A Dynamic Data/Currency Protocol for Mobile Database Design and Reconfiguration

Yanli Xia and Abdelsalam (Sumi) Helal
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA
{yxia, helal}@cise.ufl.edu

ABSTRACT
This paper presents flexible protocols for dynamic database design and reconfiguration, enabling mobile database to be designed in such a way that data location, replication and even use semantics can be changed dynamically. In particular, a dual data/currency hoarding and synchronization protocol, a metadata protocol and currency redistribution protocol are proposed. Metadata records the changing characteristics of replicas and currencies are dynamically redistributed to fit the target. Primaries may be diluted, concentrated and transferred in the system. Traditional distributed database is included as home of data. An ad-hoc database may check out its desired data/currency and check them back in when they are no longer accessed by the mobile users. We demonstrate that the flexibility of this approach can significantly improve commit performance under dynamic access and mobility behavior.

Keywords
Mobile database, Hoarding and synchronization, Metadata, Dynamic currency redistribution, Database reconfiguration, Two-tier replication, Ad-hoc database

1. INTRODUCTION

Recent advances in wireless networking technologies and the growing success of mobile computing devices are enabling new issues that are challenging to mobile database designers. What is mobile database? One idea is disconnected database, where mobile hosts are strong connected, weak connected with or disconnected from fixed network. They hoard replicas before disconnection, read/write on the local replicas, and then synchronize updates when they get reconnected with the fixed network. There is no communication among mobile hosts. This kind of database is similar to ubiquitous data access [8, 10]. Another good idea is ad-hoc database [4, 5, 7, 17], where mobile hosts weak connected in pair-wise manner or disconnected with each other. They hoard replicas from their peers and synchronize updates upon connection. When disconnected, local data accesses are performed. Broadcast disks [1, 3] is also related to mobile database, in which a broadcast server residing in fixed network pushes commonly interesting data to mobile hosts covered in its range from time to time. Traditionally, distributed database is located in a fixed decentralized network [16]. We envision that a *mobile database* should be defined as the union of distributed database, disconnected database, ad-hoc database and broadcast disks. The distributed database is treated as the home of mobile database, and the others deal with the access of mobile users. Figure 1 demonstrates the concept of mobile database.

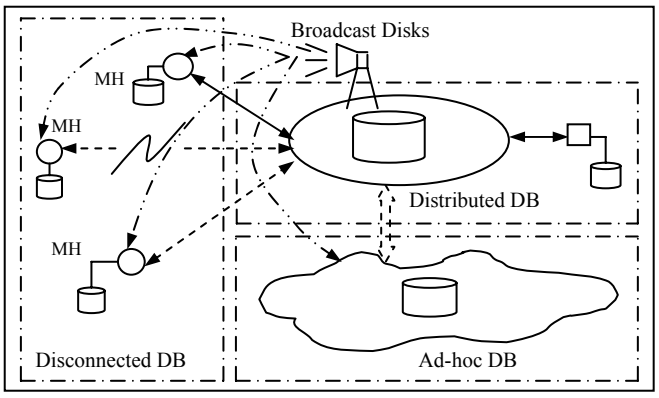

Figure 1. Mobile database: a Whole Image.

Traditional database design is static and limits the flexibility of database applications, while mobility is changing the way we design databases and their DBMS. In mobile database everything is dynamic, varying from sporadic accesses by individual users to particular data to continuous access of a particular data by a large group of user. This is the case from disconnected database access to broadcast disks. Mobile hosts have to deal with planned or unexpected disconnections when they mobile; they are likely to have scarce resources such as low battery life, slow processor speed and limited memory; their applications are required to react to frequent changes in the environment such as new location, high variability of network bandwidth; their data interests are changing from time to time and from location to location; even data semantics in mobile hosts are varying according to data access patterns, connection duration and disconnection frequencies, etc. Data partition, location and replication are always dynamic. All of these require a dynamic database design and reconfigure scheme.

To reach a generic solution for mobile database, we work on flexible protocols for dynamic database design and reconfiguration involving fixed network. We introduce a dual

SAC 2003, March 9-12, 2003, Melbourne, Florida, USA

data/currency hoarding and synchronization protocol. Data is hoarded via a variety of hoarding and synchronization mechanisms with different replica semantics indicated by amount of currencies. Metadata record the changing characteristics of mobile database and currency is dynamically redistributed to fit the target. Databases are designed in such a way that their location, replication and even use semantics can be changed dynamically. We include distributed database as home of mobile database. The focus of this paper is the interrelation of distributed database and ad-hoc database. We believe the flexibility of this approach is necessary given the uncertainty of user mobility.

The rest of this paper is organized as follows. Section 2 briefly describes related work. Section 3 provides the generalized model of hoarding and synchronization in mobile database. A simple metadata protocol is described in section 4 and section 5 is focused on the dynamic currency redistribution protocol. Section 6 presents the simulation results to assess the benefits of our protocols and section 7 concludes.

2. RELATED WORK

There is a significant body of related work. Jim Gray at al [9] showed that replication protocols go along two dimensions: master/group object ownership and lazy/eager update propagation. They argued that eager schemes are unsuitable for mobility due to frequent disconnections and proposed a two-tier lazy protocol and update anytime replication model for scalable replication. Jajodia et al [11, 20, 21, 22] improved Jim Gray's approach using adaptive data replication algorithm, which changes the replication scheme of the object as changes occur in the read-write pattern of the object. But this algorithm is designed for distributed, wholly interconnected information environment, not proper for mobile environment. CODA [14, 15] is a distributed file system with the ability to support disconnected operation for mobile computing. The scenario where all database users are ad-hoc users has been integrated in the Bayou [7, 17] and Deno [4, 5, 6, 12] projects. Both of them take asynchronous, epidemic information flow and conflict avoidance-based (i.e. pessimistic) approach to ensure that all committed updates are serialized in the same order at all servers. Bayou uses a primary-copy scheme, while Deno takes a bounded weighted-voting one. Deno's bounded weighted voting is borrowed in this paper to synchronize currencies.

3. A DUAL DATA/CURRENCY HOARDING AND SYNCHRONIZATION PROTOCOL

The concept of currency is introduced by research efforts of [18, 19, 23]. Deno uses currency extensively to data replication and voting scheme in mobile environments. What is currency? Currency is a weight associated with a replica. The amount of currency held by a replica is used as that replica's weight during voting rounds. Here, currency has the constraints that the summary of currencies for any object is fixed at 100. Additionally, the amount of a replica's currency indicates the semantics of the replica, defined as following:

Primary currency: the amount of currency is in the scope (50, 100] and it indicates the replica is a primary. An object has at most one primary and the primary has the power to commit an update immediately.

Copy currency: the amount of currency is in the scope [0, 50]. If it is equal to 0, it is a *read-only copy currency*. Otherwise, it is an *updateable copy currency*. A replica with updateable copy currency may issue tentative updates and start elections to seek commitment, while a replica with read-only copy currency is not allowed to issue tentative updates since it has no voting weight. Both of them may be involved in anti-entropy sessions to get up-to-date commitments.

3.1 Data Hoarding

Data hoarding releases currency, therefore changes object semantics. A receiver hoards replicas from a sender and each replica is denoted as (objectID, semantics), where semantics is either primary or copy.

General hoarding (G-hoarding): sender owns the primary of an object, i.e. (objectID, primary), and receiver hoards a copy from the sender. After G-hoarding, sender still owns the primary, i.e. sender (objectID, primary), and receiver (objectID, copy).

Primary hoarding (P-hoarding): sender owns the primary of the object, i.e. (objectID, primary), and receiver hoards the primary from the sender. After P-hoarding, the primary is transferred from the sender to the receiver and the one in the sender becomes a copy, i.e. sender (objectID, copy), and receiver (objectID, primary).

Copy hoarding (C-hoarding): sender owns only the copy of the object, i.e. (objectID, copy), and receiver hoards a copy from the sender. After C-hoarding, both the sender and the receiver hold copies, i.e. sender (objectID, copy), and receiver (objectID, copy).

3.2 Currency Hoarding

Currencies of an object are flowed among its replicas via currency hoardings. The total amount of currencies is fixed at any time. There're three modes of currency hoarding correspondingly to data hoarding: currency G-hoarding, currency P-hoarding, and currency C-hoarding, which are illustrated in Figure 2, where a, b and c are integers, $0 < a \leq 50$, $0 < b \leq 50$ and $0 \leq c \leq 50$.

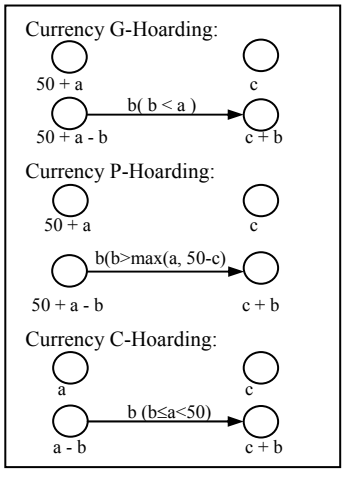

Figure 2. Currency hoarding

Currency G-hoarding: Before hoarding, the sender holds a primary currency 50 + a (a>0). After b ($0 \leq b < a$) currency goes to the receiver, the sender still owns a primary currency. Since the total currency for each object is fixed, it's definitely that the receiver owns a copy currency.

Currency P-hoarding: Before hoarding, the sender holds a primary currency. After b ($b \geq 50 - c$) currency goes to the receiver, the sender loses the primary currency and only holds a copy currency. The receiver now gets majority of currency and becomes the new primary holder.

Currency C-hoarding: Before hoarding, the sender holds a copy currency a. After b ($b \leq a<50$) currency goes to the receiver, the sender holds a copy currency. If b is equal to a, then the copy in the sender becomes read-only. The receiver now gets more currency.

There are three special cases related to primaries. *Primary dilution*: for a sender with a primary currency, if a currency hoarding-out results in a less than primary currency left, the sender holds copy currency only and we say the primary is *diluted* at the sender. *Primary concentration*: for a receiver with a copy currency, if it collects a primary currency after a currency hoarding, as we show above in currency C-hoarding, if (c + b) > 50, we say the primary is *concentrated* at the receiver. *Primary transfer*: if both primary dilution and primary concentration occur in one hoarding session, we know it is currency P-hoarding, in which the primary is *transferred* from the sender to the receiver.

3.3 Data Synchronization

The committed updates of an object are propagated via anti-entropy sessions to other replicas, hence eventually bringing the database to a consistent state. The goal of anti-entropy is for two replicas to bring each other up-to-date. Version vectors are used to compare the differences of committed versions between the communicating hosts. Each replica maintains two versions of updates: committed update version and tentative update version. The committed version numbers are global and all replicas of an object see committed updates in the same order. The replica with the larger committed version number brings the other up-to-date in pair-wise fashion. The tentative version numbers are local and will not be seen by others. It is only for local use to present the generation order of tentative updates.

3.4 Currency Synchronization

A *currency quorum* for an object consists of a set of replicas whose currency summary is the plurality of total fixed currency. An update should be accepted by a currency quorum to get commitment. In this way, no two updates can succeed simultaneously thus avoiding the possibility of conflicts. We borrow bounded weighted voting scheme from Deno to synchronize currencies. It propagates election information while collecting currencies, and commits tentative udpates when the plurality of currencies is reached, i.e. a currency quorum is formed. For details, please refer to Deno papers [4, 5, 6, 12].

From the description of the data/currency synchronization, it can be seen that an update's life cycle consists of: 1. a replica issues a tentative update; 2. the tentative update becomes a candidate in an election; 3. the candidate and its voting currencies are propagated in a lazy manner; 4. the candidate gathers voting currencies as it passes through hosts; 5. the candidate is awarded as the winner of the election when it gathers plurality of currencies, i.e. the tentative update gets committed, otherwise it is aborted; 6. the update commitment information is propagated to reach all hosts eventually via anti-entropy sessions. In this life cycle, steps 2 to 5 follow the trace of currency synchronization, and step 6 is performed by data synchronization.

4. METADATA

Metadata [3] is the activity styles and constraints data of mobile databases. It consists of connectivity, availability, data access patterns and user-defined constraints. The roles of metadata include: 1. identify data user cares about (Domain): what data interests a user? 2. specify relative worth of data (Utility): what is its relative worth? 3. indicate power of mobile host (Priority): what's my activity pattern? The focused subset of metadata in this paper includes connection duration, disconnection frequency, read frequency, tentative update frequency, commit rate, and commit delay.

Sliding window scheme is used to implement incremental metadata collection. A sliding window is a time frame with a predefined time interval, and the end point of the frame being the current local time. Window width is the time duration of the sliding window. Events in such sliding window include connection, disconnection, read, tentative updates and update commitment. Such events are logged in the time order that they occur. Since the capacity of a mobile host is limited, we cannot afford to storage them all from the very beginning. So the metadata collection algorithm is designed to log only the events in the time range of a sliding window. The log is called bounded window log. Sliding windows consume events as well as store them. When an event occurs, if the size of the sliding window has not reach its limit, the event is logged and its effect is reflected to the metadata. If the sliding window grows to its upper limit, one or more oldest events are retired and its effect is eliminated from the metadata when the new event is added. In this way, the metadata always remember the effects of events in the most recent bounded history.

The overhead to collect metadata for a replica is trivial. The main overhead is the cost to store the event and update metadata when an event occurs. There is no message cost since it happens totally locally.

5. DYNAMIC CURRENCY REDISTRIBUTION PROTOCOL

To allocate currency while taking advantages of metadata, a policy of assigning weights to metadata and mapping it to currency is needed. Intuitively, positive weights should be given to the metadata of connection duration, read frequency and tentative update frequency. While the metadata of disconnection frequency and commit delay should be assigned negative weights since they reveal the inability of a mobile host.

Weight of a replica X1 is defined as $W(X1) = \sum_{m \in M} m * w(m)$, where M is the set of involved metadata and w(m) is the weight of metadata m. Metadata catch dynamic

characteristics of mobile database and decide weights of replicas, which are proportionally mapped to replica currencies. In this way, currencies flow in the system dynamically and adapt to the use of databases.

To redistribute currencies correctly, any amount of currencies of any replica will be used exactly once in every election. The scenarios of redistributing currencies are given as follows.

5.1 Among Fixed Hosts and/or Strong Connected Mobile Hosts

Currency redistribution among wired or strong connected hosts is performed via distributed transactions. Assuming replicas of object X are hoarded by fixed hosts and/or strong connected mobile hosts numbered from 1 to k, their weights are W_1, … W_i, … W_k and their currencies before redistribution are C_1, … C_i, … C_k. The amount of currency a replica should get after currency redistribution is directly proportional to its weight:

$$Ci' = (\sum_{j=1}^{k} C_j) * W_i / (\sum_{j=1}^{k} W_j)$$

5.2 Between Two Weak Connected Mobile Hosts

In this case, mobile hosts communicate in pair-wise manner. The amounts of currencies are directly proportional to the weights of their replicas. Assuming both mobile host M1 and M2 have replicas of object X, and hold currencies C1 and C2 respectively. The weights of replicas calculated from metadata are W1 and W2, respectively. Then the target amounts of currencies of the two replicas should be:

Currency for the replica in M1 is C1' = (C1 + C2) * W1 / (W1 + W2), and Currency for the replica in M2 is C2' = (C1 + C2) * W2 / (W1 + W2). So the amount of currency hoarding from the sender to the receiver is: $\Delta C = | C1' - C2'| = (C1 + C2) * | W1 - W2| / (W1 + W2)$.

5.3 Between a Mobile Host and a Fixed Host

In this case, the currency redistribution may involve one or more neighbors of the fixed host in the distributed database. It may cause currency shrink or currency swell to the neighbors of the fixed host in the fixed network. As illustrated in Figure 3, when the replica in a mobile host hoards DC currency from a replica in a fixed host, the neighbors in the distributed database shrink their currencies to maintain the mapping policy. When the replica in a fixed host hoards DC back, its neighbors swell their currencies, as shown in Figure 4. Currency shrink and currency swell are performed via distributed transactions and are transparent to the mobile host.

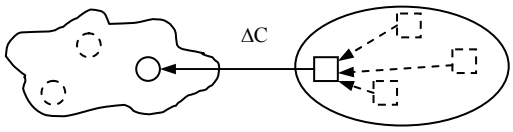

Figure 3. Currency Shrink

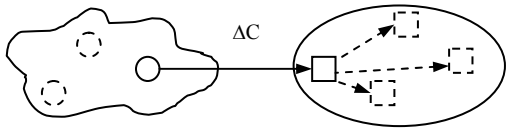

Figure 4. Currency Swell

The amount of currency a fixed host shrinks or swells is directly proportional to its weight of the replica:

$$\Delta C_i = \Delta C * W_i / (\sum_{j=1}^{k} W_j)$$

5.4 Ad-hoc Databases Check-out/Check-in Currencies

When the mobile hosts in ad-hoc network are all docked (strong connected), all the mobile and fixed hosts have good communication. Traditional distributed database management can be used to run the system, except that metadata collection is performed. When some or all of mobile hosts weak connected, they form an ad-hoc network and their database is an ad-hoc database. As shown in Figure 5, the ad-hoc database checks out some or all of currencies and perform database operations in ad-hoc manner. The DBMS should switch to dynamic, weighted voting scheme to take the advantages of database dynamic design and reconfiguration. For the replicas of an object, the currencies are redistributed referring to metadata using the scheme given in section 5.2. Primaries may be diluted, concentrated or transferred in or between ad-hoc database and the distributed database. When all mobile hosts come back to dock into fixed network, the ad-hoc database checks its currencies back in the distributed database, which is demonstrated in Figure 6.

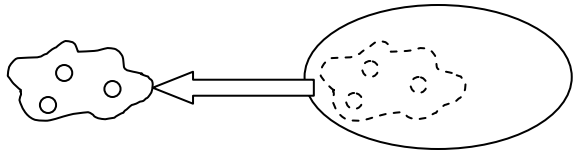

Figure 5. Ad-hoc database checks out currencies

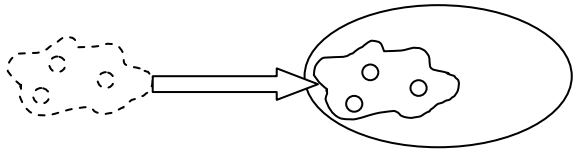

Figure 6. Ad-hoc database checks in currencies

6. SIMULATION

An experimental simulation is implemented to assess the benefits of our protocols. It simulates the behaviors of mobile hosts under two cases: dynamic currency redistribution and static currency allocation. Five mobile hosts are modeled and each of them has an event creator which generates events to simulate the activities of a real mobile host. The main assumptions are: 1. the database is a single object database, i.e. there is only one object shared

among these hosts; 2. data hoarding is assumed to be performed prior the simulation, i.e. each host has a non-zero currency of the object when the simulation is started; 3. the generated tentative updates are independent so that the abort or commitment of a tentative update does not have any effect on the other tentative or committed updates; 4. applications read committed updates only and tentative updates are unavailable before they get committed; 5. the mobile hosts are failure free, so we need not worry about currency loss.

The metrics of our focus are commit rate, commit percentage and mean commit delay. The commit rate is defined as the average number of committed updates per time sliding window. The commit percentage is the ratio of committed updates to the total tentative updates generated in the system. The commit delay of a committed update is defined as the difference from the tentative update issue time to the time that a host knows its commitment. Mean commit delay is the mean delay time of committed updates in the system averaged to all the replicas.

The metadata in consideration includes connect duration, disconnect frequency, read and tentative update frequencies, commit rate and commit delay. The simulation settings are given in Table 1.

Parameter	Setting						Description
windowWidth	6 seconds						Width of time sliding window
simuTime	80 seconds						The time of simulation running
simuSection	[0, 40], [40, 80]						Simulation time sections
updateRatio	0.1, 0.2, 0.3, 0.33, 0.4, 0.5, 0.6						The ratio of tentative updates to read events
	Hosts	Host0	Host1	Host2	Host3	Host4	
Internals between two consecutive events (secs)	1st section	1	1	1	1	0.1	Intervals between two consecutive events in different time sections
	2nd section	0.1	1	1	1	1	
Initial currency allocation (total = 100)	Uniform	20	20	20	20	20	These are the initially allocated currencies. For static cases, they are not to be changed.
	Primary	60	10	10	10	10	

Table 1. Primary simulation parameters

Two scenarios are studied to qualify the performance: one is uniform scenario, i.e. the currencies are initially allocated to each replica uniformly; and the other is primary scenario, in which the primary currency is initially given to one replica. The initially allocated currencies remain unchanged in static cases while dynamically change with weights/metadata of replicas in dynamic cases. The intervals between two consecutive events configure host 4 and host 0 to be hottest in the 1st and 2nd time sections respectively. Currencies flow to host 4 in the 1st time section then to host 0 in the 2nd time section.

The simulation results demonstrate that the dynamic design is more beneficial than the static design. The commit rate, commit percentage and mean commit delay of dynamic design versus static design of uniform case are given in Figure 7, Figure 8 and Figure 9.

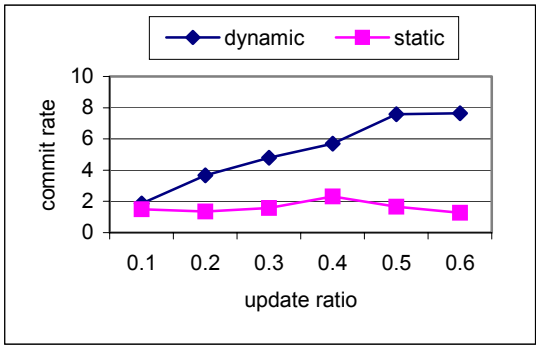

Figure 7. Commit rate of uniform case

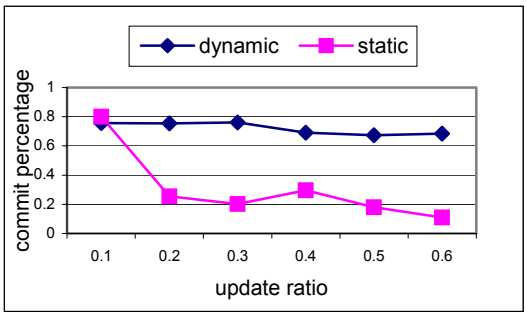

Figure 8. Commit percentage of uniform case

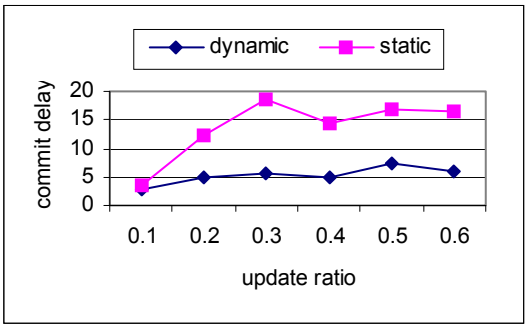

Figure 9. Mean commit delay of uniform case

It can be clearly seen that the performance of the dynamic design is significantly better than that of the static design except when the update ratio is rather low. Specifically, when the update ratio is 0.1, little benefit of dynamic design is shown over the static design. But with the increment of update ratios, the benefit

becomes significant: the commit rate of the dynamic design increases while that of the static design remains almost unchanged; the commit percentage of the dynamic design decreases only a little but that of the static design decreases rapidly; the commit delay of dynamic design is much less than that of static design. The reason is that in the dynamic design, the currency is always flowing to the hottest replicas thus there are more tentative updates to get plural currency and be committed rapidly. While in the static case, the currency allocation is fixed and there is no priority for the hottest replica, so the commit rate is almost a constant no matter the number of tentative updates generated, and the mean commit delay is higher.

Figure 10, Figure 11 and Figure 12 illustrate the commit rate, commit percentage and mean commit delay of dynamic design versus static design of primary case.

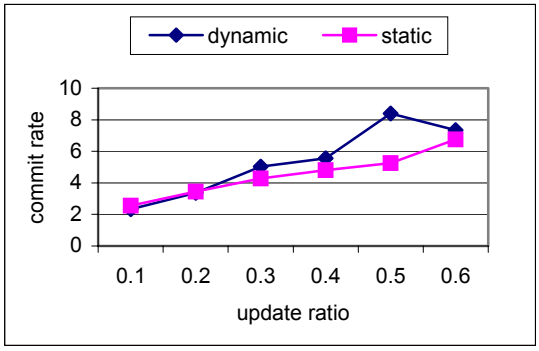

Figure 10. Commit rate of primary case

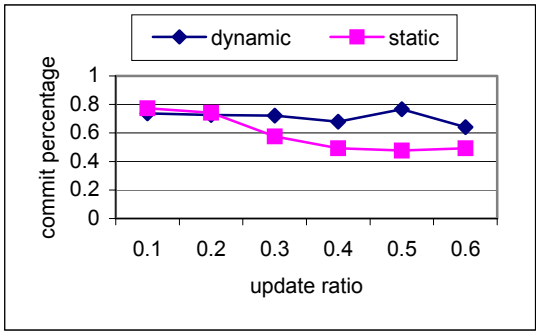

Figure 11. Commit percentage of primary case

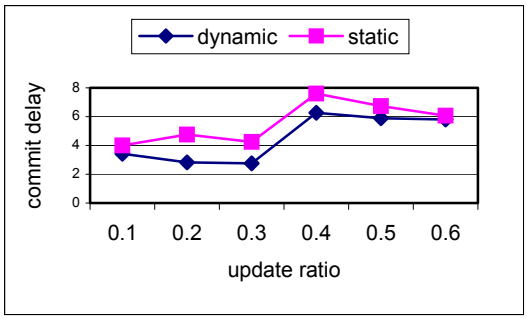

Figure 12. Mean commit delay of primary case

It is observed that in the primary case, the difference of dynamic design and static design is not so significant as the uniform one. The reason is that the primary currency happens to be initially allocated to one of the hottest replicas so the performance of static design is not too bad since a fixed primary currency makes all the primary's local tentative updates committed. It is reasonable that the benefits of the dynamic design will be more significant, even better than the uniform case, if the primary currency is unfortunately given to a cold replica.

The main overhead of the dynamic design is metadata management and currency redistribution. As stated in section 4, the overhead of metadata collection is trivial. The main overhead related to currency redistribution is message cost, but the redistribution information can be piggybacked with the anti-entropy messages and it is only a few bytes thus save bandwidth of mobile hosts. Therefore, the overhead of the dynamic design is little.

7. CONCLUSION AND FUTURE WORK

In this paper, we described a unifying concept of mobile database and proposed generalized modes of semantically rich hoarding and synchronization. Moreover, a metadata protocol and currency redistribution protocol are demonstrated to enable dynamic database design and reconfiguration in mobile environments. The benefits of dynamic design versus static design are shown via the simulation results.

Mobile databases consist of a large scope of research and application issues. There is a lot of future work to do: 1. in order to get better performance adaptive to the system changes, the full set of metadata should be considered. This requires a more completed metadata collection mechanism; 2. associate rules to predict metadata to build rule-based adaptive currency redistribution. If there is a scheme to predict what a mobile host needs in the future and hoard the data in advance, the performance will be brought up significantly; 3. detailed algorithms and more use of broadcast should be explored. It is more powerful to use broadcast to propagate update commitment, election winner awards and hint of primary location; 4. mobile transaction models will be built based on the traditional ones in that the operations in mobile databases include hoarding and synchronization as well as read and write. 5. currency-driven synchronization policies and topologies will be investigated in a mobile/weakly-connected replicated system.

8. ACKNOWLEDGEMENT

We would like to thank Hua Li and Charnyote Pluempitiwiriyamei for contributing to some of the ideas in this paper. We would especially like to thank Dr. Ugur Cetintemel for providing constructive critique.

9. REFERENCES

[1] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, Broadcast Disks: Data Management for Asymmetric Communication Environments, SIGMOD Conference, 1995

[2] D. Agrawal, A. El-Abbadi, and R. Steinke. Epidemic algorithms in replicated databases, In Proc. 16th ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems (PODS), Tucson, Arizona, May 1997

[3] M. Cherniack, M. Franklin, S. Zdonik, Data Management for Pervasive Computing, VLDB, Rome, Italy, Sep. 2001

[4] U. Cetintemel and P. J. Keleher, Light-Weight Currency Management Mechanisms in Mobile and Weakly-Connected Environments. Journal of Distributed and Parallel Databases, 11(1), pages 53-71, January 2002.

[5] U. Cetintemel, P. J. Keleher, M. Franklin, Support for Speculative Update Propagation and Mobility in Deno, The 22nd International Conference on Distributed Computing Systems, 2001

[6] U. Cetintemel and P. J. Keleher, Light-Weight Currency Management Mechanisms in Deno, In the 10th IEEE Workshop on Research Issues in Data Engineering (RIDE2000), February 2000.

[7] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch, The Bayou Architecture: Support for Data Sharing among Mobile Users, Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December 1994

[8] Michael J. Franklin, Challenges in Ubiquitous Data Management Informatics: 10 Years Back, 10 Years Ahead, LNCS #2000, R. Wilhiem (ed)., Springer-Verlag 2001

[9] J. Gray, P. Helland, P. O'Neil, D. Shasha, The Dangers of Replication and a Solution, SIGMOD'96, Montreal, Canada, 1996

[10] A. Helal, J. Hammer, A. Khushraj and J. Zhang, "A Three-tier Architecture for Ubiquitous Data Access" Proceedings of the First ACS/IEEE International Conference on Computer Systems and Applications, Beirut Lebanon, June 2001

[11] S. Jajodia and D. Mutchler, Dynamic voting algorithms for maintaining the consistency of a replicated database, ACM Trans. on Database Systems, Vol. 15, No. 2, June 1990

[12] P. J. Keleher, Decentralized Replicated-Object Protocols, In The 18th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, April 1999.

[13] G. H. Kuenning and G. J. Popek, Automated Hoarding for Mobile Computers, Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), St. Malo, France, October 5-8, 1997.

[14] J. J. Kistler, M. Satyanarayanan, Disconnected Operation in the Coda File System, In Proc. Of the ACM Symposium on Operationg Systems Principles, October 1991

[15] Y. W. Lee, K. S. Leung, M. Satyanarayanan, Operation-based Update Propagatioin in a Mobile File System, Proceedings of the USENIX Annual Technical Conference, Jun. 1999, Monterey, CA

[16] M. Tamer Ozsu and Patrick Valduriez, "Principles of Distributed Database Systems", Prentice Hall Publisher, 1999

[17] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers, Flexible Update Propagation for Weakly Consistent Replication, Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint Malo, France, October 1997

[18] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu, "Mariposa: A Wide-Area Distributed Database System," VLDB Journal, vol. 5, pp. 48-63, 1996.

[19] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, Spawn: A distributed Computational Economy, IEEE Transactions on Software Engineering, vol. 18, pp. 103-117, February 1992.

[20] O. Wolfson, S. Jajodia, and Y. Huang, An adaptive data replication algorithm, ACM Trans. on Database Systems, Vol. 22, No. 2, June 1997

[21] O. Wolfson, S. Jajodia, An algorithm for dynamic data allocation in distributed systems, Information Processing Letters, Vol. 53, No. 2, 1995

[22] O. Wolfson, S. Jajodia, Distributed algorithms for dynamic replication of data, Proc. 11th ACM PODS Symposium, San Diego, Calif., June 1992

[23] C. A. Waldspurger and W. E. Weihl, Lottery Scheduling: Flexible Proportional-Share Resource Management, in Proceedings of the First Symposium on Operating Systems Design and Implementation, Monterey, CA, November 1994