but with different costs.

We further analyze these three design alternatives later in the paper. We first motivate the analysis with some experimental results in the next section.

Parameter	Values ({default})
Num clients	1, 200, {400}, 600, 800, 1000
Client b/w (Kbps)	19.2, {57.6}, 1536, 10240, 102400

Table 1: Parameter values for experiments.

4. EXPERIMENTS

The goal of these experiments is to demonstrate the performance of *cc*, *og*, and *op* datagroup design on synchronization performance over a wide range of parameters. To do this, we define a metric, **sync time**, as the sum of the average time required to generate and transmit a covering set of update files to a client. We show the results in which we vary datagroup design strategy, client population, client communication capabilities.

4.1 Experimental Setup

The experimental database is composed of $N_P = 400$ publications. Client subscriptions to these publications is skewed, following a Zipf distribution with parameter $\theta = 1$. (We also performed experiments using a uniform distribution, but excluded them because the results were not signif-

icantly different.) Each client subscribes to between 5% and 10% of the publications, based on the distribution.

The number of updates distributed by the server is a function of client population. Each of the N_C clients is assumed to generate 100 updates, so the total number of updates is $100N_C$. The distribution of the updates to publications is based on the publications' subscription rate.

Each client is assumed to have the same network bandwidth. (We could have also modeled this as a weighted average of individual bandwidths, weighted by subscription size.) Transmission cost is therefore the amount of data a client must download divided by this bandwidth.

Some of our experimental parameters are summarized in Table 1. The default parameters are meant to model a mobile salesperson whom uses a modem to upload and download updates onto a modern laptop. We assume that the server pushes update files to clients as it generates them (e.g., by emailing them, or copying them onto an FTP site).

We built an ISDB server using commercially available ISDB and DBMS software to test the performance of the various datagroup design strategies. As message-based synchronization is similar across most of the commercially available ISDBs we know of, the general performance trends reported, though not necessarily the performance levels, should be representative.

We use two dedicated machines to host the *database server* and *synchronization server*. The database server manages access to the database. Updates to the primary database are `INSERT`s. The database server software is Sybase SQL Anywhere version 9, running on a 3.2GHz Pentium PC with 1GB RAM. The synchronization server captures and packages updates to the local database for eventual shipment to clients. (It can also incorporate into the primary database the contents of update files from clients.) The synchronization server software is Intellisync iMobile version 2.4, running on a 1.4GHz Pentium PC with 512MB of RAM. We have made modifications to it so that it can generate update files based on datagroup designs other than client-centric. The ideas behind the modifications may be found in [5].

Both servers are running Windows XP and are connected by a 100Mbps Ethernet LAN. Our setup is typical of one that might be found in a small company. More computers may be added to scale up absolute performance, but would not affect the overall trends.

4.2 The Cost of Generating Update Files

In Figure 2, we show the amount of disk space each of the design strategies consumes with increasing client population. (Notice the log scale.) The space taken by both *op* and *og* increase linearly with client population because the number of updates is set at $100N_C$. The reason that *op* consistently consumes so much disk space initially is that it must generate an update file for each publication, even if the update file is empty. Each update file, due to metadata and blocking factor, consumes at least 4KB. Because $N_P = 400$, these files consume at least 1.6MB. As the volume of updates increases, there impact of this overhead diminishes.

Eye-catching is the enormous amount of disk space that *cc* update files consume with high client population. This happens because disk space consumption increases quadratically with client population. When $N_C = 1000$, the update files consume almost 1GB of disk space. That is 2.5 orders of magnitude more space than that which is consumed by the update files of either of the other design strategies.

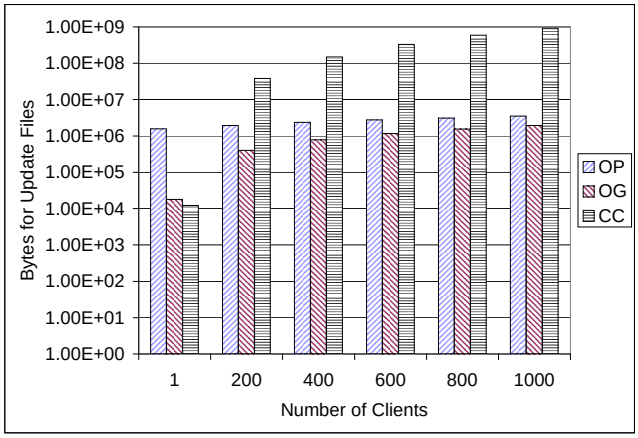

Figure 2: Disk Usage with Increasing Client Population.

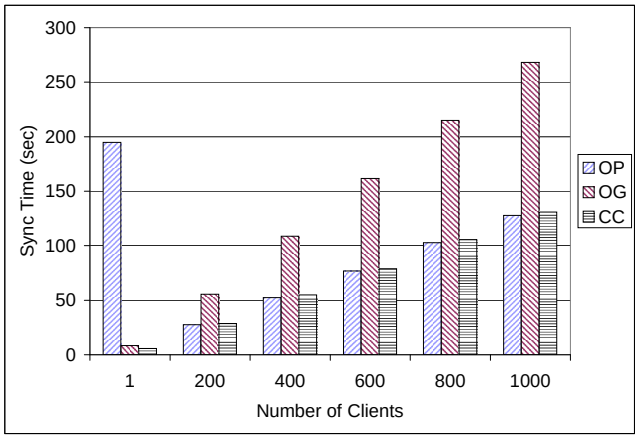

Figure 3: Per-client Sync Time with Modem-level Bandwidth (57.6Kbps).

The quadratic growth in disk usage can be explained by the assumption that update rate is proportional to the number of clients. This assumption implies that the number of updates to a publication is $O(N_C)$. The number of times a publication is subscribed to is also $O(N_C)$. The number of updates written for a publication is therefore $O(N_C^2)$.

The times required to generate the update files (not shown) follow the basic pattern suggested by the disk usage. For example, it took 10 seconds, 5 minutes and 50 minutes to generate update files for *og*, *op*, and *cc*, respectively, when $N_C = 1000$. The relative expense of *op* compared to its disk usage is due to its need to write out so many update files.

4.3 Sync Time

The sync time performances of the three datagroup design strategies are shown in Figure 3. The performances of *op* and *cc* are similar for most values of N_C, while the performance of *og* is significantly worse. The one exception to this trend is when $N_C = 1$ (or other low N_C values not shown here). When N_C is low, the cost of generating *og* and *cc* update files is low as well because there are few updates to distribute. However, there is a high lower bound to *op* cost

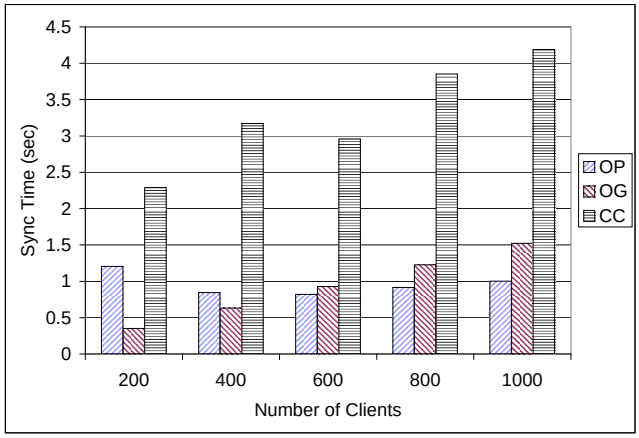

Figure 4: Per-client Sync Time with WiFi-level Bandwidth (10Mbps).

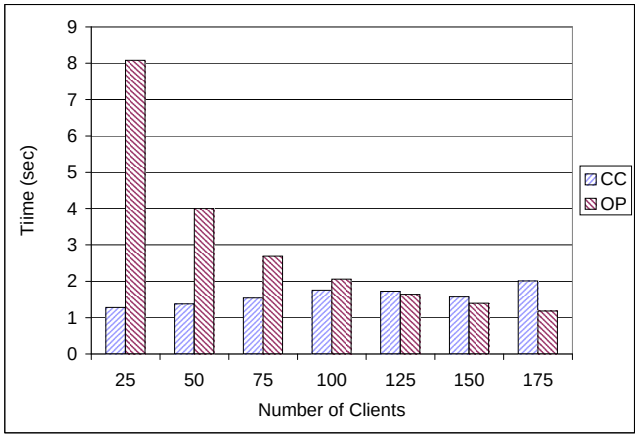

Figure 5: Per-client Update File Generation Time for 25-175 Clients (57.6Kbps).

because it must generate an update file for every publication no matter how few clients and updates there are. In Figure 3, when $N_C = 1$, $N_P = 400$ update files are still generated.

The performance of *og* is poor for obvious reasons; all clients must download all updates. But, what if downloading were less expensive? What if the client had more than a 57.6Kbps modem to communicate with?

We answer this question using results shown in Figure 4, where the bandwidth available to each client jumps to 10Mbps, equivalent to ideal WiFi connectivity. Increasing the available bandwidth makes downloading much cheaper. Consequently, the relative value of being able to quickly generate update files becomes greater.

In Figure 4, *cc* does worst because of its poor update file generation performance. The performance of *op* starts high, is best near $N_C = 600$, and then slowly worsens afterward. The initial improvement in performance is due to the amortization of the fixed cost related to generating the update files. At $N_C \approx 600$, this cost is fully amortized, and *op*'s performance steadily worsens due to the increase in volume of updates with client population. The performance of *og* is initially better than that of *op*. However, it worsens at a quicker rate because its update file includes all updates to all clients, not just the updates to individuals. For a given client, this overhead grows at a rate equal to the proportion of publications that the client *does not* subscribe to. However, experiments where bandwidth were increased to 100Mbps (corresponding to a typical LAN, not shown), *og* indeed performs best over this range of client populations.

Note that we begin Figure 4 at $N_C = 200$ because *op*'s terrible update file generation performance when $N_C = 1$ would have obscured the performance differences.

5. DISCUSSION

The results indicate that *og* does well when there is sufficient bandwidth, *cc* does well when there are not too many clients, and *op* does well when there are many clients. By capturing these performance relationships, we can decide on a datagroup design that takes advantage of the ISDB system's current performance characteristics, yielding consistently good sync time. We will no longer consider *og* as a viable datagroup design alternative. Although *og* has very good performance when bandwidth is available, such a situation is uncommon in mobile computing environments.

If we only compare *op* to *cc*, then we can restrict our consideration to cases where $N_P \geq N_C$. This should make sense because, if $N_P < N_C$, then we can assume that *op* will outperform *cc* because *op* will be guaranteed to not write updates to disk more than once and the overhead of writing unique files is equal to that of *cc*. (There is a slight overhead in downloading multiple files with *op*, however, this overhead should be relatively small for large N_C.)

In our experiments, *op* update file generation outperforms *cc* with N_C values much smaller N_P ($N_C = 125$ in Figure 5, with $N_P = 400$). This happens because, with $N_C = 125$, *cc* has increased overhead in terms of both the number of files it must generate and the number of bytes written. In comparison, the number of files *op* generates is fixed, and the number of bytes written by *op* increases slowly, as shown in Figure 2.

Specifically, the equality of performance of *cc* and *op* when $N_C = 125$ means that the time it takes *cc* to write an extra 13.5MB required by *cc* is equal to the time it takes *op* to open and write updates to 275 more files than *cc*.

If we assume that I/O is the major cost of file generation, then we can roughly estimate the point at which *op* outperforms *cc*. This is the point when the cost of writing bytes to disk using *cc* outweighs the cost of managing extra files using *op*. Using the model and concepts presented in this paper, we can readily make estimates of disk usage using either *op* or *cc* for a given number of clients. These estimates can be used to determine which datagroup design to use.

For example, our server machine has an I/O rate of 15MB per second and requires 0.003 seconds to open a file. It takes about 0.87 seconds to read 13.5MB, and about 0.83 seconds to open 275 files. The relationship between file management and volume of data written seems to hold on our ISDB platform.

A more precise point at which *op* should be chosen over *cc* requires consideration of update file transmission overhead of about 4KB per file. Estimating this penalty is straightforward, and will increase the break-even N_C an amount dependent on the available bandwidth. Furthermore, this point should be recalculated whenever there is a *significant* shift in subscription patterns or overall system capabilities.

There are complications, however, regarding *op*'s requirement that clients download multiple update files to update themselves. The first complication is in the need for the client to establish a connection with the server for each update file. Based on our experiments using the Unix `wget` utility, each connection requires on the order of a second to establish.

However, this cost may be amortized by two factors. If the files are large, the cost of establishing a connection is trivial. Second, the server may allow multiple simultaneous connections. This may be faster than establishing many single serial connections. BitTorrent, the well-known file-sharing application, for example, takes advantage of some characteristics of TCP by using multi-connection file transfer to effectively manage bandwidth [2].

Second, the client must wait until the last update file it needs has been generated before it has a complete set of updates. If a client must wait until the last file is generated, then it loses any benefit it may have anticipated by having some of its update files made available early.

This problem may be benign relative to the performance of a *cc* ISDB. Recall that the volume of updates written by the server in a *cc* ISDB grows with $O(N_C^2)$. At the same time, the volume of updates written by the server in a *op* ISDB grows with $O(N_C)$. If a client needs N_F files from a *op* ISDB, and the files are generated in random order by the server, then the expected time the *op* client must wait for its last file is $\Theta(\frac{N_F}{N_F+1}N_C)$, but the time required for a *cc* client is $O(N_C^2)$. The long-term latency for an *op* client is therefore less than that of a *cc* client.

There are heuristic steps, however, that one may take to reduce the expected latency involved in making available multiple update files: the server can create the update files for the most subscribed-to datagroups first. This is not an optimal algorithm because, in general, datagroups are not subscribed to independently. For example, the fact that a client subscribes to the most popular datagroup may, for some reason, imply that it is less likely than average to subscribe to the second most popular datagroup (perhaps because they contain redundant data). The update files of the popular datagroups may be relatively large as well, but this should not hurt performance as long as the update file's size is proportional to its datagroup's popularity. If so, then the rate at which clients subscribing to single update files are satisfied is optimal. The problem of scheduling the update files has been approached before [1], and is generally considered difficult by researchers in operations research.

Finally, the problem caused by having to download and share multiple update files may be mitigated by using modern data dissemination techniques. For example, a file that is desired by many clients may be broadcast [10]. Although the order of broadcast is still in question, this is another way of increasing scalability, as the single transmission of an update file can simultaneously satisfy many clients.

6. CONCLUSION

We describe some observations we have made regarding ISDB synchronization performance. We show that traditional client-centric (*cc*) synchronization is unscalable because it requires the server to perform significant redundant work to generate update files for clients. By altering the way that update files are designed, we can improve the cost of synchronization.

It turns out that *cc* synchronization works well for a small number of clients, but generating an update file for each database publication (*op*) works better as the number of clients increases. *Op* requires less redundant work at the server, but has a higher fixed cost. In extreme cases when client bandwidth is very high, generating a single update file containing all updates is best, as it minimizes server work at the expense of the network.

We are currently exploring ways of formalizing a means of deciding among the datagroup design strategies given the observations we presented in this paper. That is, given system performance characteristics and subscription patterns, there should be a straightforward way of deciding among client-centric, one-giant, or one-per design. Our previous attempts at near-optimal solutions have proved too computationally complex and monolithic [9]. Other future work will be directed toward optimizing the performance for specific subsets of clients. For example, there may be a subset of clients that have a uniform subscription pattern and communication capability. These clients may have their synchronization sessions grouped. Furthermore, one of these subsets may include critical users (e.g., managers). This subset should perhaps be treated as a special case and have improved synchronization performance relative to others.

7. REFERENCES

[1] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proc. IEEE/ACM MOBICOM*, Oct. 1998.

[2] B. Cohen. Bittorrent home page. Web Document. bitconjurer.org/BitTorrent.

[3] IBM, Inc. Db2 everyplace. Web Document, 2004. www-306.ibm.com/software/data/db2/everyplace.

[4] Intellisync Corp. Intellisync home page. Web Document. www.intellisync.com.

[5] S. Mahajan, M. J. Donahoo, S. B. Navathe, M. Ammar, and S. Malik. Grouping techniques for update propagation in intermittently connected databases. *Proc. IEEE Intl. Conf. on Data Eng. (ICDE)*, Feb. 1998.

[6] Microsoft, Inc. Introducing sql server mobile edition next-generation mobile database. Web Document, 2004. www.microsoft.com/sql/ce/productinfo/SQLMobile.asp.

[7] Oracle, Inc. Oracle 9i lite, a technical white paper. Web Document, 2004. otn.oracle.com/tech/wireless/papers/q4-02/Oracle_Lite_wp11-02.pdf.

[8] Sybase Workplace Database Division. SQL remote: Replication anywhere. Technical report, Sybase, Inc., www.sybase.com/products/system11/workplace, 1999.

[9] W. G. Yee, M. J. Donahoo, and S. Navathe. Framework for server data fragment grouping to improve scalability in intermittently synchronized databases. In *Proc. ACM Conf. on Information and Knowledge Mgt. (CIKM)*, pages 54–61, Nov. 2000.

[10] W. G. Yee, S. B. Navathe, E. Omiecinski, and C. Jermaine. Efficient data allocation over multiple channels at broadcast servers. *IEEE Trans. Computers, Special Issue on Mobility and Databases*, 51(10):1231–1236, Oct. 2002.