PERFORMANCE ANALYSIS OF TWO CONCURRENCY CONTROL SCHEMES FOR DESIGN ENVIRONMENTS

Show-way Yeh, Clarence A. Ellis,
Aral Ege, Henry F. Korth

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-31 August 1987

Performance Analysis of Two Concurrency Control Schemes for Design Environments

Show-way Yeh

Department of Computer Sciences
University of Texas at Austin
and
Software Technology Program
MCC
Austin, Texas

Clarence A. Ellis

Software Technology Program
MCC
Austin, Texas

Aral Ege

Software Technology Program
MCC
Austin, Texas

Henry F. Korth

Department of Computer Sciences
University of Texas at Austin
Austin, Texas

ABSTRACT

In the design environment, the concurrency control mechanisms of conventional database systems are frequently clumsy and inadequate. In this paper, some unconventional concurrency control mechanisms, soft locks and version resolution, are described and evaluated. Both mechanisms require that certain classes of conflict be resolved by interaction of designers. This paper explains these mechanisms, constructs performance analysis models for them, and derives formulas for performance measures, such as throughput and percentage of transaction time spent for conflict resolution. The formulas are then numerically evaluated for a range of parameter values that are sensible within a design environment. Some graphs of results are presented to show the performance of the mechanisms. The mechanisms are shown to perform much better than conventional locking mechanisms within design environments.

Performance Analysis of Two Concurrency Control Schemes for Design Environments

Show-way Yeh

Department of Computer Sciences
University of Texas at Austin
and
Software Technology Program
MCC
Austin, Texas

Clarence A. Ellis

Software Technology Program
MCC
Austin, Texas

Aral Ege

Software Technology Program
MCC
Austin, Texas

Henry F. Korth

Department of Computer Sciences
University of Texas at Austin
Austin, Texas

1. Introduction

The primary thesis upon which this paper is predicated is that a large software systems design environment has significantly different database concurrency requirements than the typical data processing environment. Several schemes are proposed and analyzed in this paper. The salient characteristic of these schemes is that they all rely upon a synergistic interaction among designers to enable them to greatly enhance the degree of simultaneous activity.

1.1. The DBMS and the Design Environment

In the typical modern design environment, there is a database to store artifacts of the design project, such as specifications, programs, and data [6,7,9,10,13,17,20]. These artifacts, or the user defined modular pieces of these artifacts, are called objects in this paper. The number of objects is usually large. The requirements of a large design environment are sufficiently different from the typical commercial environment that they present interesting challenges to database technology. These requirements include facilities to handle complex dynamic relations between objects; relatively seamless interfaces between designers' personal databases and global databases; exploration and navigation; and complex long-lived transactions.

The work reported in this paper is motivated by the requirements uncovered within the Software Technology Program (STP) at the Microelectronics and Computer Technology Corporation (MCC). STP has a commitment to long-term research into the upstream aspects of software design with a goal of achieving an extraordinary improvement in software productivity. The upstream aspects of design, many of which refer to the organization and brainstorming that must occur when the ideas and problems are ill-structured, partially unarticulated, and fuzzy, also present interesting challenges to database technology. These upstream aspects imply a need for the database to handle informal as well as formal documents; fuzzy and inconsistent data; diverse non-canonical data structures such as assumptions, decisions, alternatives and justifications; and finally, teams and coordination.

Internal and external studies have been undertaken within STP to understand further the information handling needs and implications of the software design environment. An emphasis has been placed upon the databases for software design [4]. Prototypes have been built within STP (an object oriented database server [7], and a hypertext system [11]) to explore alternative solutions to some of the information handling challenges. The remainder of this paper will focus on the concurrency control aspects of the software environment. Further elaboration on the challenges, and explanation of STP work has been published elsewhere [2], and we simply note here that the range of parameter values used in this paper has been derived from information and experience collected by STP.

1.2. Concurrency Control and the Design Environment

Access to the design database is characterized by teams of designers accessing a variety of types of information. The requirements of these teams of designers violate some of the assumptions of typical database concurrency control schemes.

1. In the design environment, it is no longer true that transactions are short. Transactions are mechanisms to enforce atomicity of a sequence of database operations, and to create a unit of crash recovery and user specified undo/abortion. Software designers need these transaction mechanisms when they work on pieces (modules) of the design. These design pieces include code modules (e.g. programs,) specification modules (e.g. formal models,) and requirements modules (e.g. textual documents.) A software designer may work on a module in an update mode for hours, days, or weeks. In typical DBMSs, the standard practice is for any conflicting access to simply wait until the transaction completes. Clearly, it can be intolerable for a designer to wait for such long periods of time, so alternative strategies are needed.

2. In the large systems design environment, it is no longer true that the probability of concurrent conflicting access is extremely low. In this environment, no one person can totally create or even comprehend all details of the system. Thus, there must be reliance upon the team to collectively share knowledge, and there is a critical dependence upon collective knowledge effectively shared. Information access conflicts occur more frequently because there is a tendency for design teams to dwell upon a single aspect, theme, or problem at the same time. We have found that the check-in, check-out model hinders efficient cohesive team interaction. Modern design teams take advantage of facilities for electronic meetings, computer conferencing, etc, to expedite their work. All of this implies that database access conflicts may occur much more frequently than in standard data processing environments.

3. In the design environment, it is no longer true that most database interactions are well-defined automatic transactions. In many environments, it is well-defined (i.e. known at the initiation of the transaction) which items may be read and written. This tends not to be true in our software design environment. Debugging can lead to touching many modules; a single change can necessitate changes to many seemingly unrelated objects. Another example is a hypertext network [11]: the following of links within a hypertext network, and creation of new nodes and links is typically a dynamic process leading to a wide variety of unanticipated update operations. There are other aspects of design which are not easily automated such as the abort decision. If a designer has invested many hours in a particular design update, he or she may be strongly opposed to aborting because of conflicting updates by another designer. The designers may both be well served in this case if the system does not automatically abort or roll-back. Thus we need synergistic interactive concurrency control schemes.

These differences between the design environment and the typical environment of a DBMS suggest a need for alternative concurrency control schemes for the design database. In the next section, we describe some new concurrency control schemes which we have been exploring and contrast these with related schemes in the literature.

1.3. Interactive Concurrency Control

In this discussion we separate concurrency control policies from concurrency control mechanisms. The mechanisms provide the underlying primitives out of which different policies, such as pessimistic concurrency control and optimistic concurrency control, can be constructed. Designers then typically follow the correct policies, but do not need to be concerned with the mechanisms. As an example, locking and timestamps are mechanisms; the protocol of only locking objects when they are about to be changed is one particular policy that might work well for certain groups.

Locks are mechanisms used to constrain and control access to objects. Ordinary (hard) locks when placed on objects guarantee exclusive access. One mechanism which we analyze is the soft lock, as implemented in our Gordion [7] object base management system. A soft lock is a lock on an object that can be broken if another user chooses explicitly to do so. This potentially allows several writers to access the same object simultaneously. The soft lock is useful in a cooperative environment, and can be used to attain a high degree of shared access. A record of all of the access and locking activities of an object is kept, and can be queried to find out about recent change to the object, or about broken locks. If the soft lock mechanism is coupled with the notify mechanism, then designers can be immediately alerted when a lock on an object of interest is broken. Another mechanism is the versioning mechanism which detects attempted access to an object which is being updated. This mechanism creates another version of the object. If this mechanism is coupled with the notify mechanism, then designers can be immediately alerted when a conflicting version is created.

In this paper, we investigate policies of immediate notification, and policies of delayed notification. Our immediate notification policies assume that either soft locks or versioning is coupled with the notification mechanism so that both parties are immediately alerted when a transaction attempts to access an object held by another. Then a phone call is instigated between the two parties (perhaps with a shared window of information on their screens) and they perform conflict resolution. They need to assess the situation and discuss whether they can proceed simultaneously. If not, one or the other may need to wait or to do some repair work.

Our delayed notification policies assume that either soft locks or versioning is used, but that notification of conflicts is not transmitted to the designers until a transaction termination is attempted by one of the involved designers. At this time, the same phone call protocol is carried out. This policy represents less interruptions for designers, but it may require significantly more repair work to attain consistency. The important point is that all of these policies invoke the domain expertise of the designers, and attempt to assist them rather than force a solution to the conflict upon them. In practice, this is a fruitful flexible way to resolve conflicts within a team of cooperating designers.

1.4. Related Work

As previously mentioned, we are interested in moving beyond the check-in check-out model because there are clear cases where several designers can and should be able to work on the same objects at the same time. Other models have been proposed by Katz[13], Bancilhon[3], and others for design databases. We are particularly concerned with large software design databases. Our notion of soft locks have some similarity to proposals of others such as Zdonik [23] and Greif [10].

The bulk of this paper is concerned with performance analysis. Many papers have analyzed the performance of different locking mechanisms for conventional database systems [1,5,8,12,14,15,16,18,19,21,22] but these are not suitable for the design environment. Our model is a modification of the one presented for analysis by Tay [21,22] where transactions that have acquired i locks are assumed to be either waiting, restarted, or continuing to acquire the next lock until a fixed number of locks are acquired by the transaction which in turn designates the end of transaction. We have extended Tay's notions to model our immediate notification policies and our delayed notification policies. In the next three sections, we present our model, then our formulas and their derivation, followed by our evaluation and graphs. Finally we present a comparative analysis of the policies mentioned above with respect to a more conventional approach based on hard locks.

2. The System Model and Assumptions

In our model, we assume that a number of designers are concurrently accessing the database, and that the number of designers and level of activity is constant. A designer's activity consists of repeatedly performing a sequence of transactions interspersed with non-transaction time of random duration. Therefore,the underlying system model is a closed queuing network as shown in Figure 1. Our model allows a designer to request new locks at any time during his or her transaction, and we assume that all of the designers locks are released at the end of the transaction.

At any time during the transaction, the model allows the designer to transition into an "interaction state" during which transaction data processing is suspended, and conflict resolution is performed. In the immediate notification case, this transition occurs whenever the designer tries to write lock an object which is already locked by another designer, or another designer tries to write lock an object already locked by this designer. In the delayed notification case, this transition only occurs when one designer attempts to end (commit) a transaction while holding a write lock on an object locked by another designer. In either case, when this transition does occur, there are many possible modes of conflict resolution. We envision that it typically consists of a telephone conversation with another designer followed by alteration of some local data values and then waiting for the other designer to make some alterations. Afterwards, the progress of the transaction data processing can continue. Thus in our model, deadlock and subsequent restart is avoided.

Parameters that must be supplied to the model include the distribution function for the time between consecutive lock requests, the average conflict resolution time per object per transaction (a constant), and the probability that a transaction completes after the i-th object is locked as a function of i. It seems reasonable to expect a designer to try to place locks more frequently if he holds fewer locks, and less frequently if he holds more locks. Therefore, the distribution function for the time between two consecutive lock requests by a transaction is a function of the number of locks a transaction holds. Given these assumptions the following section presents our formulas and their derivation.

3. Formulas

3.1. Delayed Notification

Lets first analyze the case in which notification is not given until the end of the transaction. Conflicts may occur when a transaction locks an object. In this case, the system allows the conflicting accesses to occur and doesn't notify any designers who are involved in the conflicts until one of them finishes his transaction and attempts to commit.

Fig. 2 shows the model where

N_i is the stage consisting of all transactions that are running and hold i locks.

q_i is the probability that a transaction holding i locks will request an i+ 1st object.

p_i is the probability that a transaction holding i objects is finishing and has conflicts with other transactions.

W_i is the stage consisting of all transactions that are finishing, hold i locks, and are resolving conflicts.

t is the throughput (number of transactions finished in a unit of time).

The stage N_i can be decomposed into two stages: One to resolve conflicts with finishing transactions, and one to process all other aspects of the transaction (the underlying job). A transaction starts at stage N_0, that is, holding zero locks. After processing time T_0, on average it requests the first lock with probability q_0. Otherwise, it finishes. If it requests a new lock, the system will grant it. It acquires the object and enters the underlying job processing sub-stage of N_1. This procedure repeats as many times as the number of objects the transaction requests. Before the transaction finishes, at any stage except stage 0, the system may notify the designer that somebody is finishing a transaction and has conflict(s) with his transaction. Then the transaction enters the conflict resolving sub-stage and the designer spends some amount of time to resolve the conflicts. Then the transaction returns to the underlying job processing sub-stage and continues the transaction. Since all transactions in the conflict resolving sub-stage of N_i will come back to the underlying job processing sub-stage, it is not necessary to distinguish these two sub-stages. The rest of this paper will not distinguish them. When a transaction is finished at stage i, that is, it holds i objects, the system, with probability p_i, may notify the designer with whom the transaction conflicts. Then, it enters stage W_i to resolve all conflicts between them. After resolving all conflicts, the transaction finally finishes.

It is assumed that a transaction does not request more than one lock on the same object. This is reasonable because the system notifies designers of conflicts only at the end of a transaction. The system must maintain a list of locks held by each transaction and the objects on which they are held. If a transaction requests two write locks or two read locks on the same object, the system can ignore the second request because it doesn't affect anything. In this paper, it is assumed that the designers do not change locking mode in a transaction.

All the transactions are assumed to lock the object prior to access, therefore we use the terms acquiring locks and accessing objects interchangibly.

Let

n_i be the number of transactions in N_i.

w_i be the number of finishing transactions which hold i locks and are resolving conflicts.

d be the probability that a given lock is a writelock.

T_i be the average time until a transaction will request the i+ 1st lock when it holds i locks.

D be the total number of objects in the system.

s_1 be the average conflict resolving time/transaction/object in N_i for all i.

s_2 be the average conflict resolving time/transaction/object in W_i for all i.

Then, given d, T_i, D, s_1, and s_2, the following shows how to analyze the average throughput of the system, the percentage of transactions which are waiting at any time, and the percentage of transaction time spent for conflict resolution after the system reaches steady state.

Let $u_{i,j,x}$ be the probability that a transaction holding i locks and a transaction holding j locks have x objects in common. Then $u_{i,j,x}$ can be computed directly. It is the probability of choosing j objects from D objects such that x of them are chosen from the i objects and j-x objects are chosen from D-i objects. So,

$$u_{i,j,x} = \frac{C(i,x)\,C(D-i,j-x)}{C(D,j)}$$

$$where \quad C(a,b) = \frac{a!}{b!(a-b)!}$$

Since all transactions check in objects at the end of the transactions, there are not any conflicts between any two transactions in N_i and in N_j for all i and j. Since all conflicts of a finishing transaction are assumed to be resolved right after the designers are notified by the system, there are not any conflicts between any two transactions in W_i and W_j for all i and j, either. Let $r_{i,j}$ be the expected number of conflicting objects between a transaction, X, in N_i and a finishing transaction, Y, in W_j. Since conflicts only occur when a transaction write locks an object which is held by others, each of the $r_{i,j}$ conflicting objects is write locked by either X or Y. Since d is the probability that a given lock is write lock, among the x objects locked by both X and Y, d*x objects are write locked by X. Similarly, d*x objects are write locked by Y. The number of objects write locked by both X and Y is d^2*x. Therefore, the number of objects which are locked by both X and Y and are write locked by either X or Y is $2d-d^2$*x. That is,

$$r_{i,j} = \sum_{x=1}^{min[i,j]} x u_{i,j,x}(2d-d^2)$$

By the Little's formula, the throughput at stage N_i is $\frac{n_i}{T_i}$. Similarly, the throughput at stage N_{i-1} is $\frac{n_{i-1}}{T_{i-1}}$. Since q_{i-1} of the $\frac{n_{i-1}}{T_{i-1}}$ transactions will flow into N_i in a unit of time,

$$\frac{n_i}{T_i} = q_{i-1}\frac{n_{i-1}}{T_{i-1}}$$

Similarly,

$$\frac{n_{i-1}}{T_{i-1}} = q_{i-2}\frac{n_{i-2}}{T_{i-2}}$$

Therefore,

$$\frac{n_i}{T_i} = t\prod_{j=0}^{i-1} q_j = tQ_{i-1} \qquad (1)$$

$$where \quad i>0 \;, \quad Q_i = \prod_{j=0}^{i} q_j \;\text{ and }\; \sum_{i=1}^{D}\left(\prod_{j=0}^{i-1} q_j\right)(1-q_i)+(1-q_0) = 1$$

For simplicity we assumed that the conflict solving does not affect the designers' underlying design work. Let

$$T_i = T_i' + T_i''$$

Where T_i' is the processing time of the underlying design work and T_i'' is the conflict resolution time between requesting the i-th and the i+1st locks of a transaction. T_i' is either given or estimated. We compute T_i'' by first determining the sum over j of the expected number of conflicts between a transaction in N_i with a transaction in W_j and then multiplying it by the average time it takes to resolve a conflict.

$$T_i'' = \sum_{j=1}^{D} s_1 r_{i,j} w_j \qquad where \;\; w_0 = 0$$

$$T_i = T_i' + \sum_{j=1}^{D} s_1 r_{i,j} w_j \qquad (2)$$

Note that a transaction in stage N_i can not get conflicts with any transaction in N_j for all j.

When a transaction has just started, it does not hold any locks. For steady state, the throughput of stage N_0 is the system throughput.

$$t = \frac{n_0}{T_0} = \frac{n_0}{T_0'} \qquad (3)$$

Substitute equations (2) and (3) into (1), the number of transactions in stage N_i is

$$n_i = n_0 Q_{i-1}\left(\frac{T_i'}{T_0'} + \frac{s_1}{T_0'}\sum_{j=1}^{D} r_{i,j} w_j\right) \qquad where \;\; i>0 \qquad (4)$$

The probability that an object is read locked by two transaction is $(1-d)^2$. That is, $1-2d+d^2$. The probability that two transactions don't conflict with each other when there are x objects are locked by both transactions is (1-2d+ d^2)^s. Let $y_{i,j}$ be the probability that two transactions hold i and j locks respectively and do not conflict with each other. Then $y_{i,j}$ can be computed directly as follows:

$$y_{i,j} = \sum_{s=0}^{min[i,j]} (1-2d+d^2)^s u_{i,j,s} \tag{5}$$

Then, p_i is as follows:

$$p_i = 1 - \prod_{j=1}^{D} y_{i,j}^{\;n_j} \tag{6}$$

Substitute (4) into (6),

$$p_i = 1 - \prod_{j=1}^{D} y_{i,j}^{\;n_0 Q_{j-1}\left(\frac{T'_j}{T'_0} + \frac{s_1}{T'_0}\sum_{l=1}^{D} r_{j,l} w_l\right)} \tag{7}$$

Then, consider the throughput in stage W_i. That is, the throughput of resolving conflicts. Let R_i be the average conflict resolution time in stage W_i. Then, by Little's formula, the throughput in stage W_i is $\frac{w_i}{R_i}$.

From Fig.2, it is easy to see that

$$\frac{w_i}{R_i} = \frac{n_i}{T_i}(1-q_i)p_i \tag{8}$$

Substitute (1) and (3) into (8),

$$\frac{w_i}{R_i} = tQ_{i-1}(1-q_i)p_i$$

$$= \frac{n_0}{T'_0} Q_{i-1}(1-q_i)p_i$$

So,

$$w_i = \frac{n_0}{T'_0} Q_{i-1}(1-q_i)p_i R_i \tag{9}$$

Similar with T''_i, R_i is

$$R_i = s_2 \sum_{j=1}^{D} r_{j,i} n_j \tag{10}$$

Substitute (4) into (10),

$$R_i = s_2 \sum_{j=1}^{D} r_{j,i} n_0 Q_{j-1} \left(\frac{T_j'}{T_0'} + \frac{s_1}{T_0'} \sum_{l=1}^{D} r_{j,l} w_l \right) \tag{11}$$

Note that a transaction in W_i will not have conflicts with any transactions in W_j for all j. Substitute (11) into (9),

$$w_i = \frac{n_0^2}{T_0'^2} Q_{i-1} p_i (1 - q_i) s_2 \sum_{j=1}^{D} r_{j,i} Q_{j-1} \left(T'_j + s_1 \sum_{l=1}^{D} r_{j,l} w_l \right)$$

Change summation base, then

$$\begin{aligned} w_i &= \frac{n_0^2}{T_0'^2} Q_{i-1} p_i (1 - q_i) s_2 \left[\sum_{j=1}^{D} r_{j,i} Q_{j-1} T'_j + \sum_{j=1}^{D} s_1 \left(\sum_{l=1}^{D} r_{l,i} Q_{l-1} r_{l,j} \right) w_j \right] \\ &= f_i \left(a_i + \sum_{j=1}^{D} c_j w_j \right) \end{aligned} \tag{12}$$

where

$$f_i = \frac{n_0^2}{T_0'^2} Q_{i-1} p_i (1 - q_i) s_2 \tag{13}$$

$$a_i = \sum_{j=1}^{D} r_{j,i} Q_{j-1} T'_j \tag{14}$$

$$c_j = s_1 \left(\sum_{l=1}^{D} r_{l,i} Q_{l-1} r_{l,j} \right) \tag{15}$$

To solve w_i in terms of f_i, a_i, and c_j, we have

$$\sum_{j=1}^{D} c_j w_j = \frac{w_i}{f_i} - a_i = \frac{w_j}{f_j} - a_j \tag{16}$$

$$w_j = \left(\frac{w_i}{f_i} - a_i + a_j \right) f_j \tag{17}$$

Substitute (17) into (16), we have

$$\sum_{j=1}^{D} c_j \left(\frac{w_i}{f_i} - a_i + a_j \right) f_j = \frac{w_i}{f_i} \sum_{j=1}^{D} c_j f_j + \sum_{j=1}^{D} c_j f_j (a_j - a_i) = \frac{w_i}{f_i} - a_i$$

So,

$$w_i = f_i \frac{\sum_{j=1}^{D} c_j f_j (a_j - a_i) + a_i}{1 - \sum_{j=1}^{D} c_j f_j} \tag{18}$$

Equations (5), (7), (13), (14), (15), and (18), form 2D equations with 2D+1 variables, p_i, w_i, and n_0. From (5) and (7), p_i are strictly increasing functions of n_0 and w_j for all j. When w_j are 0 for all j,

$$p_i = 1 - \prod_{j=1}^{D} y_{i,j}^{\, n_0 Q_{j-1} \left(\frac{T'_j}{T'_0} \right)}$$

When w_j are infinite for all j, p_i are 1 for all i. From (13), (14), (15), and (18), w_i are strictly increasing functions of n_0 and p_j for all j. When p_j are 0 for all j, w_i are 0 for all i. When p_j are 1 for all j,

$$w_i = f'_i \frac{\sum_{j=1}^{D} c_j f'_j (a_j - a_i) + a_i}{1 - \sum_{j=1}^{D} c_j f'_j} \tag{19}$$

where

$$f'_i = \frac{n_0^2}{T_0'^2} Q_{i-1} (1 - q_i) s_2$$

w_i are not infinite for all i. So, given an n_0, there is a unique solution. We can compute w_i and p_i for all i iteratively. First let all p_j be 1 and substitute them into (13). From (13), (14), (15), and (18), w_i can be computed for all i. The computed w_i are substituted into (7). From (5) and (7), new p_j are computed for all j. The new p_j are substituted into (13). This procedure repeats until the solution is obtained. Substitute n_0 and all w_i into (4), The summation of w_i for $0<i<=D$, W, is the average number of transactions in the waiting stages. The summation of n_i for $0<=i<=D$ and W is the number of transactions in the system, N. The percentage of waiting transactions is W/N. Substitute w_i into (2), T''_i for $0<i<=D$ can be obtained. Substitute n_i into (10), R_i can be obtained. Let R be the summation of T''_i times n_i and R_i times w_i for all i. Then, the quotient that R is divided by the summation of R and T'_i times n_i for all i is the average percentage of time that a designer spends for conflict resolving in a transaction. Given another n_0, another set of solution can be obtained. So, graphs of the performance (throughput) can be drawn as a function of number of transactions in the system, N, or other parameters.

3.2. Immediate Notification

This model is the same as the previous model except that the system will notify the designers of conflicts as soon as the conflicts occur instead of postponing notification until a transaction finishes. Since it is assumed that a transaction requests locks one by one, when a transaction requests a new lock, only the new lock may conflict with others. Also, there can be at most one conflicting object between any two transactions at a given time.

Fig. 3 shows the model. All parameters in Fig. 3 have similar meaning to those in Fig. 2. W_i are stages in which transactions are resolving conflicts when they request the i-th lock and conflict with other transactions. p_i are the probabilities that a transaction will conflict with other transactions when it requests the i-th lock. The internal composition of N_i is the same as in the previous model.

Since the system grants all requests, if a transaction in N_i requests a new lock and conflicts with others, the system will grant the request and the transaction will enter W_{i+1} to resolve the conflicts. A transaction starts at stage N_0, that is, holding zero locks. After processing time T_0, on average, it requests the first lock with probability q_0. Otherwise, it finishes. If it requests a new lock, which it does with probability p_1, the system will notify the designer of the transaction with which his transaction conflicts. Then his transaction enters waiting stage W_1 and spends some amount of time to resolve the conflicts. Then it enters stage N_1. If the transaction does not conflict with others, the system will grant the lock immediately and the transaction enters stage N_1. This procedure repeats as many times as the number of objects the transaction requests. Then, the transaction finishes.

Similar to the previous model, the database needs to record the locks held by each transaction and the objects on which they are held. We again require that each object can be locked at most once by a transaction. The maximum number of locks a transaction can request is the number of objects in the system.

Let n_i, w_i, d, T_i, D, s_1, s_2, $u_{i,j,x}$ be the same variables as the previous model. Recall that the probability that x objects are locked by two transactions holding i and j locks respectively is as follows:

$$u_{i,j,x} = \frac{C(i,x)\,C(D-i,j-x)}{C(D,j)}$$

Since the conflicts are assumed to be resolved right after the designers are notified, there is at most one conflicting lock between any two transactions at any time. But, a transaction may conflict with several transactions when it requests a new lock. Recall that the probability that two transactions conflict on one object is $2d-d^2$. The probability that a transaction holding j objects requests a lock on an object which is locked by a transaction holding i objects is the probability of choosing one object from the D-j objects it has not locked such that the choice is one of the objects held by the other transaction. Since there are only i-x objects that can be chosen if x denotes the number of objects held by both transactions, the probabilty is $\frac{i-x}{D-j}$. Let $m_{i,j}$ be the probability that a transaction holding i objects gets conflict due to that a transaction holding j objects requests a new lock. Then,

$$m_{i,j} = \sum_{x=0}^{min[i-1,j]} u_{i,j,x} \frac{i-x}{D-j}(2d-d^2)$$

$$= \frac{i}{D}(2d - d^2)$$

$$= m_i$$

So, when a transaction requests a new lock, the probability that it conflicts with other transactions is dependent on how many locks are held by other transactions. It is independent on how many locks are held by the requesting transaction.

Define R_i as before. Since a transaction in W_i may have conflicts with transactions in N_j and in W_k for some j and k and it will not have conflicts with itself, the average conflict resolution time of a transaction in W_i is

$$R_i = s_2 \sum_{j=1}^{D} m_j (n_j + w_j) - s_2 m_i \tag{20}$$

By the Little's formula, similar with the previous model, the throughput at stage N_i is

$$\frac{n_i}{T_i} = t \prod_{j=0}^{i-1} q_j = tQ_{i-1}$$

$$n_i = tQ_{i-1} T_i \tag{21}$$

where $Q_i = \prod_{j=0}^{i} q_j$, $\sum_{i=1}^{D} \left(\prod_{j=0}^{i-1} q_j \right) (1 - q_i) + (1 - q_0) = 1$, and $q_D = 1$

Again, assume that the conflict resolving does not affect the designers' underlying design work. Then

$$T_i = T_i' + T_i'' \qquad 0 \leq i \leq D$$

Since any two transactions in any stages N_i and N_j will not conflict with each other for all i and j, a transaction in N_i can conflict only with transactions in W_j for some j. When a transaction in W_i is resolving conflicts, it is holding i locks. So, the expected value of T_i'' is

$$T_i'' = s_1 m_i \sum_{j=1}^{D} w_j \qquad 1 \leq i \leq D, T_0'' = 0$$

$$T_i = T_i' + s_1 m_i \sum_{j=1}^{D} w_j \qquad 1 \leq i \leq D \tag{22}$$

When a transaction has just started, it does not hold any objects. For steady state, the throughput of stage N_0 is the system throughput.

$$t = \frac{n_0}{T_0} = \frac{n_0}{T_0'} \qquad (23)$$

Substitute equations (22) and (23) into (21), the number of transactions in stage N_i is

$$n_i = \frac{n_0}{T_0'} Q_{i-1} \left(T_i' + s_1 m_i \sum_{j=1}^{D} w_j \right) \qquad (24)$$

When a transaction in N_i requests a new lock, the lock may be held by some transactions in N_j and in W_k for some j and k. So, a transaction in W_i may have conflicts with some transactions in N_j or in W_k. But, it will not have conflicts with itself. This is different from the previous model. So, p_i is as follows:

$$p_i = 1 - \prod_{j=1}^{D} \left(1 - m_j\right)^{n_j + w_j} / (1 - m_i) \qquad 1 \le i \le D \qquad (25)$$

Substitute (24) into (25), we get

$$p_i = 1 - \prod_{j=1}^{D} (1 - m_j)^{w_j + \frac{n_0}{T_0'} Q_{j-1} \left(T_j' + s_1 m_j \sum_{l=1}^{D} w_l \right)} / (1 - m_i) \qquad (26)$$

Then, consider the throughput in stage W_i. That is, the throughput of resolving conflicts. By Little's formula, the throughput in stage W_i is $\frac{w_i}{R_i}$.

From Fig.3, it's easy to see that

$$\frac{w_i}{R_i} = \frac{n_{i-1}}{T_{i-1}} q_{i-1} p_i = \frac{n_0}{T_0'} Q_{i-1} p_i \qquad 1 \le i \le D$$

$$w_i = \frac{n_0}{T_0'} Q_{i-1} p_i R_i \qquad 1 \le i \le D \qquad (27)$$

Substitute (24) into (20), we have

$$R_i = s_2 \sum_{j=1}^{D} m_j \left(w_j + \frac{n_0}{T_0'} Q_{j-1} \left(T_j' + s_1 m_j \sum_{l=1}^{D} w_l \right) \right) - s_2 m_i \qquad 1 \le i \le D \qquad (28)$$

Substitute (28) into (27), we have

$$w_i = \frac{n_0}{T_0'} Q_{i-1} p_i s_2 \left(\sum_{j=1}^{D} m_j \left(w_j + \frac{n_0}{T_0'} Q_{j-1} \left(T_j' + s_1 m_j \sum_{l=1}^{D} w_l \right) \right) - m_i \right)$$

Separate the single and the nested summation loops and change the summation bases of the nested summation loop, we have

$$w_i = \frac{n_0}{T_0'} Q_{i-1} p_i s_2 \left[\left(\sum_{j=1}^{D} m_j \frac{n_0}{T_0'} Q_{j-1} T_j' - m_i \right) + \sum_{j=1}^{D} \left(s_1 \frac{n_0}{T_0'} \left(\sum_{l=1}^{D} m_l^2 Q_{l-1} \right) + m_j \right) w_j \right]$$

$$= f_i \left(a_i + \sum_{j=1}^{D} c_j w_j \right) \qquad 1 \leq i \leq D \tag{29}$$

where

$$f_i = \frac{n_0}{T_0'} Q_{i-1} p_i s_2 \tag{30}$$

$$a_i = \frac{n_0}{T_0'} \sum_{j=1}^{D} m_j Q_{j-1} T_j' - m_i \tag{31}$$

$$c_j = s_1 \frac{n_0}{T_0'} \sum_{l=1}^{D} m_l^2 Q_{l-1} + m_j \tag{32}$$

As in the previous section, w_i can be solved in terms of f_i, a_i, and c_j.

$$w_i = f_i \frac{\sum_{j=1}^{D} c_j f_j (a_j - a_i) + a_i}{1 - \sum_{j=1}^{D} c_j f_j} \tag{33}$$

Equation (33) has the same format as equation (18). But, f_i, a_i, and c_j are different for the two models.

As in the previous section, equations (26), (30), (31), (32), and (33), form 2D equations with 2D+1 variables, p_i, w_i, and n_0. From (26), p_i are strictly increasing functions of n_0 and w_j for all j. When w_j are 0 for all j,

$$p_i = 1 - \prod_{j=1}^{D} (1 - m_j)^{\frac{n_0}{T_0'} Q_{j-1} T_j'}$$

$0 < p_i < 1$ for all $i > 0$ if $n_0 > 0$. When w_j are infinite for all j, p_i are 1 for all i.

From (30), (31), (32), and (33), w_i are strictly increasing functions of n_0 and p_j for all j. When p_j are 0 for all j, w_i are 0 for all i. When p_j are 1 for all j,

$$w_i = f_i' \frac{\sum_{j=1}^{D} c_j f_j' (a_j - a_i) + a_i}{1 - \sum_{j=1}^{D} c_j f_j'}$$

where

$$f_i' = \frac{n_0}{T_0'} Q_i s_2$$

w_i are not infinite for all i. So, given an n_0, there is a unique solution. Similar with the previous section, all the performance functions can be solved in terms of the number of transactions in the system iteratively.

4. The Analytical Results

There are many parameters in the performance equations. Given our STP experience and intuition about the design environment, we have selected a range of typical parameter values, and numerically evaluated our formulas for throughput and conflict resolution time. We have also compared our results with those published in the paper of Tay [21].

The number of objects in the system is assumed to be 2000. Assume that the number of objects a transaction will request is the same for all transactions. The underlying job processing time at each stage is assumed to be constant, 1 unit of time. The conflict resolution time is measured as percentage of the underlying job processing time at each stage.

Fig. 4 to Fig. 9 are the analytical result of the first model, in which the system notifies the designers of conflicts at the end of the transactions. Fig. 10 to Fig. 15 are those of the second model, in which the system notifies the designers as soon as a conflict occurs.

Fig. 4 and Fig. 10 show the throughput, number of transactions finished per unit of time, in terms of the number of transactions in the system. Three curves represent the throughput when the conflict resolution time of each conflict are 0.1%, 1%, and 10% of the underlying job processing time in each stage respectively while the number of objects all transactions will request is assumed to be 8 and the probability that a requested object is updated is 10% Obviously the system will be saturated if the number of transactions in the system is large enough. When the conflict resolution time is small compared with the underlying job processing time, the saturation point will occur when the number of transactions in the system is huge. When the conflict resolution time is large compared with the underlying job processing time, say 10%, the saturation point will occur when the number of transactions in the system is small. As shown in Fig. 7 and Fig. 13, the percentage of time spent on conflict resolution reveals a logarithmic behavior as a function of number of transactions in the system. Fig. 7 and Fig. also show that, when the number of transactions in the system is small, the percentage of time spent on conflict resolution is exponential as a function of conflict resolution time (s_1 and s_2).

Fig. 5 and Fig. 11 show the throughput for different values for the probability that an object access is an update while the conflict resolution time is 1% of the underlying job processing time. They show that the higher d is, the smaller the number of transactions needed to saturate the system There is a limitation throughput for each d. The corresponding percentages of time spent on conflict resolution are shown in Fig. 8 and Fig. 14.

Fig. 6 and Fig. 12 show the throughput for different number of objects requested. Since the underlying job processing time at each stage is one unit of time, the transactions requesting more objects will spend more time on the underlying job. However, if the number of transactions in the system of lower j is doubled, the performance is still much better than the performance when j is doubled because the system with higher j will get saturated much easier than the system with lower j. The corresponding percentages of time spent on conflict resolution are shown in Fig. 9 and Fig. 15.

The results are compared with the conventional approach modeled by Tay [21]. Since Tay's paper has a slightly different model, the comparison needs to be explained. Fig. 16 shows the comparison. Locks in the Tay model are analogous to objects (which are shared rather than locked) in our model. The parameters are chosen to be the same. The solid lines are drawn by Tay [21]. The dashed lines are added and represent the performance of the models we are analyzing.

In Tay's model, the saturation occurs when the number of transactions in the system is around 60. In both models of this paper, it is far below the saturation point. The waiting time, R, in Tay's model is the time to wait for conflict resolution. Transactions that are holding locks are not affected by any other transactions. But, for the models in this paper, the transactions holding objects may be interrupted by other transactions at any time. So, the conflict resolution time should be doubled to be compared with Tay's waiting time. However, even if it is doubled, the conflict resolution time is still negligible.

5. Summary and Conclusions

The emphasis in conventional approach to concurrecy control is on consistency which is accomplished by locking out (or serializing) the transactions for shared objects. The models presented in this paper differs from the conventional approach in the sense that they do not enforce serialization of transactions, and the basis of consistency is notification and conflict resolution. The choice between the two approaches depends on the application environment. However, serializability will induce nested waiting. In a design environment, where the transactions are of long duration, nested waiting will have a significant negative effect upon the performance of the database system. The performance of the models presented in this paper is better than the conventional approach modeled by Tay [21]. Primarily, the reason is the avoidance of nested waiting for the queries of the waiting transactions. Recall that the figures showing the percentage of time spent on conflict resolution indicate that when the number of transactions in the system increases, the system more rapidly reaches the point where each transaction will spend substantial amount of time for conflict resolution.

Incorporating the human element in the conflict resolution loop is a key factor to enhanced concurrency. But this factor can also degrade the consistency level of the database since people may make errors. This implies that these mechanisms should be used in conjunction with consistency checking software which is interactive, intelligent, and user friendly. The system can ask questions of the designers, and may lead them through a resolution protocol which the designers can ignore or alter if necessary. Thus the combination of human resolution capability coupled with automated assistance for efficiency and enhanced consistency yields promise of an enhanced software environment. This environment will conceivably be a shared data environment with sufficient flexibility to allow software designers to have efficient access to large volumes of data and also have their needed high degree of simultaneous access.

References

1. R. Agraval, M. J. Carey, and M. Livny, "Models for Studying Concurrency Control Performance: Alternatives and Implications," Proceedings of the ACM SIGMOD International Conference on Management of Data, May 1985.

2. L. Belady, "Planning the Revolution in Software," IEEE Software, Nov 1985, pp. 68-73

3. F. Bancilhon, W. Kim, and H. F. Korth, "A Model of CAD Transactions," Proceedings of the International Conference on VLDB, August 1985.

4. T. Biggerstaff, et al, "Information Management Challanges in the Software Design Process," IEEE Database Engineering, Vol. 10, No. 1, March 1987.

5. C. Devor, and C. R. Carlson, "Structural Locking Mechanisms and Their Effect on Database Management System Performance," Inform. Systems, Vol. 7, No. 4, 1982.

6. J. Donahue, and J. Widom, "Whiteboards: A Graphical Database Tool," ACM Trans. on Office Information Systems 4, 1, Jan. 1986.

7. A.Ege, C. A. Ellis, "Design and Implementation of GORDION, an Object Base Management System," Proceedings of the Third Int. Conf. on Data Engineerig, 1987, pp. 226-234.

8. B. I. Galler, and L. Bos, "A Model of Transaction Blocking in Databases," Performance Evaluation 3, 1983, Elsevier Science Publishers B. V.

9. D. K. Giffórd, "Violet, an Experimental Decentralized System," Computer Networks 5, 6, 1981.

10. I. Greif, and S. Sarin, "Data Sharing in Group Work," Proceedings of the Computer-Supported Cooperative Work, Dec. 1986, Austin, Texas.

11. E. Gullichsen, D. D'Souza, P. Lincoln, K. C. The, "The PlaneText Book," MCC Internal Tech Report No. STP-333-86, Austin, TX, 1986.

12. K. B. Irani, and H.-L. Lin, "Queueing Network Models for Concurrent Transaction Processing in a Database System," Proceedings of the ACM SIGMOD International Conference on Management of Data, Jan. 1979.

13. R. H. Katz, M. Anwarrudin, and E. Chang, "A Version Server for Computer Aided Design Data," Proceedings of the 23rd ACM/IEEE Design Automation Conference, 1986.

14. S. S. Lavenberg, "A Simple Analysis of Exclusive and Shared Lock Contention in a Database System," Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Aug. 1984.

15. D. Mitra, "Probabilitic Models and Asymptotic Results for Concurrent Processing with Exclusive and Non-Exclusive Locks," SIAM J. Comput., Vol. 15, No. 4, Nov. 1985.

16. D. Potier, and Ph. Leblanc, "Analysis of Locking Policies in Database Management Systems," CACM, Vol. 23, No. 10, Oct. 1980.

17. S. Sarin, and I. Greif, "Computer Based Real Time Conferences," IEEE Computer 18, 10, Oct. 1985.

18. K. C. Sevcik, "Comparison of Concurrency Control Methods Using Analytic Models," Info. Proc. 83, R.E.A. Mason.

19. A. W. Shum, and P. G. Spirakis, "Performance Analysis of Concurrency Control Methods in Database Systems," Performance '81, North-Holland Publishing Company,

1981.

20. A. H. Skarra, S. B. Zdonik, and S. P. Reiss, "An Object server for an Object Oriented Database System," Proceedings of Int. Workshop on Object Oriented Database Systems, Sep. 1986.

21. Y. C. Tay, N. Goodman, and R. Suri, "Locking Performance in Centralized Databases," ACM Transactions on Database Systems, Vol. 10, No. 4, Dec. 1985.

22. Y. C. Tay, R. Suri, and N. Goodman, "A Mean Value Performance Model for Locking in Databases: The No-Waiting Case," JACM, Vol. 32, No. 3, July 1985.

23. S. Zdonik "Object Management Systems for Design environments," Database Engineering, Vol. 8, No.4, Dec. 1985, pp. 23-21.

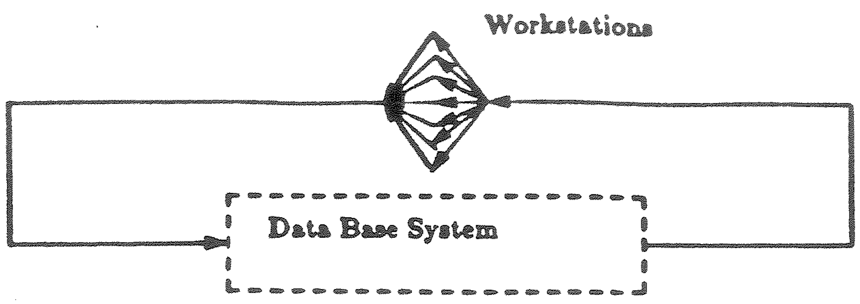

Fig. 1. The system diagram

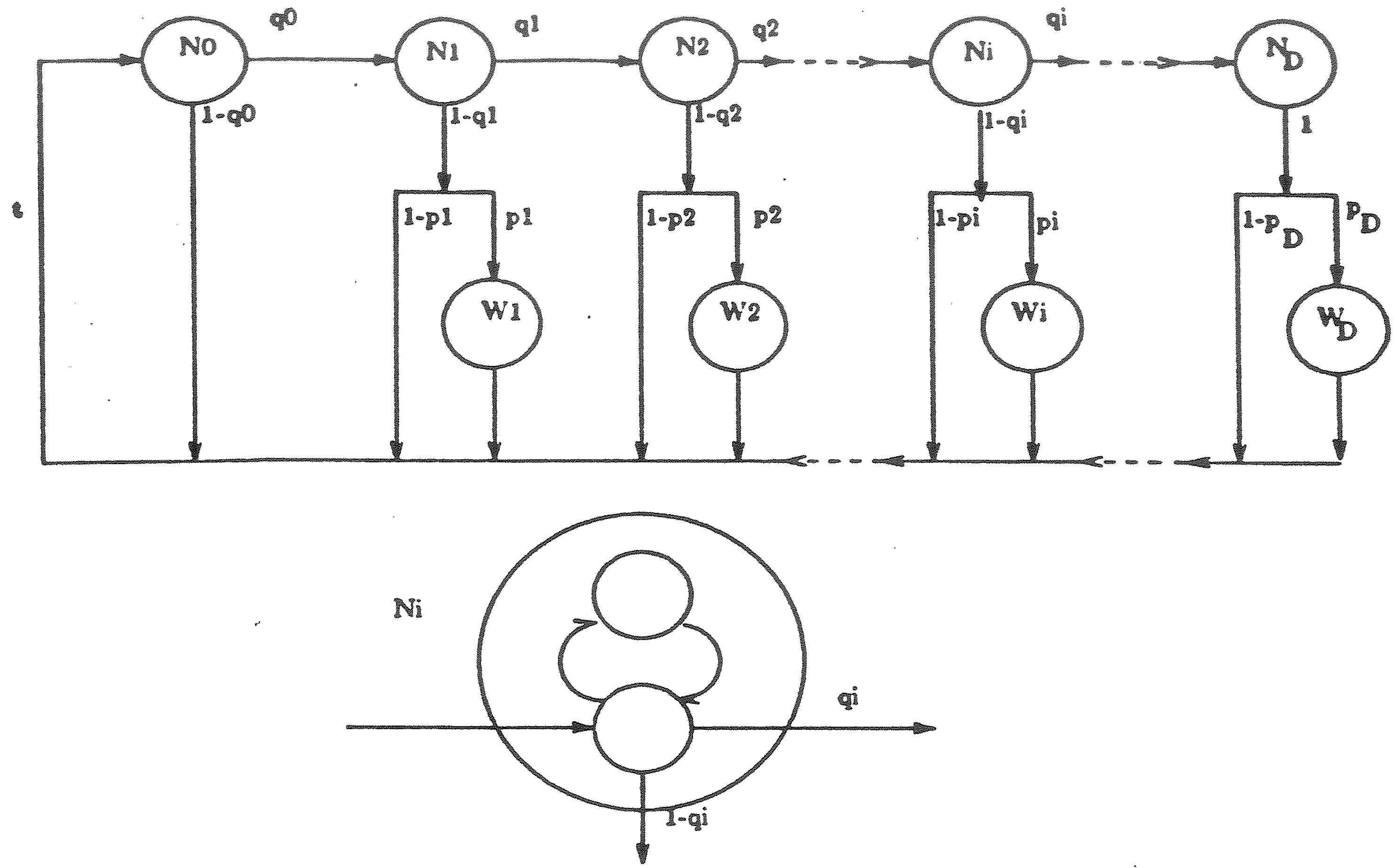

Fig. 2 The model to notify designers conflicts at the end of the transactions

$p0=0$, $q_D=0$

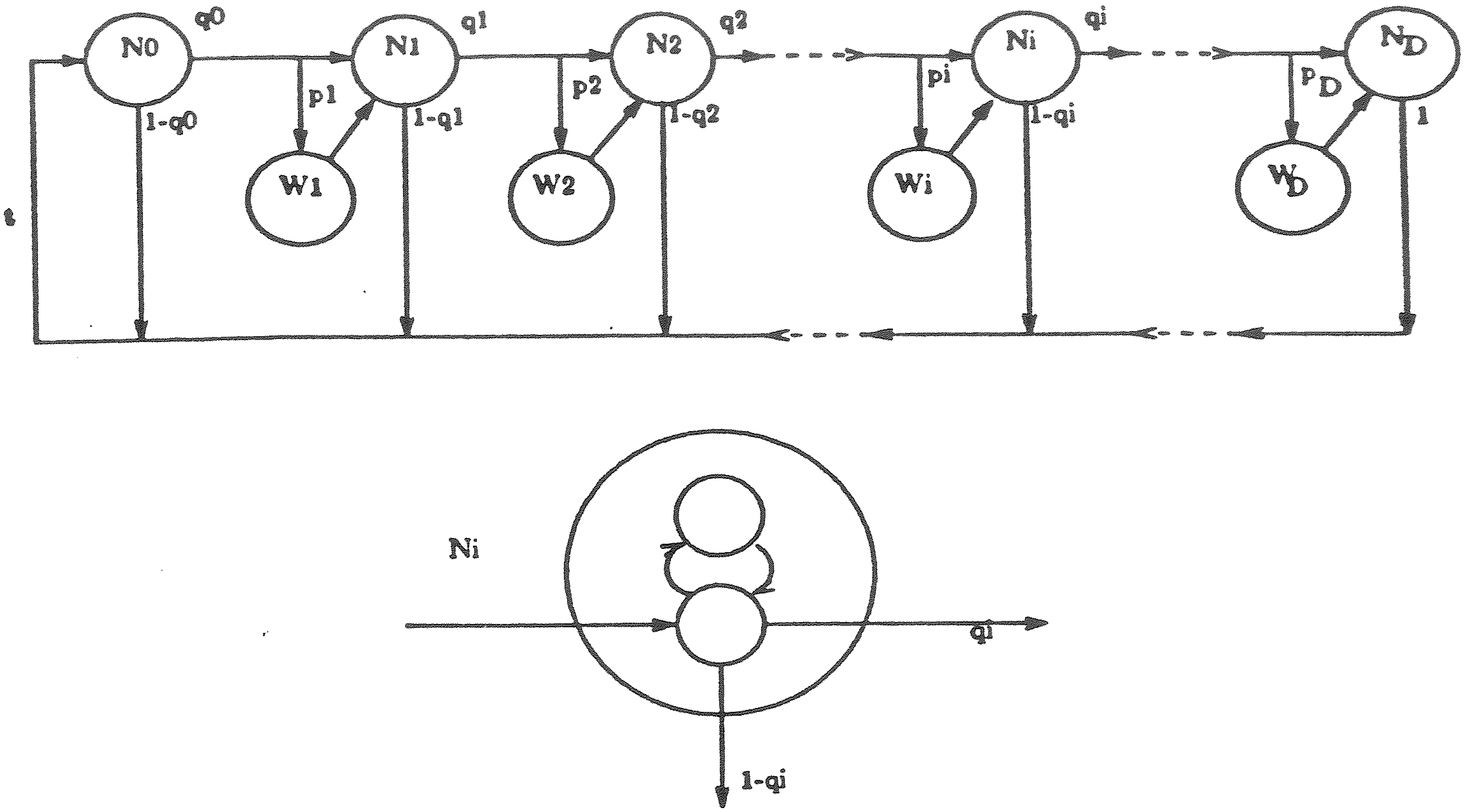

Fig. 3 The model to notify designers conflicts as soon as possible

$q_D = 0$

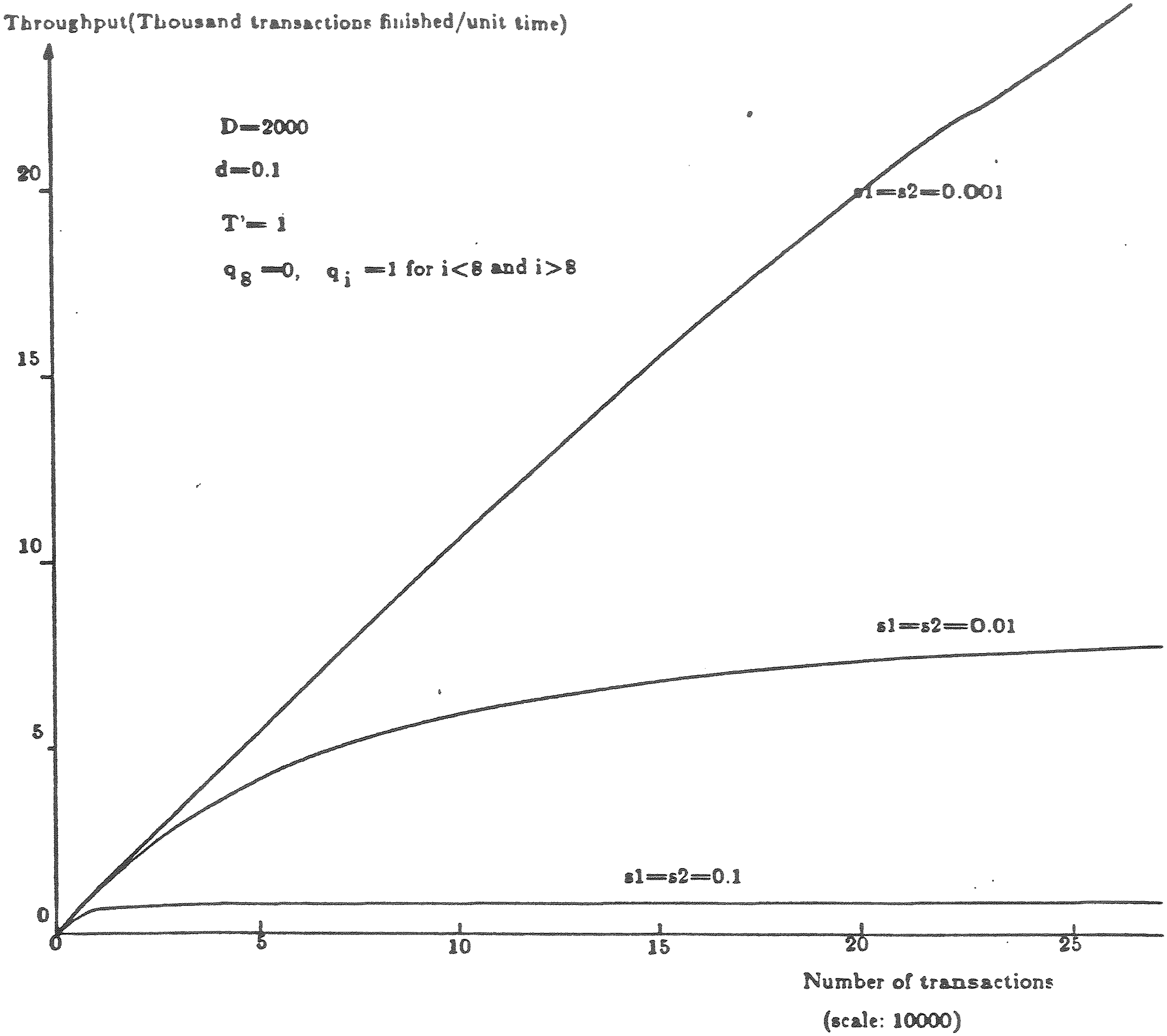

Fig. 4 The throughput for different conflict resolution time.

The first model.

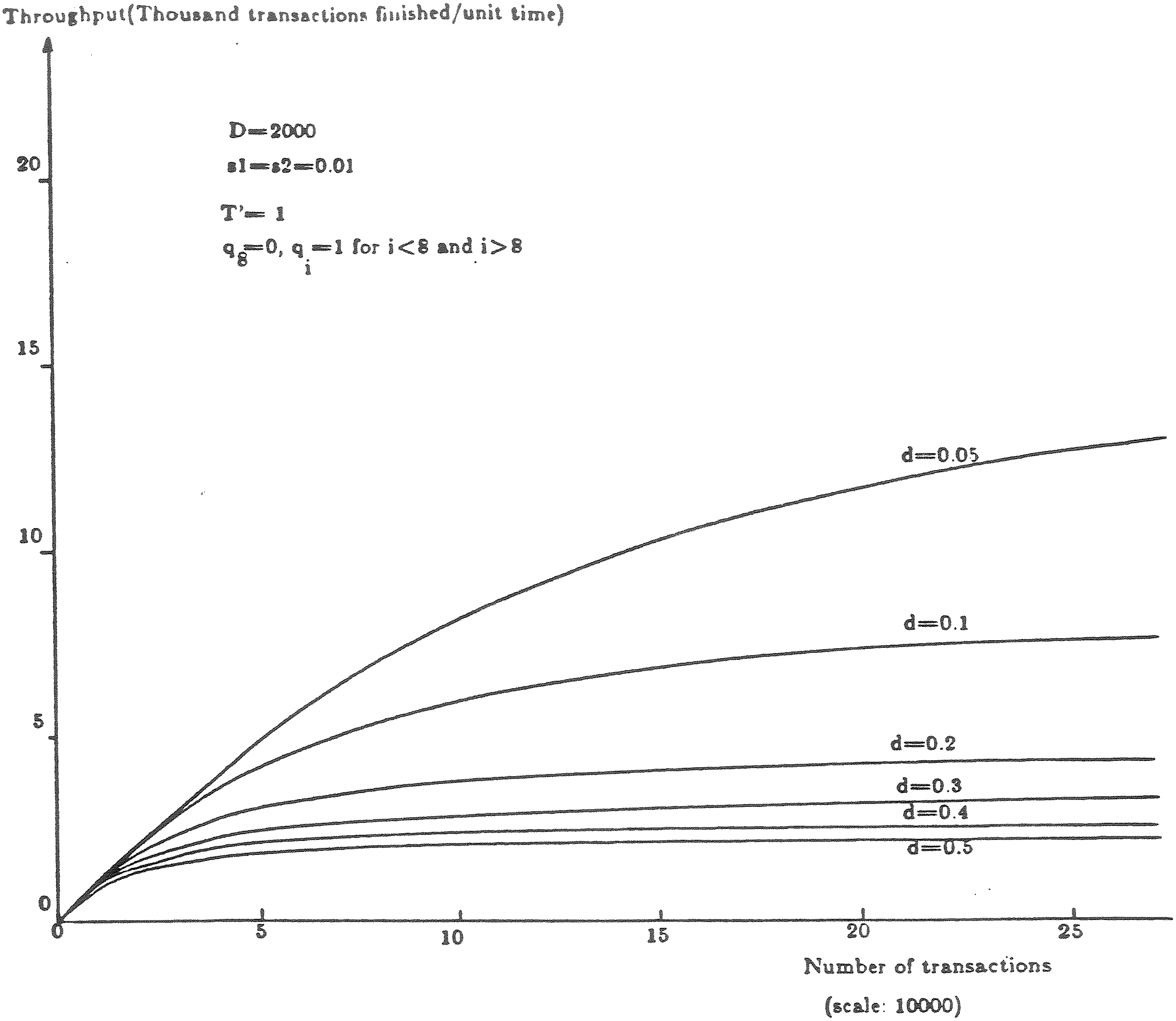

Fig. 5 Throughput for different write lock probability.

The first model.

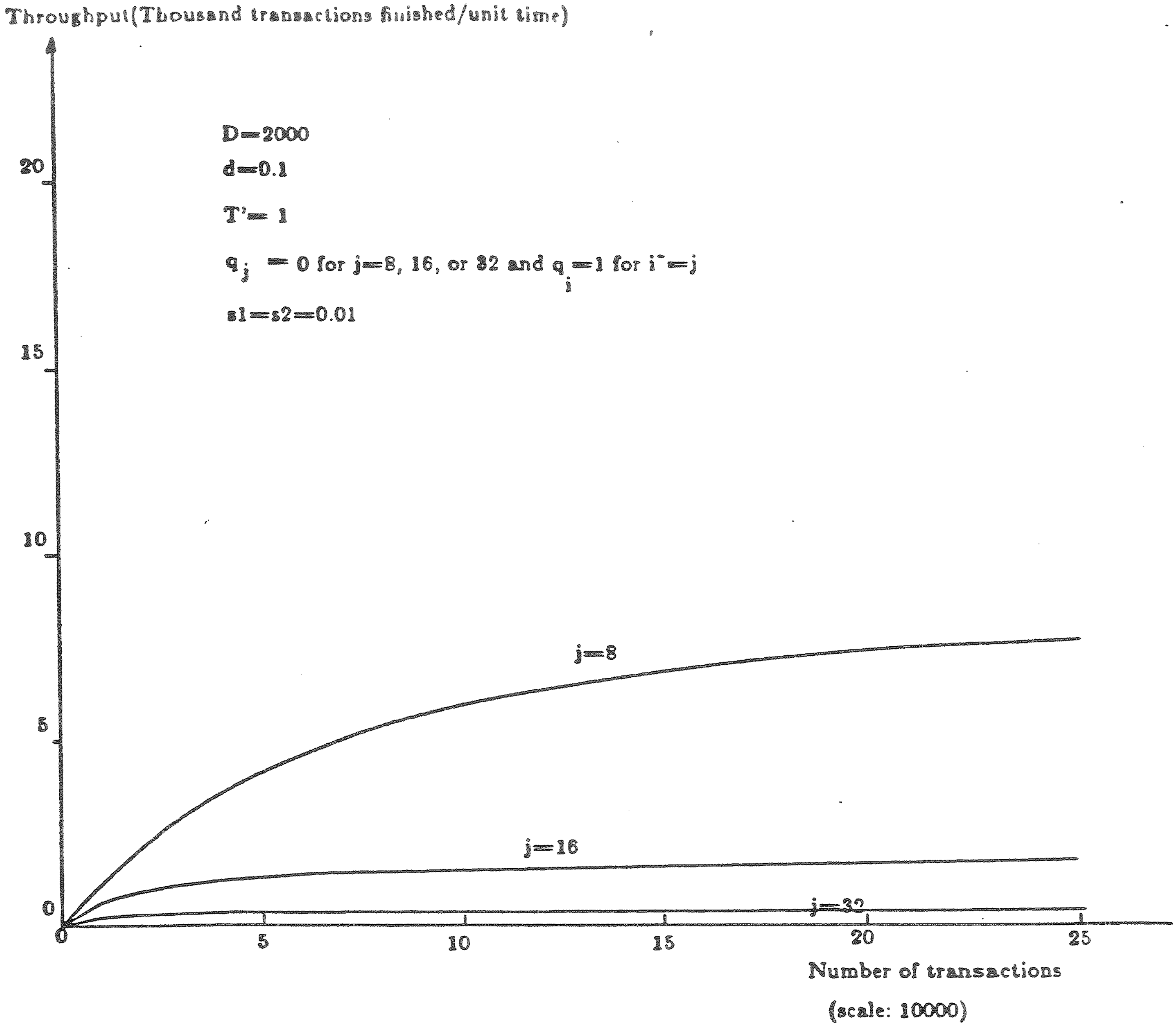

Fig. 6 Throughput for different number of locks requested.

The first model.

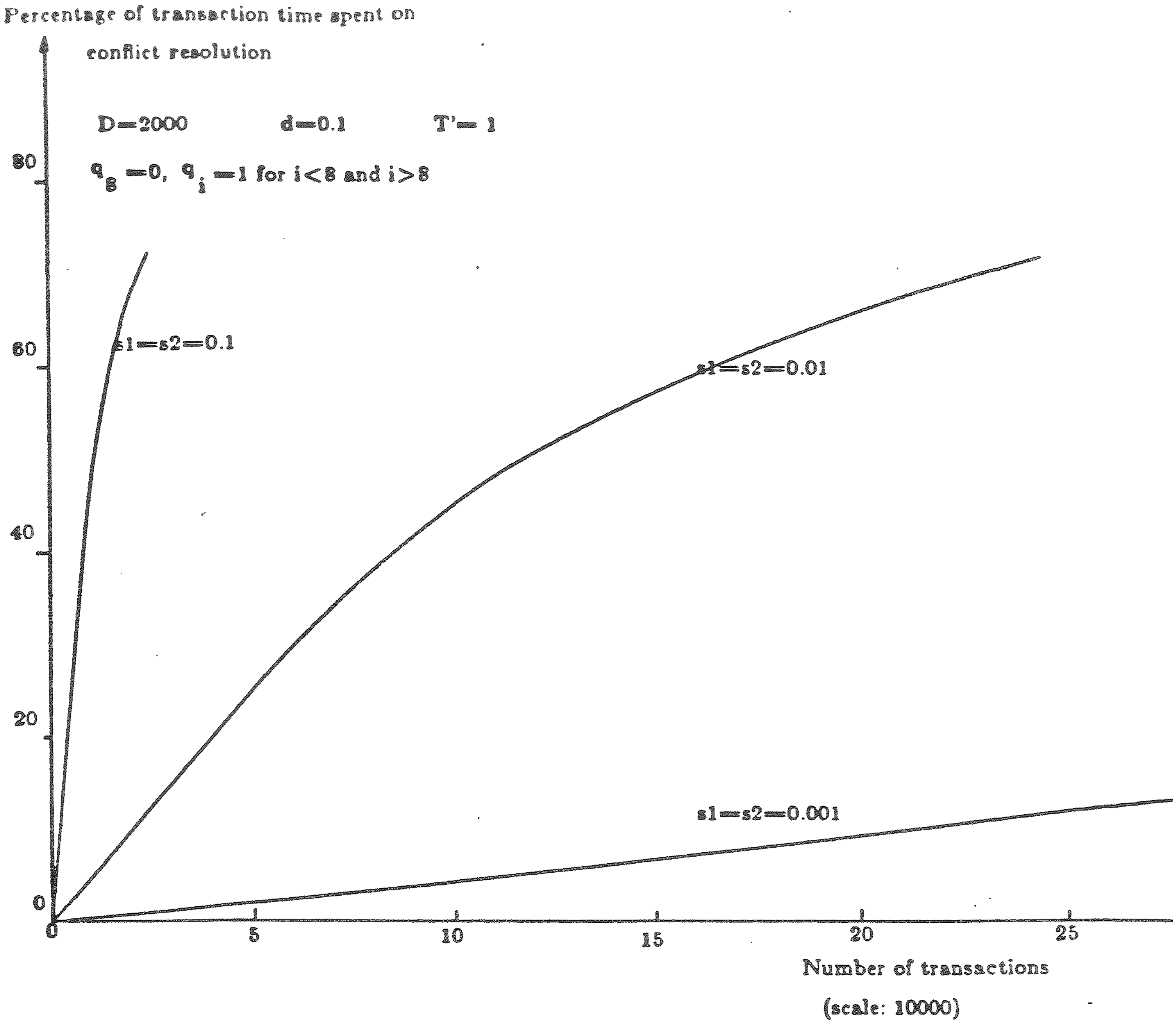

Fig. 7 Percentage of time spent on conflict resolution for different conflict resolution time, the first model.

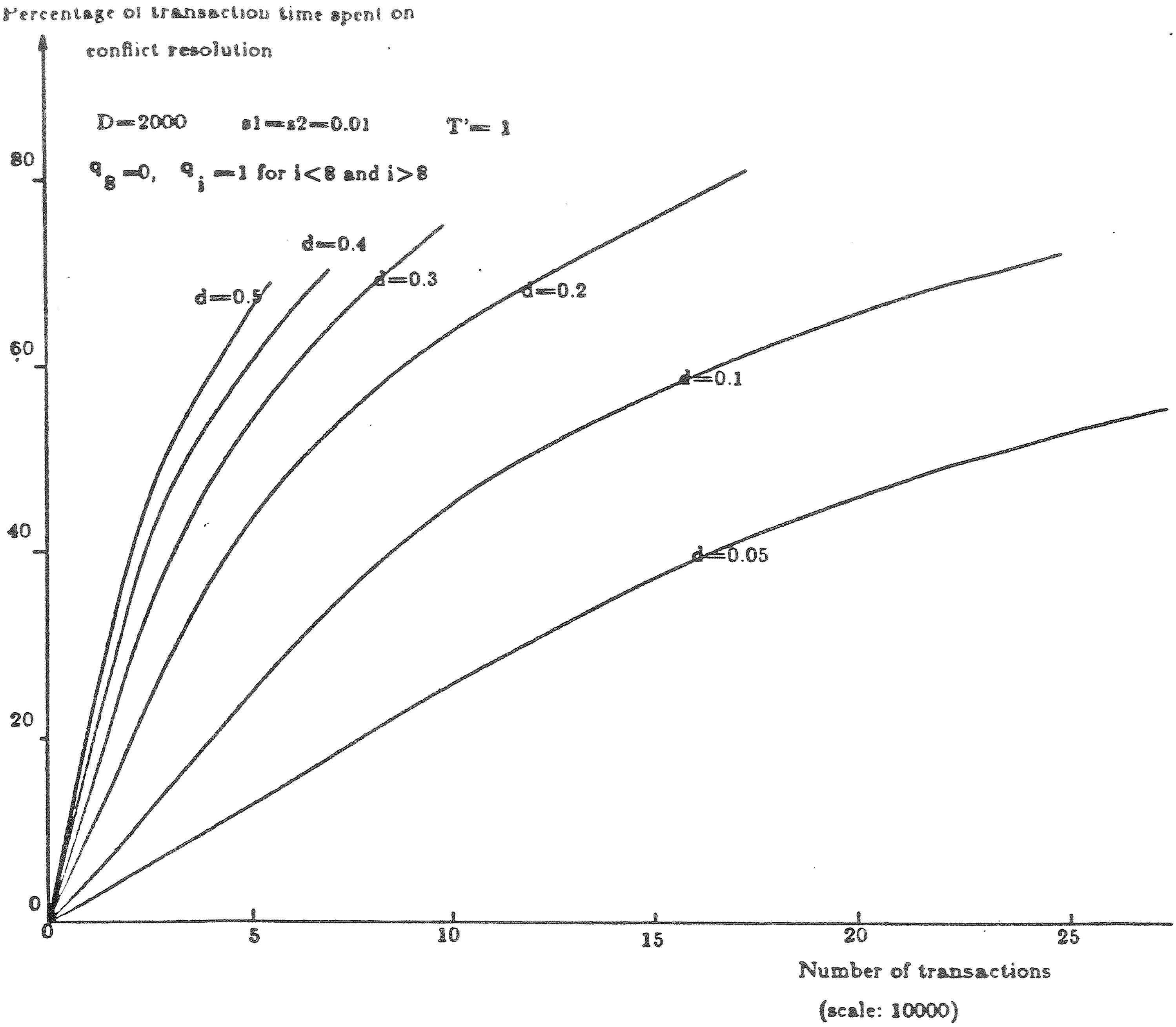

Fig. 8 Percentage of time spent on conflict resolution for different write lock probability. The first model.

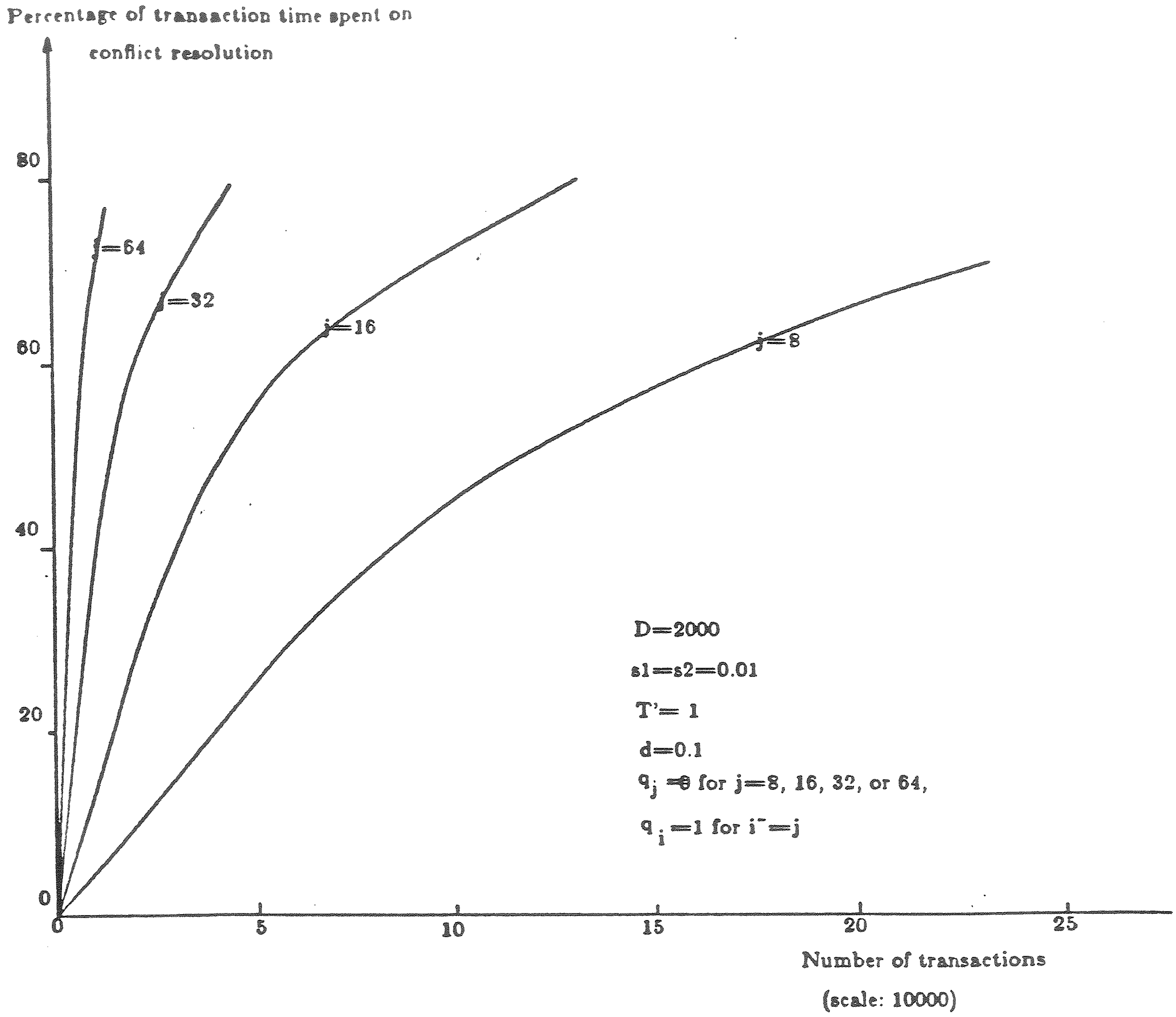

Fig. 9. Percentage of time spent on conflict resolution for different number of locks requested, the first model.

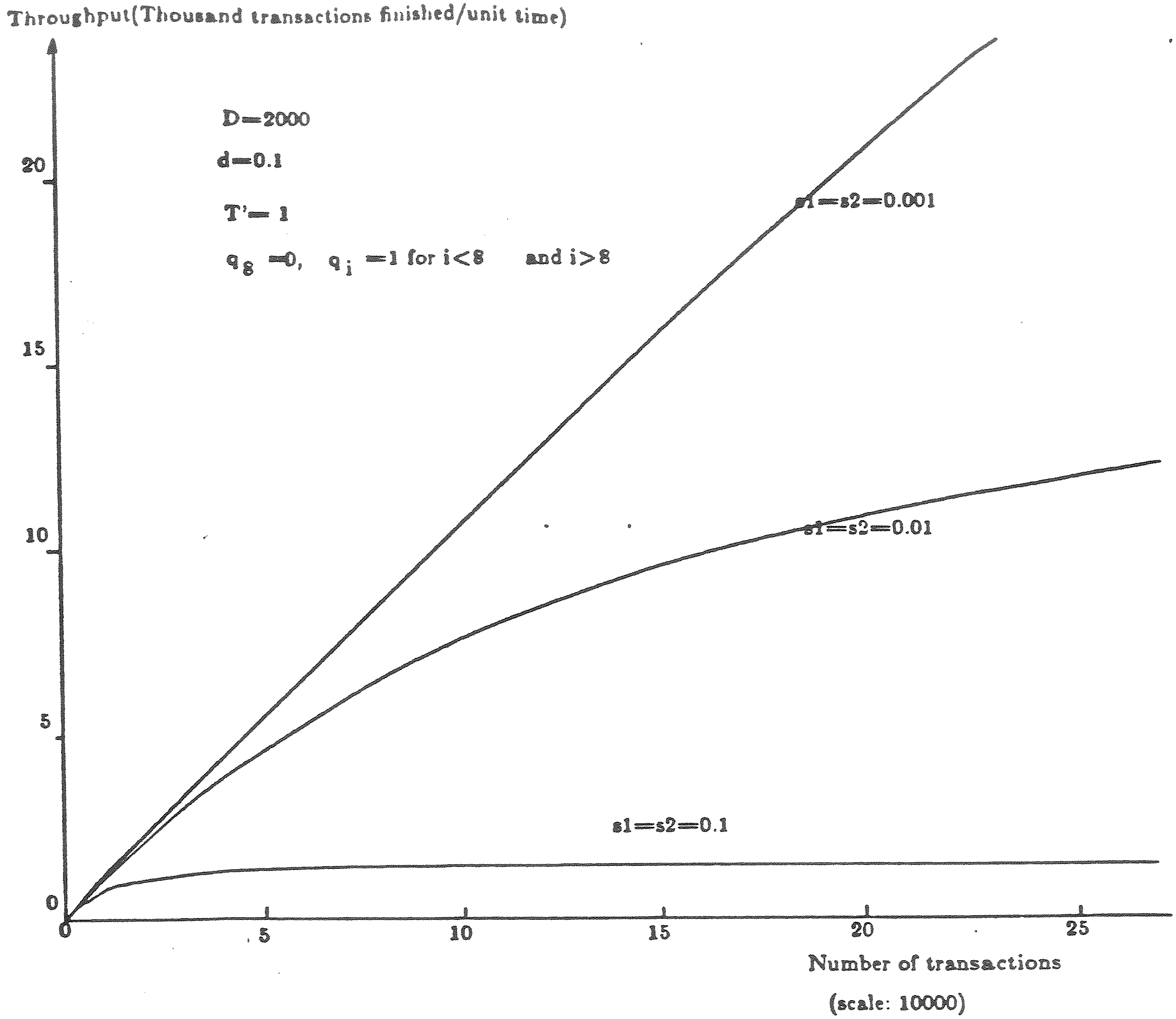

Fig. 10. Throughput for different conflict resolution time.
The second model.

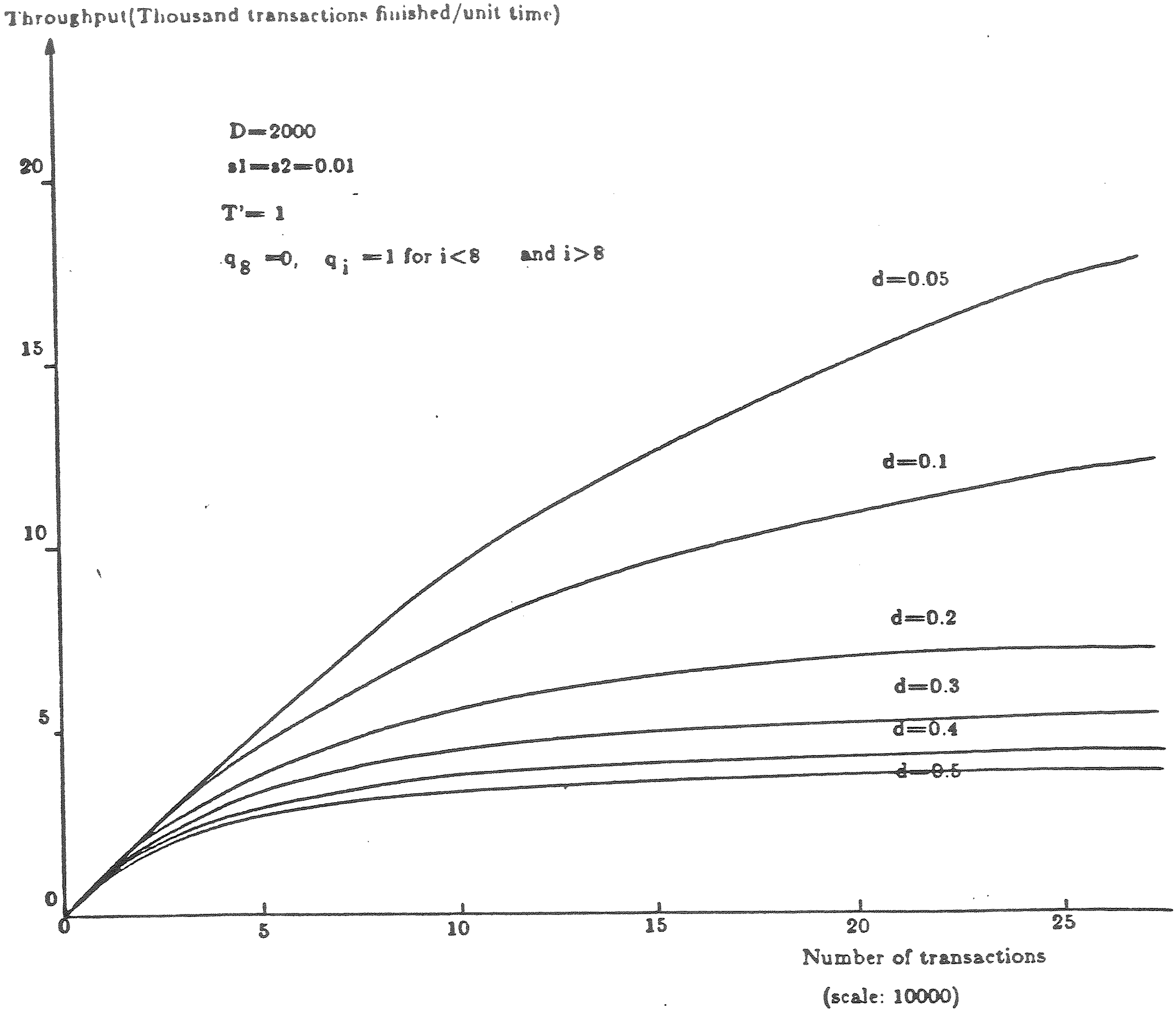

Fig. 11. Throughput for different write lock probability.

The second model.

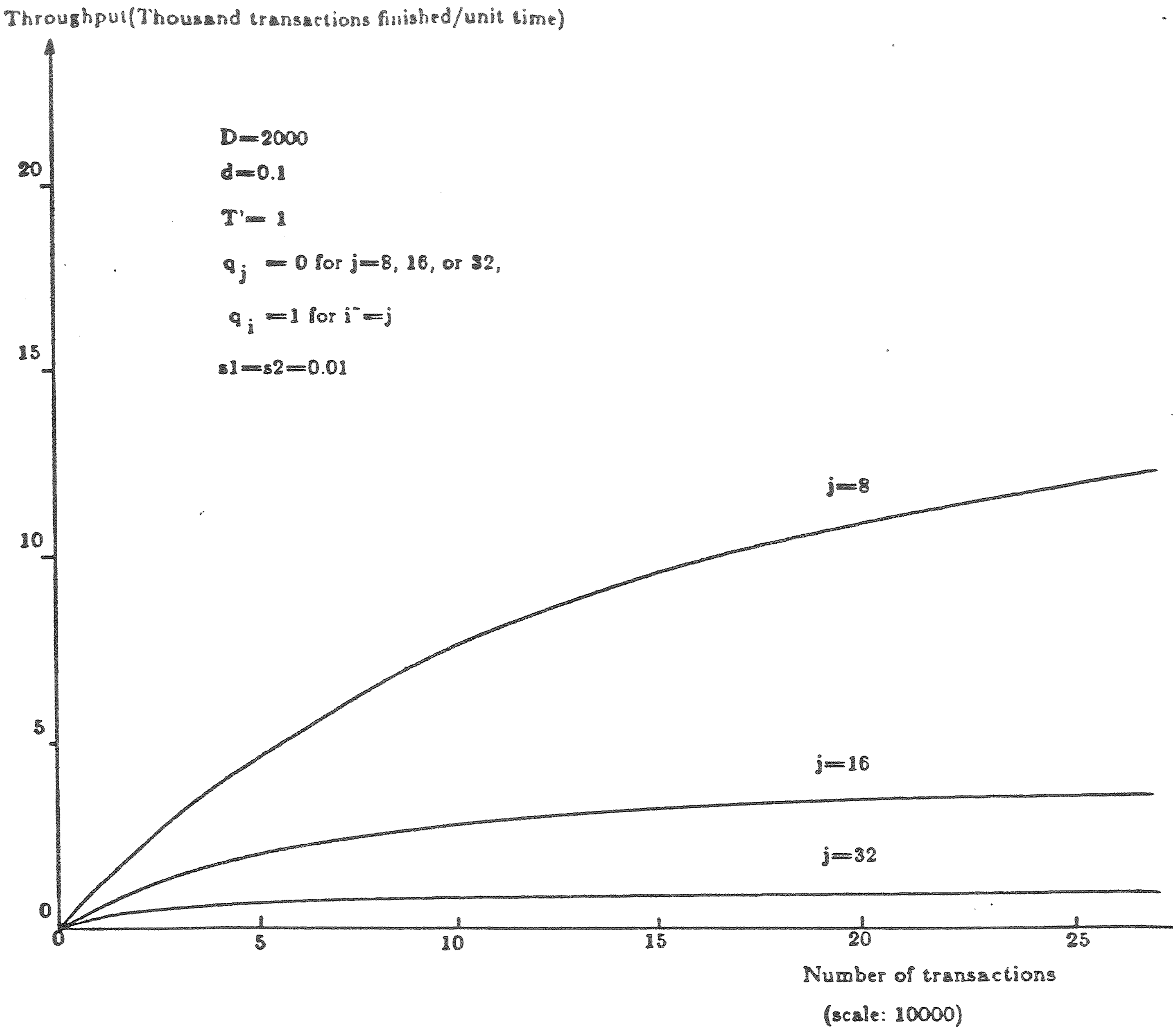

Fig. 12. Throughput for different number of locks requested.

The second model.

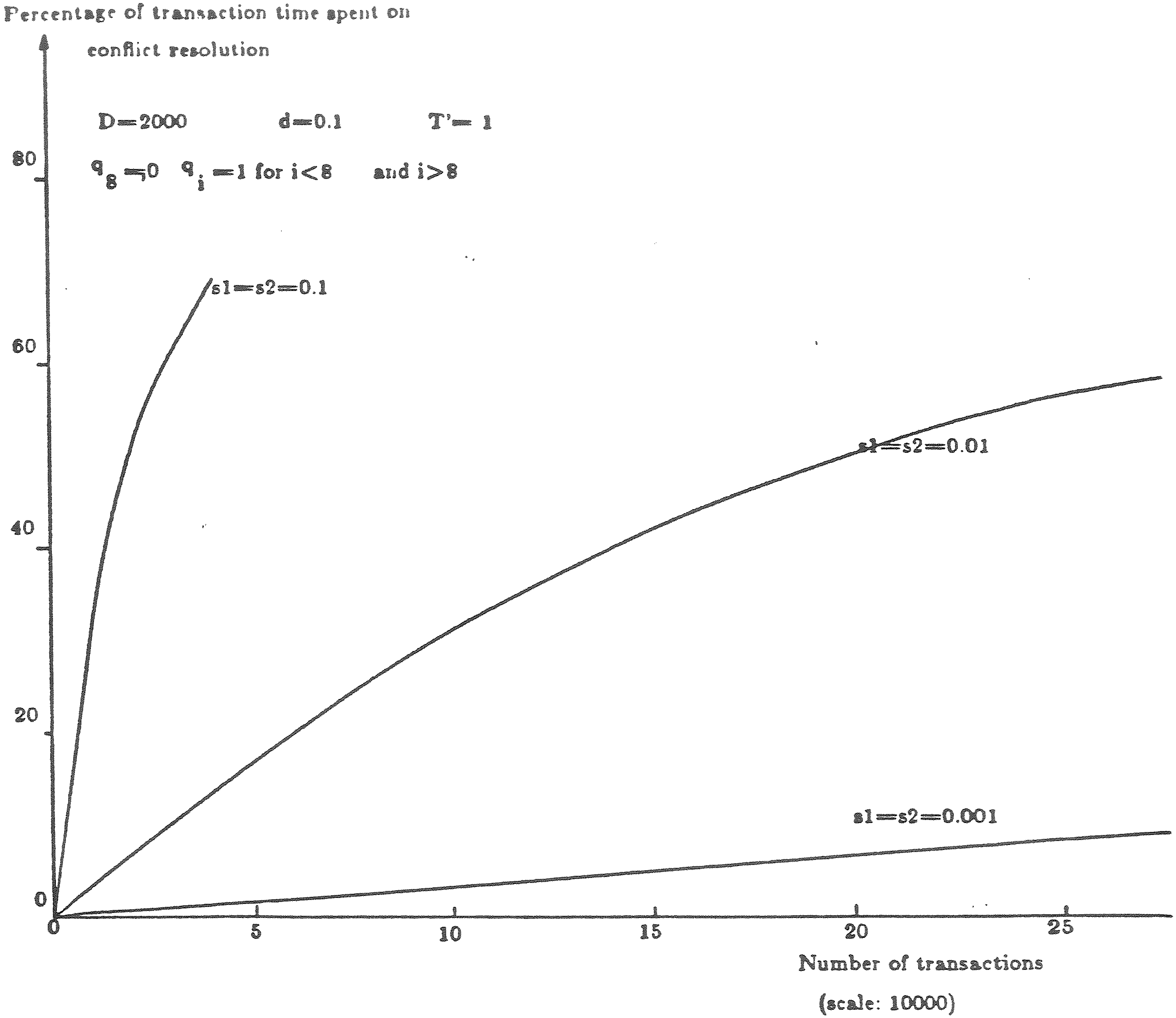

Fig. 13. Percentage of time spent on conflict resolution for different conflict resolution time, the second model.

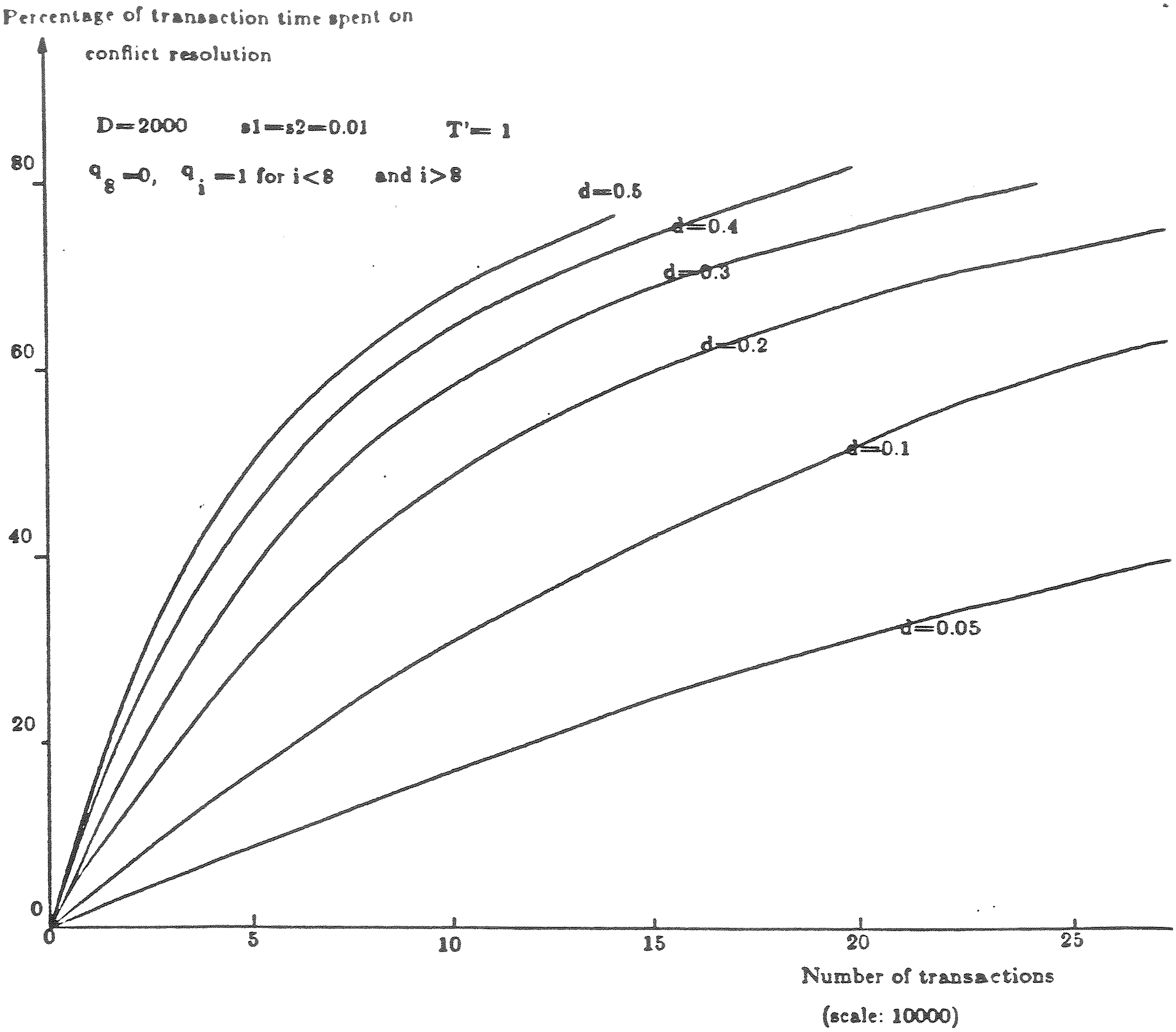

Fig. 14. Percentage of time spent on conflict resolution for different write lock probability, the second model.

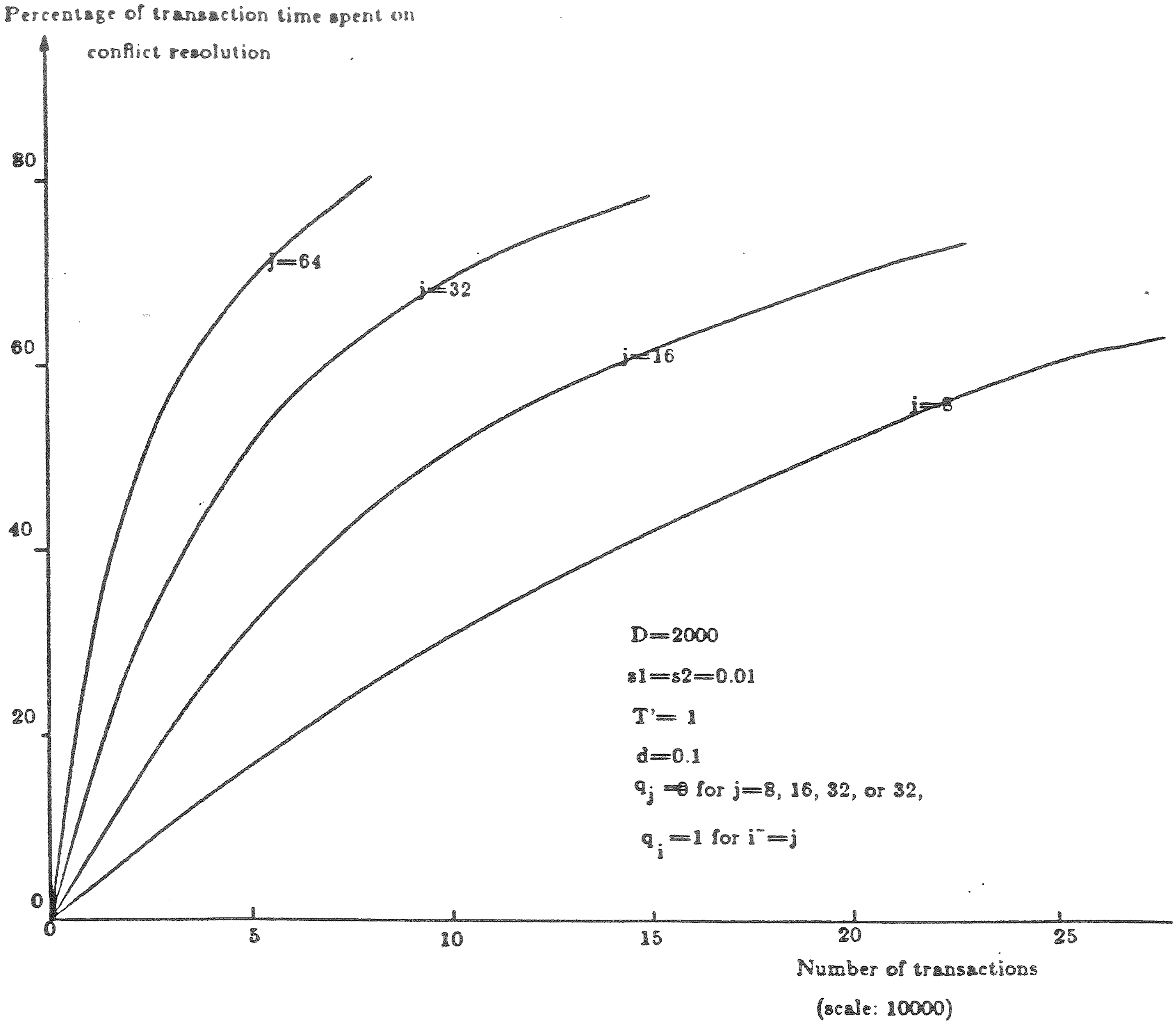

Fig. 15. Percentage of time spent on conflict resolution for different number of lock requests, the second model.

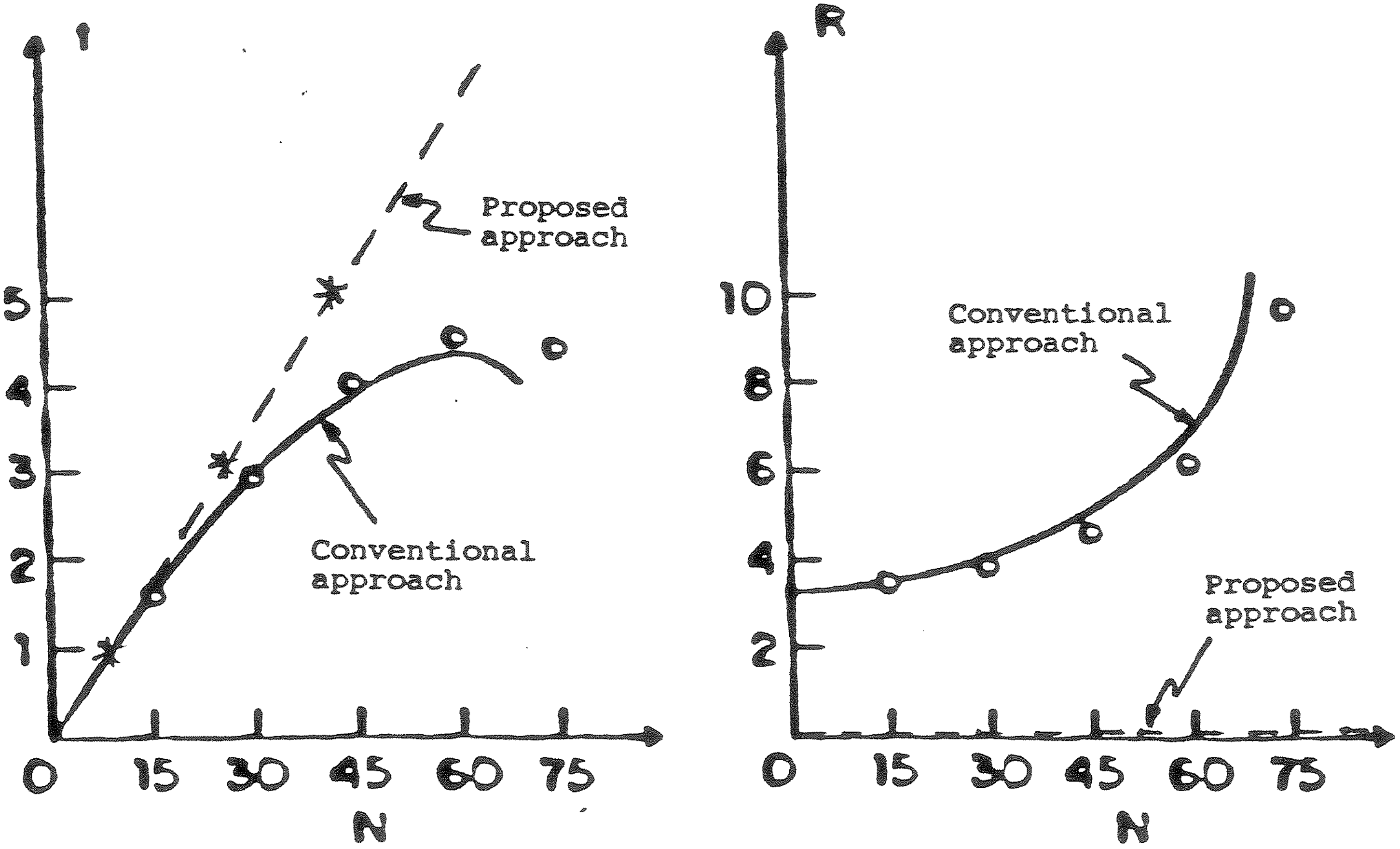

Fig. 16. Comparison of the models in this paper with Tay's model.