Data Transfer Evaluation of Optimistic Data Consistency Model

Masahiro Kuroda†, Ryoji Ono††, Takashi Watanabe†, Tadanori Mizuno†, Yoshiki Shimotsuma††

Abstract

There are two basic data consistency schemes, pessimistic consistency and optimistic consistency, with respect to data consistency and data availability. The optimistic consistency model is suitable for high latency wireless networks. This paper describes data versioning and its access control scheme to reduce data transfer for data synchronization, discuss how to reduce the data transfer size for synchronization. We evaluate the data size reduction by generating arithmetic formulas in each optimizing technique and confirm that the formulas are tolerable in determining the data size of large scale mobile data sharing system. Lastly, we recognize that two parameters to define *access domain* showed the flexibility of system configuration and effectiveness in data synchronization.

1. Introduction

The advance of the mobile computing infrastructure enables "anytime, anywhere" global information services. We have proposed an optimistic data consistency model [4,5,13] to ensure data consistency and data availability in a wide area and at most times, that enables users to access data in wireless networks, intranets, and Internet.

This model needs following four key requirements for supporting wide-area mobile network computing:

- Building an infrastructure which offers data consistency and data availability in any network;
- Supporting disconnected operations;
- Supporting data replication;
- Reducing wide-area communication costs;

So far, two basic schemes, pessimistic consistency[1] and optimistic consistency[2-3], have been established with respect to data consistency and data availability. The scheme for pessimistic consistency is not suitable

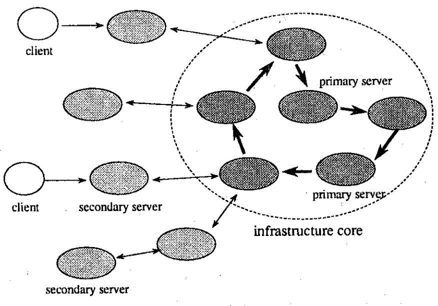


fig.1 architecture

† Shizuoka University
†† Mitsubishi Electric Corporation

for disconnected operations because this scheme all the time requires consistent. The traditional optimistic consistency scheme, such as those deployed in Coda[2] and Ficus[3], were basically aimed to offer data consistency and availability for devices using reliable and low latency networks, such as LAN. Also these schemes were targeted to Unix file systems and no features for data versioning and data synchronization controls.

This paper focuses on the reduction of communication costs in the four key requirements. We need to minimize the amount of data to exchange among all the devices and servers involved in data synchronization. We describe how to reduce data transfer and evaluate their effectiveness. In the evaluation, we assume a pattern of mobile data access, calculate the total data to exchange among mobile terminals and a primary server, and measure the total data transferred. We also propose an *access domain* scheme for data versioning to reduce the data transfer dynamically for adjusting to various kind of access model, such as a client-server system and a data collaboration system.

2. Data Consistency Architecture

The data consistency model is based on the Mobile Network Computer Reference Specification (MNCRS). The part of its goal is the standardization of data synchronization interface[6]. In this model, mobile terminals and servers in a network have a replica of some data, exchange its data versions, and converge to its consistent data.

As with MNCRS, our model is implemented using Java[7]. Therefore all descriptions in the rest of the paper will use Java's terminology. A *SyncStore* is a container which stores and maintains data. Data are represented by *Synchronizable* objects which are Java Serializable objects. The methods *put()/get()* are used to store/retrieve Java objects from the containers. Each object is defined as a subclass of either *Reconcilable* or *Diffable*, which are both a subclass of *Synchronizable*. A *Reconcilable* object compares its *time stamp* with that of the object having the same identification number in the peer *SyncStore* and replaces the original object with the other object if the original *time stamp* is older. A *Diffable* object compares its *time stamp* with that of the object in the peer and apply the log data in the other object to the original object.

2.1. Data Synchronization Method

Followings are the data synchronization methods for *Reconcilable* and *Diffable* objects.

(1) Data Replica Transfer (DRT) method

During data synchronization between two *SyncStores*, this method exchanges an object in a *SyncStore* with the other object coming from its peer *SyncStore* and also having the same object identification number (object ID). This method is used for *Reconcilable* objects and suitable for systems in which each update influences all the data in an object. This method consumes little CPU time.

(2) Differential Data Transfer (DDT) method

This method assumes version vectors (VV's) to detect concurrent update conflicts, first proposed by

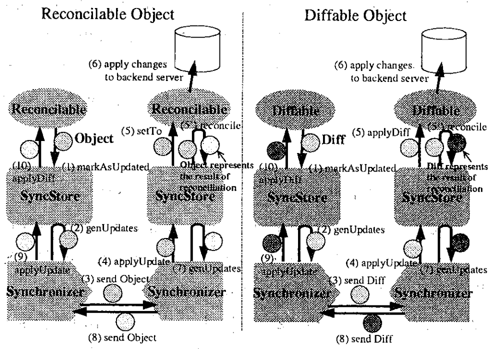


fig.2 data flow in one synchronization cycle

Parker, etc.[9], and update logs recording the changes of data in the object.

This method, at first, compares the version vectors of two *SyncStores*. According to the VV's, It brings in only the Differential part of an object from the other *SyncStore* and applies the part to the object. This method is for *Diffable* objects and is fit for systems in which each updates are local to data in an object or for collaboration systems sharing data among many participants.

3. Data Transfer for Synchronization

This architecture compares two objects in *SyncStores* on Different Java Virtual Machines, identifies the Difference and updates both objects in *SyncStores*.

3.1. Synchronization Flow

Synchronizers are the key components in bringing any pair of *SyncStores* into consistent state. Assume *SyncStore*1 is trying to synchronize with *SyncStore*2 and all the components bundled to *SyncStore* i(i=1,2) are annotated as *Synchronizer* i, etc.The basic flow of a typical synchronization is as follows:

- *Synchronizer* 1 sends *SyncStore* 1's version vector to *Synchronizer* 2. A version vector is a data structure that is used to describe a *SyncStore*'s current state.
- *Synchronizer* 2 receives the version vector from *Synchronizer* 1, compares the two *SyncStores*' version vectors, and figures out what are the updates that are logged in *SyncStore* 2 but not in *SyncStore* 1.
- *Synchronizer* 2 sends the updates collected in the previous step to *Synchronizer* 1 along with the *SyncStore* 2's version vector.
- *Synchronizer* 1 calls *SyncStore* 1's applyUpdate() method for updates received in order to reflect these updates to the *SyncStore*.

3.2. Version Vectors

A *SyncStore* keeps log data that represent the modifications to objects and consists of following information;

(1) Replica ID indicating a *SyncStore* at which logging is performed.

(2) Vt having a virtual time in the *SyncStore* at which the log data is generated.

A version vector is a list of the elements of the form ($ssid_i$, vt_i), where $ssid_i$ is the i-th Replica ID, and vt_i is a *time stamp* generated by the i-th *SyncStore* using a virtual time defined in each *SyncStore*. When two *SyncStores* synchronize each other, the version vectors are exchanged between the *SyncStores*. Using the VV's and update logs presented above, we can compute the Difference of a *SyncStore* from the other, detect concurrent update conflicts, and reduce the amount of logged updates.

3.3. *Access domain*

When information in a *SyncStore* is shared among users working in a network, the size of version vectors will become large even if some *SyncStores* are not updated so often. Mobile users may sometimes share the same data and update it concurrently at Different *SyncStores*. Or users communicate only with its server to retrieve data in the server *SyncStore*.

Data access should be flexible enough to adjust to various kind of data usage and also be minimun in size to transfer data between *SyncStores*. We introduce an *access domain* mechanism to restrict updates of version vectors which belongs to the same *access domain*.

4. Data Transfer Optimization

As for the data transfer reduction during data synchronization, there is no simple answer for it. We will discuss following three approaches for data reduction.

(1) Data simplification in application layer

Applications only know how to simplify data to send to the other side. There is no unique method in this layer, but some encoding/decoding mechanisms are popular to reduce the data size for exchanging between two *SyncStores*.

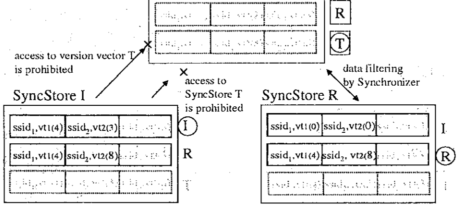

communication among Synchronizers, access to SyncStores, access to objects
SyncStore T
I
R
T
access to version vector T is prohibited
access to SyncStore T is prohibited
data filtering by Synchronizer
SyncStore I
SyncStore R
ssid1,vt1(4)
ssid2,vt2(3)
ssid1,vt1(4)
ssid2,vt2(8)
ssid1,vt1(0)
ssid2,vt2(0)
ssid1,vt1(4)
ssid2,vt2(8)

fig.4 access domain

(2) Data reduction in data object layer

Data Replica Transfer (DRT) method and Differential Data Transfer (DDT) method have Different data reduction mechanisms. Differential Data Transfer method, especially, can reduce data transfer by an *access domain* of VV's. Also the data reduction in this layer depends on the storage form of objects. Java object serialization is the mechanism to keep an object self-contained and is used for object transfer between *SyncStores*. Object serialization has also a function to hook user defined Java externalization function. This function will reduce object information in a serialized object.

(3) Data stream compression in data transfer layer

The simple technique is to use compress()/ decompress() functions to send/receive data between the *Synchronizers*.

5. Data Transfer Evaluation Model

As a data transfer reduction model, we assume mobile professionals working outside of an office and maintaining data on the mobile terminals. They visit customer sites and check equipment at the sites. They sometimes share the same data by replication and update them concurrently at different sites. Or they only communicate with servers in their office and refer to the latest data.

Figure 5 shows following two cases:

(1)A mobile professional belongs to an *access domain* and communicates only with a server. The data access does not assume any interactions with other professionals. (2)Or mobile professionals belong to the same *access domain* and update the data that others in the same domain are looking at.

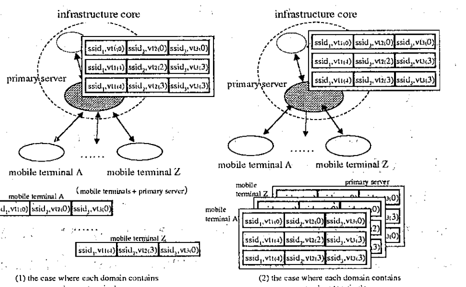

infrastructure core
primary server
ssid1,vt1(0) ssid2,vt2(0) ssid3,vt3(0)
ssid1,vt1(4) ssid2,vt2(2) ssid3,vt3(3)
ssid1,vt1(4) ssid2,vt2(3) ssid3,vt3(3)
mobile terminal A
mobile terminal Z
(mobile terminals + primary server)
mobile terminal A
ssid1,vt1(0) ssid2,vt2(0) ssid3,vt3(0)
mobile terminal Z
ssid1,vt1(4) ssid2,vt2(3) ssid3,vt3(0)
(1) the case where each domain contains only one terminal.
infrastructure core
primary server
mobile terminal A
mobile terminal Z
primary server
mobile terminal Z
mobile terminal A
(2) the case where each domain contains one or more terminals.

fig.5 structure of version vector in each evaluation models

Assuming these situations, we will, first, discuss DRT and DDT methods and lead to arithmetic formulas for estimation of data transfer size during a synchronization cycle using the data transfer optimization described above. Next, we will compare the estimated data with the measured data to confirm that the arithmetic formulas are appropriate for the data size estimation. Lastly, we will discuss about an application to a large-scale mobile network system.

The experimental system has three mobile terminals and one server. We also assume that each mobile terminal synchronizes with the server, not with other mobile terminals directly. The reason is that wireless communication system communicates with others via an access point.

5.1. Evaluation Parameter

Followings are the definitions of parameters used in the data transfer estimation.

(1) Number of terminals : n_{term}[unit]

(2) Data size per each update;

(a) Original worksheet : d_{origin} [Bytes]

(b) Encoded worksheet : $d_{encoded}$ [Bytes]

(c) Customer-code table: d_{cus}[Bytes]

(d) Mobileworker-code table: d_{per}[Bytes]

(3) Frequency of data update;

(a) Worksheet update: λ [times/hour]

(b) Customer-code table update:
λ cus[times/hour]

(4) Worker-code table update:
λ per[times/hour]

Data size in data synchronization;

(a) Original data: S_{origin}[Bytes]

(b) Encoded data: $S_{encoded}$[Bytes]

Assume t hours has passed after the last data synchronization, the average update log is λ t, and the number of sites involved in data synchronization is n [units]. The data size of data synchronization will be followings.

(5) $S_{origin} = d_{origin} \lambda \ nt$

(6) $S_{encoded} = (d_{encoded} \lambda + d_{cus} \lambda_{cus} + d_{per} \lambda_{per})nt$

5.2. Evaluation Data

The worksheet table described below shows the data used for the evaluation. As for encoding/decoding, two name-code transformation tables are used for customer names and mobile worker names.

Suppose that an worksheet is updated twice per hour, a customer code or a worker code is updated 0.1 times per hour, and synchronize with the server 0.25 times per hour in average, updates will become followings.

(7) Worksheet update: 8 records/synchronization

(8) Customer/Worker code table update:
0.4 records/synchronization

As for update of a data object, we assume that "Begin date", "Customer code", and "Worker code" are updated at the first time and second time "Finish date" and "Check item" are modified. The average size of each update is as follows.

(9) Size of average data update:
$d_{update} = \{(4+4+4)+(4+32)\}/2 = 24$[Bytes]

5.3. Object Serialization

The data synchronization utilizes Java object serialization for sending data objects to the other *SyncStore*. The serialization is a mechanism to embed object information into a serializable object to make it self-contained.

When we estimate the data transfer size, we need the size of application data and also the object information automatically serialized by Java. But Java also has an interface to define application specific serialization, named Externalization, to reduce the object size. We will use Externalization to minimize data for synchronization.

(1) The average size of the object information for a serialized log object, denoted as C_1, is 422 bytes, whereas the average size of an externalized log object, denoted as C_1', is 162 bytes as shown in Figure 6.

Table.1 structure of evaluation data

Items	Size(bytes)	Coded Size
Identification number	4	▨
Begin date	4	▨
Finish date	4	▨
Customer name	32	4
Worker name	32	4
Check items	32	▨
Total	108	52

(2) The size of object information for a version vector, denoted as $C_2(n)$, which depends on the number of mobile terminals joined in data synchronization , is 2n+91 bytes.

6. Evaluation of synchronization method

There are two synchronization methods. One is Data Replica Transfer method (DRT), the other is Differential Data Transfer method (DDT). In this section, we will estimate the amount of data in both DRT and DDT methods based on the following assumptions.

(1) 128 bytes data per record
(2) 24 bytes data for an update
(3) 5 updates per one terminal per an hour
(4) One synchronization per an hour

The total data transfer is shown in Figure 7. The graph indicates that DDT is much efficient than DRT under the above conditions. Comparing to DRT, DDT shows that the more the number of mobile terminals increases, the less the amount of data is.

7. Evaluation of Differential Data Transfer method

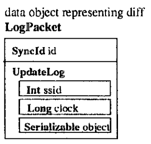
We will discuss data optimization techniques in the three software layers defined in the section 4.

(1) Data simplification in application layer

The optimization in this layer depends on the nature of applications and there is no generally acceptable technique but encoding/decoding. We will use only encoding/decoding in this layer.

There are two cases in this evaluation.

(a) Use worksheet with customer/worker names
(b) Use worksheet with encoded customer/worker names



```
LogPacket:        melco.datasync.LogPacket
Log:              melco.datasync,Log
UpdateLog:        melco.datasync.UpdateLog
TimeStamp:        melco.datasync.TimeStamp
SyncId:           javax.datasync.SyncId
Serializable:     java.io.Serializable
StatusReportDiff: evaluate.StatusReportDiff
```

type	size per item [Byte]	number	data size [Byte]
stream	4	--	4
reference to an object	1	5	5
object	1	5	5
class definition for object	15	6	90
field	3	8	24
field whose type is object	5	4	20
filed whose type is array	5	1	5
reference to an array	1	1	1
array	5	1	5
class definition for array	17	1	17
TOTAL			176

The names of classes/fields used in an LogPacket consist of 246 characters (246 bytes).

Therefore. data size added by Serialization is
176 + 246 = 422 [Bytes]
per an LogPacket.

fig.6 example of data size of an object

(2) Data reduction in data object layer

This layer has two methods for data reduction. One is to minimize version vectors by setting up *access domains*, the other one is to pack data using Externalization.

We introduced an *access domain* to restrict data sharing among mobile terminals and their server. Followings are the cases for evaluation.

(a) Not to use *access domains*
(b) Apply an *access domain* to mobile terminals and their server and not to have version vectors of mobile terminals not belong to the *access domain.*
(c) Apply an *access domain* to mobile terminals and their server and not to include any elements in a version vector that represent mobile terminals outside of the *access domain.*

As for data compaction, we will use Externalization in representing an object. There are two cases described below.

(a) Use standard Java object serialization
(b) Use Externalization for application data objects, Differential objects, and synchronization management objects.

(3) Data reduction in data stream layer

We will use simple compress/uncompress functions for data stream between two *Synchronizers*.

7.1. Estimation of Data Size for Synchronization

We will assume following data to generate arithmetic formulas for each data transfer optimization.

(a) Transfer data size $d_{encoded}$' (=57bytes) :Add to $d_{encoded}$ an object identification(=4bytes) and a status flag(=1byte) that represents which fields in the application data are modified.

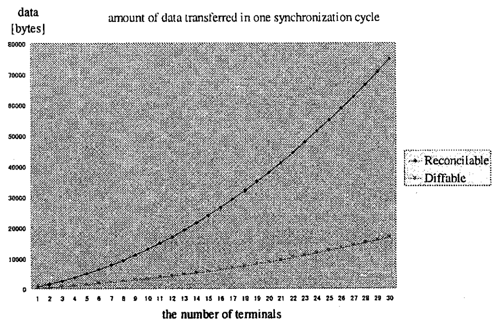


fig.7 Reconcilable vs. Diffable

(b) Transfer data size of one update d_{update}' (=29bytes): Add to d_{update} an object identification (=4bytes) and a status flag(1byte) that represents which fields in the application data are modified.

(c) Version vector size for data exchange d_{vv}' (=$36(n_{term}+1)^2$)) : $d_{vv}=12(n_{term}+1)^2$ is the size of a VV.
The data exchange transfers 3 times d_{vv} between two *SyncStores* as shown in Figure 8. The size d_{vv} is the multiplication of the number of mobile terminals+server and Replica ID (4bytes) + Vt(8bytes).

(d) *Time stamp* for version management d_{ts}(=12bytes): Replica ID(4bytes)+ Vt(8bytes)

(e) Additional data for data transfer protocol d_{proto} (=$64n_{term}$+172bytes): As showen in Figure8, the data transfer protocol exchanges *SyncStore* configuration data, VV's, Start/Continue/Stop data packet headers. The sum is $8\lambda n_{term}t+172$. In the model, it becomes $64n_{term}$+172bytes.

Followings are the data sizes for all the data exchanges of one synchronization between two *SyncStores*. S_{apl} represents the size with no application level encoding, whereas $S_{apl\text{-}encode}$ is with the encoding.

$$S_{apl} = (d_{encoded}' + d_{ts})\lambda\ n_{term}t + d_{vv}' + d_{proto} \quad (1)$$
$$= (57+12)\times 8\times n_{term}\times 1+36(n_{term}+1)^2 + 64n_{term}+172$$
$$= 36n_{term}^2 + 688\ n_{term} + 208$$

$$S_{apl\text{-}encode} = (d_{update}'+d_{ts}+d_{cus}\lambda_{cus}+d_{per}\lambda_{per})\lambda\ n_{term}t + d_{vv}'+d_{proto} \quad (2)$$
$$= (29+12+0+0)\times 8\times n_{term}\times 1+36(n_{term}+1)^2 + 64n_{term}+172$$
$$= 36n_{term}^2 + 464\ n_{term} + 208$$

The arithmetic formulas represent application data

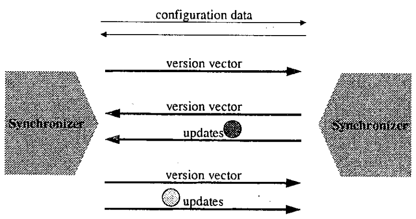


fig.8 data synchronization protocol

and object management data and do not include any additional serialization data. We will add the serialized data C_1=422 and $C_2(n_{term})=2n_{term}+91$ to the above formulas to estimate real data exchange between *SyncStores*.

$$S_{apl}' = S_{apl}+C_1\times 8\times n_{term}\times 1+C_2(n_{term})\times 3 \quad (3)$$
$$=36n_{term}^2 + 4070\ n_{term} + 481$$

$$S_{apl\text{-}encode}' = S_{apl\text{-}encode}+C_1\times 8\times n_{term}\times 1+C_2(n_{term})\times 3 \quad (4)$$
$$= 36n_{term}^2+3846n_{term}+481$$

Formulas (3) and (4) are used for estimating data simplification in the application layer.

Next is the Data object layer. The *access domain* is a candidate to reduce VV's for synchronization.

We can define an *access domain* which consists of a mobile terminal and a server. In this configurat ion, only VV's which have the terminal and the ser ver are exchanged. Suppose the number of mobile t erminals involved in an *access domain* is n_{member} , t he VV's have n_{term} +1 elements with n_{member} +1 V V's.

We need an *access domain* name(=9bytes) for ea ch VV. Each mobile terminal has 9 bytes and its se rver $9n_{domain}$ bytes.

The data $d_{vv\text{-}domain}$ consists of VV's transferred fr om mobile terminals to the server (n_{member}+1) and al l the VV's in the server sent to mobile terminals. And totally 3 VV's are exchanged between *SyncStor es*.

$$d_{vv\text{-}domain} = \{12(n_{term}+1)(n_{member}+1) +9(n_{member}+n_{domain})\}\times 2 +12(n_{term}+1)^2+9(n_{member}+n_{domain})$$
$$= 12n_{term}^2+(24n_{member}+57)n_{term} + 42n_{member}+27\ n_{domain}+ 36$$

And add the additional data for Object serialization $C_2(n_{member})= 2n_{member}+ 91$.

$$d_{vv\text{-}domain}' = d_{vv\text{-}domain} + 3\ (\ 2n_{member}+ 91)$$
$$= 12n_{term}^2+(24n_{member}+59)n_{term} + 46n_{member}+27\ n_{domain}+ 309$$

$$S_{encode\text{-}domain}'$$
$$= (d_{encode}'+d_{ts}+d_{cus}\lambda_{cus}+d_{per}\lambda_{per}+C_1)\lambda\ n_{term}t + d_{vv\text{-}domain}' + d_{proto}$$
$$= (29+12+422)\times 8\times n_{term}\times 1 +12n_{term}^2+(24n_{member}+59)n_{term} +46n_{member}+27\ n_{domain}+ 309 + 64n_{term}+172$$

$$= 12n_{term}{}^{2}+(24n_{member}+3827)n_{term}$$
$$+46n_{member}+27\ n_{domain}+ 481$$

Suppose data exchanged are generated by mobile terminals in an *access domain*, VV's need only information in the *access domain*. This means that n_{term} in the above formula can be replaced by n_{member}.

$$S_{encode\text{-}domain}{}'' = 36n_{member}{}^{2}+3873n_{member}+27n_{domain}+481$$

When we use this formula in a client/server model, which means that $n_{member}=1$ and $n_{domain}=n_{term}$, the data transfer rate becomes below.

$$S_{encode\text{-}domain}{}'' = 27n_{domain}+4390$$

This indicates that the time complexity is not $O(n_{term}{}^{2})$ but $O(n_{term})$ and the total transfer data increases in proportion to the number of mobile terminals.

Next, we will apply Externalization to the *time stamp* object and to the log object, but not to VV's.

$$S_{domain\text{-}extern} = S_{apl\text{-}encode}+(C_1{}'+5)\lambda\ n_{term}t+C_2(n_{term})\times 3$$
$$= 36n_{term}{}^{2}+1806n_{term}+481$$

Lastly, Data compression is used for data stream between *Synchronizers*. We can improve about 20% in size.

7.2. Comparison of Estimated Data Transfer and Measured Data Transfer

Table.2 shows the data size estimation and measured data regarding each optimization. The table lists up result for 3 mobile terminals case for simplicity. The data sizes of Estimation and Measurement are close each other and the Differences are between the range of 2 and 8%. This result indicates that the arithmetic formulas follow the real transferred data and that the formulas can be used to evaluate for large wireless data sharing system which consists of more than 100 mobile terminals and is Difficult to build for the measurement.

7.3. Estimation for Large Data Sharing Systems

We will apply arithmetic formulas shown in Table.3 to large scale data sharing systems.

When the number of mobile terminals involved in the data sharing is less than 10, the data exchanged is small and the additional data which represent serialized objects are larger than the original data. In this small sharing environment, Externalization and the data stream compression are effective in data reduction (fig.9).

But, the number of terminals increases up to 100 terminals, the *access domain* scheme to reduce VV's size becomes effective in data sharing (fig.10). If the number of mobile terminals increases more than 100, the time complexity of Version Vectors $O(n^2)$ will be the dominant factor of data transfer size (fig.11).

Based on these considerations, we can say that the following two parameters that dynamically change the size of VV's are important for transfer data reduction of large-scale data sharing systems.

(1) The number of mobile terminals participated in an *access domain*: n_{member}

(2)The number of *access domains*: n_{domain}

Table.2 comparison of estimation and measured data

(1)	(2)	(3)	(4)	(5)	(6)
		13015	12616		
Appli-cation	Encoding	12343	13288	-5.3	
Data object	VV reduc-tion	12269	12658	-0.3	(*)
	Externali-zation	6223	6616	47.6	40-50% cut
Data stream	Compress		9113	27.8	20-30% cut

(1)Layer, (2)Optimization, (3)Estimation,
(4)Measurement, (5)Ratio %, (6)Result
(*) no reduction

Table.3 formulas

Optimization	Arithmetic formulas for data transfer	Comments
Original	$36n^2+4070n+481$	
Application encoding	$36n^2+3846n+481$	
Version Vector	$12n^2+(24m+3827)n+46m$ $+27d+481$	$m{:}n_{member}$, $d{:}n_{domain}$
VV reduction(1)	$12n^2+3878n+527$	$m=1, d=n$
VV reduction(2)	$36m^2+3873m+27d$ $+481$	$m=n$
External-ization	$36n^2+1806n+481$	
Data compression	$(36n^2+3846n+481)\times 0.8$	

8. Conclusions

We have evaluated the data transfer optimizations for the data synchronization of optimistic data consistency model. We have also estimated the data size reduction by generating arithmetic formulas in each optimizing technique, measured the amount of data reduction in each case, and confirmed that the formulas are tolerable in determining the data size of large scale mobile data sharing system. Lastly, we recognized that two parameters to define *access domain* showed the flexibility of system configuration and effectiveness in data synchronization.

Acknowledgements

We sincerely appreciate all the MNCRS working group members for their achievements in making a de fact standard for data synchronization. We also would like to give our thanks to Mr. Hiroshi Sakurada for his valuable suggestion for the measurement system design.

References

[1] C.H. Papadimitrio, The Serializability of Concurrent Database Updates, Journal of the ACM, 26(4), October 1979.

[2] Mahadev Satyanarayanan, et al, Coda : A highly Available File System for a Distributed Workstation Environment, IEEE Transaction on Computers, 39(4), April 1990.

[3] Richard G.Guy, et al, Implementation of the Ficus Replicated File System, USENIX Conference Proceedings, USENIX, June 1990.

[4] Luosheng Peng, et al, A Star-structured Data Consistency Model for Wide-area Mobile Network Computing, SICON'98, July 1998.

[5] Luosheng Peng, et al, Using Version Vectors in the Star-structured Data Consistency Model, Principles of Distributed Computing (PODC), June 1998.

[6] Mobile network computer reference specification, http://www.mncrs.org

[7] James Gosling, et al, The Java Language Specification, Adison-Wesley, Menlo Park, California, August 1996

[8] David Ratner, et al, Dynamic Version Vector Maintenance, Technical Report CSD-970022, University of California, Los Angeles, June 1997.

[9] D. Scott Parker, et al, Detection of Mutual Inconsistency in Distributed Systems, IEEE transactions on Software Engineering, 9(3), May 1983.

[10] Peter Reiher, et al, Resolving File Conflicts in Ficus File System, USENIX Conference Proceedings, USENIX, June 1994.

[11] Calton Pu, et al, Regeneration of Replicated Objects: A Technique and Its Eden Implementation, IEEE transactions of Software Engineering, 14(7), July 1988.

[12] Susan B. Davidson, et al, Consistency in Partitioned Networks, Computing Surveys, 17(3), September 1985.

[13] Masahiro Kuroda, et al, Optimistic Data Consistency Model and its Application for Database Management, DICOMO '98, IPSJ, June 1998.

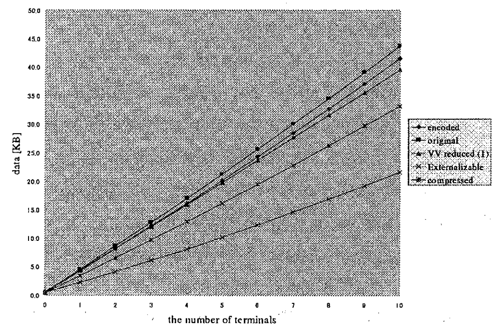


fig.9 amount of data in one synchronization cycle (1 to 10 terminals)

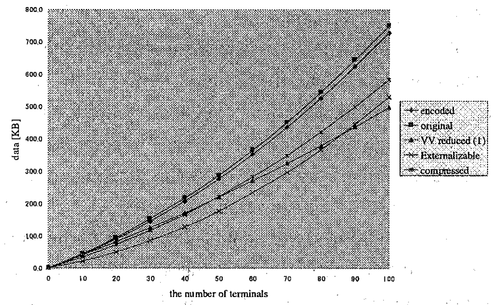


fig.10 amount of data in one synchronization cycle (10 to 100 terminals)

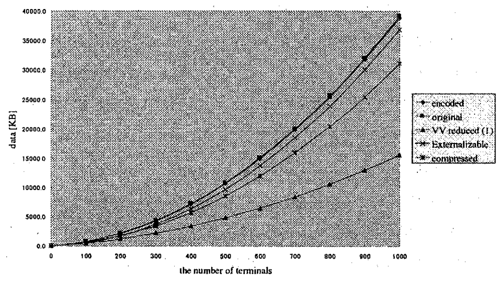


fig.11 amount of data in one synchronization cycle (100 to 1000 terminals)