ELSEVIER Performance Evaluation 25 (1996) 105–130

PERFORMANCE EVALUATION An International Journal

Performance modeling of distributed timestamp ordering: Perfect and imperfect clocks ★

C.J. Bouras [a,b,*], P.G. Spirakis [a,b]

[a] *Department of Computer Science and Engineering, University of Patras, Greece*
[b] *Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece*

Received 1 November 1992; revised 1 July 1994

Abstract

This work presents a model of a distributed database system which provides the framework to study the performance of timestamp ordering concurrency control. Locking and timestamping are two popular approaches to concurrency control in database systems. Timestamp-based algorithms have been proposed to protect distributed databases from inconsistencies during concurrent access. In these algorithms, transactions may reach a particular site in different order than their timestamps, due to unexpected network delays. This causes conflicts which the distributed concurrency control mechanism has to cope with. We exhibit an analytical solution, which has been tested with extensive simulation. The accuracy seems to be very high. We assume perfect and also imperfect clocks for synchronization and quantify the way in which local clock inaccuracies affect the phenomenon of transaction conflicts. In particular, we derive a lot of interesting performance measures such as probability of abort, mean waiting time, throughput, mean queue length and others.

Keywords: Concurrency control; Performance; Timestamp; Reordering; Conflicts; Queueing; Clock drifts; Distributed databases

1. Introduction

Database concurrency control is concerned with the problems that arise when several users access and update a database simultaneously. Concurrency control algorithms try to maintain the consistency of the database. Since the semantics of transactions are embedded in application programs, it is not easy to design concurrency control algorithms that take advantage of this knowledge. Most techniques therefore try to preserve (syntactic) serializability by trying to produce the same effect on the database as a serial execution

★ This research was partially supported by the Ministry of Education of Greece, the EEC RACE Project TELEMED and Project HELIOS/STRIDE.

* Corresponding author. Tel.: +30-61997585. Fax: +30-61991909. E-mail: Bouras@cti.gr.

SSDI 0166-5316(94)00041-7

Research in the area of concurrency control for distributed database systems has led to the development of many concurrency control algorithms. Most of these algorithms are based on one of three basic mechanisms: locking, timestamps and optimistic concurrency [4]. Given the ever-growing number of available concurrency control algorithms, considerable research has recently been devoted to evaluating the performance of concurrency control algorithms. Performance studies of concurrency control algorithms have been done using simulations as well as analytical methods [9,15,22,25,1,19,5,12,21,6–8].

Analytical queueing models of concurrency control algorithms for distributed database systems have two distinct features which are usually absent in the queueing models of conventional systems. First, multiple resource possession, where a transaction holds some data items before and during seeking a service at CPU or I/O device. Second, blocking, where due to conflicts in resource requirements or a message wait, a server may be blocked. Because of the above characteristics, queueing models of concurrency control algorithms for distributed database systems are not amenable to the conventional product form solutions. The above problems are not known to have a closed form solution for queueing networks of general topology. Simulation studies are costly with respect to computer time. It is impossible to cover the parameter space as thoroughly as one would like. The major advantage of simulation over analysis is that simulation can be used to treat some system model whose level of detail precludes an analytic solution.

There are two critical points in analyzing the performance of timestamp ordering concurrency control algorithms in distributed database systems.

- The reordering phenomenon

  For obvious reasons, due to variable network delays, transactions do not necessarily arrive at the system sites in order. From the point of view of queueing theory this phenomenon has been analyzed in [2,3,10,16,17,23]. Our work extends these results in the field of performance analysis of concurrency control algorithms in distributed database systems. It is important to mention that other works, such as [22], assume constant time for the network delay.
- The effect of clock drifts

  All models up to now assume global time for timestamping. In contrast real physically distributed database systems use local clocks which do not indicate the same time. The effect of clock drifts is an important point [6–8,14]. However, aside from relativity considerations, it usually holds that there is some bounded proportion between elapsed local time spans [18,24]. Techniques such as message passing can be used to keep local clocks almost synchronized [20]. In this work we assume that local clocks suffer a small bounded drift.

  This work studies:
- The performance of Basic Timestamp Ordering (BTO) using a new approximating solution, different from the other works, such as [19,21].
- An exact solution for the performance study of the Conservative Timestamp Ordering ($CoTO$).
- The effect of clock drifts on the performance of the BTO and $CoTO$. This happens for the first time.

2. Basic architecture of distributed databases

A database is a collection of shared data items. In a database, certain relationships hold among its data items. The set of these relationships for a database is called the consistency assertions of the database. A database is in a consistent state if the current values of its data items satisfy all of its consistency assertions. In such a system, a transaction is a program with read, write and other operations. A Distributed Database

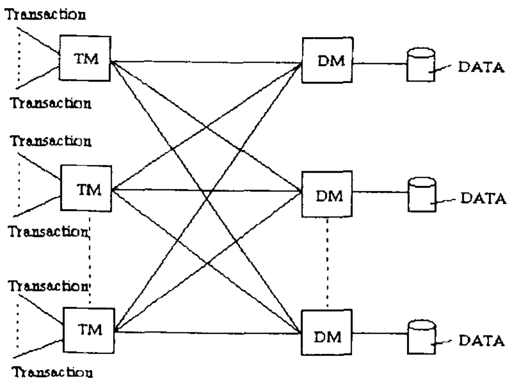


Fig. 1. Architecture of the *DDBMS* model.

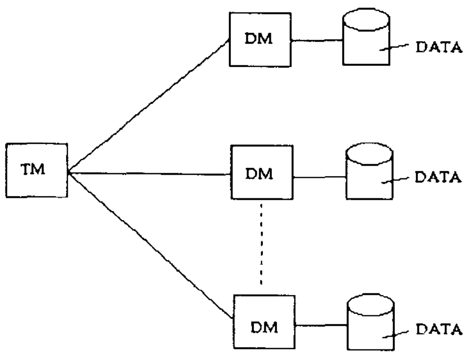


Fig. 2. *DDBMS* model for a single transaction.

Management System (*DDBMS*) may be viewed as a collection of nodes connected to a network. Each node consists of a computer running either a Transaction Manager (*TM*) or a Data Manager (*DM*) or both. The nodes communicate by sending messages over the network. The network is assumed to be completely reliable, i.e., if node *A* sends a message to node *B*, it is guaranteed that *B* will receive the message error-free. The architecture of the system is shown in Fig. 1. A *TM* coordinates the execution of the transaction. A *DM* manages a local database. From the viewpoint of a single transaction, the system consists of a single *TM* and a number of *DM*s, Fig. 2. Neither *TM*s nor *DM*s intercommunicate.

There are several reasons why *DDBMS*s are developed. Many organizations are decentralized and a *DDBMS* approach fits, more naturally, the structure of the organization. *DDBMS* is the natural solution when several databases already exist in an organization and the necessity of performing global applications arises. The recent development of small computers, providing (at a lower cost) many of the capabilities which were previously provided by large mainframes, constitutes the necessary hardware support for the development of distributed information systems. The technology of *DDBMS* is based on two other technologies which have developed a sufficiently solid foundation during the past years, computer networks technology and database technology. More details about *DDBMS* can be found in [11].

3. Time and timestamps in a distributed database

In a distributed system, it is sometimes necessary to know if an event A at some site happened before or after an event B at a different site. Determining the order of events is simple in a centralized system, since it is possible to use the same clock to determine the time at which each event occurs. In a distributed system, instead, it is not realistic to assume that perfectly synchronized clocks are available at all sites [11].

Several distributed concurrency control and deadlock prevention algorithms need the determination of an ordering of events. The determination of an ordering of events consists in assigning to each event A which occurs in the distributed system a timestamp $TS(A)$ having the following properties.

(1) $TS(A)$ uniquely identifies A (i.e., different events have different timestamps).

(2) For any two events A and B, if A occurred before B, then $TS(A) < TS(B)$.

The main inconvenience of the above definition is that the meaning of a relationship "occurred before" is not precisely defined if the two events A and B occurred at two different sites, since we do not possess a "global clock" for measuring the exact time of occurrence of all events in the distributed system. Therefore, in this section we first define accurately the meaning of the "occur before" relationship in a distributed system, and then present an algorithm which produces timestamps having the above two properties.

A precise definition of the "occur before" relationship in a distributed system is the following. Assume that we know the meaning of the statement "Event A occurred before B at site i", i.e., that we know the meaning of time ordering at a single site. The relation occurred before, denoted $\rightarrow$, can be generalized to a distributed environment by the following rules:

(1) If A and B are two events at the same site and A occurred before B, then $A \rightarrow B$.

(2) If the event A consists in sending a message and event B consists in receiving the same message, then $A \rightarrow B$.

(3) If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

4. Concurrency control based on timestamps

With this method, a unique timestamp is assigned to each transaction; transactions are processed so that their execution is equivalent to a serial execution in timestamp order. This concurrency control mechanism allows a transaction to read or write a data item x only if x has been last written by an older transaction; otherwise it rejects the operation and restarts the transaction. If the timestamps do not reflect with enough accuracy the younger–older relationship of transactions, then the following might happen. Suppose that a transaction T_i obtains a timestamp which is smaller than the timestamp of another, already completed, transaction T_j, which has written data items that are needed by T_i. T_i is aborted and restarted with a new timestamp until it obtains a timestamp which is greater than the timestamp of T_j. Note that even in this case the concurrency control mechanism produces correct executions, at the expense of needlessly restarting the same transaction several times.

4.1. *The description of BTO*

The BTO applies the following rules:

(1) Each transaction receives a timestamp when it is initiated at its site of origin.

(2) Each read or write operation which is required by a transaction has the timestamp of the transaction.

(3) For each data item x, the largest timestamp of a read operation and the largest timestamp of a write operation are recorded; they will be indicated as $RTM(x)$ and $WTM(x)$.

(4) Let TS be the timestamp of a read operation on data item x. If $TS < WTM(x)$, the read operation is rejected and the issuing transaction restarted with a new timestamp; otherwise, the read is executed and $RTM(x)$ is set to $\max(RTM(x), TS)$.

(5) Let TS be the timestamp of a write operation on data item x. If $TS < RTM(x)$ or $TS < WTM(x)$, then the operation is rejected and the issuing transaction is restarted; otherwise, the write is executed, and $WTM(x)$ is set to TS.

An interesting feature of the basic timestamp mechanism is that it is deadlock-free, because transactions are never blocked: if a transaction cannot execute an operation, it is restarted.

4.2. The description of CoTO

The main disadvantage of the BTO is the great number of restarts which it causes; $CoTO$ is a method which eliminates restarts by buffering younger operations until all older conflicting operations have been executed, so that operations are never rejected and transactions are never restarted. In order to execute a buffered operation it is necessary to know the time no more older conflicting operations exist.

The $CoTO$ is based on the following requirements and rules.

(1) Each transaction is executed at one site only and does not activate remote programs. It can only issue read or write requests to remote sites.

(2) A site i must receive all the read requests from a different site j in timestamp order. Similarly, a site i must receive all the write requests from a different site j in timestamp order.

These requirements are very simple to satisfy. First, let us assume that the communication network does not change the order of messages between two sites. Second, each site must be capable of sending request messages in timestamp order. This can be achieved in two ways. It is possible to process transactions serially at each site; this, however, is not very satisfying for a concurrency control mechanism. A more attractive solution is to execute transactions by issuing all read requests before their main execution and all write requests after their main execution. In this way, if $TS(T_i) < TS(T_j)$, it is sufficient to wait to send the R_j operations until all R_i operations have been sent and to wait to send the W_j operations until all W_i operations have been sent. The above requirements are thus satisfied, even if the two transactions run concurrently.

(3) Assume that a site i has at least one buffered read and one buffered write operation from every other site of the network. Because of requirement (2), site i knows there are no older requests which can arrive from any site. The concurrency controller at site i behaves therefore in the following way:

  (a) For a read operation R that arrives at site i.
  If there is some write operation W buffered at site i such that $TS(R) > TS(W)$, then R is buffered until these writes are executed, else R is executed.

  (b) For a write operation W that arrives at site i.
  If there is some read operation R buffered at site i such that $TS(W) > TS(R)$ or there is some write operation W' buffered at site i such that $TS(W) > TS(W')$, then W is buffered until these operations are executed, else W is executed.

5. The model

We assume that the distributed database consists of K sites. The database is not replicated. This means that each data object exists in only one site. So, in each site there exists a different Local Data Base. The number of data objects per site is N. A perfectly reliable network is assumed to connect the K sites. A key parameter in our model is the end-to-end delay which is the elapsed time from the sending of a transaction from its source until the delivery of the transaction at its destination. Transactions are generated at different sites as independent Poisson processes. We assume that local processing times are negligible compared to communication delays. We also assume that transaction generations and communication delays are statistically independent. Each transaction is assumed to access M data objects, which belong to the same Local Data Base. Each Local Data Base accepts an independent Poisson process of transactions with rate λ. Transactions travel across the network as message packets of reads and writes (one such packet per transaction). The data objects accessed by each transaction are equiprobably selected among the N data objects (uniform access).

Then the probability of two transactions having at least one common data object is

$$p_c = 1 - \frac{\binom{N-M}{M}}{\binom{N}{M}} = 1 - \frac{(N-M)!^2}{N!(N-2M)!}$$
$$= 1 - \frac{(N-M)(N-M-1)\cdots(N-2M-1)}{(N)(N-1)\cdots(N-M-1)}$$
$$= 1 - \left(1-\frac{M}{N}\right)\left(1-\frac{M}{N-1}\right)\cdots\left(1-\frac{M}{N-M+1}\right).$$

The above expression can be further simplified to

$$p_c = 1 - \left[1 - M\left(\frac{1}{N}+\frac{1}{N-1}+\cdots+\frac{1}{N-M+1}\right) + \mathrm{O}\left(\frac{M^2}{N^2}\right) + \text{higher order items}\right].$$

If we ignore second and higher order terms and we further assume that $M \ll N$ (which is usually the case in practice) then

$$p_c \approx \frac{M^2}{N}. \tag{1}$$

The above result has been independently derived in [22].

In the case of clock drifts we assume an ϵ-bounded drift [18] among the clocks. More specifically, if t is the global time and $LC(j,t)$ the indication of the clock of site j at t , then there is an $\epsilon > 0$ such that for all j

$$|LC(j,t) - t| < \epsilon. \tag{2}$$

It is obvious that the unique timestamp which each transaction receives is $LC(j,t)$. Furthermore, the values of $LC(j,t)$ are assumed to be uniformly and independently distributed in $[t \pm \epsilon]$. The constant ϵ is known from the specification of the underlying hardware clocks. Typically ϵ is very small, on the order of 10^{-5} to 10^{-6}. Note that only perfectly synchronized clocks were considered by the research on the performance of timestamps algorithms up to now [18].

The requests originating from the same site are assumed to be $FIFO$, i.e., they will arrive at the same site in the order of their timestamps. This can be ensured by communication network protocols. The $FIFO$ transmission requirement introduces resequencing delays. The distribution of such delays was analyzed in [2,3,16,17,23], mostly by modeling the network connecting two sites as an M/M/∞, an M/G/∞ or an M/M/k queueing system. Even in such a case, the PDF of the pipelined transmission delay does not have a convenient form. In our analysis, the PDF of this delay is an input parameter.

6. Performance analysis of BTO

Consider one of the sites of the $DDBMS$. It receives a sequence of transactions which affect the contents of the Local Data Base. We assume that each transaction is identified by a timestamp, and that each site of $DDBMS$ carries out the transactions in timestamp order. Let $T_1, T_2, \dots, T_n$ denote a sequence of transactions which enter the system and which are directed to each of the $DDBMS$ sites via a communication network, and let $LC(i, t_n)$ denote the timestamp associated with T_n. We use the term LC instead of TS, as in the previous section, to mention the use of logical clocks. But the two terms have the same meaning in our analysis. In the case of perfect clocks $LC(i, t_n)$ is equal to global time t_n, where t_n is the generation time for the transaction T_n. Each transaction T_n reaches the site where it must be executed after a communication delay y_n. Thus at the output of the communication network the transactions arrive at instants $t_n + y_n$, that do not necessarily respect the timestamp order. For this reason (the reordering phenomenon), there is a probability of abort, PA, for each transaction T_n. So, we have the following queueing problem:

> Every transaction T_n, with generation time t_n, timestamp $LC(i, t_n)$ and network delay y_n, must finish before the transactions which will arrive after it, and there is some fixed conflict probability p_c between them. What is the probability of abort, PA?

There are two cases for analysis: Case I, Perfect clocks, and Case II, Imperfect clocks.

6.1. Case I: Perfect clocks

$$t + d > t_i + d_i \tag{3}$$

Let us consider a particular transaction T generated at instant t which is then transmitted to one site. We assume that the transactions which conflict construct an independent Poisson process, with rate $\lambda_c = p_c\lambda$ [23]. It is obvious, that in the case of perfect clocks, each transaction must be rejected by transactions which arrive later than it. So, from Fig. 3, the transaction T, that arrives at instant t, and has a network

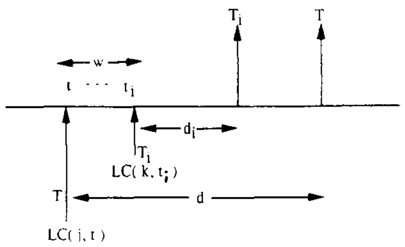


Fig. 3. The reverse order issue in case of perfect clocks.

delay d, must be rejected from each transaction T_i, which is generated at t_i, $t < t_i$, and has a delay, due to the network, d_i. Then a reverse order will be found between T and each other from T_i transactions if the following inequality holds:

Let event OR_i be the inequality $t + d > t_i + d_i$. Also note that event OR_i can be rewritten as

$$d > d_i + w. \tag{4}$$

In the sequel we assume that d and d_i are exponentially distributed with mean μ. The factor $w = t_i - t$ is the time which we must wait until the generation of the ith transaction after transaction T. It is known [13] that w has a Gamma distribution, with parameters i and λ_c. It is also known that its pdf is

$$g_i(w) = \frac{\lambda_c^i w^{i-1} e^{-\lambda_c w}}{(i-1)!}$$

and its PDF is

$$G_i(w) = 1 - e^{-\lambda_c w} \sum_{n=0}^{i-1} \frac{(\lambda_c w)^n}{(n-1)!}.$$

The probability of reverse order between transaction T and each one from T_i is

$$\begin{aligned} p_i &= \Pr\{d > d_i + w\} = \Pr\{d > w, d - w > d_i\} \\ &= \Pr\{d > w\}\Pr\{d - w > d_i \mid d > w\}. \end{aligned} \tag{5}$$

From our assumptions, the random variables d, d_i and w are independent. So from [13] we have that

$$\Pr\{d - w > d_i \mid d > w\} = \Pr\{d > d_i\}$$

Thus,

$$p_i = \Pr\{d > w\}\Pr\{d > d_i\}.$$

Calculation of $\Pr\{d > w\}$

$$\begin{aligned} \Pr\{d > w\} &= \int_0^\infty \Pr\{d \geq w\} g_i(w)\, dw = \int_0^\infty (1 - (1 - e^{-\mu w})) \frac{\lambda_c^i w^{i-1} e^{-\lambda_c w}}{(i-1)!}\, dw \\ &= \frac{\lambda_c^i}{(i-1)!} \int_0^\infty w^{i-1} e^{-(\lambda_c + \mu) w}\, dw. \end{aligned}$$

By default the integral is equal with $\Gamma(i)/(\lambda_c + \mu)^i$. Thus,

$$\Pr\{d > w\} = \frac{\lambda_c^i}{(i-1)!} \frac{\Gamma(i)}{(\lambda_c + \mu)^i}.$$

But it is known that, if $i \in \mathbb{Z}^+$ then $\Gamma(i) = (i-1)!$ Finally we have that

$$\Pr\{d > w\} = \left(\frac{\lambda_c}{\lambda_c + \mu}\right)^i. \tag{6}$$

Calculation of $\Pr\{d > d_i\}$

$$\Pr\{d > d_i\} = \int_0^\infty \Pr\{d \geq d_i\} f(d_i)\,\mathrm{d}(d_i) = \int_0^\infty (1 - (1 - \mathrm{e}^{-\mu d_i}))\mu \mathrm{e}^{-\mu d_i}\,\mathrm{d}(d_i)$$
$$= \mu \int_0^\infty \mathrm{e}^{-2\mu d_i}\,\mathrm{d}(d_i) = \frac{1}{2}. \tag{7}$$

Thus,

$$p_i = \frac{1}{2}\left(\frac{\lambda_c}{\lambda_c + \mu}\right)^i. \tag{8}$$

Now, for the calculation of the rejection probability PA of a transaction, we must take into account the following factors:

- The probability of reverse order p_i, with a younger transaction, decreases exponentially in relation to how much younger this transaction is. This means that the transaction has a bigger probability of reverse order with some transactions with which it has small differences with their birth time. Thus we observe that the reordering phenomenon has a locality.
- The interference phenomenon among transactions. This means that one of the transactions that can reject it, may already have been rejected, etc. We must note here that the interference phenomenon is one of the most difficult problems in queueing theory [10].

For the above reasons we will use an approximation so as to calculate PA. Thus we have

$$\begin{aligned} PA &= \Pr\{\text{rejection of a transaction}\} \\ &= \Pr\{\text{it is rejected by the first transaction after it}\} \\ &\quad \times \Pr\{\text{the first transaction is not rejected}\} \\ &\quad + \Pr\{\text{it is rejected by the second transaction after it}\} \\ &\quad \times \Pr\{\text{it is not rejected by the first transaction after it}\} \\ &\quad + \Pr\{\text{it is rejected by the third transaction after it}\} \\ &\quad \times \Pr\{\text{it is not rejected by the second transaction after it}\} \\ &\quad \vdots \\ &\quad + \Pr\{\text{it is rejected by the } i\text{th-transaction after it}\} \\ &\quad \times \Pr\{\text{it is not rejected by the } (i-1)\text{th-transaction after it}\} \\ &\quad \vdots \end{aligned}$$

From the above it is straigthforward to observe that the probability of abort PA, for each transaction is,

$$\begin{aligned} PA &= (1 - p_1)p_1 + (1 - p_1(1 - p_1))p_2 + \cdots + (1 - p_k(1 - p_{k-1}(\cdots)\cdots))p_{k+1} + \cdots \\ &= \sum_{i=1}^{\infty} p_i \pi_i, \end{aligned} \tag{9}$$

where

$$\pi_1 = (1 - p_1),$$

$$\pi_2 = (1 - p_1\pi_1),$$

$$\vdots$$

$$\pi_k = (1 - p_{k-1}\pi_{k-1}).$$

So, *Probability of Not Rejected* (*PNR*) is

$$PNR = 1 - PA. \tag{10}$$

Other interesting performance measures are $Throughput(THR)$,

$$THR = \lambda\, PNR \tag{11}$$

and $AbortRatio\,(AR)$,

$$AR = \lambda\, PA. \tag{12}$$

6.2. *Case II: Imperfect clocks*

In this case, due to the clock drifts, there is the possibility that each transaction must be rejected by transactions which arrive later than it or before. Let us consider a particular transaction T generated at site j. Denote by w the time we have to wait until we see the generation of ith transaction T_i (say at site k). Let t and t_i be the actual generation times of T, T_i and $LC(j,t)$, $LC(k,t_i)$ the corresponding timestamps of T and T_i. Clearly $w = t_i - t$ has a Gamma distribution. If d and d_i denote the network delays (transmission plus FIFO) for T and T_i then a reverse order will be found if and only if one of the following two sets of inequalities holds:

Either, for future $t < t_i$ (Fig. 3), T is generated earlier than T_i and T arrives later than T_i,

$$LC(j,t) < LC(k,t_i) \quad \text{and} \quad t + d > t_i + d_i,$$

or, for past $t > t_i$ (Fig. 4), T is generated later than T_i, the clocks are out of order, but T arrives later than T_i,

$$LC(j,t) < LC(k,t_i) \quad \text{and} \quad t + d > t_i + d_i \tag{13}$$

Let event E_1^i be the inequality $LC(j,t) < LC(k,t_i)$, event E_2^i be the inequality $t + d > t_i + d_i$.

In this case we define event of order reverse $OR_{i\epsilon}$ as

$$OR_{i\epsilon} = (E_1^i \wedge E_2^i). \tag{14}$$

Note that all literature up to now considered event E_1^i to be just $t < t_i$ (thus ignoring the clock synchronization issue). The probability of reverse order, in the case of the future, then is

$$\begin{aligned} p_{i\epsilon}^f &= \Pr\{OR_{i\epsilon} \text{ in case of future}\} = \Pr\{E_2^i\}\Pr\{E_1^i\} \\ &= \Pr\{d > d_i + w\}\Pr\{E_1^i\} = p_i \Pr\{E_1^i\}. \end{aligned} \tag{15}$$

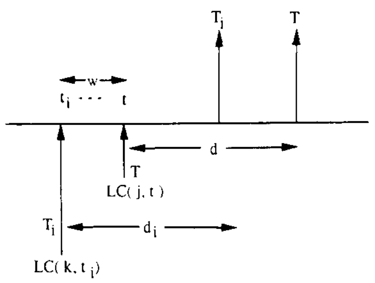


Fig. 4. The reverse order issue in case of imperfect clocks.

Also, in the case of the past we have that

$$p^{p}_{i\epsilon} = \Pr\{OR_{i\epsilon} \text{ in case of past}\} = p_i(1 - \Pr\{E^i_1\}). \tag{16}$$

As far as $\Pr\{E^i_1\}$ is concerned we have the following two cases:

Case 1 (*Fig.* 5). If $t+\epsilon \leq t_i - \epsilon$, which means that $w = t_i - t \geq 2\epsilon$, then event E^i_1 holds with conditional probability 1.

In this case

$$\begin{aligned}\Pr\{\text{event } E^i_1 \text{ in Case 1}\} &= \Pr\{LC(j,t) \leq LC(k,t_i)\}\Pr\{\text{event } E_1 \text{ given Case 1}\}\\ &= \Pr\{t_i - t \geq 2\epsilon\} * 1 = 1 - G_i(2\epsilon\lambda_c) = \mathrm{e}^{-2\epsilon\lambda_c}\sum_{n=0}^{i-1}\frac{(2\epsilon\lambda_c)^n}{n!}.\end{aligned} \tag{17}$$

Case 2 (*Fig.* 6). If $t + \epsilon > t_i - \epsilon$ then $2\epsilon > w = t_i - t$.

In this case

$$\Pr\{\text{event } E^i_1 \text{ in Case 2}\} = \Pr\{t_i - t < 2\epsilon\}\Pr\{\text{event } E^i_1 \text{ given Case 2}\}.$$

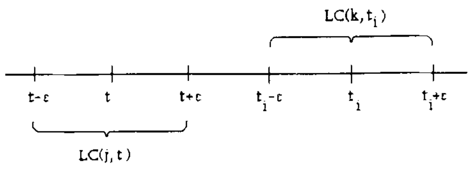


Fig. 5. Case 1.

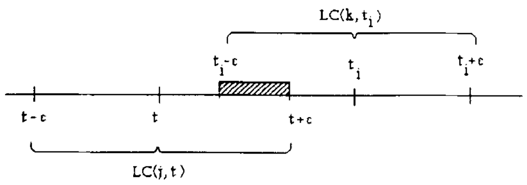


Fig. 6. Case 2.

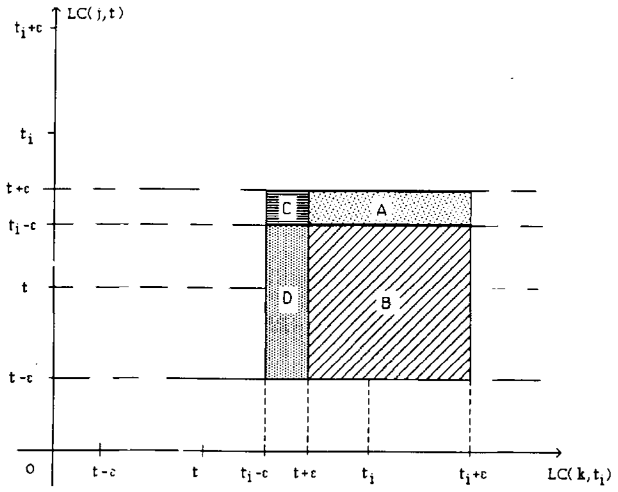


Fig. 7. The four subcases of Case 2.

It is straightforward to observe that

$$\Pr\{t_i - t < 2\epsilon\} = G_i(2\epsilon\lambda_c) = 1 - e^{-2\epsilon\lambda_c} \sum_{n=0}^{i-1} \frac{(2\epsilon\lambda_c)^n}{n!}.$$

Also, Fig. 7, by conditioning on $t_i - t = w, 0 \le w \le 2\epsilon$,

$$\begin{aligned}\Pr\{E_1^i \mid \text{Case } 2\} = {} & \Pr\{A\}\Pr\{E_1^i \mid A\} + \Pr\{B\}\Pr\{E_1^i \mid B\} \\ & + \Pr\{C\}\Pr\{E_1^i \mid C\} + \Pr\{D\}\Pr\{E_1^i \mid D\}\end{aligned}$$

It is easy to see that

$$\Pr\{E_1 \mid A\} = \Pr\{t + \epsilon < LC(k, t_i) < t_i + \epsilon \text{ and } t_i - \epsilon < LC(j, t) < t + \epsilon\} = 1.$$

Similarly

$$\Pr\{E_1^i \mid B\} = 1, \qquad \Pr\{E_1^i \mid C\} = \tfrac{1}{2}, \qquad \Pr\{E_1^i \mid D\} = 1.$$

In Fig. 7, the horizontal axis indicates the possible values of $LC(k, t_i)$ and the vertical one the possible values of $LC(j, t)$.

Also, by counting areas in Fig. 7, we get

$$\Pr\{A\} = \frac{(t_i - t)(t - t_i + 2\epsilon)}{4\epsilon^2}, \qquad \Pr\{B\} = \frac{(t_i - t)^2}{4\epsilon^2},$$

$$\Pr\{C\} = \frac{(t - t_i + 2\epsilon)^2}{4\epsilon^2}, \qquad \Pr\{D\} = \frac{(t_i - t)(t - t_i + 2\epsilon)}{4\epsilon^2}.$$

Table 1
Numerical results for $\lambda_c = 5$

i	$\Pr\{E_1^i\}$					
	$\epsilon = 10^{-6}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-2}$	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1$
1	0.9999	0.9992	0.9341	0.6978	0.6845	0.6198
2	1	1	0.9974	0.8596	0.7484	0.7144
3	1	1	0.9999	0.9639	0.7501	0.7209
4	1	1	1	0.9935	0.7609	0.7402
5	1	1	1	0.9991	0.7855	0.7662
6	1	1	1	0.9999	0.8647	0.8493
7	1	1	1	1	0.9299	0.8999
8	1	1	1	1	0.9691	0.9455

After this,

$$\Pr\{E_1^i \mid \text{Case 2}\} = \left[(t_i - t)(t - t_i + 2\epsilon) + (t_i - t)^2 + \tfrac{1}{2}(t - t_i + 2\epsilon)^2 + (t_i - t)(t - t_i + 2\epsilon)\right]\frac{1}{4\epsilon^2}.$$

Thus, by conditioning on $t_i - t = w$ we have

$$\Pr\{E_1^i \text{ in Case 2}\} = \frac{1 - \mathrm{e}^{-2\epsilon\lambda_c}\sum_{n=0}^{i-1}(2\epsilon\lambda_c)^n/n!}{8\epsilon^2}\int_{w=0}^{2\epsilon}\left[4w\epsilon - w^2 + 4\epsilon\right]\frac{\lambda_c^n w^{n-1}\mathrm{e}^{-\lambda_c w}}{(n-1)!}\,\mathrm{d}w. \quad (18)$$

Thus, finally from Cases 1 and 2 we have

$$\Pr\{E_1^i\} = \mathrm{e}^{-2\epsilon\lambda_c}\sum_{n=0}^{i-1}\frac{(2\epsilon\lambda_c)^n}{n!} + \frac{1 - \mathrm{e}^{-2\epsilon\lambda_c}\sum_{n=0}^{i-1}(2\epsilon\lambda_c)^n/n!}{8\epsilon^2}\int_{w=0}^{2\epsilon}\left[4w\epsilon - w^2 + 4\epsilon\right]\frac{\lambda_c^n w^{n-1}\mathrm{e}^{-\lambda_c w}}{(n-1)!}\,\mathrm{d}w. \quad (19)$$

Table 1 shows numerical results for $\Pr\{E_1^i\}$. From Appendix A and Table 1, we have

$$\lim_{\epsilon\to\infty}\Pr\{E_1^i\} = \frac{1}{2}$$

In a similar manner

$$\lim_{\epsilon\to 0}\Pr\{E_1^i\} = 1 \quad \text{and} \quad \lim_{i\to\infty}\Pr\{E_1^i\} = 1.$$

From the above, we conclude that,

$$\tfrac{1}{2} \le \Pr\{E_1^i\} \le 1.$$

Using the same argument, as in perfect clocks, it is easy to observe that the probability of abort PA for each transaction is

$$\begin{aligned} PA &= \Pr\{\text{rejection of a transaction}\} \\ &= \Pr\{\text{it is rejected by the first transaction in the future}\} \\ &\quad \times \Pr\{\text{this transaction is not rejected}\} \\ &\quad + \Pr\{\text{it is rejected by the first transaction in the past}\} \\ &\quad \times \Pr\{\text{it is not rejected by the first one in the past}\} \\ &\quad + \Pr\{\text{it is rejected by the second transaction in the future}\} \\ &\quad \times \Pr\{\text{it is not rejected by the first one in the past}\} \\ &\quad \vdots \end{aligned}$$

Thus,

$$\begin{aligned} PA &= (1 - p^f_{1\epsilon})p^f_{1\epsilon} + (1 - (1 - p^f_{1\epsilon})p^f_{1\epsilon})p^p_{1\epsilon} + (1 - (1 - (1 - p^f_{1\epsilon})p^f_{1\epsilon})p^p_{1\epsilon})p^f_{2\epsilon} + \cdots \\ &= \sum_{i=1}^{\infty}(p^f_{i\epsilon}\pi_i + p^p_{i\epsilon}\pi_{i+1}), \quad i = 1, 3, 5, \ldots \end{aligned}$$

where

$$\begin{aligned} \pi_1 &= (1 - p^f_{1\epsilon}), \\ \pi_2 &= (1 - \pi_1 p^f_{1\epsilon}), \\ \pi_3 &= (1 - \pi_2 p^p_{1\epsilon}), \\ \pi_4 &= (1 - \pi_3 p^f_{2\epsilon}), \\ &\vdots \\ \pi_k &= (1 - \pi_{k-1} p^f_{(k-1)\epsilon}). \end{aligned}$$

The *Probability of Not Rejected* (*PNR*) is

$$PNR = 1 - PA. \tag{20}$$

Also the *Throughput*(*THR*) is

$$THR = \lambda\, PNR \tag{21}$$

and *AbortRatio*(*AR*) is

$$AR = \lambda\, PA. \tag{22}$$

7. Performance analysis of CoTO

Due to the reordering phenomenon each transaction T_n must be waiting in a resequencing buffer for time wt_n. In general, we shall assume that each transaction needs to be ordered with respect to some subset of the preceding transactions [2,3,10,16,17]. A quite general case expresses a random association of data items and can be stated:

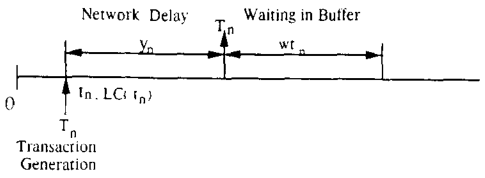


Fig. 8. The reordering issue.

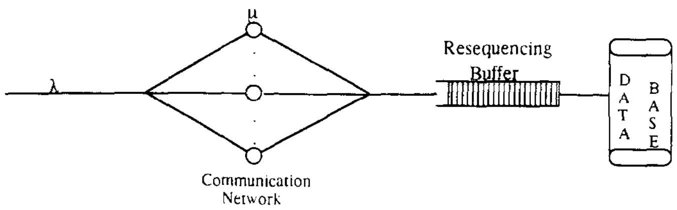


Fig. 9. The performance model.

Every transaction T_n, with generation time t_n, timestamp $LC(i, t_n)$ and network delay y_n, must respect the timestamp order with each one of the transactions which has arrived before it, with some fixed conflict probability p_c.

According to the reordering issue (Fig. 8), the PDF of random variable wt_n in equilibrium is

$$WT(x) = P(wt_n < x). \tag{23}$$

We have

$$WT(x) = \lim_{t_n \to \infty} \int_0^\infty WT(t_n, y_n, x) f(y_n)\, \mathrm{d}y, \tag{24}$$

where $WT(t_n, y_n, x)$ is the conditional probability of wt_n, given that transaction T_n arrived at time t_n and had a network delay y_n and $f(y_n)$ is the distribution of service time in the network (exponential, with rate μ).

Then

$$WT(x) = \lim_{t_n \to \infty} \int_0^\infty \sum_{k=0}^\infty \mathrm{e}^{-\lambda t_n} \frac{(\lambda t_n)^k}{k!} \sum_{j=0}^k \binom{k}{j} p_c^j (1-p_c)^{k-j} \left[\int_0^{t_n} \frac{F(z_n + y_n + x)}{t_n} \mathrm{d}z_n \right]^j f(y_n)\, \mathrm{d}y_n.$$

In order to produce the above formula we considered the probability of the event, that in the time interval $(0, t_n)$ k transactions arrive, following a Poisson process, in the Local Data Base, and all transactions that conflict with transaction T_n (j out of k transactions), leave the network before the $t_n + y_n + x$ time interval. Also we used the known property of the Poisson process, which says that given that k arrivals of the Poisson process occurred in the time interval of length t_n, then the arrival times are independent and uniformly distributed random variables in this time interval. We note that F in the above formula is the PDF of service time in network. After some algebra (Appendix B), we have

$$WT(x) = \frac{1}{\rho} \mathrm{e}^{\mu x} \left(1 - \mathrm{e}^{-\rho \mathrm{e}^{-\mu x}}\right), \quad \rho = \lambda_c / \mu. \tag{25}$$

The *Probability of No Waiting* PNW is

$$PNW = \Pr\{\text{No waiting}\} = WT(x=0) = (1/\rho)\left(1 - \mathrm{e}^{-\rho}\right). \tag{26}$$

Also *Probability of Waiting* PW is

$$PW = 1 - PNW. \tag{27}$$

The *Mean Waiting Time* (MWT) is

$$MWT = \int_0^\infty 1 - WT(x)\,\mathrm{d}x = \int_0^\infty 1 - \frac{1}{\rho}\mathrm{e}^{\mu x}\left(1 - \mathrm{e}^{\rho \mathrm{e}^{-\mu x}}\right)\mathrm{d}x = \frac{1}{\lambda_\mathrm{c}}\sum_{n=2}^{\infty}(-1)^n \frac{\rho^n}{n!(n-1)}. \tag{28}$$

The convergence and the calculation for the above improper integral can be found in Appendix C.
Finally, the *Mean Number of Transactions* (MNT) in the resequencing buffer is

$$MNT = \lambda\, MWT = \frac{1}{p_\mathrm{c}}\sum_{n=2}^{\infty}(-1)^n \frac{\rho^n}{n!(n-1)}. \tag{29}$$

In our analysis we have ignored the effect of the clock drifts, because some transactions benefit while others lose in waiting, which results in a balanced situation.

8. Numerical results and validation

In this section, we present numerical and simulation results for the performance analysis of BTO and $CoTO$. We have compared our analytical results to the results of the simulation study to validate the accuracy of our analysis. In all cases the data base size N is equal to 250 and S, A mean simulation and analysis results respectively.

The comparison of the analysis and simulation results, in the case of BTO, shows that:

- For transactions of size 2, we have high accuracy. This is also true for sizes 4 and 8. In these cases there is a difference in the results when μ is very small, 0.1–0.2. This difference is due to the fact that in these cases the transactions remain in the network for some time, since the network is slow, which, results in more frequent collisions. Another fact that results in the above difference is the value of the p_c parameter. The p_c parameter has been approximated for values of $M \ll N$.
- In all cases, for values of ϵ from 10^{-6} to 10^{-2} and 0 the performance measurements remain uninfluenced.

The comparative study of the behavior of $CoTO$ in the cases of perfect and imperfect clocks shows that:

- For transactions of size 2 and 4 we have high accuracy in all cases.
- The asynchronism of the clocks affects the performance measurements only when $\epsilon > 0.5$. But this influence is not important, since some transactions need to wait while others do not.
- Our analysis shows a deviation for small values of μ.

Tables 2 and 3 summarize our performance parameters and measures respectively. Tables 4–9 show the analytical and simulation results for BTO and $CoTO$.

Figs. 10–21 show the graphical representations for BTO and $CoTO$ respectively. These figures represent results for both the analysis and the simulation. In all graphs the horizontal axis indicates the network service rate μ and the vertical one represents our performance measures, such as PNR, THR, AR, PW, MWT and MNT.

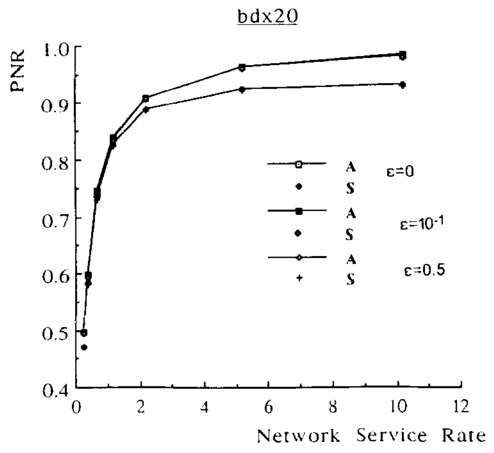


Fig. 10. BTO : PNR, $M = 4$, $\lambda = 8$.

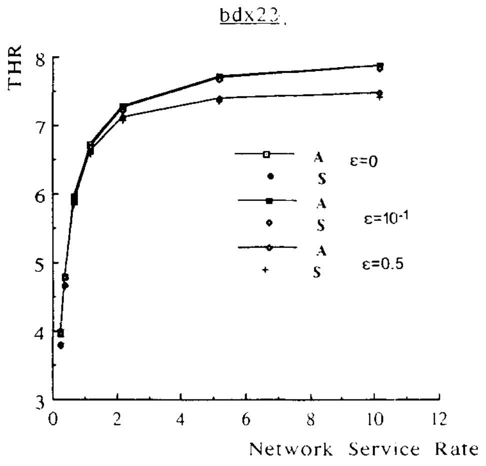


Fig. 11. BTO : THR, $M = 4$, $\lambda = 8$.

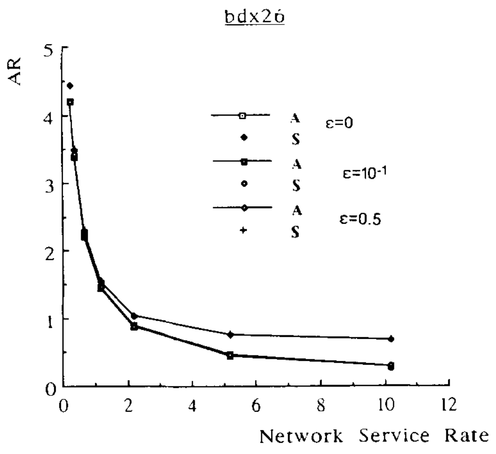


Fig. 12. BTO : AR, $M = 4$, $\lambda = 8$.

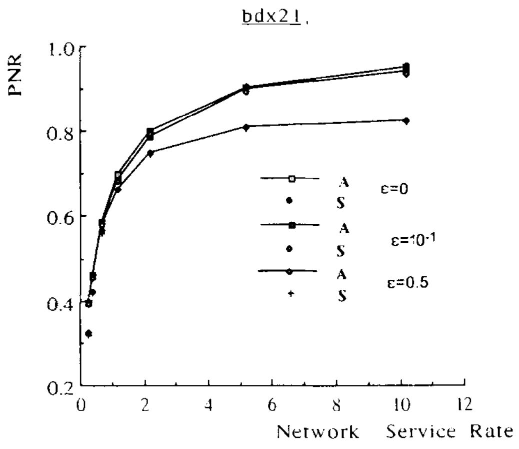


Fig. 13. BTO : PNR, $M = 8$, $\lambda = 6$.

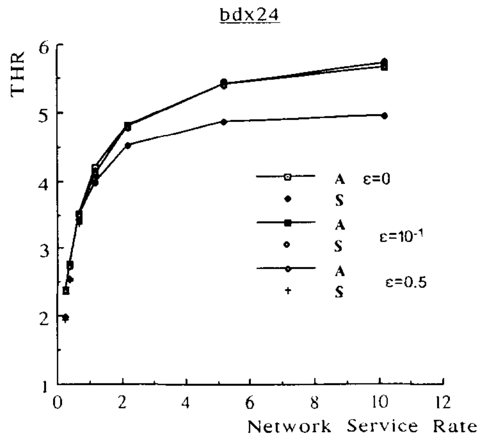


Fig. 14. BTO : THR, $M = 8$, $\lambda = 6$.

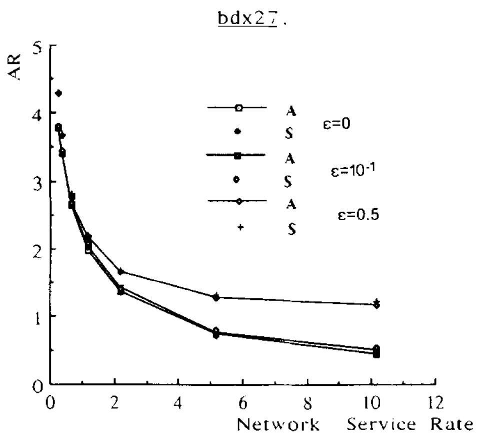


Fig. 15. BTO : AR, $M = 8$, $\lambda = 6$.

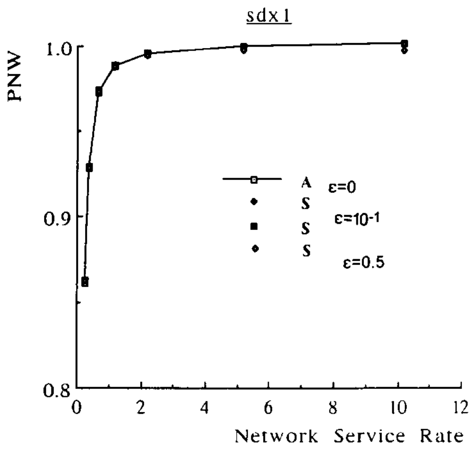


Fig. 16. $CoTO$: PNW, $M = 2$, $\lambda = 2$.

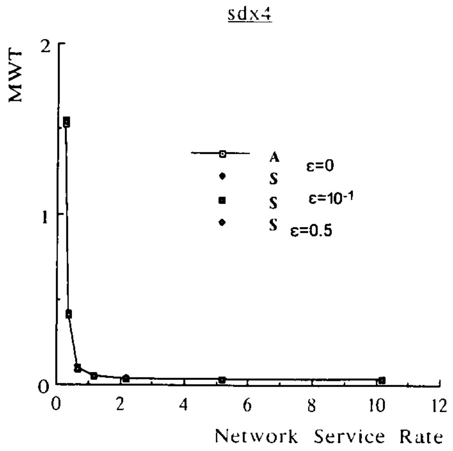


Fig. 17. $CoTO$: MWT, $M = 2$, $\lambda = 2$.

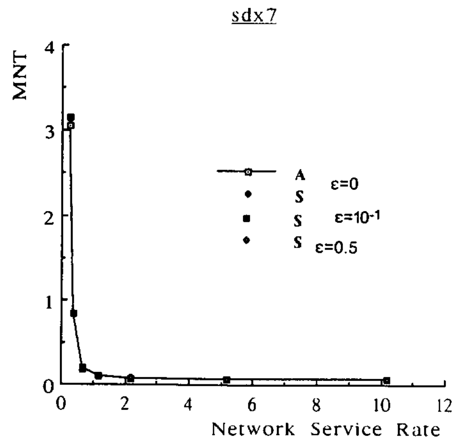


Fig. 18. $CoTO$: MNT, $M = 2$, $\lambda = 2$.

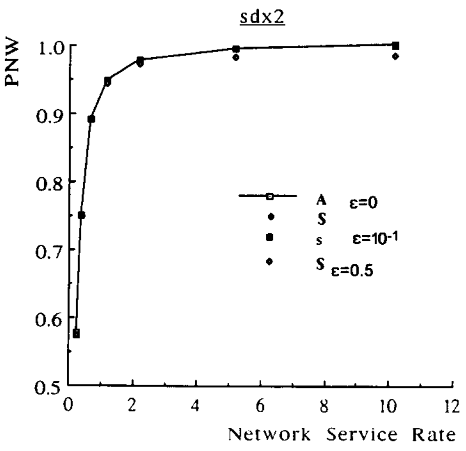


Fig. 19. $CoTO$: PNW, $M = 2$, $\lambda = 8$.

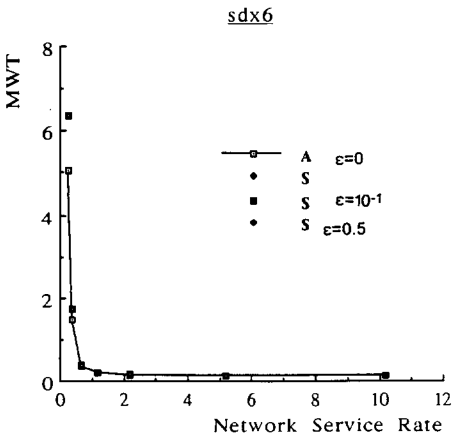


Fig. 20. $CoTO$: MWT, $M = 2$, $\lambda = 8$.

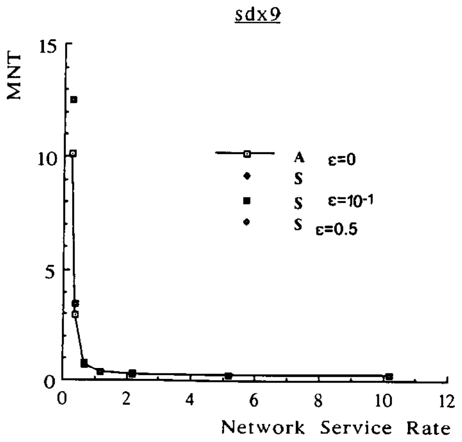


Fig. 21. $CoTO$: MNT, $M = 2$, $\lambda = 8$.

Table 2
Summary of performance parameters

N	Data base size
M	Transaction size
λ	Arrival rate
μ	Network service rate

Table 3
Summary of performance measures

PNR	Probability not rejected
THR	Throughput
AR	Abort ratio
PNW	Probability no waiting
MWT	Mean waiting time
MNT	Mean number of transactions

Table 4
BTO : PNR for $M = 8$ and $\lambda = 10$

μ	PNR							
	$\epsilon = 0$		$\epsilon = 10^{-2}$		$\epsilon = 10^{-1}$		$\epsilon = 0.5$	
	A	S	A	S	A	S	A	S
0.1	0.3523	0.2504	0.352	0.2502	0.343	0.2511	0.339	0.2496
0.2	0.3968	0.3367	0.3963	0.3366	0.3958	0.3361	0.391	0.329
0.5	0.4971	0.4724	0.4963	0.4724	0.495	0.471	0.489	0.4578
1	0.6015	0.5872	0.597	0.587	0.593	0.5855	0.575	0.557
2	0.7127	0.7059	0.71	0.7056	0.707	0.7019	0.648	0.64
5	0.8385	0.8414	0.8378	0.8409	0.826	0.8273	0.7041	0.7032
10	0.9053	0.9079	0.9047	0.9065	0.8829	0.8837	0.725	0.7243

Table 5
BTO : THR for $M = 8$ and $\lambda = 10$

μ	THR							
	$\epsilon = 0$		$\epsilon = 10^{-2}$		$\epsilon = 10^{-1}$		$\epsilon = 0.5$	
	A	S	A	S	A	S	A	S
0.1	3.523	2.5429	3.52	2.5439	3.43	2.55	3.39	2.5348
0.2	3.968	3.4193	3.963	3.4183	3.958	3.4132	3.91	3.3411
0.5	4.971	4.7465	4.963	4.7465	4.95	4.7324	4.89	4.5998
1	6.015	5.9	5.97	5.898	5.93	5.8829	5.75	5.5965
2	7.127	7.0926	7.1	7.0896	7.07	7.0525	6.48	6.4305
5	8.385	8.4512	8.378	8.4462	8.26	8.3096	7.041	7.0631
10	9.053	9.119	9.047	9.1051	8.829	8.8761	7.25	7.275

Table 6
BTO : AR for $M = 8$ and $\lambda = 10$

μ	AR							
	$\epsilon = 0$		$\epsilon = 10^{-2}$		$\epsilon = 10^{-1}$		$\epsilon = 0.5$	
	A	S	A	S	A	S	A	S
0.1	6.477	7.6125	6.48	7.6115	6.57	7.6054	6.61	7.6206
0.2	6.032	6.7361	6.037	6.7371	6.042	6.7422	6.09	6.8143
0.5	5.029	5.3011	5.037	5.3011	5.05	5.3152	5.11	5.4478
1	3.985	4.1477	4.03	4.1497	4.07	4.1647	4.25	4.4511
2	2.873	2.955	2.9	2.958	2.93	2.9952	3.52	3.6171
5	1.615	1.593	1.622	1.598	1.74	1.7346	2.959	2.9811
10	0.947	0.925	0.953	0.9391	1.171	1.1681	2.75	2.7692

Table 7
$CoTO$: PNW for $M = 4$ and $\lambda = 2$

μ	PNW			
	$\epsilon = 0$		$\epsilon = 10^{-1}$	$\epsilon = 0.5$
	A	S	S	S
0.1	0.5875	0.5647	0.5646	0.5641
0.2	0.7557	0.7386	0.7385	0.7379
0.5	0.8909	0.8805	0.8801	0.8788
1	0.9433	0.9360	0.9356	0.9329
2	0.9711	0.9694	0.969	0.964
5	0.9883	0.9878	0.986	0.977
10	0.9941	0.9939	0.9923	0.9801

Table 8
$CoTO$: MWT for $M = 4$ and $\lambda = 2$

μ	MWT			
	$\epsilon = 0$		$\epsilon = 10^{-1}$	$\epsilon = 0.5$
	A	S	S	S
0.1	4.926	6.232	6.23	6.229
0.2	1.3409	1.621	1.621	1.618
0.5	0.2267	0.2664	0.267	0.2687
1	0.0577	0.06517	0.0655	0.0688
2	0.01458	0.01614	0.00163	0.0203
5	0.00234	0.00271	0.00291	0.0073
10	0.00058	0.00074	0.00094	0.0054

9. Conclusions and future work

9.1. Conclusions

This work studied the performance of the BTO and $CoTO$ in the cases of perfect and imperfect clocks. The conclusions that we reached are:

Table 9
$CoTO$: MNT for $M = 4$ and $\lambda = 2$

μ	MNT			
	$\epsilon = 0$		$\epsilon = 10^{-1}$	$\epsilon = 0.5$
	A	S	S	S
0.1	9.852	12.29	12.289	12.3
0.2	2.682	3.199	3.196	3.198
0.5	0.453	0.523	0.523	0.53
1	0.1155	0.1282	0.1283	0.1365
2	0.02916	0.03167	0.03184	0.0407
5	0.00469	0.00538	0.0056	0.015
10	0.001175	0.00147	0.00179	0.0114

- Our approximative analytical method gives high accuracy in the study of the BTO and the $CoTO$ behavior.
- The asynchronism of the clocks affects the performance only for values of $\epsilon > 0.5$.
- Our results suffer for small values of μ, because of the appearance of difficult classical problems of queueing theory, which do not have exact solutions (in product form).
- In general we can say that for transactions that access up to and including, 1% of the data in the database, we have high accuracy.

9.2. *Future work*

This work can be expanded in two different directions, a general and a specific one.

In relation to the latter one, steps can be taken for a more realistic model of the $DDBMS$. This means a replicated $DDBMS$ and transactions that demand access to more than one location. Also, the size of the transactions should not be static. Instead there should be a mix of small, medium and large transactions. Also, the distribution of ϵ should not be uniform, but some other distribution with small deviation (e.g., normal distribution). Finally, the collision probability p_c can be defined in such a way as to express more complicated situations, as percentages of readings/record, number of common data, etc. It is also interesting to find a better approximation, which will remote the differences between analysis and simulation for small values of μ.

Important steps can also be made towards the general direction. In a distributed system the asynchronism of the clocks frequently creates the need for a new synchronization of the whole system. It is quite interesting to study situations and cases where some small asynchronism is preferred to the cost of a new synchronization. In simple words we believe that the analysis technique, we discussed in this work, offers new capabilities of approaching certain aspects of distributed systems from a different research viewpoint.

Appendix A

We shall examine, what happens for $\Pr\{E_1^i\}$ when $\epsilon \to \infty$ in the case of $i = 1$. From (19), we have

$$\Pr\{E_1\} = \left(e^{-\lambda_c 2\epsilon} + \frac{1 - e^{-\lambda_c 2\epsilon}}{4\epsilon^2}\left(2\epsilon^2 + \frac{2\epsilon}{\lambda_c} + \left(\frac{1}{\lambda_c^2} - 4\epsilon^2\right)e^{-\lambda_c 2\epsilon} - \frac{1}{\lambda_c^2}\right)\right).$$

From the above equation we get

$$\lim_{\epsilon\to\infty} \Pr\{E_1\} = \lim_{\epsilon\to\infty} \mathrm{e}^{-\lambda_c 2\epsilon} + \lim_{\epsilon\to\infty}\left[\frac{1-\mathrm{e}^{-\lambda_c 2\epsilon}}{4\epsilon^2}\left(2\epsilon^2 + \frac{2\epsilon}{\lambda_c} + \left(\frac{1}{\lambda_c^2} - 4\epsilon^2\right)\mathrm{e}^{-\lambda_c 2\epsilon} - \frac{1}{\lambda_c^2}\right)\right].$$

The first term converges to 0. So

$$\lim_{\epsilon\to\infty} \Pr\{E_1\} = \lim_{\epsilon\to\infty} \frac{1-\mathrm{e}^{-\lambda_c 2\epsilon}}{4\epsilon^2}\left(2\epsilon^2 + \frac{2\epsilon}{\lambda_c} + \left(\frac{1}{\lambda_c^2} - 4\epsilon^2\right)\mathrm{e}^{-\lambda_c 2\epsilon} - \frac{1}{\lambda_c^2}\right).$$

Now it is obvious that

$$\lim_{\epsilon\to\infty} \Pr\{E_1\} = \frac{1}{2}.$$

Using the same technique we have

$$\lim_{\epsilon\to\infty} \Pr\{E_i\} = \frac{1}{2}.$$

Appendix B

$$\begin{aligned}
WT(t_n, y_n, x) &= \sum_{k=0}^{\infty} \mathrm{e}^{-\lambda t_n}\frac{(\lambda t_n)^k}{k!}\left[p_c \int_0^{t_n} \frac{F(z_n + y_n + x)}{t_n}\,\mathrm{d}z_n + 1 - p_c\right]^k \\
&= \mathrm{e}^{-\lambda t_n}\sum_{k=0}^{\infty} (\lambda t_n)^k \left[p_c \int_0^{t_n} \frac{F(z_n + y_n + x)}{t_n}\,\mathrm{d}z_n + 1 - p_c\right]^k \Big/ k! \\
&= \mathrm{e}^{-\lambda t_n}\mathrm{e}^{\lambda t_n} e^{-\lambda t_n p_c} \exp\left[-\lambda p_c \int_0^{t_n} F(z_n + y_n + x)\,\mathrm{d}z_n\right] \\
&= \exp\left[-\lambda p_c \int_0^{t_n} 1 - F(z_n + y_n + x)\,\mathrm{d}z_n\right] \\
&= \exp\left[-\lambda p_c \int_0^{t_n+y_n+x} 1 - F(v_n)\,\mathrm{d}v_n + \lambda p_c \int_0^{y_n+x} 1 - F(v_n)\,\mathrm{d}v_n\right].
\end{aligned}$$

Since,

$$\lim_{t_n\to\infty} WT(t_n, y_n, x) = \lim_{t_n\to\infty} \exp\left[-\lambda p_c \int_0^{t_n+y_n+x} 1 - F(v_n)\,\mathrm{d}v_n + \lambda p_c \int_0^{y_n+x} 1 - F(v_n)\,\mathrm{d}v_n\right].$$

It is known that

$$\overline{y_n} = \lim_{t_n \to \infty} \int_0^{t_n} 1 - F(v_n)\,\mathrm{d}v_n,$$

where $\overline{y_n}$ is the mean of random variable y_n. Then

$$\lim_{t_n\to\infty} WT(t_n, y_n, x) = \lim_{t_n\to\infty} \exp\left[-\lambda p_c \overline{y_n} + \lambda p_c \int_0^{y_n+x} 1 - F(v_n)\,\mathrm{d}v_n\right]$$

and

$$WT(x) = \int_0^{\infty} \exp\left[-\lambda p_c \overline{y_n} + \lambda p_c \int_0^{y_n+x} 1 - F(v_n)\,\mathrm{d}v_n\right] f(y_n)\,\mathrm{d}y_n.$$

From our assumptions, we have

$$F(v_n) = 1 - \mathrm{e}^{-\mu v_n} \quad \text{and} \quad f(v_n) = \mu \mathrm{e}^{-\mu v_n}$$

and then

$$\overline{y_n} = 1/\mu$$

So,

$$WT(x) = \mathrm{e}^{-\rho} \int_0^{\infty} \exp\left[\lambda p_c \int_0^{y_n+x} \mathrm{e}^{-\mu z_n}\,\mathrm{d}z_n\right] \mu \mathrm{e}^{-\mu y_n}\,\mathrm{d}y_n = \frac{1}{\rho}\mathrm{e}^{\mu x}\left(1 - \mathrm{e}^{-\rho \mathrm{e}^{-\mu x}}\right),$$

where $\rho = \lambda p_c/\mu$.

Appendix C

C.1. Convergence

First, we shall prove that

$$\mathrm{e}^{-x} \le 1 - x + \frac{x^2}{2}, \quad x \ge 0.$$

Let

$$g(x) = \mathrm{e}^{-x} - 1 + x - \frac{x^2}{2}, \quad x \ge 0.$$

According to mean value theorem in $[0, x]$ there is $a \in (0, x)$ such that

$$g'(a) = \frac{g(x) - g(0)}{x - 0} \Rightarrow -\mathrm{e}^{-a} + 1 - a = \frac{g(x)}{x} \Rightarrow g(x) = x(-\mathrm{e}^{-a} + 1 - a) \le 0.$$

But it is known that $e^{-a} \leq a - 1$.

So, it is true that

$$e^{-x} \leq 1 - x + \frac{x^2}{2}, \quad x \geq 0.$$

After this we have

$$\int_0^\infty 1 - WT(x)\,dx = \int_0^\infty 1 - \frac{1}{\rho}e^{\mu x}\left(1 - e^{-\rho e^{-\mu x}}\right) dx$$
$$\leq \int_0^\infty 1 - \frac{1}{\rho}e^{\mu x}\left(\rho e^{-\mu x} - \frac{\rho^2}{2}e^{-2\mu x}\right) dx = \int_0^\infty \frac{\rho}{2}e^{-\mu x}\,dx = \frac{2\rho}{\mu}.$$

The above result shows the convergence of improper integral.

C.2. Calculation

It is known that

$$e^{-x} = \sum_{n=0}^{\infty}(-1)^n \frac{x^n}{n!}.$$

From this we have

$$e^{-\rho e^{-\mu x}} = \sum_{n=0}^{\infty}(-1)^n \frac{(\rho e^{-\mu x})^n}{n!} \quad \Rightarrow \quad 1 - e^{-\rho e^{-\mu x}} = \sum_{n=1}^{\infty}(-1)^{n+1} \frac{(\rho e^{-\mu x})^n}{n!}.$$

Then

$$\int_0^\infty 1 - \frac{1}{\rho}e^{\mu x}\left(1 - e^{-\rho e^{-\mu x}}\right) dx = \int_0^\infty 1 - \frac{1}{\rho}e^{\mu x}\sum_{n=1}^{\infty}(-1)^{n+1}\frac{(\rho e^{-\mu x})^n}{n!}\,dx = \frac{1}{\lambda_c}\sum_{n=2}^{\infty}(-1)^n \frac{\rho^n}{n!(n-1)}.$$

References

[1] R. Agrawal, M. Carey and M. Livny, Concurrency control performance modelling: Alternatives and implications, *ACM Trans. Database Systems* **12**(4) (1987) 609–654.

[2] S. Agrawal and R. Ramaswamy, Analysis of the resequencing delay for M/M/∞ systems, *ACM SIGMETRICS 1987*, pp. 27–35.

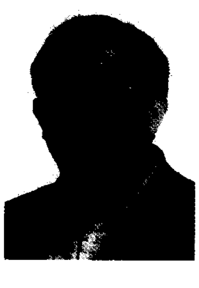
[3] F. Bacelli, E. Gelenbe and B. Plateau, An end to end approach to the resequencing problem, *J. Assoc. Comput. Mach.* **31**(3) (1984).

[4] A. Bernstein and M. Goodman, Concurrency control in distributed database systems, *Computing Survey* **13**(2) (1981) 185–221.

[5] C. Bouras and P. Spirakis, Simplified performance models of the reordering issue in timestamp ordering concurrency control in distributed databases, *ISCIS V*, Cappadocia, Turkey, 1990

[6] C. Bouras and P. Spirakis, The effect of clock drifts on the performance of distributed timestamp ordering, *3rd COMAD*, Bombay, India, 1991.

[7] C. Bouras and P. Spirakis, Performance models for perfect and imperfect clocks on timestamp ordering in distributed databases, *MASCOTS'93*, San Diego, California, January. 17–20, 1993.
[8] C. Bouras and P. Spirakis, The perfect and imperfect clocks–approach to performance analysis of basic timestamp ordering in distributed databases, *ICCI '93*, Sudbury, Canada, May 26–28, 1993.
[9] M. Carey and M. Stonebraker, The performance of concurrency control algorithms for database management systems, *Proc. 10th Int. Conf. on Very Large Data Bases*, Singapore, August 1984, pp. 107–118.
[10] W. Cellary, E. Gelenbe and T. Morzy, Concurrency control in distributed database systems, in: H. Kobayashi and M. Nivat (Eds.), *Studies in Computer Science and Artificial Intelligence*, North-Holland, Amsterdam (1988).
[11] S. Ceri and G. Pelagatti, *Distributed Databases. Principles and Systems*, McGraw-Hill, New York (1984).
[12] M. El-Toweissy, El-Makky, M. Abougabal and S. Fouad, The mean value approach to performance evaluation of the timestamp ordering algorithms, *ICCI'91*, Ottawa, Canada, pp. 276–281, Lecture Notes in Computer Science, Vol. 497, Springer, Berlin (1991).
[13] W. Feller, *An Introduction to Probability Theory and its Applications*, Vols. 1–3, 2nd ed., Wiley, New York (1971).
[14] C. Fidge, Logical time in distributed computing systems, *IEEE Computer* (August 1991) 28–33.
[15] B.I. Galler, Concurrency control performance issues, Report CSRG-147, Ph.D. Thesis, University of Toronto, 1984.
[16] F. Kamoun, M. Ben Djerad and G. LeLann, Queueing analysis of the ordering issue in a distributed database concurrency control: A general case, *Proc. 3rd Int. Conf. on Distributed Computing Systems*, 1982, pp. 447–452.
[17] F. Kamoun, L. Kleinrock and R. Muntz, Queueing analysis of the ordering issue in a distributed database concurrency control mechanism, *Proc. 2nd Int. Conf. on Distributed Computing Systems*, 1981, pp.13–23.
[18] E. Korach, G. Tel and S. Zaks, Optimal synchronization of ABD networks, Technical Report RUU CS-88-23, Utrecht.
[19] V. Li, Performance models of timestamp-ordering concurrency control algorithms in distributed databases, *IEEE Trans. Comput.* **C-36**(9) (1987) 1041–1051.
[20] J. Lundelins and N. Lynch, An upper and lower bound for clock synchronization, *Information and Control* **62**(2,3) (1984) 190–204.
[21] M. Singal, Performance analysis of the basic timestamp ordering algorithm via Markov modelling, *Performance Evaluation* **12** (1991) 17–41.
[22] M. Singal and A. Agrawala, Performance analysis of an algorithm for concurrency control in replicated database systems, *ACM SIGMETRICS 1986*, pp. 159–169.
[23] A. Stafylopatis and E. Gelenbe, Delay analysis of resequencing systems with partial ordering, *Performance'87*, pp. 433–445.
[24] P. Vitanyi, Distributed algorithms in an Archimedean ring of processors, *ACM STOC 1984*, pp. 542–547.
[25] C. Wang and V. Li, Queueing analysis of the conservative timestamp ordering concurrency control algorithm, *Proc. IEEE Int. Computing Symposium*, 1986, pp. 1450–1455.



Christos J. Bouras obtained the Ph.D. from Patras University (Greece). He is currently a research associate at Computer Technology Institute (Greece). His research interests include telematics and new services, performance analysis of computer systems and database concurrency control. He has published 11 papers in various well-known refereed conferences. He is an author of two books. He has been a referee in ACM SIGMETRICS, MASCOTS and ICCI.

Paul G. Spirakis obtained the Ph.D. from Harvard University. He is currently a full Professor in the Department of Computer Science and Engineering, Patras University (Greece). His research interests include probabilistic, parallel and distributed algorithms and protocols, telematics, exact analysis of algorithms, algorithms and complexity of graph theoretic problems, performance analysis of computer systems and database concurrency control. He has extensively published in most of the important Computer Science journals and refereed conferences including the ACM STOC, the ACM/IEEE FOCS, most of the other ACM conferences, ICALP, STACS, etc. He is an author of three books. He has edited various conference proceedings and is currently an editor of three journals. He is a member of the steering committees of SPAA and WDAG. He is a member of ACM, EATCS, and The Math. Assoc. of America.