An Analytic Model for Performance Analysis of Concurrency Control Strategies in Mobile Environments

GUAN-CHI CHEN AND SUH-YIN LEE

Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu, Taiwan 300, ROC
Email: gcchen@csie.nctu.edu.tw

Many performance studies have shown that locking based protocols outperform other concurrency control protocols in most database configurations and workloads. However, in mobile environments there are many special characteristics, such as long network delay and the expensive wireless communication. Besides, transactions may be aborted due to forced termination when handoff occurs for the consideration of concurrency control. The performances of concurrency control strategies should be re-evaluated in such environments. In this paper, we develop an analytic model to evaluate the performances of concurrency control strategies. The accuracy of this model is verified by simulation. According to the experimental results we show that the behavior of mobile users and the degree of data contention have a significant impact on the relative performances of the concurrency control strategies. In particular, we point out that the optimistic concurrency control strategy outperforms other strategies over a wide range of system configurations. We also explain why the optimistic algorithm is well suited in mobile environments.

Received August 1, 1998; revised April 21, 1999

1. INTRODUCTION

Recent technological advances in portable computers (notebooks) and wireless communication have made the mobile computing environment a reality. An example of the future mobile environment is the so-called personal communication service (PCS) network. The PCS network is based on current cellular network architecture. In cellular network architecture, a mobile unit, which is a notebook or laptop computer with a wireless network card, communicates with servers on the wire network through a base station. The communication area covered by a base station is called a cell. When a mobile unit is within the cell of a base station, the base station will provide a communication channel to the mobile unit if there are some idle channels available in the cell. A general mobile information system model, as shown in Figure 1, has been proposed [1, 2, 3].

This model consists of stationary and mobile units. Stationary units are classified as either fixed hosts or base stations. Fixed hosts are information servers with associated databases connected to the existing wire network, and are not capable of connecting to a mobile unit. A base station is equipped with a wireless interface and is capable of connecting to a mobile unit. In other words, the base station plays the role of coordinator and communication interface between the mobile units and the stationary units.

The mobile user accesses the database by submitting transactions, called mobile transactions. A transaction

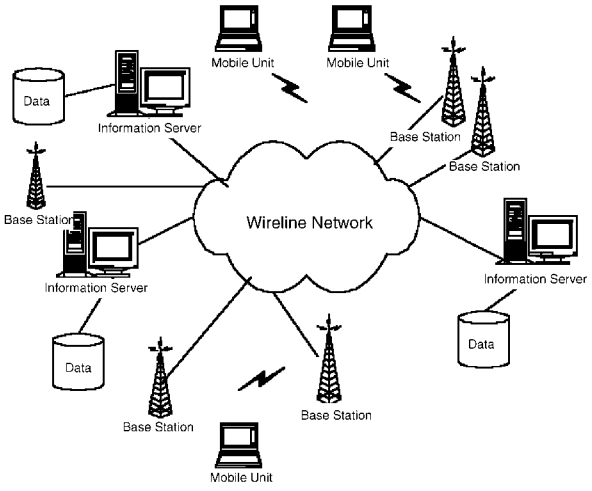

FIGURE 1. Mobile information system model.

submitted from a mobile unit is sent to a base station through a wireless link and then sent to the information server via the existing fixed network.

During the executing time, a transaction may need the user's participation to input data. Many studies have pointed out that the cost of a call set-up is very expensive [4]. In order to avoid re-establishing the communication each time the transaction needs the user's interaction, we assume that the communication link must be kept while the transaction is executing.

When a mobile unit is within the cell of a base station, the base station will provide a communication channel to the mobile unit if there are some idle channels available in this cell. Due to the movement of a mobile user, when a mobile user enters a new cell, the new base station should provide an idle channel to the mobile unit to continue its communication. This process is called handoff. If there is no idle channel in the new cell to provide for the user, then the connection will be forced to drop or terminate. The forced termination of an active transaction not only interferes with the users, but also wastes the system resources, since the database should be rolled back and the transaction should be restarted later. As we know, handoff will occur when a mobile unit enters a new cell. Due to the long network delay of the wireless link, the transaction will become a long-lived transaction in mobile environments. In addition, with the growth of the number of mobile users and the aim to provide a good quality service, it is a trend to set up more base stations. As a consequence, the handoff frequency will increase in the future. Therefore, the probability of forced termination of transactions will also increase.

Several problems, such as lost update, temporary update and incorrect summary, can occur when concurrent transactions execute in an uncontrolled manner. In database management systems (DBMSs), many concurrency control algorithms have been proposed to ensure the correct execution of concurrent transactions in controlled manner [5]. Earlier studies about concurrency control algorithms for conventional DBMSs have concluded that locking based protocols outperform others under most operating circumstances [6]. In light of the significant difference between the mobile database systems and the conventional database systems outlined above, it is possible that the previous result may not hold true in mobile environments. This possibility motivated our investigation of the performances of concurrency control algorithms in mobile environments.

The rest of this paper is organized as follows. We introduce the concurrency control algorithms in the next section. The analytic and simulation models are proposed in Section 3 and Section 4, respectively. In Section 5, we present and analyze the experimental results. Finally, we make a conclusion and point out our future work in Section 6.

2. CONCURRENCY CONTROL ALGORITHMS

The concurrency control software is included in a DBMS to ensure the correct execution of concurrent transactions. Concurrency control algorithms proposed can be roughly classified into locking based protocols, time-stamp ordering protocols and optimistic concurrency control protocols [5]. In our study, we will compare the performance for a specific protocol in each class. These particular instances were chosen because they are of comparable complexity and are general in their applicability. We briefly introduce these algorithms in this section.

2.1. Two-phase locking protocol

The two-phase locking (2PL) protocol is the most popular concurrency control protocol and is widely used in most of today's database systems. In the 2PL protocol, two types of lock are offered—share lock (*Slock*) and exclusive lock (*Xlock*). A transaction is required to set an *Slock* on a data item before reading it and to set an *Xlock* before writing on it. Multiple transactions can set *Slocks* on the same data item simultaneously and, therefore, can read the data item concurrently. On the other hand, once an *Xlock* on a data item is set by a transaction, no other transactions are allowed to set any types of lock on the data item. If a lock cannot be acquired, a transaction has to wait until the lock-holding transaction releases the related lock. For correctness, once a transaction releases a lock, it cannot acquire any other lock. Furthermore, almost all of the implementations choose to release locks held by a transaction only after the corresponding transaction aborts or commits. In addition to the lock request and release rules, the 2PL protocol needs a deadlock detection and resolution algorithm to solve the deadlock caused by the circular wait of locks among transactions. The most commonly used deadlock detection algorithms are periodical checking and intermediate checking.

2.2. Time-stamp ordering protocol

A time-stamp is a unique identifier created by the DBMS to identify a transaction [5]. Typically, time-stamp values are assigned in the order in which the transactions are submitted to the system, and a time-stamp can be thought of as the transaction start time. Time-stamp ordering (TO) protocol uses time-stamps to order the transactions. A schedule in which the transactions participate is then serializable, and the equivalent serial schedule has the transactions in the order of their time-stamp values. To guarantee that a schedule does not violate serializability, we associate with each data item X two time-stamp values:

read_TS(X)	The read time-stamp of item X is the largest time-stamp among all the time-stamp of transactions that have successfully read item X.
write_TS(X)	The write time-stamp of item X is the largest time-stamp of all the transactions that have successfully written item X.

Whenever a transaction T tries to issue a read or write operation on item X, the following rules are checked:

(1) *Transaction T issues a read operation on item X:*
if time-stamp of T < write_TS(X)
then reject this read operation
else accept this read operation

(2) *Transaction T issues a write operation on item X:*
if time-stamp of T < read_TS(X) or write_TS(X)
then reject this write operation
else accept this write operation

Like the 2PL protocol, the TO protocol guarantees serializability of schedules. Since the TO protocol does not use locks, deadlock cannot occur in this protocol. However, when a transaction T aborts, any effect it has incurred must be undone and the transactions that have used the results written by T must be also rolled back. This effect is known as cascading rollback or cascading abort, and is one of the problems associated with the TO protocol.

2.3. Optimistic concurrency control protocol

The optimistic concurrency control (OCC) protocol is based on the premise that the conflict is rare and just lets transactions execute first [5]. When a transaction reaches the commit point, the system will check for conflicts. If there is a conflict, the transaction will be terminated and restarted again. So no checking is done during the transaction executing time. A transaction executes in three phases.

- *Read Phase*: this phase represents the body of the transaction up to commit. A transaction can read values of data items from the database. However, updates are applied only to local copies of the data items in the transaction working place.
- *Validation Phase*: the system will examine if any conflict occurs to ensure that serializability will not be violated.
- *Write Phase*: if the transaction is validated, then the transaction updates are applied to the database. If conflicts which would result in a loss of integrity are detected during the validation phase, then the transaction is rolled back and restarted.

The set of data items read by a transaction is called the transaction *readset*, while the set of data items written by a transaction is called the transaction *writeset*. During the *validation phase*, the system will check if any data item of the *readset* of the transaction could be found in the *writesets* of other transactions. If found, the transaction will be rolled back and restarted.

2.4. Comparisons of concurrency control methods

All of these concurrency control methods carry some overheads in time and space. The TO protocol is similar to 2PL in the sense that both use pessimistic approaches in which the system checks the conflicts between transactions as each data item is accessed. The TO protocol decides the serialization order statically—depending on the time-stamp of the transaction, which is the submit time of the transaction. On the other hand, the 2PL protocol decides the serialization dynamically—according to the order in which the data items are accessed. When a conflict occurs, the TO protocol aborts the transaction immediately. However, the 2PL protocol makes the transaction wait, but with a large possibility of aborting later to avoid deadlock.

When the OCC protocol is used, all transactions are allowed to process, but some may be aborted when they attempt to commit. Because there is no need to wait for other transactions, the transaction execution time when the OCC protocol is used is shorter than when the 2PL protocol is used. We will see the performances of these three methods in our analytic model.

3. ANALYTIC MODELS

This section proposes analytic models for the 2PL, TO and OCC protocols in mobile environments. We first describe the traffic model in a PCS system, and then describe the performance models for 2PL, TO and OCC protocols respectively.

3.1. The traffic model for PCS

There are two types of calls—voice call (VC) for voice service and transaction call (TC) for data transaction. We assume that VC and TC arrivals in a cell form Poisson processes with mean λ_v and λ_t respectively. Let λ_{hv} and λ_{ht} be the arrival rate of handoff VC and TC, respectively. At this moment, we assume λ_{hv} and λ_{ht} are known (both arrival rates will be derived in later discussion). The residence time of a mobile unit in a cell is assumed to be an independent, identical, random variable with an exponential distribution, and the mean value equals $1/\eta$. Let t_c be the call holding time of a voice call, which is assumed to be exponentially distributed with the density function $f_c(t_c) = \mu\, e^{-\mu t_c}$, where the mean call holding time is $E[t_c] = 1/\mu$. We first assume that the mean call holding time of a transaction call is S (S will be derived in later discussion). Thus, the mean call holding time of all calls would be

$$\frac{1}{\mu'} = \frac{1}{\mu}\frac{\lambda_v}{\lambda_v + \lambda_t} + S\frac{\lambda_t}{\lambda_v + \lambda_t}.$$

Since λ_t is comparatively smaller than λ_v, we assume that the call holding times of all calls remain exponentially distributed.

In Figure 2, we show the state-transition-rate diagram of the system. Assume that there are C channels in a cell. Let $P(i)$ be the steady-state probability of total i calls in the system. Using the state-transition-rate diagram, we easily obtain the probability $P(i)$ as

$$P(i) = \left(\frac{\lambda_T}{\eta + \mu'}\right)^i \frac{P(0)}{i!}, \quad 0 < i \le C \tag{1}$$

and

$$P(0) = \left(\sum_{i=0}^{C} \frac{[\lambda_T/(\eta + \mu')]^i}{i!}\right)^{-1} \tag{2}$$

where $\lambda_T = \lambda_v + \lambda_t + \lambda_{hv} + \lambda_{ht}$.

Since no special handoff scheme is used, the handoff calls and the new calls are not distinguishable. A call is blocked when there are no idle channels. Thus the blocking probability P_b is given by

$$P_b = P(C). \tag{3}$$

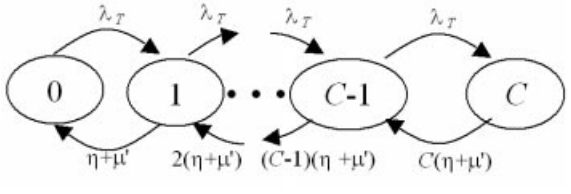

FIGURE 2. The state-transition-rate diagram.

Since the residence time of a mobile unit in a cell is exponentially distributed, according to [7], the arrival rates of handoff VC and TC will be

$$\lambda_{hv} = \frac{\eta(1-p_b)\lambda_v}{\mu+\eta p_b} \quad \text{and} \quad \lambda_{ht} = \frac{\eta(1-p_b)\lambda_t}{(1/S)+\eta p_b}. \qquad (4)$$

By the equations derived in the above description, now we can use an iterative algorithm to compute p_b as follows:

Algorithm Compute_blocking_probability
Input : λ_v, λ_t, μ, η, C, S
Output : p_b
Step 1: Select initial values for λ_{hv} *and* λ_{ht}.
Step 2: Compute p_b *by using (3)*

$$p_b = P(C).$$

Step 3: $\lambda_{hv,old} \leftarrow \lambda_{hv}$ *and* $\lambda_{ht,old} \leftarrow \lambda_{ht}$.
Step 4: Compute λ_{hv} *and* λ_{ht} *by using (4)*

$$\lambda_{hv} = \frac{\eta(1-p_b)\lambda_v}{\mu+\eta p_b} \quad \textit{and} \quad \lambda_{ht} = \frac{\eta(1-p_b)\lambda_t}{(1/S)+\eta p_b}.$$

Step 5: If $(|\lambda_{hv} - \lambda_{hv,old}| > \delta\lambda_{hv})$ *and*
$(|\lambda_{ht} - \lambda_{ht,old}| > \delta\lambda_{ht})$
then go to Step 2. Otherwise go to Step 6.
Here δ *is a pre-defined value.*
Step 6: Output p_b.

By the above algorithm, we can compute the value of p_b under the initial value of S. Whether the above iterative approach will eventually converge is an open issue, and will be our future research direction.

Now the mean residence time of a transaction in the database system can be derived as follows. Let h denote the number of handoffs occurring for a TC before it terminates. Since we assume the mean call holding time of a transaction call equals S, h will be $S\eta$. The mean residence time of a transaction in the system, S', will be

$$S' = \left(\sum_{i=1}^{h-1} \frac{(1-p_b)^{i-1}}{\eta}\right) + \left(S - \frac{(h-1)}{\eta}\right)(1-p_b)^{h-1}. \qquad (5)$$

The second term of Equation (5) is the dwelling duration of the transaction call in the last cell. The value computed by Equation (5) is also the mean service time of a transaction in the database system. It will be compared with the values computed from the analytic models of concurrency control protocols, which we will discuss in the following sections.

In the following description, we make the following assumptions. (1) There are N data items in the database. (2) The update transactions are about P_u percentage of total transactions. Write operations in update transactions are about P_w percentage of the total operations performed by update transactions. In other words, the probability that an operation is a write operation is $P_u P_w$. (3) Each transaction will access K data items before it commits. (4) The execution time for each operation of a transaction equals t (including CPU time, disk I/O delay and network delay).

3.2. Transaction residence time in two-phase locking protocol

In this section, we will analyze the mean residence time of a transaction when the 2PL protocol is employed in mobile environments. It has been shown that the performance of locking protocols for transactions that read and write a database of size N is the same as that for transactions that write a database of size $N/[1-(1-P_uP_w)^2]$ [8]. To simplify the analysis of 2PL, we can assume the K operations of a transaction are all write operations by changing the size of database from N to $N/[1-(1-P_uP_w)^2]$, denoted as N' [8, 9]. However, it is easy to extend the model to the case of shared as well as exclusive mode accesses to the database [8]. Since the arrival rate of TC is λ_t and the mean residence time of a transaction in the database system equals S', the average number of transactions in the database system would be $M = B\lambda_t(1-p_b)S'$, where B is the number of cells in the system. In order to compute the probability of lock conflict, we have to know how many locks a transaction will request before it terminates. In many previous analytic studies of the performance of 2PL, the effect of deadlock is ignored. However, in mobile environments, the network delay is longer so that the residence time of a transaction will be longer. Thus the average number of transactions in the database system will be more and, as a consequence, the deadlock probability will be higher than that in the conventional database systems. Therefore, here we cannot ignore the effect of deadlock. We first assume the probability of deadlock occurring when a transaction requests a lock, p_d, is known (p_d will be derived in later discussion). Then the mean number of locks a transaction will request before it commits or aborts without the consideration of forced termination will be

$$K_{exp} = \sum_{i=1}^{K} (1-p_d)^{i-1}. \qquad (6)$$

When we combine the effect of forced termination to Equation (6), we can get the mean number of locks a transaction will request before it terminates (commit, abort

or been forced to terminate) as follows:

$$K' = \left(\sum_{i=1}^{h-1} \frac{K_{\text{exp}}}{S\eta}(1-p_{\text{b}})^{i-1}\right) + \frac{K_{\text{exp}}}{S}\left(S - \frac{(h-1)}{\eta}\right)(1-p_{\text{b}})^{h-1}. \quad (7)$$

In Equation (7), K_{exp}/S is the number of locks a transaction will request in a time unit. As in Equation (5), the second term in Equation (7) is the mean number of locks a transaction may request in the last cell and $S-(1/\eta)(h-1)$ is the mean residence time of a transaction in the last cell.

In the analysis of lock conflict probability, it is assumed that each transaction, on average, has acquired half of its locks at steady state [8, 9]. When a transaction requests the ith lock, the probability of lock conflict, $P_{\text{c}}(i)$, can be expressed as a function of M, K' and N' as follows:

$$P_{\text{c}}(i) = \frac{(M-1)(K'/2)}{N'-i+1}, \quad i = 1, \ldots, K'. \quad (8)$$

In the following we use the approximation $P_{\text{c}}(i) \simeq p_{\text{c}} = (M-1)K'/(2N')$, $1 \le i \le K'$, since $K' \ll N'$. In other words, the probability that a transaction encounters a lock conflict remains the same regardless of the number of locks that it holds [8, 9].

Now the deadlock probability can be derived as follows. Let $P_{\text{d}}(m)$ be the probability of deadlock with cycle length m (involving m transactions). For a deadlock with cycle length 2, $T1 \rightarrow T2 \rightarrow T1$ ($\rightarrow$ represents that the former transaction requests a lock held by the latter transaction), $P_{\text{d}}(2)$ is computed by the product of $Prob(T1 \rightarrow T2)$ and the number of ways to choose $T2$ among blocked transactions. The number of blocked transactions, M_{b}, equals $M\beta$, where β is the probability that a transaction is in blocked state. The value of β is not known *a priori*, but can be approximated in the case of low lock contention levels. Assume that most of the lock conflicts are with active transactions, such that $\beta = K' p_{\text{c}} W_1/S'$. W_1 denotes the delay encountered by a transaction blocked by an active transaction and S' is the mean transaction residence time in the database system. The above equation implies that β is expressed as the ratio of the mean transaction delay in the blocked state and the mean transaction residence time. From [9], $W_1 = ((k'-1)t/3)+(t/2)$, where t is the mean process time of a transaction operation. When K is sufficiently large we have $W_1/S' \simeq \frac{1}{3}$. Substituting p_{c} by Equation (8), we have

$$\beta = \frac{K' p_{\text{c}} W_1}{S'} = \frac{(M-1)K'^2}{6N}. \quad (9)$$

Now $P_{\text{d}}(2)$ can be expressed as follows:

$$\begin{aligned} P_{\text{d}}(2) &= Prob(\textit{lock conflict occurs}) \\ &\quad * Prob(T1 \rightarrow T2 \mid \textit{lock conflict occurred}) \\ &\quad * Prob(T2 \rightarrow T1 \mid T2 \textit{ is currently blocked}) \\ &\quad * Prob(T2 \textit{ is currently blocked}) \\ &\quad * (\textit{number of candidates for } T2) \\ &= p_{\text{c}} \frac{1}{M-1} \frac{1}{M-1} \beta M\text{–}1 \\ &= \frac{p_{\text{c}}\beta}{M-1}. \end{aligned} \quad (10)$$

The probability of deadlock with cycle-length m is generalized as

$$\begin{aligned} P_{\text{d}}(m) &= \frac{p_{\text{c}}}{M-1} \frac{1}{\prod_{i=1}^{m-1} M-i} \beta^{m-1} \prod_{i=1}^{m-1} (M-i) \\ &= \frac{p_{\text{c}}\beta^{m-1}}{M-1}. \end{aligned} \quad (11)$$

Therefore, the probability that a transaction will encounter a deadlock of any cycle-length is

$$p_{\text{d}} = \sum_{m=1}^{M-1} \frac{p_{\text{c}}\beta^{m-1}}{M-1}. \quad (12)$$

To compute the mean residence time of a transaction in the database system, we should know the mean waiting time of a transaction when a lock conflict occurs. Assume that $T2$ is blocked by $T1$. $T1$ may be active or blocked. In the former case, transaction $T2$ waits until $T1$ completes. In the latter case, we have a chain of transactions as in $T2 \rightarrow T1 \rightarrow T0$. The mean waiting time of $T1$ from the time it had a lock conflict with $T0$ was W_1, but since $T2$ had a lock conflict with $T1$ at a random time after the first conflict, the additional delay is $W_1' \simeq W_1/2$. Since the probability of a transaction in blocked state is β, the probability that a transaction is blocked by an active transaction is $(1-\beta)$. It follows that the overall mean waiting time of $T2$ equals $(1-\beta)W_1 + \beta(W_1 + W_1') = W_1 + 0.5\beta W_1$.

The probability that a transaction encountering a lock conflict blocked at level i is approximated by $P_i = \beta^{i-1}$, $i \ge 1$ (active transactions at the top of the chain correspond to level zero). Also, the delay incurred by transactions blocked at level i, $i > 1$, is approximated by $W_i = (i-0.5)W_1$. The mean waiting time, W, of a transaction when a lock conflict occurs will be

$$\begin{aligned} W &= \left(1 - \sum_{i=1}^{M-1} \beta^i\right) W_1 + \sum_{i=2}^{M-1} (i-0.5)\beta^{i-1} W_1 \\ &\simeq \left[1 + \sum_{i=1}^{M-1} (i-0.5)\beta^i\right] W_1. \end{aligned} \quad (13)$$

Finally, we can compute the mean residence time of a transaction in the database system as follows:

$$S' = K' t + p_{\text{c}} K' W. \quad (14)$$

Now we have the iterative algorithm to compute p_{b}, p_{d} and S as follows:

Algorithm performance_2PL
Input: λ_v, λ_t, μ, η, C, S, N, P_u, P_w.
Output: p_b, p_d *and* S'.
Step 1: select initial values for λ_{hv}, λ_{ht} *and* S.
Step 2: compute N' *by* $N' = N/[1-(1-P_uP_w)^2]$.
Step 3: compute p_b *by Algorithm compute_blocking_probability.*
Step 4: compute S' *by*

$$S' = \left(\sum_{i=1}^{h-1}\frac{(1-p_b)^{i-1}}{\eta}\right) + \left(S - \frac{(h-1)}{\eta}\right)(1-p_b)^{h-1}.$$

Step 5: $S'_{old} \leftarrow S'$.
Step 6: compute M *by* $M = \lambda_t(1-p_b)S'$.
Step 7: select an initial value for p_d.
Step 8: $p_{d,old} \leftarrow p_d$.
Step 9: compute K_{exp} *by using Equation (6)*

$$K_{exp} = \sum_{i=1}^{K}(1-p_d)^{i-1}.$$

Step 10: compute K' *by using Equation (7)*

$$K' = \left[\sum_{i=1}^{h-1}\frac{K_{exp}}{S\eta}(1-p_b)^{i-1}\right] + \frac{K_{exp}}{S}\left(S - \frac{(h-1)}{\eta}\right)(1-p_b)^{h-1}$$

Step 11: compute p_c *by* $p_c = (M-1)K'/(2N')$.
Step 12: compute p_d *by using Equation (12)*

$$p_d = \sum_{m=1}^{M-1}\frac{p_c}{M-1}\beta^{m-1}.$$

Step 13: If $(|p_d - p_{d,old}| > \delta p_d)$ *then go to Step 8. Otherwise go to Step 14. Here* δ *is a pre-defined value.*
Step 14: compute W *and* S' *using*

$$W = \left[1 + \sum_{i=1}^{M-1}(i-0.5)\beta^i\right]W_1$$

and $S' = K't + p_cK'W$.
Step 15: if $(|S' - S'_{old}| > \delta S')$ *then go to Step 3. Otherwise go to Step 16.*
Step 16: output p_b, p_d *and* S'.

In addition to the probability of forced termination of transactions, the system throughput is also an important metric of performance evaluation. A transaction can commit only if no deadlock or forced termination occurs before it completes. Assume there are B cells in the system, then the system throughput (transactions) will be

$$T = \lambda_t(1-p_b)B(1-p_d)^{K-1}\sum_{i=1}^{h}(1-p_b)^{i-1} \quad h = S\eta. \tag{15}$$

3.3. Analytic model for time-stamp ordering protocol

In order to compute the probability that a conflict occurs when a transaction $T1$ accesses a data item, we have to know how many data items a transaction will access before it terminates. Let $p_{i,r}$ and $p_{i,w}$ denote the probability that a conflict occurs when $T1$ attempts to access its ith data item and the operation is read and write, respectively. Assume there are N_i data items which have been accessed by transactions whose time-stamp is larger than $T1$ when $T1$ makes its ith data request. Out of N_i data items, on the average $N_iP_uP_w$ will have their last access as write. Since a write operation will conflict with the read and write operations on the same data item while a read operation only conflicts with write operations, $p_{i,r}$ and $p_{i,w}$ can be computed as follows:

$$p_{i,r} = \frac{N_iP_uP_w}{N-i} \tag{16}$$

$$p_{i,w} = \frac{N_i}{N-i}. \tag{17}$$

Computation of N_i is difficult because the model does not keep track of other transactions. N_i has been computed for the TO protocol in [10] and is given by the following expression:

$$N_i = \frac{i}{2}\left(\frac{i}{t}\lambda_{txn} - 1\right). \tag{18}$$

In Equation (18), λ_{txn} denotes the arrival rate of transactions to the database and equals $\lambda_tB(1-p_b)$. Therefore, when a transaction makes its ith access, the probability that it conflicts is

$$p_i = (1-P_uP_w)\frac{N_iP_uP_w}{N-i} + \frac{P_uP_wN_i}{N-i} = \frac{(2P_uP_w - P_u^2P_w^2)i}{2(N-i)}\left(\frac{i}{t}\mu_{txn} - 1\right). \tag{19}$$

An important observation is that $p_i \propto i^2$. That is, the abort probability of a transaction increases quadratically with its age. Next, we compute the performance measures of interests.

The mean number of data items a transaction will access before it terminates without the consideration of forced termination will be

$$K_{exp} = \sum_{i=1}^{K}(1-p_i)^i. \tag{20}$$

Similar to 2PL, when combined with the effect of forced termination, the mean number of data items a transaction will access before it terminates will be

$$K' = \sum_{i=1}^{h-1}\frac{K_{exp}}{S\eta}(1-p_b)^{i-1} + \frac{K_{exp}}{S}\left(S - \frac{(h-1)}{\eta}\right)(1-p_b)^{h-1}, \quad h = S\eta. \tag{21}$$

Thus, the expected execution time of a transaction will be the product of the value of K' and the execution time of each operation and is given by

$$S' = tK'. \tag{22}$$

The iterative algorithm to compute p_b and S is similar to that of the 2PL protocol. For the sake of brevity, we do not present it here. Similar to the 2PL protocol, by the expressions of S, K_{exp} and p_i, $1 \leq i \leq K_{exp}$, we can compute the system throughput as follows:

$$T = \lambda_{txn}\left[\prod_{i=1}^{k_{exp}}(1-p_i)\right]\sum_{i=1}^{h}(1-p_b)^{i-1}, \quad h = S\eta. \tag{23}$$

3.4. Analytic model for optimistic concurrency control protocol

Since there is no blocking effect in the OCC protocol, it is very simple to estimate the execution time needed for a transaction. As in the description in Section 2.3, there are three phases of transaction execution—*read phase*, *validation phase* and *write phase*. In the *read phase*, the transaction reads the values of data items from database and computes the results. The execution time needed in this phase equals the product of the number of operations and the execution time of each step. After the transaction completes all operations, it enters the *validation phase*. We assume the times to validate a transaction and the disk I/O delay are constants and equal to t_v and t_{io} respectively. The execution time in the last phase (*write phase*) equals to the product of the mean number of write operations and the disk I/O delay. Let p_v be the probability of validation conflict occurring. Therefore, the execution time needed for a transaction when the OCC protocol is employed is:

$$S = tK + t_v + (1-p_v)KP_uP_wt_{io}. \tag{24}$$

We are left with the task of obtaining an expression for p_v. Consider the validation of a transaction $T1$. For the OCC protocol, $T1$ has to be validated against two sets of transactions during its *validation phase* [11]:

(1) transactions that completed writing after $T1$ has entered the system but before $T1$ has started its validation, since they may have modified the data item read by $T1$, and
(2) transactions that were validated before $T1$ but have not yet completed writing by the time $T1$ started its validation, i.e., the transactions that are in the *write phase* when $T1$ is in the *validation phase*.

To derive an expression for p_v, we first have to determine the number of data items updated or to be updated by the first and second sets of transactions. A transaction can enter the *write phase* if there is no validation conflict or forced termination occurs. As in the description in Section 3.2, let h be the average number of handoffs before a transaction terminates ($h = S\eta$). Since the execution time of the *validation* and *write phases* is comparatively shorter than that of the *read phase*, we assume that no handoff occurs in the *validation* and *write phases*. By inputting the initial value of S, we can compute the value of p_b using the algorithm *compute_blocking_probability*. Therefore, the throughput will be

$$T = \lambda_{txn}(1-p_v)\sum_{i=1}^{h}(1-{p_b}^{i-1}), \quad h = S\eta. \tag{25}$$

The rate at which data items are updated is given by TKP_uP_w. Therefore the number of data items updated by the first set of transactions is equal to the rate at which data items are updated multiplied by the execution time of $T1$, i.e. $T(KP_uP_w)(KS)$. Similarly, the number of data items to be updated by the transactions of the second set is $T(KP_uP_w)t_v$. Hence, the total number of updated or to be updated data items that have to be checked is $T(KP_uP_w)(KS + t_v)$. We call this set of data items the *validation_set*. We ignore the effect that the *validation_set* includes the data items updated by $T1$. This is reasonable when the number of transactions is sufficiently large [11]. The probability that a single data item accessed by $T1$ does not lie in the *validation_set* is $1-[T(KP_uP_w)(KS+t_v)/N]$, and the probability that none of K data items lies in the *validation_set* is $1-[T(KP_uP_w)(KS+t_v)/N]^K$. Therefore,

$$p_v = 1 - \left(1 - \frac{T(KP_uP_w)(KS+t_v)}{N}\right)^K. \tag{26}$$

By the substitution of T, we have

$$p_v = 1 - \left\{1 - \lambda_{txn}(1-p_v)\left[\sum_{i=1}^{h}(1-p_b)^{i-1}\right] \times (KP_uP_w)(KS+t_v)/N\right\}^K, \quad h = S\eta. \tag{27}$$

The expected execution time of transactions and the system throughput can be determined after obtaining the value of p_v by Equations (24) and (25) respectively. The iterative algorithm for the OCC protocol is similar to that of the 2PL protocol. For the sake of brevity, we do not present it here.

4. SIMULATION MODEL

In order to analyze the performances of concurrency control strategies, we developed a discrete event simulation model using SIMSCRIPT II.5 [12]. To simulate a very large PCS network, we use the (8×8) mesh topology with 64 BSs as shown in Figure 3. The mobile unit may move to one of the four neighboring cells depending on the routing function of the mesh network, with the same probability of 0.25. There are three types of events in the PCS part for the simulation, which are arrival, handoff and complete. For the arrival event, the calls are divided into two types—VC and TC. When a TC arrives, a transaction is submitted to the database transaction simulator and then the TC will behave like a VC in the PCS network simulation part.

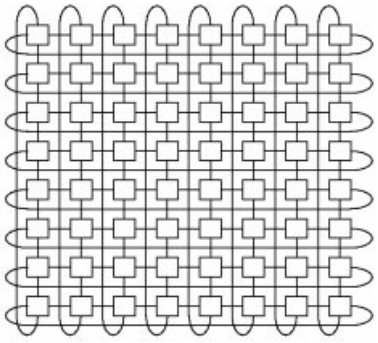

FIGURE 3. (8×8) mesh network as PCS network topology in this model.

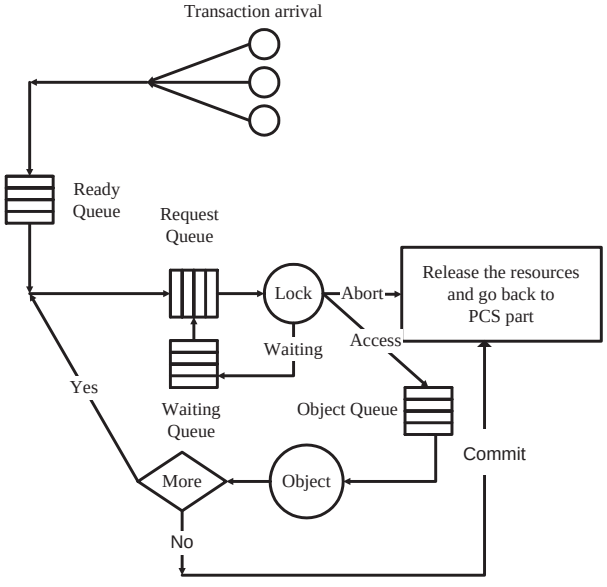

FIGURE 4. The simulator of the 2PL.

In the transaction simulator, we have constructed a database simulation model for the 2PL protocol as in Figure 4 [6]. A concurrency control agent will handle the lock requests from the transactions according to the rules of the 2PL protocol. If the lock is granted, the transaction will access the data item. If more than one data item is to be accessed by the transaction, the transaction will cycle through these queues several times. If the transaction cannot get the lock of the data item, it will enter the waiting queue of the object until the object is released by other transactions. As in the discussion in Section 2.4, the TO protocol is similar to the 2PL protocol and will not be described here. As to the simulator of the OCC protocol, the transaction reads all the data items it needs and writes them to a temporary space. After all the accesses of data are performed, a validation procedure is executed to guarantee the serializability. If a transaction is permitted to commit, it will write all its results to the database. The logical model of transaction execution when the OCC is used is shown in Figure 5.

There are two physical resources in this simulator, CPUs and disks. A certain amount of resource overhead is

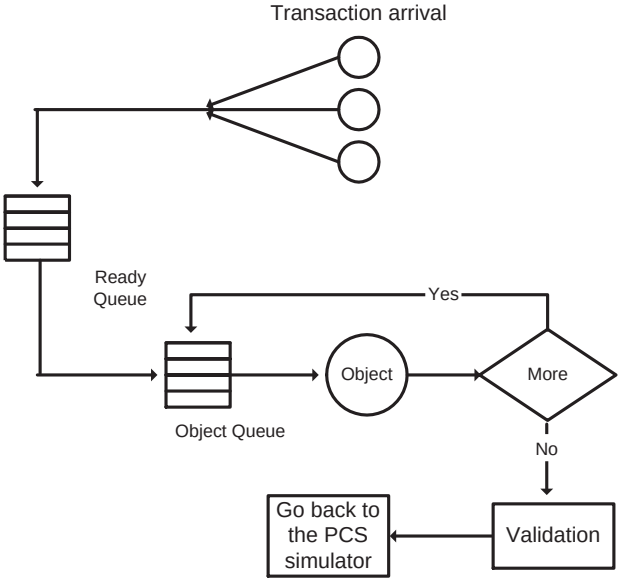

FIGURE 5. The simulator of the OCC.

associated with each concurrency control operation and database access. Concurrency control operations require only CPU service, and database accesses require both CPU and disk services. The CPU servers are modeled as a pool of servers serving a common CPU queue in first-come-first-served (FCFS) discipline. Unlike the CPU servers, there is an associated queue with a queue policy of FCFS for each disk. Data objects are randomly distributed among these disks. The size of the database, N, is set to 8000 objects, a relatively small size in order to attain a reasonable degree of data contention. The transaction size, which represents the number of operations performed by a transaction, is set to a value of uniform distribution between 15 and 25. The update transactions are about 60% of total transactions. Write operations in update transactions are about 50% of the total operations performed by update transactions. To simulate a distributed environment, there are four CPUs and eight disks in our simulator. The CPU access delay is set to 12 ms, and the I/O access delay is set to 35 ms. The CPU time for handling a lock request is set to 3 ms. These settings are similar to those in other papers on performance study and represent rough estimates of what realistic values might be [6, 13].

5. EXPERIMENTAL RESULTS

In this section, we present and analyze the results of the experiments. In order to achieve a 90% confidence level, we run each simulation for three batches with 800,000 calls and discard the first 80,000 calls of each batch to account for initial transient conditions. For the first 10% of calls the degree of channel contention and data contention is relatively low compared with the rest of calls [7]. First, we study the system performances of different protocols in different mobilities of mobile users. The metrics of performance considered here are the probabilities of forced

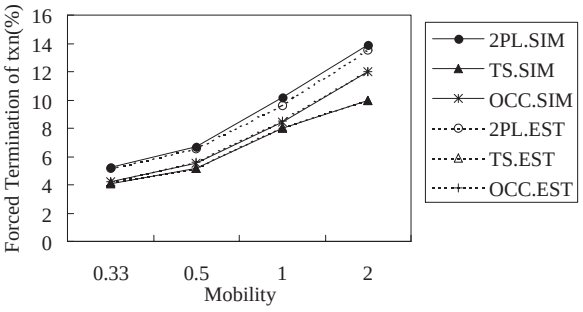

FIGURE 6. The probabilities of forced termination of transactions in different mobilities.

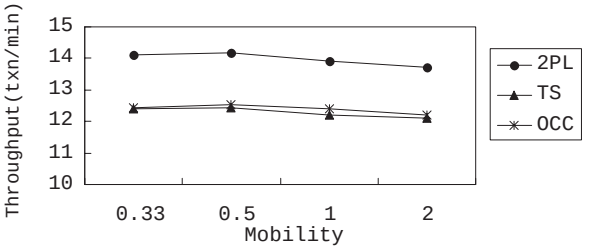

FIGURE 7. The transaction throughputs in different mobilities.

termination of transactions, transaction throughput and transaction complete time. Then we discuss the system performances in different degrees of data contention. In each experiment, we assume that there are 10 channels in each base station. The average call holding time of a voice call is set to 3 minutes and the network delay (including the wireless link) is set to 3 seconds.

5.1. Mobility-related experiments

Figure 6 shows the probability of a transaction being forced to terminate before it completes for these protocols in different mobilities of mobile users. Similar to the derivation in [7], the probability of a transaction being forced to terminate before it completes equals $(\lambda_{ht}/\lambda_t)p_f$. In Figure 6, for each protocol, SIM and EST represent the values obtained from the simulation and analytic model, respectively. The mobility of a user, η, is the expected number of cells that the user may cross per minute. In order to evaluate the effect of mobile behavior of a user, we vary the value of mobility (η) from 0.33 to 2. The reciprocal of the mobility is the excepted residence time which a user stays in a cell. In other words, the excepted residence time is varied from 0.5 to 3 minutes in the experiments. We can see that the probability of forced termination of transactions grows when a high value of mobility is set. Because in a fast mobility environment, a transaction is less likely to complete in one cell, so the frequency of handoffs may increase. This situation increases the probability of forced termination of transactions. In addition to the probabilities of forced termination of transactions, the transaction throughput is also an important metric when we evaluate the system performance. The transaction throughputs of mobility-related experiments are shown in Figure 7.

From Figures 6 and 7, we can find that these three protocols present almost the same behavior curve in the probabilities of forced termination of transactions, and the 2PL protocol has better transaction throughput. In Figure 6, the results obtained from the simulation and the analytic model are very close. However, the difference between the simulation and predicted results in the 2PL protocol is larger than those in TO and OCC. The reason is that we make some simplifications in the analytic models, especially in the estimation of deadlock probability in the 2PL protocol. Some studies ignore the effect of deadlock. However, since the execution time of a transaction in a mobile environment is longer compared with that in a conventional environment, the deadlock probability is also higher in a mobile environment. So the effect of deadlock cannot be ignored. Since the results obtained from the simulation and the analytic model are quite close, for clarity of illustration, the lines standing for the theoretical results are not drawn in the figures in the following context.

In Figure 7, there is no significant degradation in throughput. The reason for this is that when the mobility of users increases, the probability of forced termination of transactions will increase, so that the degree of data contention will decrease. As a consequence, the degradation of transaction throughput is not significant. In these experiments, the transaction complete time is relatively much smaller than the phone call holding time (about 1:3). In addition, the arrival rate of TC is relatively much smaller than that of VC (about 1:10). The performance of 2PL is slightly better than those of TS and OCC in this situation. However, when the arrival rate of TC increases, the blocking effect of 2PL will become severe. We will analyze the performances of these protocols in different arrival rates of TC in next section.

5.2. Data-contention-related experiments

Different degrees of data contention may affect the system performance when these protocols are applied. In these experiments, we simulate the transaction execution at different arrival rates of TCs. When the arrival rate of TC increases, the number of transactions submitted to the database will also increase. As a consequence, the degree of data contention will become higher. Figures 8 and 9 show the probabilities of forced termination of transactions and transaction throughputs in different arrival rates of TCs. The mobility of users is set to 0.5 in these experiments.

From Figure 8, we can find that the probabilities of forced termination of transactions of TO and OCC almost do not increase when arrival rate increases. However, the probability of forced termination of transactions of 2PL dramatically increases when the arrival rate increases. From Figure 9, we observe that the transaction throughputs of 2PL start degrading when the arrival rate of TCs is more than

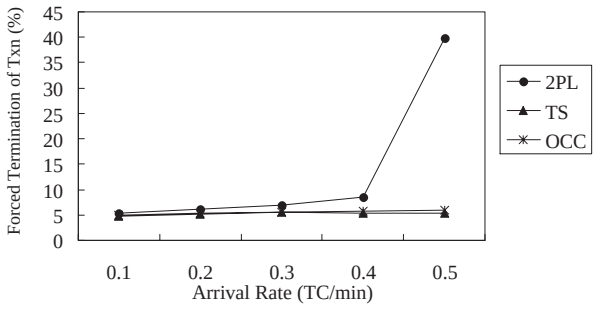

FIGURE 8. The probabilities of forced termination of transactions in different arrival rates.

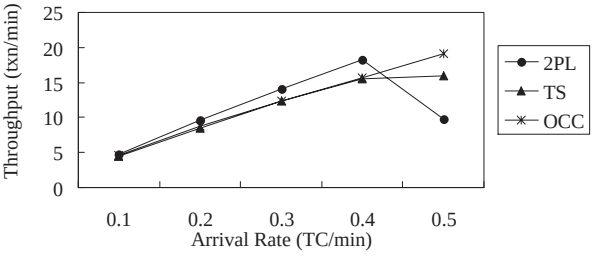

FIGURE 9. The transaction throughputs in different arrival rates.

0.4. The cause of this phenomenon is that 2PL performs poorly when the degree of data contention is high. When the arrival rate of TC increases, the transaction complete time of 2PL rapidly increases. However, the transaction complete times of TO and OCC do not increase when the arrival rate increases. This is because transactions in these two protocols do not hold the data items and need not wait for other transactions. Since the complete time of a transaction using 2PL is longer than those using the other two protocols, so the time duration of channels being held by transactions is also longer. As a consequence, the probability of forced termination of VC when handoff occurs is also higher in 2PL. Figures 10 and 11 show the average complete time of transactions and the probabilities of forced termination of VC of mobility-related experiments, respectively.

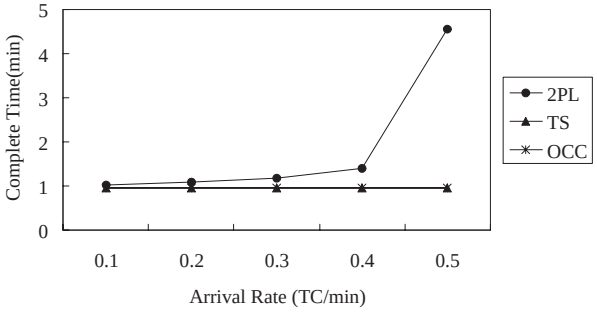

FIGURE 10. The average turnaround time of transactions in different arrival rates.

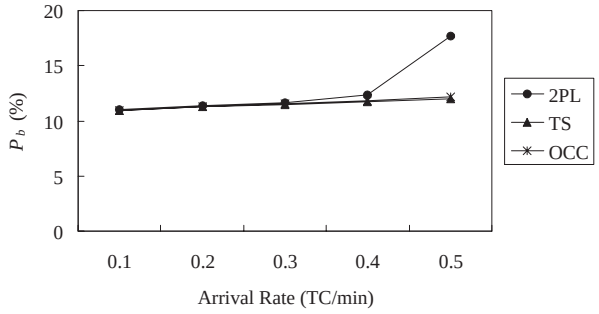

FIGURE 11. The probability of forced termination of VC in different arrival rates.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented a quantitative study and comparison of the performances of different concurrency control protocols in mobile environments. The performance metrics used here are the probability of forced termination of transactions, transaction throughput and transaction completion time. The effects of concurrency control protocols on the probability of forced termination of phone calls are also evaluated. In a conventional DBMS, locking-based algorithms generally outperform others. However, in mobile environments, optimistic algorithms show an improved performance because there are no blocking effects. Also, the policy of delayed resolution of data conflict results in achieving a greater chance of completing transactions in OCC when compared with the TO protocol. To evaluate the performance of concurrency control strategies in mobile environments, some detailed experiments were carried out. From the experimental results, we can conclude that OCC is of comparable performance with 2PL and TO when the degree of data contention is low. However, when the degree of data contention increases, the performance of 2PL starts degrading, and OCC outperforms the others in such a situation. In addition, the time to complete a transaction using 2PL is longer than that of OCC, implying that the expensive wireless channel is held longer by a transaction in 2PL. A quantitative guideline is very helpful for choosing a particular protocol. In the future, we will try to develop a quantitative guideline on the operating conditions of a mobile environment which favors a particular protocol.

No matter which concurrency control protocol is employed, some transactions may still be forced to terminate when handoff occurs. We are currently working on developing an algorithm which can reduce the probability of forced termination of transactions or prevent transactions from being terminated if there is no idle channel when handoff occurs.

REFERENCES

[1] Imerielinski, T. and Badrinath, B. R. (1994) Wireless mobile computing: challenges in data management. *Commun. ACM*, **37**, 19–28.

[2] Evaggelia, P. and Bharat, B. (1995) Maintaining consistency of data in mobile distributed environments. In *Proc. 15th Int. Conf. Distributed Computing Systems*, pp. 404–413.

[3] Walborn, G. D. and Chrysanthis, P. K. (1995) Supporting semantics-based transaction processing in mobile database applications. In *Proc. 14th IEEE Symp. on Reliable Distributed Systems*, pp. 215–224.

[4] Dunhan, M. H. and Helal, A. S. (1995) Mobile computing and databases: anything new? *ACM SIGMOD Record*, **24**, 5–9.

[5] Elmasri, R. and Navathe, S. B. (1994) *Fundamentals of Database Systems*, 2nd Edn, Addison-Wesley.

[6] Agrawal, R., Carey, M. J. and Livny, M. (1987) Concurrency control performance modeling: alternatives and implications. *ACM Trans. Database Syst.*, **12**, 609–654.

[7] Lin, Y.-B. (1997) Performance modeling for mobile telephone networks. *IEEE Networks*, **11**, 63–68.

[8] Tay, Y. C. (1985) Locking performance in centralized databases. *ACM Trans. on Database Syst.*, **10**, 415–462.

[9] Thomasian, A. and Ryu, I. K. (1991) Performance analysis of two-phase locking. *IEEE Trans. Software Engng*, **17**, 386–402.

[10] Singhal, M. (1991) Analysis of the probability of transaction abort and throughput of two time-stamp ordering algorithm for database systems. *IEEE Trans. Knowledge Data Engng*, **3**, 261–266.

[11] Dan, A., Towsley, D. F. and Kohler, W. H. (1988) Modeling the effects of data and resource contention on the performance of optimistic concurrency control protocols. In *Proc. 4th Int. Conf. on Data Engineering*, pp. 418–425.

[12] *SIMSCRIPT II.5 Programming Language* (1987) Los Angeles, CA: CACI.

[13] Agrawal, D., El Abbadi, E. and Lang, A. E. (1994) The performance of protocols based on locks with ordered sharing. *IEEE Trans. Knowledge Data Engng*, **6**, 805–818.

APPENDIX A. NOTATION

This appendix lists the notation used in the equations.

λ_v The voice call arrival rate to a cell.

λ_{hv} The arrival rate of handoff VCs to a cell.

λ_t The transaction call arrival rate to a cell.

λ_{ht} The arrival rate of handoff TCs to a cell.

λ_{txn} The transaction arrival rate to the database.

$1/\eta$ The mean mobile unit residence time in a cell.

$1/\mu$ The mean call holding time of a voice call.

$1/\mu'$ The mean call holding time of a call (VC or TC).

S The mean call holding time of a transaction call.

S' The mean residence time in the database system for a transaction.

p_b The blocking probability of a call.

p_c The probability of conflict occurs when a transaction accesses a data item.

p_d The deadlock probability.

β Fraction of transaction which is in blocked state.

P_u Fraction of update transactions among all transactions.

P_w Fraction of write operations in an update transaction.

N Number of data items in the database.

K Number of data items requested by a transaction.

K' The mean number of data items accessed by a transaction before it terminates.

t_c The mean call holding time of a voice call.

t The execution time of a transaction operation.

t_v The validation time in OCC protocol.

t_w The I/O delay in the database system.

W_1 The mean waiting time for a transaction blocked by a active transaction.

W The mean waiting time for a transaction when a lock conflict occurs.