Journal of Theoretical and Applied Information Technology
10th March 2013. Vol. 49 No.1
© 2005 - 2013 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A LIGHT-WEIGHTED TRANSACTION PROCESSING SYSTEM BASED ON INTERNET MESSAGE ACCESS PROTOCOL

[1]JIAN SU, [2]CHONG ZHOU, [3]WENYONG WENG

[1,3]School of Computer and Computing Science, Zhejiang University City College, Hangzhou, China

[2]Department of Computer Science, Zhejiang University, Hangzhou, China

E-mail: [1]suj@zucc.edu.cn , [2]zhouchonghz@gmail.com , [3]wengwy@zucc.edu.cn

ABSTRACT

A few methods are proposed to build a light-weighted transaction processing system, which is based on the Internet Message Access Protocol (IMAP). The main idea underlying these methods is to use an email service as ubiquitous lightweight data storage and provide means to assure transaction processing in a new application environment. Two kinds of operation schemes based on advisory shared-exclusive locks and optimistic concurrency control are implemented respectively. Performance tests are also offered to help understanding the pros-and-cons of these two schemes in different application conditions. The light-weighted transaction processing system will be used as an on-line storage system for many on-line light-weighted applications which do not require a full-feature database system.

Keywords: *Transaction Processing, Internet Message Access Protocol (IMAP), Concurrency Control, Storage System*

1. INTRODUCTION

Internet is one of the most important information infrastructures for most of people. More and more applications will be deployed on the Internet. Many applications, especially those accessible through Internet, need an on-line database system. Obviously, a few well-known database software, such as MySQL or Oracle, are quite often been used to play the role of on-line database system [1]-[4]. However, many applications, such as mobile application, just need an on-line storage system to store a few data, not necessarily a full-feature database system or a data cloud service[5][6].

In this paper, we will discuss a light-weighted transaction processing system which is based on the widely-used Internet Message Access Protocol (IMAP) [7]. As we know, IMAP is a standard protocol for email management. This transaction processing system use the email box as the underlying storage infrastructure, and provide the means on the basis of IMAP to assure transaction processing [8].

Designing and implementing the transaction processing mechanism on the infrastructure of e-mail service will make the following interesting things possible: a mailbox on an e-mail server can be used as a database server anywhere anytime, requiring only a Internet connection. There is no need to establish a mobile database server or to rent a cloud data service, it will be secure enough with SSL link, and it will not easily collapse due to a professional maintain of e-mail service provider.

We will present a transaction processing library called LibMTP, which implemented the atomicity and isolation semantics of transaction based on IMAP [9]. The LibMTP is a client side library worked in the way of peer to peer negotiation without any central arbitration. Two kinds of concurrency control policy and transaction services are compared and discussed to find out the best implement scheme in different kinds of application situations [10]. Only the most fundamental features of various extensions of the IMAP are utilized in order to gain a wider compatibility.

2. ARCHITECTURE

Since IMAP allows multiple clients to connect to the same mailbox simultaneously, but do not support any request scheduling, LibMTP establishes the transaction semantic by client-side negotiation without any central arbitration. Structure of application with LibMTP can be denoted as Figure 1.

© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

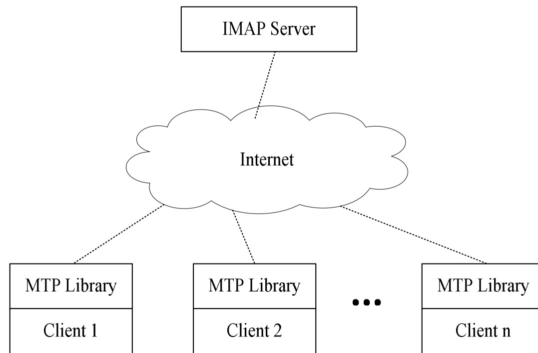


Figure 1. Application architecture

3. ARCHITECTURE

In this section, some IMAP elements and features will be discussed and formalized for an easier description about the upper level protocol and algorithms.

3.1. Message

LibMTP uses a tuple called Message to represent an e-mail, which is defined as

Message::=<Seq, Title, Content, Answered>

In this definition, Seq, Title and Content are the sequence number, subject and plaintext body of an email respectively. Answered is a flag indicating whether or not an email is answered. In fact, LibMTP utilizes the subject, plaintext body, and answered flag of an email to store and represent data, in addition to the message sequence number. Also, LibMTP provide means to establish the lock, serialize latch, and design some other functional mechanisms.

3.2 Message Sequence Number

Message sequence number is used by IMAP servers to uniquely identify e-mails within an IMAP session. When an IMAP session is established, the server will assign a unique sequence number to each e-mail in the order of receiving time. Any new incoming e-mail will obtain a sequence number which equals to the number of messages in the mailbox. Since emails tagged with a deleted flag will not be deleted immediately until the session is reestablished, delete operation will not result in a fluctuation of message sequence number. LibMTP will filter the e-mails with deleted flag with any access behavior, so that it makes delete operation logically taking effect immediately. IMAP message sequence number is used to implement the lock and latch mechanism, as will be discussed later in the following sections.

3.3 Imap Commands

IMAP works in a request-response mode. All the requests are called commands. LibMTP uses four core IMAP commands to build its functionality. These four commands are APPEND, SEARCH, FETCH and STORE. Four procedures are designed to use these four commands respectively, i.e., for every command there is a corresponding procedure to use it. These commands and their corresponding procedures are listed in the Table 1:

Table 1. IMAP command and procedures

Command	Procedure Definition
APPEND	Prototype: ATOM_APPEND(Title, Content)
	Title: the title of the new append message. Content: the content of the new append message. return SUCCEED | FAILED.
SEARCH	Prototype: ATOM_SEARCH(Title_Pattern, Answered)
	Title_Pattern: title pattern to search. Note that any e-mail whose title contains the Title_Pattern will be included in the result set. Answered: whether the Answered Flag of an e-mail is set. Answered belongs to the set of {ANSED,NONANS,IGNORE}. return Ary_Of_MsgSeq.
FETCH	Prototype: ATOM_FETCH(Which_Part, Ary_Of_MsgSeq)
	Which_Part belongs to the set of {TITLE, CONTENT, TIT_N_CNT}. return Array_Of_Title | Array_Of_Content | Array_Of_<Title,Content>
STORE	Prototype: ATOM_STORE(Ary_Of_MsgSeq,Flag)
	Flag belongs to the set of {ANS_FLAG, DEL_FLAG}. return SUCCEED| FAILED.

4. SCHEME 1: LOCK PROTECTED TRANSACTIONS

SCHEME 1 uses advisory shared-exclusive locks to protect the data to be accessed concurrently [11]-[13]. Clients are required to cooperate with each other by acquiring the lock before accessing the corresponding data [14] [15]. Clients use exclusive locks to obtain an exclusive access right over Data Items, while use shared locks to prevent reading of uncommitted data. After critical CRUD operations a transaction should be committed if everything is

© 2005 - 2013 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

all right, or rolled back if there is any exception or error detected.

4.1. Lock Protocol

4.1.1. Protocol sketch

The lock protocol can be divided into two phases. During the first phase, clients acquire locks as needed, and during the second phase, clients release all the acquired locks so as to release critical Chunks [16].

Before executing a read operation, clients need to acquire a shared lock first. Before executing a write, update or delete operation, clients must acquire an exclusive lock. Exclusive locks are privileged than shared locks, i.e., clients can read data safely if they have assigned an exclusive lock to a Chunk, but they shall never write to a Chunk if they only have acquired a shared lock. Exclusive locks and share locks are represented by X_LOCK_MSGs and S_LOCK_MSGs. To apply an exclusive lock over a Chunk, clients append a Message whose Type field is X_LOCK_MSG, and the Name field is equal to the Name field of the target Chunk by issuing an IMAP APPEND command through invoking the ATOM_APPEND function. Similarly, to apply a shared lock over a target Chunk, clients have to append a S_LOCK_MSG Message.

Since multiple clients may try to apply locks over a Chunk simultaneously, SCHEME 1 protocol has to decide who can acquire lock immediately and who have to wait. SCHEME 1 takes a FCFS (First Come First Served) policy and use the Message Sequence Number as a criterion to decide which lock request is issued first. All clients check the mailbox to see whether there is any conflicted locks applied to the same target Chunk before accessing the data, and if there is, the client have to back off some time and do a recheck later, until the client have obtained the lock message with a smallest message sequence number.

Clients will maintain a set of locks that have been acquired locally, and there is no need to acquire a lock repeatedly. Locks can not be upgraded, and clients always have to acquire an exclusive lock again if they have not acquired it, even if they have already obtained a shared lock on the same target.

4.1.2. Lock acquire algorithm

Procedure LOCK_ACQUIRE implements the algorithm used to acquire an exclusive lock or shared lock on the target Chunk. It accepts two parameters. The first parameter is used to indicate which kind of lock it will acquire, and the value can be X_LOCK or S_LOCK. The Second parameter is the name field of the target Chunk. If the procedure can acquire the lock successfully, it will return a SUCCEED result, while a FALSE if there is something wrong or the lock-acquiring request has failed for too many times, which imply that there may be a dead lock[17] .

There are some auxiliary subroutines used by the procedure LOCK_ACQUIRE, some of which is also used by the lock release algorithm. The first pair of the subroutine is LOCK_IS_OBTAINED and LOCK_ADDED. These two procedures are used to maintain and use the set of locks which have already been acquired by the client. LOCK_IS_OBTAINED accepts two parameters: the first parameter is a lock type, and the second one indicates the target chunk. LOCK_IS_OBTAINED checks the lock list to find out whether the lock to be acquired has already been acquired: if it is, LOCK_IS_OBTAINED will return the Message Sequence Number of the lock Message, while if it is not, -1 will be returned. LOCK_ADD_TO_SET is used to add the acquired lock message and the message sequence number into the lock list. XLOCK_IS_OCCUPIED and SLOCK_IS_OCCUPIED are used to check whether the exclusive lock or shared lock append by the client is the lock message targeted to the same Chunk with a smallest sequence number. If the answer is yes, the procedure will return FALSE indicating that the target chunk now is available for critical operation, otherwise TRUE will be returned. SLOCK_IS_OCCUPIED accepts two parameters: the target Chunk name and the message sequence number of the exclusive lock message. XLOCK_IS_OCCUPIED accepts three parameters: the first and second parameter works just as the first two parameters of S_LOCK_IS_APPLIED, and the last parameter is used to send a shared lock message on the same target, and is acquired already. This is used to avoid an exclusive lock which conflict with a shared lock already acquired.

4.1.3 lock release algorithm

After acquiring a lock, clients can enter the critical section and execute the CRUD (Create, Read, Update and Delete) operations. These operations will be handled by TXN module services. Since a sequence of CRUD operations within a transaction will be finished after committing or rolling back of a transaction, all acquired locks will be released. Procedure LOCK_RELEASE provide the function of lock releasing.

© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.2 Transaction Algorithms

SCHEME 1 use Transaction interfaces to bind a sequence of CRUD operations into a customized atomic action [18]. After having acquired the lock, client can now enter the critical section safely. A local buffer is used to cache a previously read chunk and delay the write operations until transaction has been committed. All the dirty chunks and delete messages will be write to the mailbox and validated by a single atomic operation. After that, a few of clean up operations will be carried out, and all the locks will be released.

4.2.1. Buffer management algorithms

As both exclusive locks and shared locks will protect Chunks from being modified, it is safe to maintain a cache of Chunks which have been read at the client side. Meanwhile, the buffer is used to delay the dirty write to the global storage, so as to establish the atomic semantic. Transactions use two arrays of Messages as the buffer pool: ary_of_dirties and ary_of_copies. The ary_of_dirties is used to buffer the dirty chunks and deleted messages that will be written to the mailbox when the transaction is committed, and the ary_of_copies is used to cache the chunks that are read and not modified.

The TXN_BUFFER_INSERT function is used to insert a new buffer item into the client buffer. It accepts two parameters: the first parameter is the message that is going to be buffered, and the second one indicates whether this is a dirty one, which can be value of DIRTY or NONDIRTY.

The TXN_BUFFER_REMOVE function is used to remove an element from the buffer, it accepts two parameters: the first parameter is the type of the message, and the second one is the message name.

The TXN_BUFFER_CONTAINS function is used to check whether the target message is cached already. If it is, then it can be accessed directly from the buffer, or it is indicating that it has to be fetched from the mailbox through an IMAP operation. It will return the message if there is a buffer hit or NOTHING_FOUND if it is missed.

4.2.2. Crud algorithms

The algorithm presented in this section is the CRUD procedures which are used directly by client applications to operate on Data_Items .

Procedure TNX_READ will read the value field of a Data_Item if its key field is given. The auxiliary TXN_CLEANUP is used to remove the low version chunks and duplicated, expired or deleted messages. It is invoked under protections of locks. It accepts only a single parameter that is the name of the Chunk which is going to be cleaned. After cleaning up, only one unique Chunk with the highest version will be retained, or all Chunks will be deleted. Procedure TXN_WRITE is used to write a Data_Item to the data set. It accepts two parameters, i.e., the key and value field of a Data_Item. To reduce the number of exposed interface, we grouped write and update operation into a single procedure that is TXN_WRITE. If the key is a new one, then the TXN_WRITE acts as a create operation; if the key is existed already, TXN_WRITE will replace the old value with new value, just as a write operation does. Procedure TXN_REMOVE is used to delete a Data_Item given its key.

4.2.3. Transaction commit and roll back

As soon as all the CRUD operations are finished, the transaction should be committed if everything is all right and the client decides to take the modifications into effect, otherwise the transaction should be rolled back to give up all efforts done before [19] [20]. Procedure TXN_ROLLBACK implements the function of rolling back transactions.

5. SCHEME 2: OCC TRANSACTIONS

The lock based transaction schema previously described will suffer from a significant slowing down during deadlock resolve course. Thus, we developed an alternative transaction scheme which use OCC (Optimistic Concurrency Control) method to avoid deadlock tangles [21].

An OCC transaction is divided into three phases, namely READ, VALIDTAE and WRITE, which will be carried out one by one. The READ phase is a non-critical phase. This phase can be paralleled with any other transaction's READ phase or any other phases. VALIDATE phase and WRITE phase are critical, and shall never get paralleled. The concurrency controlling policy of the three phases can be denoted as Figure 2.

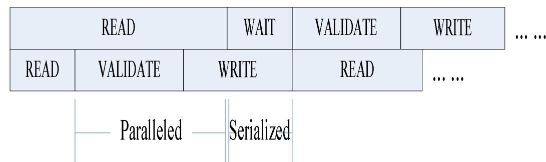


Figure 2. Concurrency controlling policy of the 3-phased transaction

In this OCC Transaction scheme, LibMTP will maintain a local buffer to store the Data_Items that

Journal of Theoretical and Applied Information Technology
10th March 2013. Vol. 49 No.1
© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

have been touched. The fields of the local buffer are described as Table 2.

Table 2. Fields of the local buffer

Field	Function
Key	The key of the Data_Item
Value	The value of the Data_Item
ReadFlag	Indicates whether the Data_Item is read from global storage
Sequence	If this is a Data_Item read from global storage(the ReadFlag is setted), this field will record the Message Sequence Number

During the READ phase, all the CRUD operations may get executed. Read operation will look at the buffer first, if a read from the buffer is missed, the read routine will go on to read the Data_Item from global storage with the ATOM_FETCH() procedure, new read Data_Items will be added into the local buffer, and the corresponding ReadFlag and Sequence field will be setted. All of the write, update and delete operations are performed to the local buffer only (omitting ReadFlag and Sequence out). The read phase of OCC transactions can be described as Figure 3.

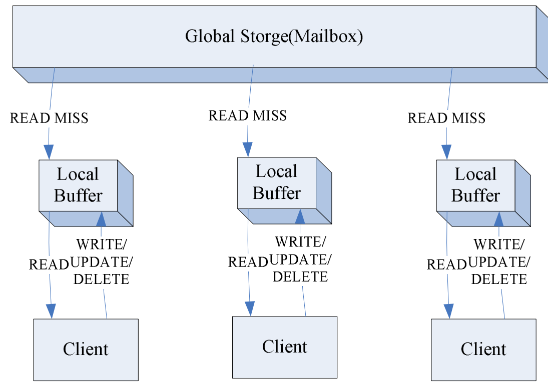


Figure 3. The read phase of OCC transactions

Before the VALIDATE phase begins, LibMTP will lock the whole data set by invoking the LOCK_ACQUIRE() procedure. After the whole data set is locked, we are going to check out whether any of the read data is polluted by other transactions by iterating through the local buffer to see whether any of the Data_Item with a ReadFlag has a new version which has a bigger Message Sequence Number. If the read set is polluted, the transaction can not carry on committing, but having to be rolled back. Otherwise, if none of the read data get modified, the transaction can be committed safely.

As is validated successfully, transaction now is going to write data into the global storage during WRITE phase. All the polluted Data_Items will be updated as well as delete operation will be carried out.

6. PERFORMANCE EVALUATION

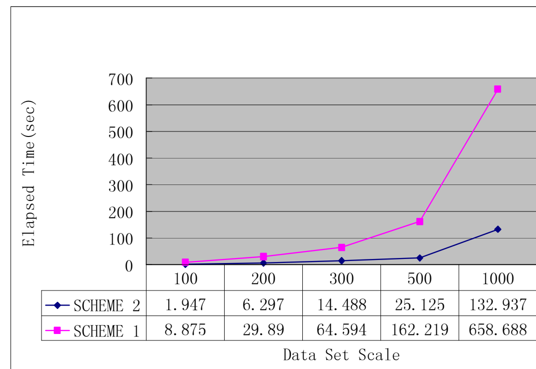
In this section, we are going to evaluate the performance of the two schemes on write oriented transactions and read-write balanced operations [22]. All performance results are obtained on a Dell desktop PC.

6.1 Write Orinted Transactions

The two schemes will be tested with a bulk write transaction. We let the two scheme clients write data items into data set, but do no read operations. First we carried out the benchmark, which running one single thread. The test result is show as Figure 5.

Then we tested them in a multiple user mode. Five clients are used for testing. The test result is show as Figure 6.

As displayed, SCHEME 1 allows more concurrency, but costs more lock overhead. To reveal more details about the relationship between the client number and performance, we tested the schemes with different number of clients to do the same number of write operation, and the result is collected in Figure 7.



	100	200	300	500	1000
SCHEME 2	1.947	6.297	14.488	25.125	132.937
SCHEME 1	8.875	29.89	64.594	162.219	658.688

Figure 5: Read-only Trans with Single Client

© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

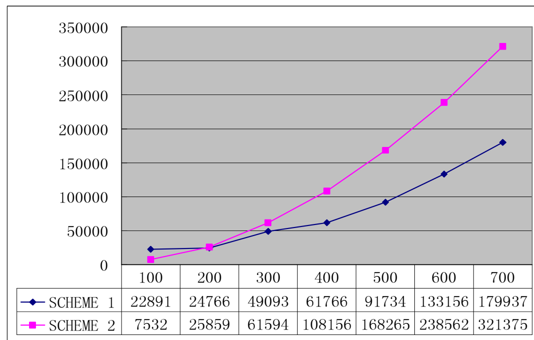


Figure 6: 5 Read-only Trans with 5 Clients

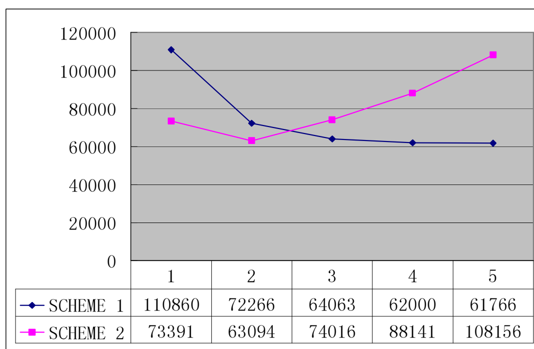


Figure 7: Read-only Trans with Different Client Numbers

6.2 Debit-And-Credit Transactions

In this section, we tested the two schemes with a read-write balanced transaction in a debit-and-credit style [23][24]. We first tested the schemes with a single client mode. Result is shown in Figure 8. Then we tested the schemes in a multiple clients mode, five clients are chosen to do the debit-and-credit transaction simultaneously. The result is displayed in Figure 9.

To reveal the relationship between the operation data set and performance we carried out test described as Figure 10, in which we did the debit-and-credit transactions on different scale of account number by five clients.

Finally, to analysis the performance of the two schemes with different number of clients, we tested the schemes on the same conditions but different number of clients, the results is described in Figure 11.

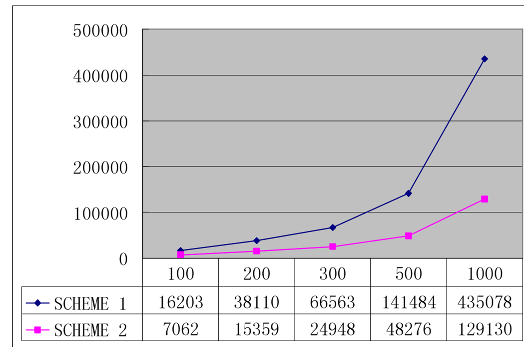


Figure 8: Debit & Credit Trans with Single Client

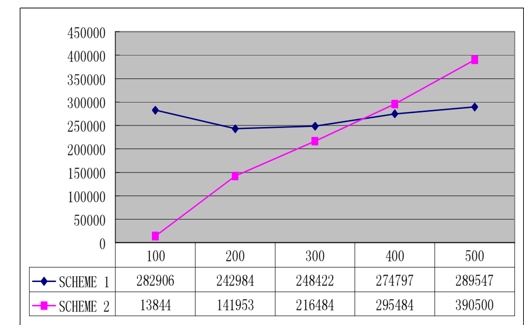


Figure 9: Debit & Credit Trans with 5 Clients

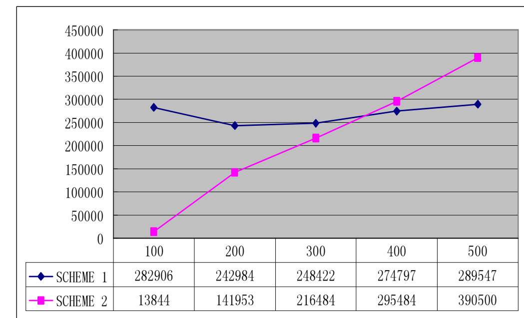


Figure 10: Debit & Credit Trans with 5 Clients on Different Scale Data Sets

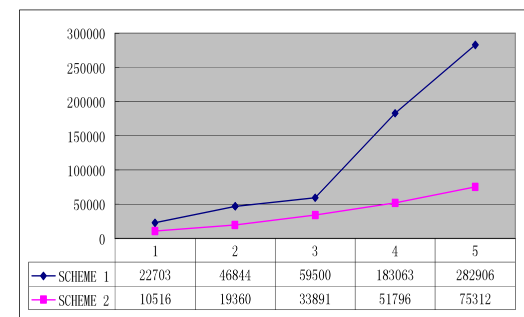


Figure 11: Debit & Credit Trans on Different Client Numbers

Journal of Theoretical and Applied Information Technology
10th March 2013. Vol. 49 No.1
© 2005 - 2013 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7. CONCLUSION

This paper proposes an approach to build a light-weighted transaction processing system based on the widely-used IMAP infrastructures. It is an interesting attempt to make use of the historical, wide spread, charge-free E-Mail service for ubiquitous data storage in a new internet application environment. Two schemes of transaction were implemented. One of them uses advisory shared-exclusive locks to establish the atomicity and isolation semantics based on peer-to-peer negotiation. The other scheme takes an optimistic concurrency control policy to provide a deadlock-free implement, but sacrifice the throughput possibility. Neither of the two schemes relies on a central arbitration mechanism. In stead, both of these schemes completely depend on peer-to-peer negotiation.

For on-line light-weighted applications which do not require a full-feature database system, the light-weighted transaction processing system can be used as an on-line storage system

ACKNOWLEDGEMENT

The work of this paper is supported by the Science and Technology Program of Zhejiang Province (No.2012C33078).

REFRENCES:

[1] Xiaoying Wang, Xuhan Jia, Lihua Fan, Weitong Huang, "Research on performance modeling of transactional cloud applications", Journal of Theoretical and Applied Information Technology, Vol.44, No.2,2012, pp.166-171.

[2] Zheng Hua, "A development model for domain-oriented information service based on cloud computing infrastructure", Journal of Theoretical and Applied Information Technology, Vol. 46, No. 2, 2012, pp. 594-598.

[3] H. Yadava, The Berkeley DB Book, Apress, Berkely, CA, 2007.

[4] W. Litwin, “Linear hashing: A new tool for file and table addressing”, Proceedings of the 6th International Conference on Very Large Data Bases, Vol.6, 1980, pp.212-223.

[5] J.M. Hellerstein, M. Stonebraker, "Anatomy of a Database System", Readings in Database Systems 4th edition, The MIT Press, 2005, pp.42-93.

[6] J.M. Hellerstein, M. Stonebraker, "What Goes Around Comes Around", Readings in Database Systems 4th edition, The MIT Press, 2005, pp. 2-41.

[7] M. Crispin, "Internet Message Access Protocol − Version 4rev1", Internet Engineering Task Force RFC3501, 2003.

[8] D. Comer, "The Ubiquitous B-Tree", Computing Surveys, Vol.2, No.2, 1979, pp.121-137.

[9] M. M. Astrahan, W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griggiths, "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, Vol.1, No.2, 1976, pp. 97-137.

[10] R. Sears, E. Brewer, "Stasis: flexible transactional storage", Proceeding of OSDI, 2006, pp.29-44.

[11] J. N. Gray, R. A. Lorie, G. R. Putzulo, I.L.Traiger, "Granularity of locks and degrees of consistency in a shared database", IBM Research Report, RJ1654, 1975.

[12] A. Adya, R. Gruber, B. Liskov, U. Maheshwari, "Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks", Proceeding of SIGMOD, 1995, pp.23-24.

[13] C. Mohan, "ARIESIKVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes", Proceedings of the 16th VLDB Conference, Brisbane, August 1990, pp.392-405.

[14] R. Bayer, M. Schkolnick, "Concurrency of Operations on B-Tree", Acta informatica, 1977.

[15] P. L. Lehman, S. B. YAO, "Efficient Locking for Concurrent Operations on B-Trees", ACM Transactions on Database Systems, Vol.6, No.4, 1981, pp.650-670.

[16] E. Sciore, "SimpleDB: a simple java-based multiuser system for teaching database internals", Proceeding of SIGCSE, 2007, pp.561-565.

[17] C. Mohan, Don Haderle, B. Lindsay, H. Pirahesh, P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging", ACM Transactions on Database Systems, Vol 17, No. 1, 1992, pp.94-162.

[18] M. Kornacker, C. Mohan, J.M. Hellerstein, "Concurrency and recovery in generalized search trees", Proceeding of SIGMOD, 1997, pp.62-72.

© 2005 - 2013 JATIT & LLS. All rights reserved·

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

[19] M. Stonebraker, "Retrospection on a Database System", ACM Transactions on Database Systems, Vol.5, No.2, 1980, pp.225-240.
[20] M. Stonebraker, E. Wong, P. Kreps, G. Held, "The Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1, No.3, 1976, pp.189-222.
[21] H.T. Kung, "On Optimistic Methods for Concurrency Control", ACM Transactions on Database Systems, Vol.6, No.2, 1981, pp.213-226.
[22] J.N. Gray, Database and Transaction Processing Performance Handbook, Morgan Kaufmann Pub, 1993.
[23] D. D. Chamberlin, "A history and evaluation of System R", Communications of the ACM, Vol.24, No.10, 1981, pp.632-646.
[24] C. Mohan, "ARIES/IM: An Efficient and High Concurrency index Management Method Using Write-Ahead Logging", Proceeding of SIGMOD, 1992, pp.371-380.