The VLDB Journal (1997) 6: 40–52

The VLDB Journal
© Springer-Verlag 1997

Analysis of locking behavior in three real database systems

Vigyan Singhal[1], **Alan Jay Smith**[2]

[1] Cadence Berkeley Laboratories, Cadence Design Systems Inc., 1919 Addison St., Suite 303, Berkeley, CA 94704-1144, USA
[2] Computer Science Division, EECS Department, University of California, Berkeley, CA 94720-1776, USA

Edited by H. Garcia-Molina. Received April 5, 1994 / Accepted November 1, 1995

Abstract. Concurrency control is essential to the correct functioning of a database due to the need for correct, reproducible results. For this reason, and because concurrency control is a well-formulated problem, there has developed an enormous body of literature studying the performance of concurrency control algorithms. Most of this literature uses either analytic modeling or random number-driven simulation, and explicitly or implicitly makes certain assumptions about the behavior of transactions and the patterns by which they set and unset locks. Because of the difficulty of collecting suitable measurements, there have been only a few studies which use trace-driven simulation, and still less study directed toward the characterization of concurrency control behavior of real workloads. In this paper, we present a study of three database workloads, all taken from IBM DB2 relational database systems running commercial applications in a production environment. This study considers topics such as frequency of locking and unlocking, deadlock and blocking, duration of locks, types of locks, correlations between applications of lock types, two-phase versus non-two-phase locking, when locks are held and released, etc. In each case, we evaluate the behavior of the workload relative to the assumptions commonly made in the research literature and discuss the extent to which those assumptions may or may not lead to erroneous conclusions.

Key words: Concurrency control – Workload characterization – Trace-driven simulation

1 Introduction

Concurrency control is essential to the correct functioning of a database due to the need for correct, reproducible results. For this reason, and because concurrency control is a well-formulated problem, there has developed an enormous body of literature studying the performance of concurrency control algorithms. Most of this literature uses either analytic modeling or random number-driven simulation, and explicitly or implicitly makes certain assumptions about the behavior of transactions and the pattern by which they set and unset locks. Because of the difficulty of collecting suitable measurements, there have been only a few studies which use trace-driven simulation, and we are aware of no studies which have been principally directed at characterizing the concurrency control behavior of real workloads.

There have been a few studies of database systems using traces. Some have addressed the issue of database buffer management, e.g., Smith (1978), Rodriguez-Rosell (1976) and Kearns and DeFazio (1989) studied IMS, Verkamp (1985) studied a CODASYL system, and Hawthorn and Stonebraker (1979) have analyzed reference behavior in INGRES, a relational database system. These studies, however, have not looked at the issue of concurrency control. Studying concurrency control requires locking activity characteristics (both lock and unlock events), and information about transaction boundaries.

There has also been one group of researchers (Yu et al. 1985, 1986, 1987, 1993) who have used locking traces from database systems for concurrency control studies. However, the thrust of their research has not been an extensive data characterization, and they have provided limited characterization of the transaction workloads they have used. Our main objective in this paper is to provide a more useful characterization of the locking behavior and transaction lengths.

A large number of researchers have studied concurrency control algorithms from a performance point of view. The usual approach is to define a transaction model consisting of four sub-models: database model, transaction model, user model and system model. Given the model description, there are two popular methods of analysis – through analytical means or by stochastic simulations based on artificially constructed workload parameters (as opposed to trace-driven simulations). It is well known that analytic modeling is limited in the range of behavior assumptions that can employed if solutions are to be expected. Because of the absence of an agreed-upon model of transaction locking behavior, or even the availability of a variety of data and measurements, the simulation studies also make many assumptions. We hope to rectify this problem through this paper.

Section 2 will briefly discuss some of the concepts and terminology we will need in this paper. In Sect. 3, we discuss our traces, and how they were collected and analyzed. Section 4 describes the generic concurrency control model and describes the various assumptions which are present in most studies. A detailed look at the database-related assumptions is provided in Sect. 5, where we also consider their validity and the sensitivity of system performance to these assumptions. Section 6 does the same for transaction behavior assumptions.

2 Terminology and background

In this section, we define some terms used later in our analysis. Singhal and Smith (1994) provide a considerably more detailed discussion of the terminology and methodology used in concurrency control research, as well as additional analysis and modeling.

There are various alternatives to implementing concurrency control in databases – locking (with blocking, and with restarts), optimistic concurrency control (OCC) and timestamping. In this paper we will use the terms *Lock* and *Unlock* because the traced systems use locking to provide concurrency control. Our workloads, however, are equally applicable to the other approaches – OCC and timestamping. For these options the first access to a data item can be interpreted as a Lock request and the last access can be interpreted as an Unlock request.

2.1 Two-phase locking versus non-two-phase locking

If we choose locking to implement concurrency control, an important issue is two-phase locking (2PL). 2PL means that transactions do not acquire new locks after they have released one or more other locks. 2PL is typically ensured by releasing all locks only at the end of transactions. None of the studies we are aware of have considered workloads where transactions release locks before the end of each transaction. In real database systems, however, such as the ones we have traced, locks may actually be released before the end of a transaction to provide better performance. In our traces, some Read locks are released shortly after they are acquired [*short duration* locks, in the terminology of Gray and Reuter (1993)]; Write locks, however, are never released before the end of the transaction. While, in general, this can lead to non-serializable transaction behavior, we presume that the access order to the data has some restrictions and that these are sufficient to preserve serializability, or at least to not affect the validity of the database.

2.2 Cursor locks

We use the term *Cursor locks* to refer to the Read locks which are unlocked before the end of transactions. We call them Cursor locks because usually the purpose of these locks is to guarantee the stability of the cursor pointer on the relation while data is being accessed from the relation (Gray and Reuter 1993, p. 397). These locks are held only while the object is being accessed. The non-Cursor locks are simply referred to as *Read* or *Write* locks. Although Cursor locks are typically held for a much shorter time than Read locks, they can still have a significant effect on performance, since the transaction acquiring the Cursor lock may have to wait a long time for that lock, even though it then holds it for only a short period.

2.3 Index locks

Index locks (used to provide fast access to data), such as B-trees, also have to be locked to maintain transaction isolation. Most concurrency control performance studies have ignored the contention for Index locks, although some recent studies have considered this issue specifically (Johnson and Shasha 1993; Srinivasan and Carey 1991). We do not know of any study which has considered the performance of both index and data locking simultaneously. Because of the high use of Index locks, they have the potential to cause significant performance problems. The fact that the systems we measured use subpages as the granularity of Index locks (as opposed to pages for data) suggests that index contention may have been a problem in the past with some systems, although this has not been documented. In this paper, locks on index subpages and data pages are simply referred to as Index locks and Data locks, respectively.

2.4 Locktimes and lockfractions

The *locktime* of a lock is the length of time that the lock is held. All measured times in this paper are real (not virtual) times on the traced database systems. The locktime of a transaction is defined as the sum of the locktimes for all the locks it acquires. Locktime is an important quantity – it indicates how long a transaction may have to wait to acquire a lock which conflicts with a lock already held by another transaction. Note that the ratio of the locktime of a transaction to the length of the transaction gives the average number of locks held by the transaction during its lifetime. Because Read and Write locks are released only at the the end of the transaction, locktimes for Read and Write locks should therefore be closely related to transaction length, unless most locks are acquired at the end of transactions. We define *lockfraction* as the ratio of locktime to transaction length. For Read and Write locks, the lockfraction also indicates the fraction of the remaining lifetime when the lock is acquired.

3 Description of traces

In this section, we describe our traces and the systems from which they were collected.

The data analyzed in this paper have been collected from three different commercial installations: Security Pacific Bank (referred to as *bank*), Crowley Maritime Corp. (*transport*) and an anonymous telecommunications company (*phone*). The description of the relevant hardware and software configurations of the three sites appears in Table 1.

Two of the traces, *bank* and *transport* have been traced using the IBM DB2 GTF tracing facility (IBM Corporation 1987). The process of gathering these traces and a detailed description of the traced events is presented by Viavant

Table 1. Description of the trace sites and the traces. *Entries* refers to the total number of trace entries per traces (buffer manager and locking entries). *TPS* refers to transactions/second

	Site (company)		
	Phone (anonymous)	*Bank* (Security Pacific)	*Transport* (Crowley Maritime)
Trace date	10/15/90	3/16/88	7/25/88
Hardware system (# of units)			
CPU	IBM 3090-600J (12)	IBM 3090-200 (2)	Hitachi/NAS XL80
Disks	IBM 3380/90 (35)	IBM 3380 (15)	IBM 3380 (20)
Software			
OS	MVS-XA 3.1	MVS-XA 2.1.7	MVS-XA 2.1
DB2 release	Release 1 version 2.1	Release 3 version 1.3	Release 3 version 1.3
Trace			
Size	216.91 MB	307.69 MB	567.58 MB
Time	30 min	1 h 15 min	3 h
Entries	8112768	830478	1834422
TPS	4.955	0.341	0.544

(1989). The *phone* trace has been gathered using a new and much more efficient tracing package (Ted Messenger, IBM Almaden Research Center, personal communication 1990), discussed further below. The DB2 tracing facility is able to activate tracing for any subset of 14 different *event-classes*, for example *SQL events*, *I/O events*, and *Lock events*. Each event class causes trace records to be logged for a number of different system events.

The traces studied are trace segments containing contiguous trace information, without any breaks in the tracing period, collected from multiprocessor shared-memory (centralized) systems. The database management system at all three sites was DB2, IBM's relational database management system for IBM 3090 mainframes (Date and White 1993), running under the MVS operating system using 3380 disks. These specific configurations of the database system might limit the generality of our observations. While the three traces may not completely represent the entire spectrum of transaction-processing applications, our paper still provides a good characterization of locking behavior in three very different transaction-processing environments. To our knowledge, no other work in the published literature covers such a varied workload.

3.1 Locking entries

For our analysis, we used the Lock and Unlock entries from the traces. Data pages are locked in one page (4 KB) units. Indexes may be locked on a subpage partition; each index page is a B-tree node. The number of partitions is user-defined and variable per trace. We observed up to 16 partitions per page for some indexes in our traces. The DB2 systems we traced had all lock entries at the page or subpage level; for the next generation DB2 products, locks at a finer level of granularity (record-level) have been proposed (Mohan et al. 1992).

The DB2 traces contain three kinds of locks – Read, Write and Update. Write locks conflict with each other and with Read locks. Update locks guarantee the same semantic consistency as Read locks; however, only Update locks are allowed to be upgraded to Write locks. Update locks do not conflict with Read locks, but conflict with each other and with Write locks. The reason for having Update locks, instead of directly upgrading Read locks, is to avoid possible future deadlocks at the cost of higher contention (Date 1987). The performance tradeoffs of Read locks versus Update locks are not obvious and, to our knowledge, such tradeoffs have never been studied in the literature. Because of this and since the functionality provided by Update locks is identical to Read locks, in this paper we have mapped all Update locks to Read locks.

3.2 Transactions

Individual transactions in the trace are identified with a field called the *ACE address*, which identifies the address space of the process. There are no *Begin Transaction* or *End Transaction* entries in the traces. We have, therefore, chosen the time the first lock request is issued by a transaction as the beginning of the transaction lifetime. There is, however, a special Unlock entry which says *Unlock all locks*, owned by this ACE address. We use this to decide when a transaction ends. Our definition of transaction boundaries is smaller than the actual boundaries – we neglect the work done before the first lock request and after the last unlock request. However, we include all the work done while any locks were held. Since we are interested in concurrency control contention, this should be sufficient. Note that the actual system load and the transaction lengths will be greater than what is reported here. We refer to the number of active transactions at any moment as the multiprogramming level (MPL) of the system.

In this paper, we measure various characteristics of the transactions. Using the method described above we have associated a transaction identifier with each trace entry. Then we unraveled the trace to separate the entries belonging to one transaction from the entries belonging to other interleaved transactions. Most measurements in this paper have been done from this pool of transactions. This pool is what would be required, along with the ordering of transactions, if we were to use our traces for a trace-driven concurrency control model simulation.

Note that a transaction sometimes represents a smaller unit of work than a job, since a user job may spawn a number of transactions. A transaction is a complete unit of work, however, in that it releases all locks it owns when it finishes.

3.3 Timestamps

In two of the traces, *bank* and *transport*, all trace entries are tagged with a timestamp. In the *phone* trace, the timestamps have been synthesized by linearly interpolating time epochs from some entries which record timestamps. There is a special trace entry which periodically records the time (every 32 KB, or 1142 trace entries), and some of the trace entries (like *Commit* and a few others which we do not use in our analyses) do have timestamps.

Since our objective is to characterize the transaction workloads in real databases, we have removed lock-wait intervals from the traces for all our analyses, except for Fig. 1 which plots the system activity for the traces. This is done for a particular transaction by advancing the timestamps of the transaction entries that follow the lock-wait by the duration of the lock-wait interval. The motivation to exclude existing lock-wait intervals is clear – we want to characterize the transaction workload that is loaded on the system, and not how the traced DB2 system processed that workload. Lock-waits represent one instantiation (one particular interleaving of transactions) of the workload; they are not part of the inherent workload. In fact, lock-waits represent a *performance index*, an output, of concurrency control studies.

We obtained *bank* and *transport* traces using the GTF tracing package, which imposes a substantial system overhead [of the order of 100–200% is reported by Viavant (1989)]. GTF logs an enormous amount of information per trace entry, much of which (about 88% by volume) is not used for our analysis. This overhead may affect us in two ways. It may affect the validity of the durations of the transactions. However, in this paper we are not interested in the absolute time values. All time values must be considered relative to each other. We make the reasonable assumption that all transactions are affected by the overhead equally. Another side effect of the overhead is that the users' behavior may change due to this overhead. Although this may reflect on the system MPL, we have no reason to believe that it affects the mix of the transactions in the workload substantially. The *phone* trace has been logged using a very efficient new tracing package which incurs an overhead of approximately 2–5% (T. Messenger, IBM Almaden Research Center, personal communication 1990). In fact, we have been told that the users of the database system did not perceive any change in the system performance. Clearly, this eliminates the possibility of the second side effect we just discussed. Note that we also assume that all transactions are stretched equally due to multiprogramming effects.

The timing information about the occurrence of lock requests may also be perturbed due to database configurations, load controller policy, and the second-order effects of data contention on the number of active transactions and resource contention. Therefore, in this paper, we are only interested in studying time values relative to each other. This, of course, assumes that perturbations on the time values are proportional. Notice that we do not assume that the effects on lock contention are also proportional; this is because, as we stated in the beginning of this subsection, we have excluded the lock-wait intervals from the transaction lengths. Thus, we allow transactions with different locking requirements but otherwise equal transaction lengths under an MPL to have different total lengths under a different MPL, due to different lock-wait lengths for the two transactions.

3.4 System activity in the traced systems

In this subsection, we present the level of system activity in the database systems we traced. Figure 1 shows the transaction MPL and the average number of transactions waiting on

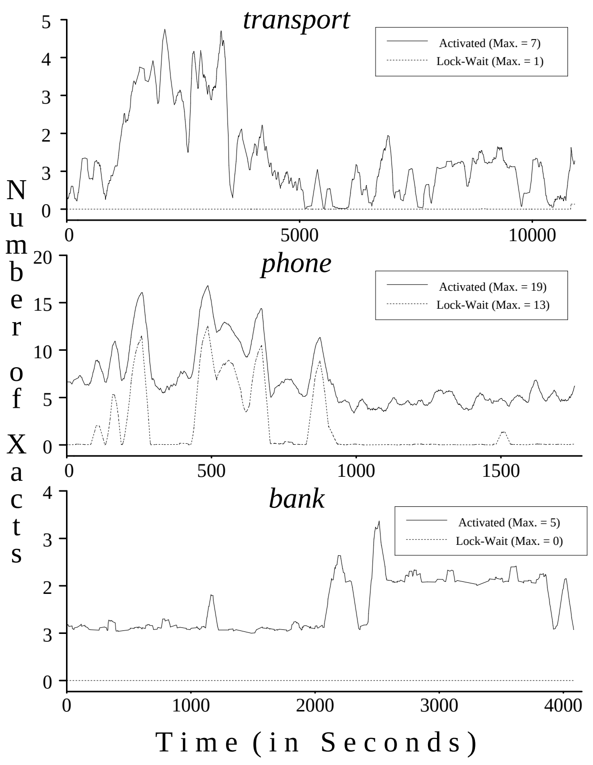


Fig. 1. Transaction load for the traces. The number of transactions (*Xacts*) denotes an average over a time-period. The time-period is 120 s, 15 s and 30 s, respectively, for the three traces. *Lock-wait* denotes the average number of transactions waiting because of lock conflicts

lock requests. The figure represents an average number of transactions calculated using a continuous window of time (2 min, 1 min and 30 s for *transport*, *bank* and *phone*, respectively). The window size has been chosen for each trace to provide reasonable smoothing.

Because of the way we have defined transaction lifetimes earlier in this section, Fig. 1 represents a lower level of activity than the actual system – processes which do not lock any items are not counted; also portions of transaction lifetimes before the acquisition of first lock and after the release of all locks, are not counted.

The figure shows a relatively low MPL for the *transport* and *bank* traces. One of the reasons for this might be that the heavy overload of GTF tracing caused the transaction load to be less than the usual load, due both to the scheduler decreasing the MPL, and to users withdrawing from the system due to unresponsiveness. Even though the MPL of the traced system may be high, it does not mean that transaction mix in the database system is not representative of the mix under the regular load. These two traces also show a near-zero level of lock contention. On the other hand, the *phone* plot shows a significant degree of contention. This trace also contained some transaction aborts because transactions had deadlocked due to circular dependencies for locks. The high degree of contention will affect the timing information about the locking events in the transaction load. However, we are only interested in time-values relative to each other. Also, as explained in the previous section, we have removed the lock-wait periods from the trace. One interesting observation from the *phone* plot is the fact that the contention level mimics the MPL very closely. This suggests that the system is trying to maintain the number of active transactions (those not blocked for locks) at a constant. This would suggest a load control policy which is based solely on the number of active transactions, and ignores the number of transactions blocked for locks. We refer the reader to Monkeberg and Weikum (1991) and Carey et al. (1990) for a detailed discussion of load control policies.

4 Concurrency control modeling

In this section we describe the generic concurrency control model used in most concurrency control analyses, and note also the various assumptions made in such modeling. Most assumptions have two flavors – behavioral assumptions (e.g., selection of exponential distributions) and assumptions about parameter settings (e.g., the value of the mean for that exponential distribution).

There are four main components of a concurrency control model: the database model, the user model, the transaction model and the system model.

4.1 Database model

The database model captures the characteristics of the database, such as the database size, data distribution on multiple nodes, data replication, and the access pattern. A relational database, for example, is composed of several relations or tables. Each of the relations might be stored in one or more files. A relation is comprised of a set of tuples, such that each of these tuples is defined over the same attributes.

In concurrency control modeling studies, typically, a database is modeled as a collection of fixed size *data items*. Data items are grouped into *granules* to form units of access and concurrency control. With the exception of studies analyzing effects of granularity on concurrency control performance (Carey and Stonebraker 1984; Ries and Stonebraker 1979), one granule usually comprises one data item, and is the unit of I/O access or concurrency control. In most commercial systems, granules are defined on a physical scale and not on a logical scale; locks are usually per page and are not related to the record sizes or the table sizes. In the context of this paper, one lockable item or object can be interpreted to be one granule.

Clearly the number of the data items is an important parameter because it directly affects the amount of contention. The higher the number of granules, other parameters remaining identical, the lower is the contention. However, as we will see later, the number of lockable items in the database has been frequently underestimated in order to obtain non-negligible (and “interesting”) levels of contention in the system.

The access pattern of the data items has often been modeled as being uniform over the entire database. To model non-uniform access some models have used an 80–20 or a 50–5 (Dan et al. 1994) access behavior. A b–c access behavior means that b% of the accesses are uniformly distributed over c% of the data items and the remaining accesses are uniformly distributed over $(100 - c)$% of the data[1].

Although concurrency control performance for indexes has been studied in isolation (Srinivasan and Carey 1991), we do not know of a study that has analyzed the effects of index and data locking in the same framework. Since index pages have a B-tree structure, as opposed to the flat structure of the data items, it is reasonable to expect different conflict patterns for data and index locks.

Many, particularly early, studies have assumed that all locks were Write locks to simplify the analyses. Even if we model both Read and Write locks, we must be careful to allow Read locks to conflict with other Read locks in our model, in case a Write lock request is queued ahead for the same object. Also, not only should we use a realistic Read/Write lock ratio, we must also ascertain if the Read/Write ratio is dependent on how frequently an item is locked. Clearly, if a more frequently accessed object is Write-locked less frequently (but Read-locked much more often) than another object, this page would be less of a factor in the overall contention. This sort of behavior may, for example, be expected in the index root pages which are accessed much more often than the leaf pages.

Another aspect of modeling Read and Write locks is Write-lock acquisition. The issue here is whether Write locks are set on items to be updated when they are first read, or whether such items are Read-locked and later the Read locks are upgraded to Write. Agrawal et al. (1987) have studied this assumption and concluded that there is a significant difference in performance between the two cases. Also, different locking algorithms are affected to different degrees by this assumption.

4.2 User model

The user model describes the arrival process of the user transactions at the system. The arrival process is generally modeled either as an open system (Carey and Stonebraker 1984) (invariably as an exponential inter-arrival distribution) or as a closed system (Tay et al. 1985) where the users circulate through the system and resubmit transactions after the previous ones are executed. The users’ transactions may be batch-type (non-interactive) or interactive. For a closed system, an external think time (deterministic or variable) may be modeled as the mean time between a transaction completion and the next submission from the same terminal. Open system models are invariably modeled using a Poisson arrival process. The transactions in our workloads come from both batch-mode submissions and interactive transactions. We do not have enough information in our traces to distinguish between the two.

4.3 Transaction model

For our purposes, a transaction can be characterized by a string of concurrency control requests, CPU and I/O process-

[1] Note that this is slightly different from the b–c access rule as proposed by Knuth (1973), p. 397), which means that b% of the accesses are over c% of the data, b% of b% of accesses are over c% of c% of data, and so on

ing requests and intra-transaction think times. The transaction model defines how these various components combine to form a transaction. The various parameters are the total length of the transactions, order of the CPU, I/O, concurrency control and think time requests, total number of these requests per transaction, duration between requests, portion of database accessed by these requests, etc. A simple model would be to have one class of transactions, each transaction comprised of the same number of concurrency control requests, identical duration between the requests without explicitly modeling CPU and I/O computation, and all data items being equally likely to be accessed on any access.

The modeling of transaction lengths is very important. While the mean of transaction lengths is an important parameter, the variation of the transaction lengths is also important, since it is well known in queueing systems that flow times increase with service time variance as well as mean (Kleinrock 1975).

4.4 System model

The system model captures the relevant characteristics of the system design (both hardware and software), including the physical resources (CPUs and disks), their performance parameters, and their associated schedules. The schedule refers to the sequencing of CPU and I/O requests as well as the amounts of service needed. The CPU and I/O time per logical service are specified as model parameters. The models also include the CPU service discipline, the number of CPUs for multiprocessors and the CPU configuration for distributed systems. With the growing interest in distributed database systems, varying the system model configuration and studying its effects on concurrency control performance has become a popular area of research (Wang and Rowe 1991; Carey and Livny 1988).

A few studies have not modeled the system resources at all. They assume that the rate of processing for an individual transaction remains constant independent of the MPL. This is equivalent to assuming infinite system resources. Although this approach might appear to limit the applicability of such studies to real systems, they do isolate the effects of data contention from resource contention on the system performance.

To the extent that the physical system configuration is sufficient for the offered workload, the issue of the system model is largely separable from modeling the transaction workload. This paper addresses the latter issue, and we do not consider the system model further here.

5 Modeling the database

In this section we will present the database-related characteristics of our workloads, and how the trace characteristics correlate with the assumptions in the literature. Specifically, we will look at the access distribution of locks over the database, the extent of index locking, the distribution of and correlation between Read, Write and Cursor locks, Write locks, and the importance of Cursor locks.

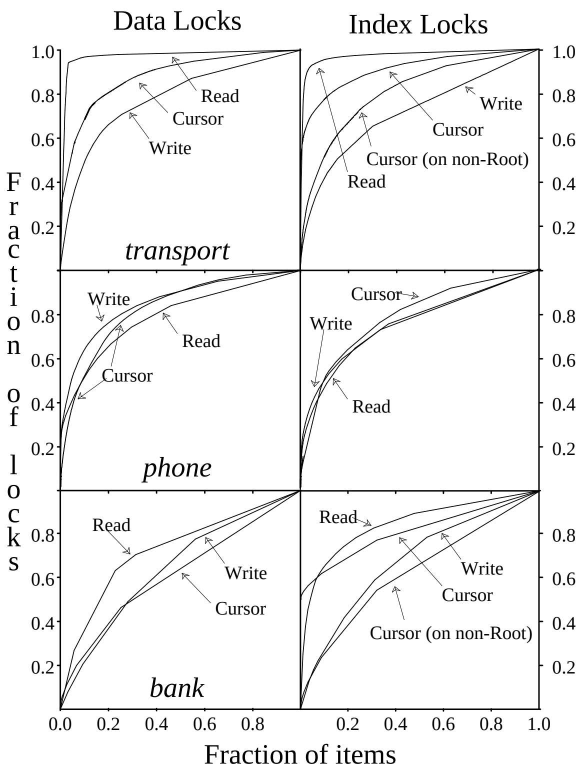


Fig. 2. Distribution of locks over the index and database items. The three sets of plots are for *transport*, *phone* and *bank*, respectively. Root refers to the root of B-tree indexes. Read and Write (and Cursor for *phone*) lock distributions for non-Root index items are almost identical to the ones in the figure because Root pages acquire these kinds of locks very rarely, as seen in Table 2

5.1 Access distribution

The access distribution of locks has a significant impact on the contention. A uniform distribution would equally distribute the contention equally over all items and would be the best for system performance. With an increase in skew of the distribution, the items locked more often tend to become bottlenecks [similar to the *convoy phenomenon* presented by Blasgen et al. (1979)]. The knowledge of the skew in real database systems should be useful for future modeling studies; here, we present the actual pattern of distribution of all locks over the data and index items in the database. In Fig. 2, we show the skew in the distribution separately over the index and data items [the data for this figure is provided by Singhal and Smith (1994)]. Table 2 gives the complete statistics about number of locks, type of locks and locktimes.

The three traces show differing amounts of skew. *Transport* appears to have the highest skew and *bank* appears to have the lowest. These plots indicate that the common 80–20 and 50–5 estimates both appear to be reasonable. The Root pages of the B-tree indexes form a significant portion of the total Cursor locks for *transport* and *bank* (even though the number of Root items is insignificant compared to the non-Root ones, as seen in Table 2); so in Fig. 2 we have also shown the skew for Cursor non-Root items. The figure shows that if we disregard the Root locks, the skew is much lower.

Table 2. Distribution of Locks among various types of lockable items. *Total items* denotes the total number of distinct items in the trace; since some items are locked as more than one lock type, this total is less than the sum of item total for the three lock types. The *upper percentages* in each box represent the ratio of the statistic to the sum of statistics in that column; the *lower percentages* are the horizontal percentages reflecting the contribution of one lock type among the three lock types. *L-time* refers to locktime in seconds

	Total	Cursor locks			Read locks			Write locks		
	items	Items	Locks	L-time	Items	Locks	L-time	Items	Locks	L-time
Transport										
Data	37956	37291	541262	17235.5	780	29189	75522.1	416	1476	2511.1
	73.9%	75.8%	80.1%	62.8%	53.0%	58.0%	64.3%	17.8%	27.8%	41.8%
	100.0%	98.2%	94.6%	18.1%	2.1%	5.1%	79.3%	1.1%	0.3%	2.6%
Index	255	233	74772	990.1	12	126	147.8	21	59	22.4
Root	0.5%	0.5%	11.1%	3.6%	0.8%	0.3%	0.1%	0.9%	1.1%	0.4%
	100.0%	91.4%	99.8%	85.3%	4.7%	0.2%	12.8%	9.2%	0.1%	1.9%
Index	13157	11644	59821	9211.0	680	21036	41783.0	1899	3783	3469.9
non-Root	25.6%	23.7%	8.9%	33.6%	46.2%	41.8%	35.6%	81.3%	71.1%	57.8%
	100.0%	88.5%	70.7%	16.9%	5.2%	24.9%	76.7%	14.4%	4.5%	6.4%
Phone										
Data	82859	82260	873599	16049.6	1367	4609	446.7	1413	9787	2266.2
	20.2%	20.4%	38.6%	50.6%	30.6%	37.1%	34.0%	13.6%	31.1%	22.1%
	100.0%	99.3%	98.4%	85.5%	1.7%	0.5%	2.4%	1.7%	1.1%	12.1%
Index	76	73	10493	7.9	2	4	0.9	7	16	2.9
Root	0.0%	0.0%	0.5%	0.0%	0.0%	0.0%	0.1%	0.1%	0.1%	0.0%
	100.0%	96.1%	99.8%	67.5%	2.6%	0.0%	7.7%	9.2%	0.2%	24.8%
Index	327872	320966	1379964	15678.5	3093	7820	865.8	8956	21599	7968.5
non-Root	79.8%	79.6%	60.9%	49.4%	69.4%	62.9%	65.9%	86.3%	68.8%	77.9%
	100.0%	97.9%	97.9%	64.0	0.9%	0.6%	3.5%	2.7%	1.5%	32.5%
Bank										
Data	96009	95974	134052	10038.4	35	82	16362.1	32	63	139.3
	60.9%	61.0%	42.0%	47.7%	26.9%	14.6%	66.2%	36.8%	34.1%	34.3%
	100.0%	100.0%	99.9%	37.8%	0.0%	0.1%	61.7%	0.0%	0.0%	0.5%
Index	59	57	93088	1287.9	3	4	0.0	0	0	0.0
Root	0.0%	0.0%	29.2%	6.1%	2.3%	0.7%	0.0%	0.0%	0.0%	0.0%
	100.0%	96.6%	93.4%	100.0%	5.1%	6.6%	0.0%	0.0%	0.0%	0.0%
Index	61500	61399	91858	9730.8	92	474	8365.2	55	122	266.4
non-Root	39.1%	39.0%	28.8%	46.2%	70.8%	84.6%	33.8%	63.2%	65.9%	65.7%
	100.0%	99.8%	99.4%	53.1%	0.1%	0.5%	45.7%	0.1%	0.1%	1.2%

Table 2 provides data about the number of lockable items in the database, and the distribution of the three lock types over these items. It also shows the extent of index locking. Judging from the number and locktimes of index locks in this table, it would appear that index locking should indeed be a part of concurrency control analyses.

5.2 Cursor locks

Cursor locks have not generally been modeled. This might be because it makes analytical models too complex or it might be because the use of these locks can lead to semantic inconsistency, since they allow non-two-phase locking and permit non-serializable behavior. In commercial systems, however, Cursor locks are indeed used, as is evident from Table 2. The table shows that Cursor locks dominate the total number of locks, accounting for greater than 96% of all locks in all cases, with the single exception of Index non-Root locks in *transport*, where they still account for more than 70% of the total. One may argue that Cursor locks are locked for relative insignificant amounts of times compared to other locks, and hence they can cause little contention. Looking at the table, this is clearly not the case, because the total of locktimes of Cursor locks is not insignificant when compared to Read and Write locktimes, even though the individual locktimes for Cursor locks might be more than an order of magnitude lower than that of Read and Write locks. It is important to note that while a Cursor lock may be only held for a short period of time, there may be a significant delay in obtaining a Cursor lock, during which time the transaction waits.

5.3 Correlation between lock types

In this subsection we will study whether for a certain lock item, the number of locks of one lock type is correlated with the number of lock types of another type. For example, a positive correlation between Cursor or Read locks and Write locks would indicate that we can expect more contention from Cursor-Write or Read-Write conflicts. The system performance is sensitive to the correlation between pairs of lock types; negative correlation is desirable for lower contention.

In the studies that have assumed a uniform access distribution of locks over all items, the item identifier for each lock request is chosen randomly and independently of anything else from the set of all items. Likewise, independently of the item, with a certain probability, the Lock request is a Write request. Thus there is no correlation between Read and Write locks. Studies which have modeled a non-uniform *b–c* distribution have also chosen Write requests with a probability independent of anything else. Thus, pages more likely to be Read-locked are also assumed to be more likely Write-locked.

In real systems, for an arbitrary item, the number of locks belonging to the three types may not be independent of one another. There may be objects which will almost always be read and rarely updated. On the other hand, for another set of objects, Write locks for that object may far outnumber the two other types.

To check whether there is any correlation between the lock types we look at correlation between ranks of the lock types; rank correlation is measured using the standard Spearman rank correlation test (Conover 1980). For each page we have measured three quantities – number of times it is Cursor-locked, Read-locked and Write-locked. In Table 3, we show the measured correlation between all three pairs of these quantities.

The null hypothesis we want to test is that each pair of quantities is mutually independent versus the alternate hypothesis that the two variables are either positively or negatively correlated. We reject the null hypothesis at 0.05 level of significance. Since a significant fraction of Cursor Index locks belong to Root pages for *transport* and *bank*, we have also included the test for non-Root pages for these traces. Note that even though the absolute correlation statistics are low, they are very significant statistically because of the very large number of objects. Statistical significance, however, does not imply that these low correlations significantly affect simulation results from experiments which do not take this correlation into account; using a contention model we have shown (Singhal and Smith 1994) that this correlation is significant enough to affect contention.

The statistic for *Cursor-Read* correlation is not important for contention as these two types of locks do not conflict. The statistics for all index objects and non-Root index objects are virtually the same, indicating that the large number of Cursor locks for Root pages do not affect the overall correlation. The table shows very significant positive correlations between Read-Write locks for all sets of objects except data objects for *bank*. This would indicate that Read-Write contention would be higher than if we assumed independence. However, since the correlation is much lower than one, there will be less contention than the studies which use a *b–c* distribution of data and choose Write locks using independent probability values (implying a high positive rank correlation between Read and Write locks). In the table, most Cursor-Write pairs have a significant negative correlation, which makes Cursor-Write contention less probable than for the independence assumption.

5.4 Write-lock assumption

The “Write-lock” assumption addresses the issue of whether Write locks are acquired directly or are upgraded from Read locks. Some studies (e.g., Ryu and Thomasian 1990) have assumed that Write locks are acquired directly. On the other hand, Agrawal et al. (1987) have argued that assuming that Write locks are acquired directly (which they call the *no-lock upgrades* assumption) is incorrect, and that performance results are sensitive to this assumption. The no-lock upgrades assumption leads to lower contention in both the blocking version and the restart version of locking algorithms. In the former, the assumptions prevents some deadlocks from happening; in the restart version, the transaction restarts which are inevitable, restart sooner if the no-lock upgrades assumption is made. However, the performance of the restart version improves more than the performance of the blocking version.

We compare the Write locks acquired directly with the Write locks upgraded from Read locks in Table 4. The table shows that most of the Write locks (approximately every nine out of ten Write locks) are acquired directly and the number of Write locks upgraded from Read locks is an order of magnitude lower. This is contrary to the assumption in some papers (e.g., Agrawal et al. 1987) that Write locks are always upgraded from Read locks. As discussed in the previous paragraph, our statistics would imply that studies which model Write locks through upgrades not only underestimate the system performance but also relatively underestimate the performance of locking with restarts more than that of locking with blocking.

6 Transaction behavior modeling

In this section, we study the modeling of transaction behavior. The issues involved are modeling of transaction lengths, locktimes and the distribution of the three kinds of locks and transaction classes.

An issue that we will not address in this section is modeling of resource processing. This refers to the breakdown of resource requirements (e.g., CPU and I/O requests) during a transaction lifetime, the sequence of these requests and the distribution of processing requirements. As noted earlier, this is outside the scope of this study and cannot be done properly with the data we have available.

6.1 Transaction length

The length of transactions is a very important factor in the analysis of database contention, since long transactions usually imply long lock-waits because Read and Write locks are released only when transactions end. Further, even the distribution of transaction lengths is important, since the system performance is sensitive to the second and third moments of transaction lengths (Thomasian 1993). This is because short transactions can be blocked for long periods by long transactions holding necessary locks.

The transaction length distribution affects the relative performance of the various concurrency control schemes as well as their absolute performance. In locking with restarts, since transactions restart on all lock conflicts, the fact that locktimes may be large for long transactions will have a less negative effect on performance than locking with blocking.

Most concurrency control studies have used fixed-length transactions. In Fig. 3, we characterize transaction lengths for our traces [the data for the plot is provided by Singhal and Smith (1994)]. Since a few researchers have used exponential and gamma distributions to model transaction lengths, we have also plotted both the exponential fits and

Table 3. Spearman test for checking independence between two different lock types

	Lock type pair	*No. of objects*	*Test statistic* $\hat{\rho}$	*.95 test quantile*	Reject H_0 ?
Transport					
Data	Cursor-Read	37956	0.1884	0.0101	Yes
	Cursor-Write	37956	0.2587	0.0101	Yes
	Read-Write	37956	0.1662	0.0101	Yes
Index	Cursor-Read	13412	–0.2094	0.0169	Yes
	Cursor-Write	13412	-0.3192	0.0169	Yes
	Read-Write	13412	0.2757	0.0169	Yes
Non-Root Index	Cursor-Read	13157	–0.2078	0.0170	Yes
	Cursor-Write	13157	–0.3185	0.0170	Yes
	Read-Write	13157	0.2778	0.0170	Yes
Phone					
Data	Cursor-Read	82859	0.0053	0.0068	No
	Cursor-Write	82859	–0.0033	0.0068	No
	Read-Write	82859	0.3591	0.0068	Yes
Index	Cursor-Read	327948	–0.0458	0.0034	Yes
	Cursor-Write	327948	–0.1828	0.0034	Yes
	Read-Write	327948	0.3291	0.0034	Yes
Bank					
Data	Cursor-Read	96009	–0.0110	0.0063	Yes
	Cursor-Write	96009	–0.0201	0.0063	Yes
	Read-Write	96009	–0.0003	0.0063	No
Index	Cursor-Read	61559	–0.0502	0.0079	Yes
	Cursor-Write	61559	–0.0269	0.0079	Yes
	Read-Write	61559	0.1789	0.0079	Yes
Non-Root index	Cursor-Read	61500	–0.0501	0.0079	Yes
	Cursor-Write	61500	–0.0269	0.0079	Yes
	Read-Write	61500	0.1818	0.0079	Yes

Table 4. Comparing Write locks acquired by upgrading Read locks to Write locks acquired directly. The latter is referred to as the no-lock upgrades assumption. *L-time* refers to locktime in seconds

Trace	*Data locks*				*Index locks*			
	Upgrade Write		*Direct Write*		*Upgrade Write*		*Direct Write*	
	Locks	*L-time*	*Locks*	*L-time*	*Locks*	*L-time*	*Locks*	*L-time*
Transport	48	178.67	1428	2332.45	377	1537.19	3465	1955.16
phone	1921	375.75	7866	1890.47	4023	1175.62	17592	6795.73
bank	0	0.00	63	139.28	14	1.36	108	265.03

the gamma distribution function[2] fits. Both these fits[3] have been generated using the method of *Maximum Likelihood* (Larson 1982). Our analysis of the statistical goodness of fit (Singhal and Smith 1994) shows that both fits are poor. In particular, both exponential and gamma distribution fits yield much lower figures for variance for all three traces than is measured.

6.2 Locktimes

As we have defined in Sect. 2.4, locktimes denote the length of time locks are held on the database items. For Cursor locks, both start and end points need to be known; for Read and Write locks, only start times are necessary, since both are released only at the end of the transaction.

[2] Gamma probability functions are a more general form of exponential probability functions. The density function of a gamma random variable X is defined as $f_X(x) = \frac{\lambda^n x^{n-1} e^{-\lambda x}}{\Gamma(n)}, x > 0$

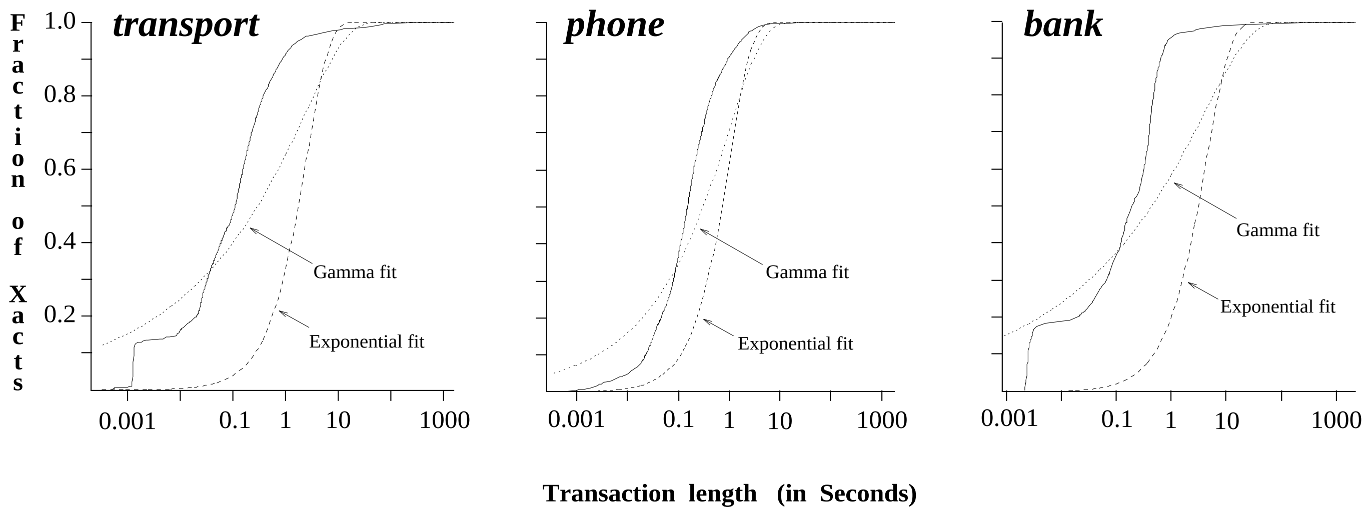
[3] The fits in Fig. 3 are not as poor as they appear to be – this is because the plots use a log scale and the fits have been obtained before taking the logarithms. It is possible to get exponential fits (by just horizontally shifting the current fits) which will look much closer on this log scale, but are actually much poorer fits, as it becomes evident if we look at those fits on a linear scale plot

In the literature, locktimes have not been explicitly modeled. In the static locking studies (Morris and Wong 1985), all locks were acquired at the beginning of transactions and thus the locktimes are identical to transaction lengths. However, many researchers have argued that static locking models are unrealistic because at the start of a transaction one may not know which pages are going to be locked. We know of no real database system which uses static locking. In most later studies, dynamic locking has been assumed, and in almost all those studies, Read/Write locks have been uniformly distributed over the entire transaction lengths.

There is no reason to believe that in real systems the locking epochs are uniformly distributed. We examine this distribution below.

6.2.1 Cursor locks

As we have discussed in Sect. 2.2, Cursor locks are very common, especially in very long transactions where serializability is traded off for higher performance. Even if Cursor locktimes are short, however, they cannot be ignored because of the possible waits to set cursor locks. It may, however, be reasonable to model Cursor lock times as zero, i.e., they



Trace	*Min.*	*Max.*	*Median*	*Mean*	*Std. dev.*	$E(T^2)$	$E(T^3)$	$E(T^4)$
Transport	0.000504	1525.37	0.110703	2.50572	32.19	1042.5	1.062×10^6	1.341×10^9
Phone	0.000214	1778.91	0.149048	1.04052	23.33	545.4	7.530×10^5	1.218×10^9
Bank	0.002182	2226.72	0.194977	4.47598	82.60	6842.8	1.364×10^7	2.843×10^{10}

Fig. 3. Transaction length distribution. The *dotted lines* show the fitted exponential and gamma distribution to the actual distribution. The last three columns in the table provide the 2nd, 3rd and the 4th moments of transaction lengths

are immediately unlocked after locking. We present the necessary measurements of Cursor locks in this subsection.

In Fig. 4, we report the distribution of a statistically representative sample of locktimes of Cursor locks [the data for the plot is provided by Singhal and Smith (1994)]. The plot shows that the locktimes for Cursor locks are more than an order of magnitude lower than the transaction lengths. This can also be observed from Table 2, where we see that the ratio of Read and Write locktimes to the total locktimes is at least an order of magnitude more than the ratio of number of Read and Write locks to the number of all locks. It should also be noted that for Cursor locks, a very small fraction of the locktimes are abnormally high and these locktimes distort the mean and variance of Cursor locktime distributions.

We also tested for independence between transaction length and Cursor locktimes using the Spearman rank correlation test. Since Cursor locks are acquired only to keep the Cursor pointer on the data stable while data is being accessed, we did not expect to see any dependence between transaction length and Cursor locktimes. However, as shown by the data (Singhal and Smith 1994) we found a significant positive correlation between the transaction length and locktimes for Cursor locks. This surprising observation could be due to the possibility that longer transactions are executed at lower priority.

If the long transactions do not lock a different set of Cursor objects than the short transactions, the effect of these significant correlations can be simulated in a model by parameterizing the distribution for Cursor locktime as a function of the length of the transaction that required it. However, if long transactions do lock a different set of Cursor objects, the lockable items have to be divided among the long and short transactions and the parameterization of Cursor locktimes has to be based on whether the object is more frequently locked by long transactions or short ones. This

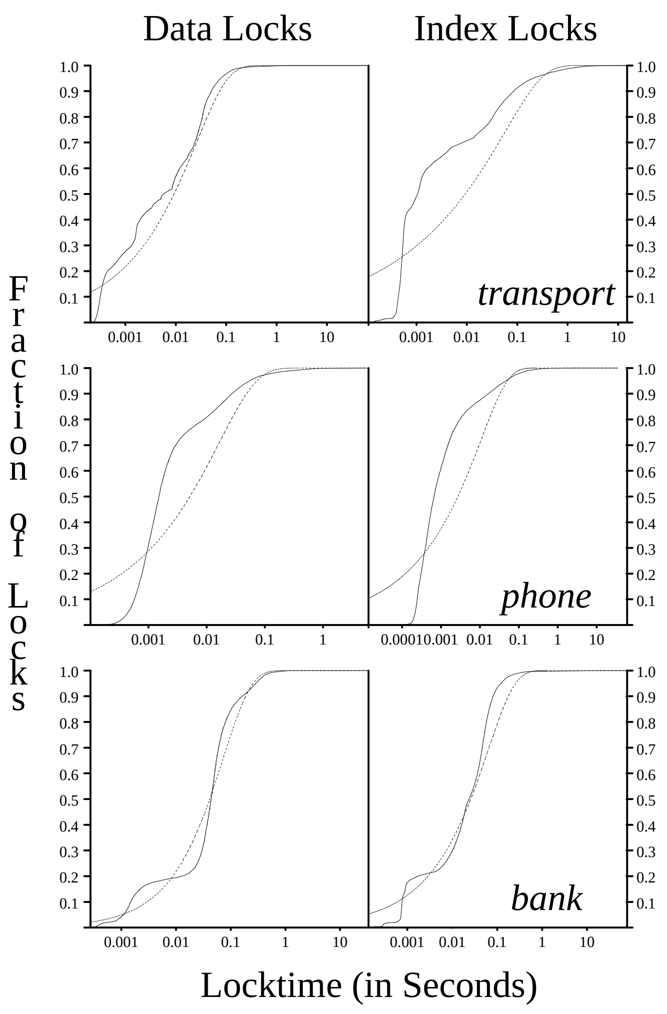


Fig. 4. Distribution of locktimes for Cursor locks. The *dotted lines* are the gamma distribution fits to the actual distribution

clearly requires more detailed database modeling – requiring us to model individual objects and associating them with long and short transactions. In real systems it is very reasonable to expect that long transactions lock a different set of Cursor objects than short transactions. Long transactions generally have a very different nature than short transactions – they access data in a sequential fashion, usually run at a low priority, and are usually batch-submissions (T. Messenger, private communication 1990). All this shows that it may be very difficult to accurately model Cursor locktimes, without resorting to using trace-driven simulations.

Since the locktimes for Cursor locks are much smaller than the transaction lengths and the locktimes for Read/Write locks, small errors in locktimes for Cursor locks should not affect the concurrency control performance analyses to a significant degree. In fact, as we discussed at the beginning of this section, a reasonable approximation would be to model Cursor locks with a locktime of zero – only the acquisition of Cursor locks is important. It would be interesting to investigate this further by experimentally determining the importance of accurate modeling of Cursor locktimes for the final performance results, but we do not do so in this paper.

6.2.2 Read/Write locks

For reasons discussed above, we want to know the distribution of locktimes for Read and Write locks. Since Read and Write locks are held until the end of the transaction, we want to test the assumption that these locks are uniformly distributed over the transaction length.

For reasons discussed in Sect. 2.4, we will characterize lockfractions instead of locktimes for Read and Write locks. By definition, the values of lockfraction lie between zero and one. Static locking is equivalent to assuming a lockfraction of one for all locks. On the other hand, assuming that locks are acquired uniformly over the transaction length means that the lockfraction is uniformly distributed over the interval $[0, 1]$.

In Fig. 5 we plot the distribution of lockfraction for Read and Write locks [the data for the plot appear in Singhal and Smith (1994)]. We also distinguish between index and data locks. Both the Read- and Write-lock acquisitions exhibit considerable skew relative to the uniform distribution assumption. This large variation in locktimes will cause more contention than would be the case for a uniform distribution of locktime, so we can expect that a naive model which assumes a uniform distribution to underestimate contention due to Read-Write lock conflicts.

It is useful to check whether the lockfractions are related to transaction lengths. Positive correlation between lockfractions and transaction lengths would indicate a high variance for locktimes. The results are reported by Singhal and Smith (1994): we observe a significant positive correlation in almost all cases for Cursor lockfractions; for Read and Write lockfractions, the results vary widely from positive to negative. In the literature, static locking models have assumed that all locks are acquired at the start of the transactions, implying a lockfraction of exactly one. More realistic dynamic locking models have assumed that locks are acquired uniformly over the transaction duration. This would again mean

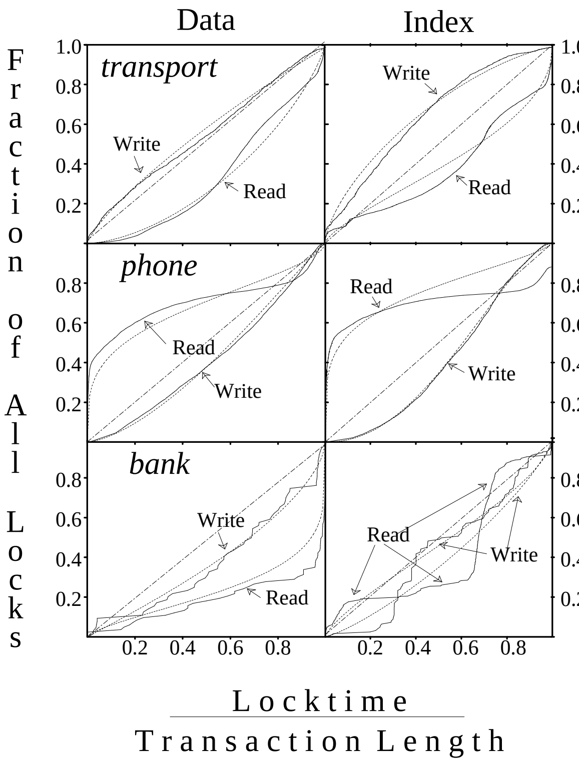


Fig. 5. Distribution of locktimes for Read/Write locks. The *dotted lines* are the beta distribution fits to the actual distribution. The *dashed line* represents the line with slope one

that they have implicitly assumed that lockfractions are independent of transaction lengths. Given the wide swings between positive and negative correlations, assuming no correlation is probably a reasonable assumption for most studies.

6.3 Transaction classes

Transaction classes refer to the different kinds of transaction applications running on the system. Most studies have used only one transaction class. One of the few studies that use multiple transaction classes – a mix of short transactions with random access and higher probability of Write with long transactions doing sequential access with a smaller probability of Write – is that conducted by Carey and Stonebraker (1984). If real transaction behavior varies widely, it is important for the models of many studies to realistically represent this variation, since most performance measures decrease with increased variance in the workload. This can be done by incorporating multiple classes of transactions, each class with significantly different behavior. Using a low variance workload also gives an unfair advantage to the blocking version of locking because there is a shorter bound on the amount of lock-waits, due to low variance in transaction lengths. Another reason for modeling multiple transaction classes is that in the presence of multiple transaction classes, only a subset of them may become the contention bottleneck; modeling a single transaction class means that when contention becomes the bottleneck, no transactions can progress. We also refer the reader to the work of Thomasian (1994), where it has been shown that the effect of multiple transaction classes cannot be simulated using a single transaction class.

The characterization for the transactions present in our workload appears in a set of tables in Singhal and Smith (1994). There are two classifications. The first is based on the application plan names as given by the system; the second classification is based on three factors: transaction length, Read/Write nature of transactions and two-phase versus non-two-phase transactions. These characterizations can be used to guide a realistic selection of parameters for concurrency control analysis.

6.3.1 Plan-based characterization

Each transaction in our traces has a *plan name* attached to it. This plan name refers to the application type of the transaction. We first classified the various transactions based on their plan names. The quantities measured for the transactions include transaction length, Write locktimes and Read locktimes. For each plan type, and for both index and data items, we tabulate the number of Read locks acquired, number of Write locks acquired, number of Read locks upgraded to Write, number of locks unlocked and number of distinct items locked. It is important to distinguish the Write locks directly acquired from the Write locks upgraded through Read locks because, as discussed in Sect. 5.4, this distinction can significantly impact simulation results.

Not surprisingly, we find very high variation between the characteristics of the different applications. Average transaction lengths vary a lot over different plans. An interesting observation is the fact that, for most plans, the number of locks acquired by transactions outnumbers the number of distinct items locked. Since Read and Write locks cannot be released before Commit, this fact can be attributed to Cursor locks being acquired on the same items repeatedly.

6.3.2 Another transaction grouping

The plan-based characterization described above exhibits high variability within many plans (i.e., groups). To get a better characterization for modeling, we can group the transactions on three mutually orthogonal axes:

1. Two-phaseness: since two-phaseness of transactions is an important property, we use that as one of attributes for our grouping.
2. Read-Write behavior: a large fraction of the transactions do not acquire Write locks. These transactions may have behavior dissimilar from the transactions that acquire Write locks. Therefore, we distinguish between these two classes of transactions.
3. Length of transactions: we have also divided the transactions into classes based on their length.

In our previous publication (Singhal and Smith 1994), we provide tables which characterize each of our workloads based on the above three parameters. The statistics from these tables provide the parameters which one could use to realistically model a database model.

On the average, over all traces, Read transactions greatly outnumber the Write transactions. Read transactions have a greater variance of transaction lengths, number of locks and locktimes. Two-phase transactions outnumber the non-two-phase transactions. The non-two-phase transactions are much longer and acquire many more locks (mostly Cursor) than two-phase transactions.

7 Conclusion

In this paper we have looked at the locking behavior of three real relational database systems running three different kinds of real applications. We have used our traces to obtain a comprehensive characterization of transaction workloads in real commercial database systems. We believe that this data will be valuable not only to the researchers who can use it to create more accurate models, but also to database designers who can use this information for a deeper understanding of real-world transaction workloads. It is important to note, however, that our characterization does not necessarily have predictive power; that is, we have measured various parameters of the workload, but that does not allow us to be confident that we can predict the values of parameters that were not measured.

We have also looked at the various assumptions made in the concurrency control performance analysis literature, have measured their validity in the traces, and have analyzed the sensitivity of predicted system performance to these assumptions. A future direction of research would be to use trace-driven simulation to analyze the sensitivity of the assumptions more rigorously.

Acknowledgements. We would like to acknowledge Ted Messenger of IBM Almaden Research Center and Hervé Touati of University of California at Berkeley for their help in making the DB2 traces available to us. Also, we thank Steve Luzmoor of Cray Research for helping us transfer some of the traces to our machines. This research has been supported in part by the National Science Foundation under grants MIP-9116578 and CCR-9117028, by NASA under grant NCC 2-550, by the State of California under the MICRO program, and by the International Business Machines Corporation, Digital Equipment Corporation, Mitsubishi Electric Research Laboratories, Sun Microsystems, Philips Laboratories/Signetics, Intel Corporation, and Apple Computer Corporation.

References

1. Agrawal R, Carey MJ, Livny M (1987) Concurrency control performance modeling: alternatives and implications. ACM Trans Database Syst 12(4): 609–654
2. Blasgen M, Gray J, Mitoma M, Price T (1979) The convoy phenomenon. Oper Syst Rev 13(2): 20–25
3. Carey MJ, Livny M (1988) Distributed concurrency control performance: a study of algorithms, distribution and replication. In: Proc of the 14th Intl. Conf. on Very Large Database Systems, Los Angeles, California
4. Carey MJ, Stonebraker M (1984) The performance of concurrency control algorithms for database management systems. In: Proc of the 10th Intl. Conf. on Very Large Database Systems, Singapore
5. Carey MJ, Krishnamurthy S, Livny M (1990) Load control for locking: the “Half and Half” approach. In: Proc of the 9th ACM Symp. on Principles of Database Systems, Nashville, Tennessee
6. Conover WJ (1980) Practical nonparametric statistics. Wiley, New York

7. Dan A, Dias DM, Yu PS (1994) Buffer analysis for a data sharing environment with skewed data access. IEEE Trans Knowl Data Eng 6(2): 331–337
8. Date CJ (1987) An introduction to database systems (vol. II). Addison-Wesley, Reading, Mass
9. Date CJ, White CJ (1993) A guide to DB2: a user's guide to the IBM product IBM DATABASE 2 and its major companion products, 4th edn. Addison-Wesley, Reading, Mass
10. Gray J, Reuter A (1993) Transaction processing: concepts and techniques. Morgan Kaufmann, San Mateo, Calif
11. Hawthorn P, Stonebraker M (1979) Performance analysis of a relational data base management system. In: Proc of the ACM SIGMOD Int Conf on Management of Data, Boston, Mass
12. IBM Corporation IBM Database 2: system planning and administration guide (SC26-4085-3) (1987) IBM Corporation, White Plains, New York, May
13. Johnson T, Shasha D (1993) The performance of current B-tree algorithms. ACM Trans Database Syst 18(1): 51-101
14. Kearns JP, DeFazio S (1989) Diversity in database reference behavior. Performance Eval Rev 17(1): 11–19
15. Kleinrock L (1975) Queueing systems (vols I, II) Wiley, New York
16. Knuth DE (1973) The art of computer programming (vol III). Addison-Wesley, Reading, Mass
17. Larson HJ (1982) Introduction to probability theory and statistical inference. Wiley, New York
18. Mohan C, Haderle D, Lindsay B, Pirahesh H, Schwarz P (1992) ARIES: a transaction recovery method supporting fine-granularity locking and partial rollback using Write-Ahead locking. ACM Trans Database Syst 17(1): 94–162
19. Monkeberg A, Weikum G (1991) Conflict-driven load control for the avoidance of data-contention thrashing. In: Proc of the 7th Int Conf on Data Engineering, Kobe, Japan
20. Morris RJT, Wong WS (1985) Performance analysis of locking and optimistic concurrency control algorithms. Performance Eval 5(2): 105–118
21. Ries DR, Stonebraker MR (1979) Locking granularity revisited. ACM Trans Database Syst 4(2): 210–227
22. Rodriguez-Rosell (J (1976) Empirical data reference behavior in data base systems. IEEE Comput 9(11): 9–13
23. Ryu IK, Thomasian A (1990) Analysis of database performance with dynamic locking. J ACM 37(3): 491–523
24. Singhal V, Smith AJ (1994) Characterization of contention in real relational databases. (Technical report no UCB/CSD 94/801) Computer Science Division, University of California, Berkeley, March
25. Smith AJ (1978) Sequentiality and prefetching in data base systems. ACM Trans Database Syst 3(3): 223–247
26. Srinivasan V, Carey MJ (1991) Performance of B-Tree concurrency control algorithms. In: Proc of the ACM SIGMOD Int Conf on Management of Data, Denver, Colorado
27. Tay YC, Goodman N, Suri R (1985) Locking performance in centralized databases. ACM Trans Database Syst 10(4): 415–462
28. Thomasian A (1993) Two-phase locking performance and its thrashing behavior. ACM Trans Database Syst 18(4): 579–625
29. Thomasian A (1994) On a more realistic lock contention model and its analysis. In: Proc of the 10th Int Conf on Data Engineering, Houston, Texas
30. Verkamo AI (1985) Empirical results on locality in database referencing. In: Proc of the ACM SIGMETRICS Conf, Texas, Austin
31. Viavant S (1989) Collection, reduction and analysis of DB2 trace data. In: MS Report, University of California, Berkeley, California, July
32. Wang Y, Rowe LA (1991) Cache consistency and concurrency control in a client/server DBMS architecture. In: Proc of the ACM SIGMOD Int Conf on Management of Data, Denver, Colorado
33. Yu PS, Dias DM, Robinson JT, Iyer BR, Cornell DW (1985) Modeling of Centralized Concurrency Control in a Multi-System Environment. In: Proc of the ACM SIGMETRICS Conf, Austin, Texas
34. Yu PS, Cornell DW, Dias DM, Thomasian A (1986) On coupling partitioned database systems. In: Proc of the 6th Ann Symp on Distributed Computing, May
35. Yu PS, Dias DM, Robinson JT, Iyer BR, Cornell DW (1987) On coupling multi-systems through data sharing. Proceedings of the IEEE 75(5): 573–587
36. Yu PS, Dias DM, Lavenberg SS (1993) On the analytical modeling of database concurrency control. J ACM 40(4): 831–872