INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Performance analysis of database systems under non-uniform data access distribution

Yesha, Yelena, Ph.D.
The Ohio State University, 1989
PERFORMANCE ANALYSIS OF DATABASE SYSTEMS UNDER NON-UNIFORM DATA ACCESS DISTRIBUTION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By
Yelena Yesha, B.Sc., M.Sc.

* * *

The Ohio State University
1989

Dissertation Committee:
Dr. Mukesh Singhal
Dr. Ming T. Liu
Dr. Dik Lee

Approved by
Advisor
Department of Computer and Information Science
ACKNOWLEDGEMENTS

I would like to thank Dr. Mukesh Singhal for introducing me to the area of databases, and for his help and guidance. He provided me with constant encouragement and valuable guidance throughout my graduate years at The Ohio State University. He read promptly and carefully many drafts of the dissertation, and his suggestions greatly contributed to the final form of the dissertation.

I would also like to express my sincere gratitude to Dr. Mike Liu. His thorough reading of this dissertation resulted in many useful comments which helped in substantially improving the presentation of the research results. In particular, he suggested to add an induction based verification of the computational technique.

My thanks also go to Dr. Dik Lee. He patiently read this dissertation and provided valuable suggestions and constructive criticism.

Finally, I thank all members of my family for their love and encouragement.
VITA


June, 1984 ........................................................Honors Bachelor of Science in Computer Science. 
York University, Toronto, Canada.

June, 1984 ........................................................Honors Bachelor of Science in Applied Mathematics. 
York University, Toronto, Canada.

June, 1986 ........................................................M.Sc., Computer and Information Science, The Ohio State University

PUBLICATIONS


4-7, 1987).


**FIELD OF STUDY**

Computer and Information Science
TABLE OF CONTENTS

ACKNOWLEDGEMENTS ...................................................................................................ii

VITA .......................................................................................................................................iii

TABLE OF CONTENTS .......................................................................................................v

LIST OF TABLES .............................................................................................................viii

LIST OF FIGURES ...............................................................................................................x

CHAPTER PAGE

I. INTRODUCTION ..........................................................................................................1

1.1 Motivation for Performance Analysis of Concurrency Control

Algorithms ....................................................................................................................3

1.2 Motivation for the Dissertation .............................................................................4

1.3 Summary of Research Results .............................................................................6

1.3.1 An Efficient Algorithm ................................................................................7

1.3.2 Performance Analysis of Static Locking .............................................8

1.3.3 Performance Analysis of Two-Phase Locking .........................................9

1.4 Organization of the Dissertation .........................................................................10

II. SURVEY OF PREVIOUS WORK ...........................................................................12

2.1 The b-c Access Model ..........................................................................................12

2.2 The Multiple Class Model ..................................................................................15
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Probability of Conflict (for ( \mu = 500 ))</td>
<td>31</td>
</tr>
<tr>
<td>2. Results for Probability of not Blocking (( \mu=500, n=10 ))</td>
<td>33</td>
</tr>
<tr>
<td>3. Results for Probability of not Blocking (( \mu=500, t=5 ))</td>
<td>34</td>
</tr>
<tr>
<td>4. The Probability of Conflict among ( 't' ) Transactions (for ( \mu = 500 ))</td>
<td>63</td>
</tr>
<tr>
<td>5. The Probability of a Transaction not Blocking (for ( \mu = 500 ))</td>
<td>64</td>
</tr>
<tr>
<td>6. Convergence for Throughput</td>
<td>76</td>
</tr>
<tr>
<td>7. The Probability of Conflict (for ( \mu = 500, K=8, M=500 ))</td>
<td>77</td>
</tr>
<tr>
<td>8. System Throughput - I</td>
<td>78</td>
</tr>
<tr>
<td>9. The probability of Conflict (for ( \mu = 500, M=1000, t=4 ))</td>
<td>80</td>
</tr>
<tr>
<td>10. System Throughput - II</td>
<td>81</td>
</tr>
<tr>
<td>11. Response time (( K=8, \mu =500, M=500 ))</td>
<td>82</td>
</tr>
<tr>
<td>12. Response time (( tr=4, \mu =500, M=1000 ))</td>
<td>82</td>
</tr>
<tr>
<td>13. Throughput Convergence</td>
<td>92</td>
</tr>
<tr>
<td>14. Probability of Deadlock (( \mu=500, K=8, M=500 ))</td>
<td>95</td>
</tr>
<tr>
<td>15. The Probability of Blocking (( \mu=500, K=8, M=500, S=0.1 ))</td>
<td>96</td>
</tr>
<tr>
<td>16. System Throughput (( K=8, \mu =500, M=500 ))</td>
<td>97</td>
</tr>
<tr>
<td>17. The Probability of Blocking (( tr=4, \mu =500, M=1000 ))</td>
<td>98</td>
</tr>
<tr>
<td>18. System Throughput (( tr=4, \mu =500, M=1000 ))</td>
<td>99</td>
</tr>
</tbody>
</table>
19. The Response Time ($t_r=4$, $\mu=500$, $M=1000$)……………………………………………………………101

20. The Response Time ($K=8$, $\mu=500$, $M=500$)……………………………………………………………101
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. A Possible Data Access Distribution</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2. The b-c Data Access Model</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3. The Multiple Class Model</td>
<td>16</td>
</tr>
<tr>
<td>Figure 4. Generalized Data Access Model</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5. The Queueing Network Model of a Database</td>
<td>21</td>
</tr>
<tr>
<td>Figure 6. The Queueing Network Model</td>
<td>72</td>
</tr>
<tr>
<td>Figure 7. The Markov Chain</td>
<td>73</td>
</tr>
<tr>
<td>Figure 8. System Throughput - I</td>
<td>79</td>
</tr>
<tr>
<td>Figure 9. System Throughput - II</td>
<td>81</td>
</tr>
<tr>
<td>Figure 10. Response Time - I</td>
<td>83</td>
</tr>
<tr>
<td>Figure 11. Response Time - II</td>
<td>83</td>
</tr>
<tr>
<td>Figure 12. The Queueing Network Model of the System</td>
<td>88</td>
</tr>
<tr>
<td>Figure 13. System Throughput - III</td>
<td>98</td>
</tr>
<tr>
<td>Figure 14. System Throughput - IV</td>
<td>100</td>
</tr>
<tr>
<td>Figure 15. Response Time - III</td>
<td>103</td>
</tr>
<tr>
<td>Figure 16. Response Time - IV</td>
<td>103</td>
</tr>
</tbody>
</table>
A database is a collection of related data objects, which is shared by several users with potentially diverse interests. A user interacts with a database by executing transactions which consist of a sequence of read and write actions on the data objects of the database. Two transactions conflict if one attempts to write a data object which is being read or written by another transaction.

In a database system, several users concurrently access the data objects by running transactions. Since the actions of concurrently running transactions may access the same data objects, some control is necessary to coordinate the concurrent access to the database to ensure its consistency (the problem is known as concurrency control). One way to resolve the concurrency problem is by executing transactions serially, i.e., one at a time. However, a serial execution of transactions is very inefficient: it results in poor utilization of system resources and poor response time to user requests.

Database systems handle this problem by using concurrency control algorithms which achieve the coordination by having a transaction block other concurrent conflicting transactions so that they cannot see some data objects left temporarily inconsistent by the former transaction. Most concurrency control algorithms use
locking [8], timestamps [2], or transaction restart [18] to serialize conflicting transactions accesses to data objects. In locking based algorithms a transaction must lock a data object before accessing it. A transaction is allowed to lock a data object if it is not already locked by some other transactions. In timestamp based algorithms, every site maintains a logical clock which is incremented by one when a transaction arrives at that site and is updated whenever the site receives a message with higher clock value. Each transaction is assigned a unique timestamp. Conflicting actions are resolved according to the timestamps of their transactions. In transaction restart based algorithms a transaction which suffers a conflict gets aborted and resubmits its requests to the system [2].

Consequently, if a transaction requests a data object which is currently held by another transaction (that is, a conflict occurs), it waits (i.e., gets blocked) until the requested data object is released. The probability of blocking due to conflicts and the length of the time a transaction has to wait in case of a conflict (called blocking delays) are parameters of great importance in performance analysis of concurrency control algorithms. Note that the probability of a request getting blocked depends upon the data access distribution of transaction requests. (In addition, it depends upon the transaction size, the number of granules or data objects in the database, and the number of concurrently running transactions.)
1.1 Motivation for Performance Analysis of Concurrency Control Algorithms

Although the correctness of concurrency control algorithms, particularly locking, is very well understood, there were not many studies done in the area of their performance. The area of performance of concurrency control algorithms is very important. The reason is that the performance of a database system is greatly determined by its underlying concurrency control algorithm because it determines the performance degradation in presence of software contention. With many alternatives now available to design a database system, there is a great need to assess the suitability of a design to a particular environment and need.

Most of the previous performance studies of concurrency control algorithms have been done at a very high level and the results obtained have been very contradictory. For example, the performance of locking algorithms has been compared to that of optimistic algorithms, and the results in [22] show that optimistic algorithms outperform locking algorithms. However, completely opposite results were reported in [41]. Studies reported in [3] suggest that an algorithm that uses blocking instead of restarts has a better performance, but studies reported in [12] suggest that better performance can be achieved by using restarts. This incoherence of the results is due to the fact that almost every study made its own unique set of assumptions regarding database system resources, transaction behavior and many other important system related parameters. The behavior of locking has been observed using different analytical models [19, 12, 38, 41, 25, 23].
The majority of analytical models of locking algorithms use queueing networks or Markov processes. Most of them make the following assumptions: data access is uniformly distributed, there is only one transaction class, transactions predeclare their locks, transactions have fixed lengths, the locks may not be shared, etc. Even with the above simplification of the transaction model, however, the existing analytic models have the following shortcomings: some work best when a transaction requires to lock a single data object; some discard a transaction when it is blocked; some guarantee increasing throughput even under heavy workload. The aforementioned problems demonstrate the difficulty of modeling locking behavior analytically.

It is very important that the performance of concurrency control algorithms should be studied in an environment that emulates database systems in the real world, one in which the system assumes non-uniform access to data objects [43, 5].

1.2 Motivation for the Dissertation

In real-life database systems, it is likely that some data objects are accessed more frequently than others. However, past performance analyses of concurrency control algorithms have assumed that data access distribution of transaction requests is uniformly distributed across the entire database (e.g., [4, 10, 11, 22, 41, 25, 30, 38]). This assumption has largely been adopted because computation of the probability of conflict under an arbitrary data access distribution is very expensive (due to combinatorial explosion) even for a database of small size. Although this assumption simplifies the analysis considerably, it is unrealistic because it fails to capture the
locality of data reference of transactions [33,40].

Intuitively, one can visualize that the assumption of uniform data access distribution should give a lower bound on the probability of conflict among transactions (i.e., it results in the least number of conflicts among transactions). This is because data access of transactions is spread over the entire database as much as possible. It has been formally shown that this assumption indeed gives a lower bound for the probability of conflict (or equivalently, upper bound on the probability of no conflict) among transactions [29]. Since having more conflicts causes more blocking (thus, a degradation in performance), an important implication of this result is that previous performance studies of concurrency control algorithms have computed upper and lower bounds on the performance measures [31]. Therefore, elimination of the uniform data access distribution assumption is essential for more realistic performance analysis of concurrency control algorithms.

In real-life database systems, transactions exhibit locality of data reference [33,40]. Therefore, it is important that the assumption of uniform data access distribution be eliminated for a more realistic performance analysis of concurrency control algorithms. In this dissertation we remove this assumption from the performance analyses of concurrency control algorithms and also apply the technique developed to performance studies of static and two-phase locking algorithms.

Note that the data access distribution of transactions affects the performance of concurrency control algorithms by affecting the probability of conflict (or blocking) among transactions; therefore, elimination of the assumption of uniform data access
distribution from the performance analyses of concurrency control algorithms essentially entails computing the probability of conflict among transactions under non-uniform data access environment.

1.3 Summary of the Research Results

In Chapter 3 of this dissertation, we discuss a model of a database system and develop a taxonomy. We also derive an equation for the probability that an arriving transaction does not block due to conflicts with already running transactions and show that it is extremely computation-intensive in case of non-uniform data access. Then we present an algorithm for computing the probability of no conflict between two transactions. This algorithm has polynomial complexity. This algorithm is then extended to the case of multiple transactions. We illustrate the algorithms using numerical examples. The numerical results confirm that the assumption of uniform data access distribution gives an optimistic estimate (that is, a lower bound) of the probability of conflict. We also extend our results to variable transaction size.

In Section 3.5 of this dissertation we introduce an approximate computational technique, which enables us to compute the probability of conflict among transactions very efficiently. The approximate technique is illustrated using numerical examples. Then, we present an efficient (i.e., polynomial time) algorithm for computation of the probability of conflict among transactions for databases where data access distribution is arbitrary. The algorithm permits inclusion of any data access distribution in the performance analyses of concurrency control algorithms at a moderate cost of
computation. We illustrate the algorithm by using a numerical example to compute the probability of blocking of a transaction under non-uniform data access distribution. A comparison of the results for non-uniform and uniform data access distributions indicates that the assumption of uniform access distribution can introduce significant error in the probability of transaction blocking.

In Chapters 5 and 6 we apply the technique developed in Section 4.2 to performance analysis of static and two-phase locking algorithms respectively, and show that the equations for such quantities as the probability of lock being granted, the mean wait time for a blocked request (blocking delay) and response time can be derived analytically. The technique developed is also applied to computation of the probability of deadlock under the assumption of arbitrary data access distribution.

1.3.1 An Efficient Algorithm

Past performance analyses of concurrency control algorithms have assumed that data access distribution of transactions is uniformly distributed across the entire database. The reason was that the amount of computation required to accurately capture any arbitrary data distribution was not justified due to high computational costs. Since in practice transactions exhibit locality of data reference, elimination of the assumption of uniform data access distribution is essential for more accurate performance analysis of real-life databases. In this dissertation, we present an efficient algorithm for computing the probability of transaction blocking under arbitrary data access distribution. This algorithm makes it possible to accurately
reflect any data access distribution in the performance analysis of concurrency control algorithms at a moderate cost of computation.

We illustrate the applicability of the algorithm by using numerical examples in which we compute the probability of conflict among \( t \) transactions and the probability that an arriving transaction does not get blocked due to conflicts. The examples confirm that the assumption of uniform data access distribution indeed underestimates the probability of conflict and the probability of blocking, and in some cases the error can be substantial. The results of the numerical studies also show that the probability of a transaction not blocking is more sensitive to the number of concurrently running transactions when data access distribution is non-uniform as compared to the case when data access distribution is close to uniform. When the data access distribution is closer to uniform, the fall in the probability of not blocking is much steeper as the number of running transactions increases.

1.3.2 Performance Analysis of Static Locking

In order to analyze the behavior of locking mechanisms under more realistic conditions, we present in this dissertation a performance study of locking mechanisms under the assumption of non-uniform data access distribution. The main contribution of this analysis is the use of analytic techniques as opposed to simulation techniques to study the problem. We present a simple, yet very powerful queueing network model and show that the values for such system performance measures as throughput, response time and blocking delays can be derived analytically. The
analyses are straightforward and easy to understand.

The study presented is the first one which incorporates the non-uniformity of data access in the model without making oversimplified assumptions. The analytical results obtained confirm the fact that the incorporation of non-uniformity is essential for more realistic and accurate studies of the performance analysis of concurrency control algorithms.

1.3.3 Performance Analysis of Two-Phase Locking

We extend our performance study to two-phase locking policy (where locks on data objects are acquired as and when needed during the transaction execution). We present a queueing network model for a two-phase locking mechanism and derive analytical equations for quantities such as the probability of lock being granted, the probability of deadlock, mean wait time for the blocked request (blocking delay) and throughput of the system. In our analysis we assume a wide variety of realistic conditions such as: a non-uniform data access distribution, variable transaction size, different database size. The model presented is simple, but at the same time very powerful. We also discuss the merits of the numerical results and the implication that non-uniformity of data access has on the performance of two-phase locking. The main contribution of the analysis is the use of analytic techniques and incorporation of non-uniformity of data access in the model.
1.4 Organization of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2 we present the existing data access models and describe how they were used in the performance studies of concurrency control algorithms.

In Chapter 3, we develop a necessary taxonomy and present a database model under non-uniform data access distribution. We derive an equation for the probability that an arriving transaction does not block due to conflicts with already running transactions and show that it is extremely computation-intensive in case of arbitrary data access. Next, we present a polynomial algorithm to compute the probability of conflict between two transactions under arbitrary data access distribution. We illustrate the algorithm using numerical examples. We also present an efficient approximate technique for the computation of the probability of conflict between transactions under arbitrary data access distribution.

In Chapter 4, we present an efficient algorithm for computation of the probability of conflict among transactions for databases in which data access distribution is non-uniform. The algorithm is extended to the cases of variable transaction size and dynamic acquisition policy. We illustrate the algorithm by using a numerical example to compute the probability of blocking of a transaction under non-uniform data access distribution.

In Chapter 5, we present the analysis of static locking mechanism under non-uniform data access. We describe a queueing network model of static locking and
derive equations for system performance measures such as throughput, response time and blocking delays. The numerical results obtained from the analytical study confirm that the incorporation of non-uniformity in the performance study of locking mechanisms is essential for more accurate and realistic analysis.

In Chapter 6, we extend our performance study to two-phase locking policy. We present a queueing network model for a two-phase locking algorithm and derive equations for system performance measures and the probability of deadlock. We also provide a discussion of the numerical results.

Chapter 7 contains concluding remarks and some directions for future research in the area of performance analysis of database systems.
CHAPTER II
SURVEY OF PREVIOUS WORK

In this chapter we examine several models of arbitrary data access distributions and discuss their merits and limitations. We describe the existing models of arbitrary data access and a tradeoff between accuracy of non-uniform data access models and the computational cost of their analysis. The more accurate and powerful a model of non-uniform access is, the more computations its analysis requires.

Very few performance studies of database systems have incorporated non-uniform data access distribution in their analyses [4, 5, 24, 43]. The previous performance studies have modeled non-uniform data access by using the "b-c access model", the "multiple class model" and the "generalized access model". In the next sections we will describe several models, which are used to represent non-uniform data access and discuss how they are used in the performance analysis of database systems.

2.1 The b-c Access Model

For a real-life database system it is possible to assign to each database object a relative frequency access value by using the following technique: we would measure the number of accesses to the database object for a sufficiently long period and then
divide this number by the total number of accesses to the database over that period. Although it is not difficult to compute the relative frequency of access for existing databases, it is fairly difficult to incorporate this information into analytical performance models of database systems.

Figure 1 is a representation of the possible data access distribution. Along the horizontal axis are data objects, which are numbered 1 to M, and along the vertical axis is the relative frequency of accesses of the data objects of the database.

One of the 'simplest' models of non-uniform data access distribution is the b-c access model, which captures non-uniformity at a very coarse level. In the b-c

![Figure 1. A Possible Data Access Distribution](image-url)
access model, a request accesses a fraction \( c \) of data objects with probability \( b \). Basically, in this model we are dividing data objects of the database into two groups: one contains \( c \times 100 \) percents of the data objects of the database and the other contains \( (1-c) \times 100 \) percents of the data objects of the database.

Thus, \( (1-b) \times 100 \) percents of the requests for data objects are done within \( (1-c) \times 100 \) percents of the data objects. Inside each group, each data object has the same relative frequency of access. It is clear from the description that the \( b-c \) access model is an approximation and does not capture non-uniformity at a very fine level. The behavior of the \( b-c \) access model is depicted in Figure 2.

![Figure 2. The b-c Data Access Model](image.png)
The b-c access model was first used by Lin and Nolte and by Munz and Krenz in their performance studies of concurrency control algorithms [19,24]. Later on it was used by Tay, Suri and Goodman [34]. In their analysis they used the so called "80-20 rule", i.e., they made an assumption that 80 percent of the data is accessed with 20 percent probability. This assumption was incorporated in the computation of the probability of conflicts among the running transactions. The analytic results obtained have shown that non-uniformity has a degrading effect on systems performance measures, such as throughput, response time and blocking delay.

Although the b-c model is an approximation and captures non-uniformity at a very coarse level, it still enables us to visualize the effect that the non-uniform data access distribution has on the performance of database systems.

2.2 The Multiple Class Model

In the multiple class model (see Figure 3), M data objects are divided into r classes $D_1, D_2, ..., D_r$, and a request accesses a data object in class $D_i$ with the probability $q_i$, for $i=1,2,...,r$, and $\sum q_i=1$. Within each class the data is uniformly accessed. There are two special cases: $r=1$ (the data access is uniform over the entire database) and $r=M$ (the data access is completely non-uniform). Basically, the multiple class model is a generalization of the "b-c access model".

The multiple class model was used by Chesnais, Gelenbe and Mitrani in [4].
effect on system performance. However, the multiple class model is helpful and easy to use—when the number of classes is small. Otherwise we have to deal with expressions which have exponential complexity.

It is clear from the description above that the multiple class model is an approximation, but it captures non-uniformity of data access at a much finer level and with much higher accuracy than the b-c access model. The price paid for this increased accuracy is additional computational complexity. It is possible to increase accuracy by increasing the number of classes, but this will correspondingly increase the cost of computation. Thus, there is a tradeoff between accuracy and the cost of

![Diagram](image)

Figure 3. The Multiple Class Model
In the next section, we will discuss a generalized data access model in which we assign to each data object a probability of access (i.e., data object $d_i$ is accessed with probability $p_i$, or equivalently $r=M$ in the multiple class model).

### 2.3 Generalized Data Access Model

In the generalized data access model (see Figure 4), the data access distribution of user requests across the database is specified by a probability vector $\overline{p} = (p_1, p_2, ..., p_M)$, where $p_i$, $1 \leq i \leq M$, is the relative frequency of access of data object $d_i$ by the transactions. Upon arrival in the system, the transaction attempts to lock the required data objects selected from the $M$ data objects according to access distribution $\overline{p}$ before it starts executing (that is, we make an assumption of static locking). The probability that a data object is accessed by a transaction is independent of the probability of other data objects accessed by that transaction.

The generalized data access model was first used by Christodoulakis in [5], where he formally examines the implications of non-uniformity and dependencies of attribute values on database design and database performance evaluation. Later on Sevcik, Zahorjan and Bell used the generalized data access model to study the effect of non-uniformity on estimating the number of block transfers [43]. Their studies concluded that the estimation of the number of blocks transferred based on assumption of uniform access probabilities can be substantially in error when access
probabilities are not in fact uniform. Their technique overcomes this inaccuracy at the price of a moderate increase in computations.

However, due to the complexity of the analysis, no studies have been carried out to incorporate the generalized data access model into the analysis of transaction blocking and computation of blocking delays. In this dissertation we employ the generalized data access model for analysis of transaction blocking and use it in the performance studies of static and two-phase locking algorithms under the assumption of non-uniform data access distribution.

Figure 4. Generalized Data Access Model
CHAPTER III
THE MODEL OF TRANSACTION BLOCKING

In this chapter, we first discuss a model of a database system under non-uniform data access distribution. We then present a polynomial algorithm to compute the probability of conflict between two transactions under arbitrary data access distribution. Next, we extend this algorithm to compute the probability of no conflict among 't' transactions (i.e., the probability that 't' transactions can run concurrently). The algorithm is illustrated using numerical examples. Finally, we present an efficient approximate technique for the computation of the probability of conflict between transactions under arbitrary data access distribution and discuss its applicability to performance studies of concurrency control algorithms.

3.1 The Model

We assume that the database consists of M data objects indexed from 1 to M. Each data object can be considered a record or a group of records which are locked simultaneously. A transaction always accesses 'n' distinct data objects (i.e., the transaction size is 'n'). Data access distribution of transactions across the database is specified by a probability vector \( \mathbf{p} = (p_1, p_2, ..., p_M) \), where \( p_i \), for \( 1 \leq i \leq M \), is the relative frequency of access of data object i by transactions. We assume that the
probability that a data object is accessed by a transaction is independent of the probability of other data objects accessed by that transaction. Although in real-life databases, there is some dependency in transaction accessing, here we make the assumption of independence in order to make the analysis computationally tractable. Such assumption has been made and justified in several performance studies of database systems [5, 43, 16]. It has been shown using simulation in [16] that the error due to the assumption of independence is small.

The data request of a transaction T is denoted by a request vector RV = (a₁, a₂,...,aₘ), where aᵢ ∈ {0,1}. If aᵢ=1, then T accesses data objects i, else it does not; Σᵢ aᵢ gives the total number of data objects accessed by T, which is equal to 'n' here. Since a transaction accesses data object i with probability pᵢ, the probability that T makes a data request shown in RV is (from the assumption of independence),

\[
P(RV) = \prod_{\forall a_i=1} p_i \prod_{\forall a_i=0} (1-p_i) = \prod_{i=1}^{i=M} p_i^{a_i}(1-p_i)^{1-a_i}
\]  

In case of general data access distribution, the probabilities of the \(\binom{M}{n}\) request vectors are usually different. In case of uniform data access distribution, however, every request vector is equally likely. We model the centralized database by the following queueing network model (see Figure 5):

When a transaction arrives in the system, it attempts to lock 'n' data objects (that is, we assume static locking policy; in Section 4.2 we extend our results to dynamic locking policy [8]) chosen according to data access distribution \(\overline{p}\) from M
data objects. Since the database is shared among several users, it may so happen that an arriving transaction finds that some of the required data objects are already locked by some other running transactions. In such situations, the transaction suffers a conflict and remains blocked until all the data objects required by it become available to it. Otherwise (i.e., if none of the required data objects are locked by the other running transactions), the transaction locks all the needed data objects, executes by entering the Computer System block, and then releases all the locks after completion and before departing the system. When a transaction departs, all the blocked transactions are checked to see if their requirements can be met by the released data objects. A transaction does not lock any data objects while in blocked/wait state.
The probability of conflict (i.e., the probability that a transaction does not get all the needed data objects on arrival) and blocking delays (i.e., the average wait time due to unavailability of needed data objects) are important parameters in the performance analysis of the underlying concurrency control algorithm (here static locking). In this chapter, we limit our discussion only to the computation of the probability of conflict under non-uniform data access distribution.

3.1.1 The Expression for the Probability of not Blocking

The probability that an arriving transaction is immediately executed when ‘t’ other transactions are already running is the probability that the arriving transaction gets all the data objects required by it given that ‘t’ transactions are already running. We express the probability that an arriving transaction is immediately executed when ‘t’ other transactions are already running in terms of the probability that ‘t’ and ‘t+1’ transactions are running concurrently (i.e., are conflict free) in the following way:

\[ P(\text{An arriving trans. does not block | } t \text{ trans. are already active}) = \frac{\text{Probability that } t+1 \text{ transactions run concurrently}}{\text{Probability that } t \text{ transactions run concurrently}} \]  

(2)

If ‘t’ independent transactions \( T_1, T_2, ..., T_t \) have their request vectors as \( RV_1, RV_2, ..., RV_t \) respectively, then the probability that there is no conflict among these transactions (denoted by \( P_{NC}(t) \)) is given by the following equation:
\[
\left( \sum_{\forall RV_1 \land \forall RV_2 \land \ldots \land \forall RV_t} P(RV_1) \ast P(RV_2) \ast \ldots \ast P(RV_t) \right)^{-1} \sum_{\forall RV_i} P(RV_i) \sum_{\forall RV_j} P(RV_j) \sum_{\forall RV_k} P(RV_k) \ldots \sum_{\forall RV_v} P(RV_v) \left( \forall RV_s \right) \left( \forall RV_t \right) \left( \forall RV_u \right).
\]

If \( D = (1, 1, \ldots, 1) \) is an \( M \)-dimensional unit-vector, then the above equation can be rewritten as

\[
\left( \sum_{\forall RV_1 \land \forall RV_2 \land \ldots \land \forall RV_t} \prod_{j=1}^{t} P(RV_j) \right) \left( \sum_{\forall RV_i} P(RV_i) \right) \left( \sum_{\forall RV_j} P(RV_j) \right) \left( \sum_{\forall RV_k} P(RV_k) \right) \ldots \left( \sum_{\forall RV_v} P(RV_v) \right) \left( \forall RV_s \right) \left( \forall RV_t \right) \left( \forall RV_u \right).
\]

Let request vector \( RV_i \) denote a transaction of size \( n_i \). In case of uniform data access distribution, for every \( n \), all the request vectors of size \( n \) are equally likely; therefore, in this case the probability of no conflict is equal to:

\[
P_{NC}(t) = \frac{\prod_{i=1}^{t} \left( \binom{M}{n_i} \right) \ast \left( \binom{M-N_i}{n_i} \right) \ast \ldots \ast \left( \binom{M-N_{t-1}}{n_i} \right)}{\prod_{i=1}^{t} \left( \binom{M}{n_i} \right)}
\]

where \( N_i = \sum_{j=1}^{i} n_j \).

The above equation can be obtained from the general formula as follows: In the uniform case, all the \( p_j \) are equal to \( p = 1/M \). Using equation (1), \( P(RV_i) \) is equal to \( d_i = p^n \ast (1-p)^{M-n} \). Therefore, the numerator of equation (4) is equal in the uniform case to \( \prod_{i=1}^{t} \left( \binom{M}{n_i} \right) \ast \left( \binom{M-N_i}{n_i} \right) \ast \ldots \ast \left( \binom{M-N_{t-1}}{n_i} \right) \ast \prod d_i \), and the denominator is equal to \( \prod_{i=1}^{t} \left( \binom{M}{n_i} \right) \ast \prod d_i \). Dividing the numerator by the denominator gives the above.
equation for the uniform case.

In the uniform case, if all the \( n_i \) are equal to \( n \), the probability that an arriving transaction does not conflict with \( t \) running transactions is equal to \( \frac{\binom{M-tn}{n}}{\binom{M}{n}} \), which can be interpreted as follows: \( tn \) data objects are already taken up by \( t \) running transactions; therefore, the only way an arriving transaction has no conflict is that it chooses its \( n \) data objects from the remaining \((M-tn)\) data objects. Some performance studies have approximated \( P_{NC}(t) \) by \((1-tn^2)/M\) in case of uniform access distribution.

In case of general data access distribution, however, the probability of each of the \( \binom{M}{n} \) request vectors is usually different. Consequently, the complexity of the number of summations in the numerator of equation (4) is \( \binom{M}{n} \times \binom{M-n}{n} \times \ldots \) \( \binom{M-(t-1)n}{n} \). Therefore, direct computation of the probability of no conflict among a set of transactions from equation (4) is very expensive even for a database of modest size. It is largely due to this reason that past performance analyses of concurrency control algorithms have assumed that data access is uniformly distributed.

Now we derive the formula for the b-c access model. In the b-c data access model, a fraction \( c \) of the data objects is accessed with probability \( b \). Let \( h=Mc \). We can renumber the data objects such that \( p_i=q=b/h \) for \( i=1,...,h \), and \( p_i=r=(1-b)/(M-h) \) for \( i=h+1,...,M \). We then apply equation (4) to these probabilities. Suppose \( RV_t=(a_1,\ldots,a_M) \) denotes a transaction of size \( n_t \), and that \( \sum_{i=1}^{M} a_i=n_t \). Then
\[ \sum_{h+1 \leq j \leq M} a_j = n_i - m_i, \text{ and} \]

\[ P(RV_i) = d(m_i, n_i) = q^{m_i} (1-q)^{n_i-m_i} (1-r)^{(M-h)-(n_i-m_i)} \]  \hspace{1cm} (6)

So for given \( m_1, ..., m_i, P(RV_1) \cdot \cdots \cdot P(RV_i) \) is equal to

\[ R(m_1, ..., m_i) = \prod_{1 \leq i \leq i} d(m_i, n_i) \]  \hspace{1cm} (7)

Also, the number of possibilities for choosing non conflicting \( RV_1, ..., RV_i \) with given \( m_1, ..., m_i \) is:

\[ L(m_1, ..., m_i) = \prod_{1 \leq i \leq i} \binom{h-M_{i-1}}{m_i} \binom{M-h-M'_{i-1}}{n_i-m_i} \]  \hspace{1cm} (8)

where \( M_0 = M'_0 = 0, M_i = \sum_{1 \leq j \leq i} m_j, M'_i = \sum_{1 \leq j \leq i} n_j - m_j \). The total number of possibilities of choosing \( RV_i \) with a given \( m_i \) is:

\[ J(m_i) = \binom{h}{m_i} \binom{M-h}{n_i-m_i} \]  \hspace{1cm} (9)

So the probability of no conflict for the b-c model is:

\[ \sum_{0 \leq m_1, ..., m_i} \cdots \sum_{0 \leq m_1, ..., m_i} L(m_1, ..., m_i) R(m_1, ..., m_i) \]

\[ \prod_{1 \leq i \leq i} \left( \sum_{0 \leq m_i} J(m_i) d(m_i, n_i) \right) \]  \hspace{1cm} (10)

In the remainder of the chapter, we develop a series of efficient algorithms for computing the probability of conflict in the following sequence: First, we present a polynomial time algorithm for computing the probability of no conflict between two transactions. The algorithm avoids combinatorial explosion by splitting the state space. Subsequently, we extend this algorithm to multiple transactions. Then we present an efficient approximate technique for computation of the probability of
conflict under arbitrary data access distributions. The algorithms are illustrated using numerical examples.

3.2 A Polynomial Algorithm for Computing the Probability of Conflict between Two Transactions

In this section, we present an algorithm for computing the probability of no conflict between two transactions. The algorithm has polynomial complexity. The algorithm is formulated in terms of two functions $P(n_1, n_2, k)$ and $Q(n, k)$, where $n, n_1, n_2$, and $k$ are integers. These functions are defined recursively as follows:

$$P(n_1, n_2, k) = (1-p_k)^2 P(n_1, n_2, k-1) + p_k (1-p_k) P(n_1-1, n_2, k-1)$$
$$+ (1-p_k) p_k P(n_1, n_2-1, k-1)$$

Boundary Conditions:

$$P(i, j, k) = \begin{cases} 0 & \text{if } i+j>k \\ 1 & \text{if } i=j=k=0 \\ 0 & \text{if } i \text{ or } j \text{ or } k < 0 \end{cases}$$

$$Q(n, k) = (1-p_k) Q(n, k-1) + p_k Q(n-1, k-1)$$

Boundary Conditions:

$$Q(i, j) = \begin{cases} 1 & \text{if } i=j=0 \\ 0 & \text{if } i > j \\ 0 & \text{if } i \text{ or } j < 0 \end{cases}$$

Note that function $Q(n, k)$ is similar to function $F(r, m)$ of [43]; however, we have different boundary conditions.

Explanation
Function $Q(n, k)$:

For a transaction of size $n$, $Q(n, k)$ gives the probability that all the data objects accessed by it are concentrated over the first $k$ data objects of the database, i.e., if $\Psi$ denotes the collection of all vectors $RV=(a_1, a_2, \ldots, a_M)$ such that $a_1 + \ldots + a_k = n$ and $a_{k+1} + \ldots + a_M = 0$, then

$$\sum_{RV \in \Psi} P(RV) = Q(n, k)$$

(15)

Note that each term in the denominator of equation (5) is nothing but $Q(n, M)$, which can be computed in $4nM$ operations rather than in $\binom{M}{n} \times M$ operations.

Function $P(n_1, n_2, k)$:

For two transactions $T_1$ and $T_2$ of size greater than or equal to $n_1$ and $n_2$, respectively, $P(n_1, n_2, k)$ gives the probability that $T_1$ and $T_2$ do not conflict given that $n_1$ accesses of $T_1$ and $n_2$ accesses of $T_2$ are concentrated over first $k$ data objects of the database. Note that numerator of equation (4) is nothing but $P(n, n, M)$ which can be computed in about $11Mn^2$ operations rather than in $\binom{M}{n} \times \binom{M-n}{n} \times M$ operations. If we precompute the terms $(1-p_k)^2$ and $(1-p_k)p_k$, $1 \leq k \leq M$, and store them into arrays, then we can use these ready-made values during the computation of $P(n, n, M)$, bringing down the number of operations to $5Mn^2$, which is less than half of the previous number. At any time, we need only those elements of $P(i, j, k)$ which have the same value of $k$; therefore, the space complexity is $O(n^2)$ rather than $O(Mn^2)$. 
Therefore, for transactions with arbitrary data access distribution, the probability
of conflict between them can be given by the following equation:

\[
1 - \frac{P(n,n,M)}{Q(n,M)Q(n,M)}
\]  

(16)

3.3 An Algorithm for Multiple Transactions

In this section, we present an efficient algorithm to compute the probability of
no conflict among any number of transactions. The algorithm is an extension of the
previous algorithm (Section 3.2) and is formulated in terms of two functions: \(P(n_1,\)
\(n_2,\ldots,n_t, k)\) and \(Q(n,k)\), where \(n_1, n_2, n_3,\ldots,n_t\) and \(k\) are integers. These functions are
defined recursively as follows:

\[
P(n_1, n_2,\ldots,n_t, k) = (1-p_k)^t \cdot P(n_1, n_2, n_3, \ldots, n_t, k-1) + 
\]

\[
p_k \cdot (1-p_k)^{n-1} [P(n_1-1, n_2,\ldots,n_t, k-1) 
\]

\[
+ P(n_1, n_2-1, n_3,\ldots,n_t, k-1) + \ldots 
\]

\[
+ P(n_1, n_2,\ldots,n_{t-1}, n_{t-1}, k-1)] 
\]

(17)

Boundary Conditions:

\[
P(n_1, n_2,\ldots,n_t, k) = \begin{cases} 
0 & \text{if } \sum_{i=0}^{t} n_i > k \\
1 & \text{if } n_1 = n_2 = \ldots = n_t = k = 0 \\
0 & \text{if } n_1 \text{ or } n_2 \text{ or } \ldots n_t \text{ or } k < 0 
\end{cases}
\]

(18)

\[
Q(n,k) = (1-p_k) \cdot Q(n,k-1) + p_k \cdot Q(n-1,k-1) 
\]

(19)

Boundary conditions:
\[
Q(i, j) = \begin{cases} 
1 & \text{if } i = j = 0 \\
0 & \text{if } i > j \\
0 & \text{if } i \text{ or } j < 0
\end{cases}
\]  
(20)

Explanation:

Function \(Q(n, k)\): The same as function \(Q(n, k)\) in Section 3.2.

Function \(P(n_1, n_2, n_3, \ldots, n_t, k)\):

For \(t\) transactions \(T_1, T_2, T_3, \ldots, T_t\) of size greater than or equal to \(n_1, n_2, n_3, \ldots, n_t\), respectively, \(P(n_1, n_2, \ldots, n_t, k)\) gives the probability that \(T_1, T_2, \ldots, T_t\) do not conflict given that \(n_1\) accesses of \(T_1\), \(n_2\) accesses of \(T_2\), \ldots, and \(n_t\) accesses of \(T_t\) are concentrated over first \(k\) data objects of the database.

Therefore, for an arbitrary number of transactions with arbitrary data access distribution, the probability of conflict between them can be given by the following equation:

\[
1 - \frac{P(n, n, n, \ldots, n, M)}{[Q(n, M)]^t}
\]  
(21)

It is easy to show that the complexity of computing the numerator of the above equation is \(O(n^tM)\).

3.4 Numerical Examples

In this section, we present numerical results to illustrate the polynomial algorithm (Section 3.2) for computing the probability of conflict between two
transactions under non-uniform data access distribution. Assuming $M=1000$ and data access distributed according to truncated Normal distribution (that is, $p_i = C e^{-\frac{1}{2}(i-\mu)^2}, 1 \leq i \leq M$, where $C$ is a constant such that $\sum p_i = \text{transaction size over all } i$), we compute the probability of conflict using equation (16) for various values of transaction size ($n$) and variance of the Normal distribution ($\sigma$). To elucidate the difference in the probability of conflict which results from non-uniform data access distribution, we also compute the probability of conflict for uniform access distribution and compile the results in Table 1.

The choice of truncated Normal distribution to model data access distribution is for the sake of illustration of the algorithm; nevertheless, it does capture the bell-shaped property of data access distribution. By choosing different values for $\mu$ and $\sigma$, we can simulate various bell-shaped curves for data access distribution. A smaller choice of $\sigma$ gives a narrow bell-shaped curve, thereby signifying that data access is largely concentrated over a small portion of the database.

The numerical results confirm that the assumption of uniform data access distribution gives an optimistic estimate (that is, lower bound) of the probability of conflict. Also, as data access distribution drifts away from uniform distribution (i.e., $\sigma$ gets smaller), the probability of conflict becomes larger. This is intuitive because now more transaction accesses are concentrated over a smaller portion of the database, causing more conflicts. The results also indicate that an increase in the transaction size ($n$) results in a higher probability of conflict - the larger the number of data objects accessed by a transaction, the higher the chances of overlap.
Table 1. The Probability of Conflict (for n= 500)

<table>
<thead>
<tr>
<th>n</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=400</th>
<th>σ=500</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.05928266</td>
<td>0.04437109</td>
<td>0.03942000</td>
<td>0.03747333</td>
<td>0.035552</td>
</tr>
<tr>
<td>8</td>
<td>0.10316290</td>
<td>0.07765577</td>
<td>0.06912923</td>
<td>0.06576831</td>
<td>0.062449</td>
</tr>
<tr>
<td>10</td>
<td>0.15675470</td>
<td>0.11886585</td>
<td>0.10608316</td>
<td>0.10103271</td>
<td>0.096032</td>
</tr>
<tr>
<td>12</td>
<td>0.21811870</td>
<td>0.16689604</td>
<td>0.14968725</td>
<td>0.14352525</td>
<td>0.135565</td>
</tr>
<tr>
<td>14</td>
<td>0.28512185</td>
<td>0.22130661</td>
<td>0.21475087</td>
<td>0.20653133</td>
<td>0.180195</td>
</tr>
</tbody>
</table>

3.5 An Approximate Computational Technique

Though the algorithm presented in the previous section considerably reduces the complexity of the computation of equation (21), it is still impractical if the number of transactions, transaction size, and database size are very large. In this section, we present a very fast, but approximate method to compute equation (21). The method is based on the principle of pair-wise conflicts — t transactions do not conflict if no pair of those transactions has a conflict. If all t transactions have size n and the probability of pair-wise conflict is p, then the probability that \( \binom{t}{2} \) distinct pairs do not conflict is \( p^{\binom{t}{2}} \). However, in this computation we assume that the probability of a pair-wise conflict is independent of other pair-wise conflicts, and this is what introduces error in the computation.
The probability that a transaction does not conflict when \( t \) transactions are running is (equation (16)):

\[
\Pr\left(\binom{t+1}{2} \right) = \Pr\left(\sum_{i=1}^{t} \binom{i}{2} \right) = p^t
\]  

(22)

Therefore, the probability that a transaction does not conflict when \( t \) transactions are already active is the same as the probability that an incoming transaction does not have a pair-wise conflict with any of the \( t \) transactions. In the next section, we give several numerical examples which show that the error due to this approximation is very small.

3.5.1 Numerical Results

In this section, we present numerical results to illustrate the algorithm of Section 3.3 and its approximation, which enable us to compute the probability that a transaction does not conflict/block with a set of non-conflicting transactions under non-uniform data access distribution. Assuming \( M=1000 \) and data access distributed according to truncated Normal distribution (that is, \( p_i = Ce^{-\frac{1}{2}(i-\mu)^2}, 1 \leq i \leq M \), where \( C \) is a constant such that \( \sum p_i = \text{transaction size over all } i \)), we compute the probability that a transaction does not conflict with already active transactions. We compile the results in Tables 2 and 3.

The choice of truncated Normal distribution to model data access distribution is for the sake of illustration of the algorithm and its approximation; nevertheless, it
does capture the bell-shaped property of data access distribution.

In Table 2, we keep the transaction size (n=10) constant and vary the number of transactions (tr). As expected, the probability of no conflict/blocking becomes smaller as the number of transactions increases. The error due to approximation also increases with the number of transactions, but remains less than 6 percents. In Table 3 we fix the number of transactions (tr=5) and increase the transaction size (n). Again, as expected the probability of no conflict/blocking becomes smaller as the transaction size becomes larger. The error introduced due to approximation is less

Table 2. Results for Probability of not Blocking (μ=500, n=10)

<table>
<thead>
<tr>
<th>t</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=400</th>
<th>σ=500</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact</td>
<td>0.70973</td>
<td>0.77546</td>
<td>0.79822</td>
<td>0.80731</td>
<td>0.94240</td>
</tr>
<tr>
<td>Approx</td>
<td>0.71106</td>
<td>0.77639</td>
<td>0.79908</td>
<td>0.80814</td>
<td>0.94269</td>
</tr>
<tr>
<td>Error</td>
<td>0.18740</td>
<td>0.1199</td>
<td>0.10774</td>
<td>0.10281</td>
<td>0.03077</td>
</tr>
<tr>
<td>Exact</td>
<td>0.59622</td>
<td>0.68162</td>
<td>0.71198</td>
<td>0.72421</td>
<td>0.91445</td>
</tr>
<tr>
<td>Approx</td>
<td>0.59960</td>
<td>0.68411</td>
<td>0.71431</td>
<td>0.72649</td>
<td>0.91528</td>
</tr>
<tr>
<td>Error</td>
<td>0.56690</td>
<td>0.36530</td>
<td>0.32720</td>
<td>0.31483</td>
<td>0.09076</td>
</tr>
<tr>
<td>Exact</td>
<td>0.49900</td>
<td>0.59841</td>
<td>0.63437</td>
<td>0.64896</td>
<td>0.88703</td>
</tr>
<tr>
<td>Approx</td>
<td>0.50561</td>
<td>0.60279</td>
<td>0.63854</td>
<td>0.65309</td>
<td>0.88866</td>
</tr>
<tr>
<td>Error</td>
<td>1.3246</td>
<td>0.73190</td>
<td>0.65750</td>
<td>0.63640</td>
<td>0.18376</td>
</tr>
<tr>
<td>Exact</td>
<td>0.41832</td>
<td>0.52470</td>
<td>0.56455</td>
<td>0.58087</td>
<td>0.86018</td>
</tr>
<tr>
<td>Approx</td>
<td>0.42635</td>
<td>0.53114</td>
<td>0.57080</td>
<td>0.58711</td>
<td>0.86282</td>
</tr>
<tr>
<td>Error</td>
<td>1.92070</td>
<td>1.2279</td>
<td>1.1070</td>
<td>1.07425</td>
<td>0.30808</td>
</tr>
<tr>
<td>Exact</td>
<td>0.34937</td>
<td>0.45945</td>
<td>0.50184</td>
<td>0.51935</td>
<td>0.83386</td>
</tr>
<tr>
<td>Approx</td>
<td>0.35952</td>
<td>0.46800</td>
<td>0.51025</td>
<td>0.52779</td>
<td>0.83773</td>
</tr>
<tr>
<td>Error</td>
<td>2.9052</td>
<td>1.86090</td>
<td>1.67580</td>
<td>1.62511</td>
<td>0.46411</td>
</tr>
<tr>
<td>Exact</td>
<td>0.28768</td>
<td>0.40184</td>
<td>0.44562</td>
<td>0.46384</td>
<td>0.80807</td>
</tr>
<tr>
<td>Approx</td>
<td>0.30316</td>
<td>0.41237</td>
<td>0.45612</td>
<td>0.47447</td>
<td>0.81338</td>
</tr>
<tr>
<td>Error</td>
<td>5.38090</td>
<td>2.62040</td>
<td>2.35620</td>
<td>2.29170</td>
<td>0.65712</td>
</tr>
</tbody>
</table>
Table 3. Results for Probability of not Blocking ($\mu=500$, $t=5$)

<table>
<thead>
<tr>
<th>n</th>
<th>$\sigma=200$</th>
<th>$\sigma=300$</th>
<th>$\sigma=400$</th>
<th>$\sigma=500$</th>
<th>$\sigma=\infty$ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>0.5746</td>
<td>0.6634</td>
<td>0.6950</td>
<td>0.7078</td>
<td>0.8948</td>
</tr>
<tr>
<td>Approx</td>
<td>0.5801</td>
<td>0.6675</td>
<td>0.6989</td>
<td>0.7116</td>
<td>0.8964</td>
</tr>
<tr>
<td>Error</td>
<td>0.9726</td>
<td>0.6210</td>
<td>0.5686</td>
<td>0.5445</td>
<td>0.1877</td>
</tr>
<tr>
<td>Exact</td>
<td>0.4183</td>
<td>0.5247</td>
<td>0.5645</td>
<td>0.5808</td>
<td>0.8601</td>
</tr>
<tr>
<td>Approx</td>
<td>0.4263</td>
<td>0.5311</td>
<td>0.5708</td>
<td>0.5870</td>
<td>0.8628</td>
</tr>
<tr>
<td>Error</td>
<td>1.9207</td>
<td>1.2279</td>
<td>1.1070</td>
<td>1.0742</td>
<td>0.3080</td>
</tr>
<tr>
<td>Exact</td>
<td>0.3088</td>
<td>0.3898</td>
<td>0.4372</td>
<td>0.4664</td>
<td>0.8329</td>
</tr>
<tr>
<td>Approx</td>
<td>0.2922</td>
<td>0.4013</td>
<td>0.4445</td>
<td>0.4608</td>
<td>0.8287</td>
</tr>
<tr>
<td>Error</td>
<td>3.2080</td>
<td>2.9537</td>
<td>1.6719</td>
<td>1.2155</td>
<td>0.5182</td>
</tr>
<tr>
<td>Exact</td>
<td>0.1985</td>
<td>0.2735</td>
<td>0.3082</td>
<td>0.3092</td>
<td>0.7992</td>
</tr>
<tr>
<td>Approx</td>
<td>0.1867</td>
<td>0.2863</td>
<td>0.2985</td>
<td>0.3145</td>
<td>0.7945</td>
</tr>
<tr>
<td>Error</td>
<td>6.3524</td>
<td>4.6743</td>
<td>3.2405</td>
<td>1.7182</td>
<td>0.5958</td>
</tr>
<tr>
<td>Exact</td>
<td>0.1262</td>
<td>0.1908</td>
<td>0.1743</td>
<td>0.2181</td>
<td>0.7746</td>
</tr>
<tr>
<td>Approx</td>
<td>0.1168</td>
<td>0.2040</td>
<td>0.1806</td>
<td>0.2224</td>
<td>0.7605</td>
</tr>
<tr>
<td>Error</td>
<td>8.0845</td>
<td>6.9332</td>
<td>3.6044</td>
<td>1.9967</td>
<td>1.8518</td>
</tr>
<tr>
<td>Exact</td>
<td>0.0728</td>
<td>0.1312</td>
<td>0.0998</td>
<td>0.1423</td>
<td>0.7420</td>
</tr>
<tr>
<td>Approx</td>
<td>0.0667</td>
<td>0.1411</td>
<td>0.1032</td>
<td>0.1455</td>
<td>0.7270</td>
</tr>
<tr>
<td>Error</td>
<td>9.5106</td>
<td>7.5241</td>
<td>3.3383</td>
<td>2.2942</td>
<td>2.0572</td>
</tr>
</tbody>
</table>

than 10 percents. By comparing the numerical results in Tables 2 and 3, we also come to a very interesting conclusion. For the same number of locked data objects, the probability of no conflict/blocking will be higher if these data objects are locked by fewer transactions of larger size. The numerical results also confirm that the magnitude of error introduced due to approximation is small (<10 percents).
3.6 Summary

In this chapter, we have presented an algorithm to compute the probability that a transaction conflicts with a set of active transactions under arbitrary data access distributions. We have also presented an approximation technique which provides a very fast method to compute the probability of no blocking/conflict. We have illustrated the algorithms by using numerical examples which show that the magnitude of error introduced due to approximation is very small when the number of locked data objects is not very large. The results of this chapter provide a very cost-effective way to analyze the performance of database systems under non-uniform data access distributions.
CHAPTER IV
AN EFFICIENT COMPUTATIONAL TECHNIQUE

In this chapter, we first present an efficient (i.e., polynomial time) algorithm for computation of the probability of conflict among transactions for databases in which data access distribution is arbitrary. The algorithm permits inclusion of any data access distribution in the performance analyses of concurrency control algorithms at a moderate cost of computation. Then, we extend our results to dynamic acquisition policy, and we illustrate the algorithm by using a numerical example to compute the probability of blocking of a transaction under non-uniform data access distribution. A comparison of the results for non-uniform and uniform access distributions indicates that the assumption of uniform access distribution can introduce significant errors in the probability of transaction blocking. In the Appendix, we present an efficient technique for computing the functions used in the algorithm.

4.1 A Polynomial Algorithm for Computing the Probability of Conflict among 't' Transactions

In this section, we present an algorithm to compute the probability of conflict among 't' transactions under non-uniform data access distribution. The complement of this probability is the probability that 't' transactions can run concurrently, which
is required in equation (3). Our algorithm has polynomial complexity and the analysis is based on the database model and taxonomy developed in Chapter 3. The basic idea underlying the algorithm is that we recursively express the probability of conflict among 't' transaction of size 'n' in terms of the probability of conflict among 't-1' transactions, one of size '2n' and the remaining 't-2' of size 'n'. The algorithm requires merging of 'k' non-conflicting transactions into a big transaction. We denote the request vector of such a merged transaction, which accesses 'kn' data objects out of M data objects, by W. The algorithm uses the following two functions (where RV = \bar{\alpha}, is a request vector of a transaction of size 'n'): Q(kn, M) and A(kn, n, M).

\[ Q(kn, M) = \sum_{W} P(W) \]  

(23)

Computation of Q(kn, M) requires \( \frac{M}{kn} \) \* M operations. In the Appendix, we give an algorithm which computes Q(kn, M) in only 4knM operations. Note that each term in the numerator of equation (3) is Q(n, M).

\[ A(kn, n, M) = \sum_{W} P(W) \cdot P(RV) \]  

(24)

Computation of A(kn, n, M) requires \( \binom{M}{n} \cdot \binom{M-n}{kn} \) \* 2M operations. In the Appendix, we give an algorithm which computes A(kn, n, M) in 5kn^2M operations.

4.1.1 The Algorithm

We first illustrate the algorithm for three transactions and then generalize it to 't' transactions. (Illustration of the algorithm with three transactions not only
enhances the understanding of the algorithm, but also serves as an explanation of the boundary conditions of its recursion.) If three independent transactions $T_1$, $T_2$, and $T_3$ have their request vectors $RV_1$, $RV_2$, and $RV_3$ defined as $RV_1 = \alpha = (a_1, a_2, ..., a_M)$ and $RV_2 = \beta = (b_1, b_2, ..., b_M)$, and $RV_3 = \gamma = (c_1, c_2, ..., c_M)$, respectively, then the probability that these transactions conflict (i.e., $0 \leq i \leq M$, $a_i \land b_i \neq 0$ or $b_i \land c_i \neq 0$ or $a_i \land c_i \neq 0$) is given by the following equation:

$$\sum_{\forall RV_1} \sum_{\forall RV_2} \sum_{\forall RV_3} \frac{P(RV_1) \cdot P(RV_2) \cdot P(RV_3)}{\text{such that } \alpha \land \beta \neq 0 \text{ or } \alpha \land \gamma \neq 0 \text{ or } \beta \land \gamma \neq 0}$$

$$= \left( \sum_{\forall RV_1} P(RV_1) \right) \cdot \left( \sum_{\forall RV_2} P(RV_2) \right) \cdot \left( \sum_{\forall RV_3} P(RV_3) \right)$$

(25)

Computation of the denominator of the above equation is straightforward (it is equal to $Q^3(n, M)$). However, computation of the numerator is non-trivial because the state space of the summation is extremely large due to combinatorial explosion. We handle this explosion by splitting the state space. We decompose numerator $H^{(3)}(n, n, n)$ (the superscript of $H$ denotes the number of transactions involved and the three arguments denote the size of the transactions) of the above equation into two functions $E^{(3)}(n, n, n)$ and $F^{(3)}(n, n, n)$ by splitting the state space defined by the condition of the summation of $H^{(3)}(n, n, n)$ into two smaller disjoints spaces - first where $\alpha \land \beta \neq 0$ and second where $\alpha \land \beta = 0$ and ($\alpha \land \gamma \neq 0$ or $\beta \land \gamma \neq 0$).

$$H^{(3)}(n, n, n) = E^{(3)}(n, n, n) + F^{(3)}(n, n, n)$$

(26)

where,
\[
E^{(3)}(n, n, n) = \left\{ \sum_{\forall \text{RV}_1, \forall \text{RV}_2, \forall \text{RV}_3} P(\text{RV}_1) \cdot P(\text{RV}_2) \cdot P(\text{RV}_3) \right\} \left\{ \sum_{\forall \text{RV}_1, \forall \text{RV}_2} P(\text{RV}_1) \cdot P(\text{RV}_2) \right\} \left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \right\} \tag{27}
\]

\[
F^{(3)}(n, n, n) = \left\{ \sum_{\forall \text{RV}_1, \forall \text{RV}_2, \forall \text{RV}_3} P(\text{RV}_1) \cdot P(\text{RV}_2) \cdot P(\text{RV}_3) \right\} \tag{28}
\]

Next, we proceed with the computation of functions \(E^{(3)}(n, n, n)\) and \(F^{(3)}(n, n, n)\).

\[
E^{(3)}(n, n, n) = \left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \right\} \left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \cdot P(\text{RV}_2) \right\} \tag{29}
\]

The second term on the right hand side of the above equation is:

\[
\left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \cdot P(\text{RV}_2) \right\} = \tag{30}
\]

\[
\left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \cdot P(\text{RV}_2) \right\} - \left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \right\} = Q(n, M)^2 - A(n, n, M) \]

Consequently,

\[
E^{(3)}(n, n, n) = Q(n, M)^2 - A(n, n, M) \tag{31}
\]

Next, we examine function \(F^{(3)}(n, n, n)\), which is equal to:

\[
F^{(3)}(n, n, n) = \left\{ \sum_{\forall \text{RV}_1, \forall \text{RV}_2, \forall \text{RV}_3} P(\text{RV}_1) \cdot P(\text{RV}_2) \cdot P(\text{RV}_3) \right\} = \tag{32}
\]

\[
\sum_{\forall \text{RV}_1} P(\text{RV}_1) \cdot P(\text{RV}_2) \sum_{\forall \text{RV}_1} P(\text{RV}_1) \cdot P(\text{RV}_3) \sum_{\forall \text{RV}_1} P(\text{RV}_1) \cdot P(\text{RV}_2, \forall \text{RV}_3) \left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \right\} \left\{ \sum_{\forall \text{RV}_1} P(\text{RV}_1) \right\} \tag{32}
\]
Note that $F^{(3)}(n, n, n)$ is the summation of the product $P(RV_1) \times P(RV_2) \times P(RV_3)$ over the domain where $\bar{a}$ and $\bar{b}$ do not conflict but $\bar{c}$ does conflict with $\bar{a}$ or $\bar{b}$. To simplify the computation of $F^{(3)}(n, n, n)$, we perform a transformation on the domain of the summation. We lump $\bar{a}$ and $\bar{b}$ into $\bar{w} = (w_1, w_2, ..., w_M)$ such that $w_i = a_i \lor b_i$ for every $1 \leq i \leq M$; consequently, the request vector $\bar{w}$ denotes the request vectors $\bar{a}$ and $\bar{b}$ combined and $\sum_{1 \leq i \leq M} w_i = 2n$ because $\bar{a}$ and $\bar{b}$ are disjoint/non-conflicting. As a result, $F^{(3)}(n, n, n)$ becomes the summation of product $P(RV_1) \times P(RV_2) \times P(RV_3)$ over the domain where $\bar{w}$ and $\bar{c}$ conflict.

Let $P(W) = \prod_{w_i=1} p_i \prod_{w_i=0} (1-p_i)$. (That is, $P(W)$ denotes the probability of choosing data objects according to request vector $\bar{w}$.) However, it should be noted that

$$\sum_{\forall RV_1, \forall RV_2} P(RV_1) \times P(RV_2) \neq \sum_{\forall W} P(W)$$

This is because of two reasons: First, we should note that $P(RV_1) \times P(RV_2) \neq P(W)$. For an explanation, let us next examine both sides of this inequality:

$$P(RV_1) \times P(RV_2) = \prod_{a=1} p_i \prod_{a=0} (1-p_i) \prod_{b=1} (1-p_i) \prod_{b=0} p_i$$

$$P(W) = \prod_{w_i=1} p_i \prod_{w_i=0} (1-p_i)$$

The first two terms of $P(RV_1) \times P(RV_2)$ give the first term of $P(W)$; however, the remaining two terms of $P(RV_1) \times P(RV_2)$ fail to give the second term of $P(W)$. This is because $P(W)$ falls short of $P(RV_1) \times P(RV_2)$ by $\prod_{i=1}^{i=M} (1-p_i)$. Therefore,
\[ P(RV_1) \cdot P(RV_2) = P(W) \prod_{i=1}^{i=M} (1-p_i) \] (36)

Second, there are several possible pairs of \( \overline{a} \) and \( \overline{b} \) which can give the same \( \overline{w} \). Given that \( \overline{w} \) has 2n 1s, \( \overline{a} \) can choose its n 1s from those 2n in \( \binom{2n}{n} \) ways. Therefore, there are \( \binom{2n}{n} \) pairs of \( \overline{a} \) and \( \overline{b} \) which will result in the same \( \overline{w} \). With these two observations, we can claim the following:

\[ \sum_{\forall RV_1, \forall RV_2, \text{such that } \overline{a} \cup \overline{b} = \overline{\theta}} P(RV_1) \cdot P(RV_2) = \binom{2n}{n} \prod_{i=1}^{i=M} (1-p_i) \sum_{\forall W} P(W) \] (37)

With accesses \( \overline{a} \) and \( \overline{b} \) lumped together as an access \( \overline{w} \), we can rewrite equation (2) for \( F^{(3)}(n, n, n) \) as follows:

\[ F^{(3)}(n, n, n) = \] (38)

\[ \binom{2n}{n} \prod_{i=1}^{i=M} (1-p_i) \sum_{\forall W} P(W) \sum_{\forall RV_1, \text{such that } \overline{w} \cup \overline{a} \cup \overline{b} = \overline{\theta}} P(RV_3) = \]

\[ \left( \binom{2n}{n} \prod_{i=1}^{i=M} (1-p_i) \left[ \sum_{\forall W, \forall RV_1, \text{such that } \overline{w} \cup \overline{a} \cup \overline{b} = \overline{\theta}} P(W) \cdot P(RV_3) \right] \right) \]

Note that the rightmost term in the above equation is \( H^{(2)}(2n, n) \). This algorithm is based on representing \( H^{(3)}(n, n, n) \) in terms of \( H^{(2)}(2n, n) \). We can further simplify \( F^{(3)}(n, n, n) \) as:

\[ F^{(3)} = \binom{2n}{n} \prod_{i=1}^{i=M} (1-p_i) \left[ \sum_{\forall W, \forall RV_1} P(W) \cdot P(RV_3) - \sum_{\forall W, \forall RV_1, \text{such that } \overline{w} \cup \overline{a} \cup \overline{b} = \overline{\theta}} P(W) \cdot P(RV_3) \right] \] (39)

Using the notations developed above, we can express \( F^{(3)} \) as:
In Section 4.2 we will see that the above equation serves as the boundary condition for the recursion of the general algorithm.

4.1.2 Generalization

In this section, we present an efficient algorithm to compute the probability of conflict among 't' transactions under arbitrary data access distribution. The algorithm requires merging of 'k' non-conflicting transactions, 2\leq k \leq t-3, of size \(n_1, n_2, \ldots, n_k\) into a big transaction. We denote the request vector of such a merged transaction which accesses \(n_1+n_2+\ldots+n_k (=N_k)\) data objects out of \(M\) data objects, by \(W_k = \overline{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kM})\). The algorithm uses the following two functions:

\[
Q(N_k, M) = \sum_{W_k} P(W_k).
\]  

(41)

and

\[
A(N_k, n_{k+1}, M) = \sum_{W_k} P(W_k) + P(RV_{k+1}),
\]

(42)

such that \(w_{k1} + \ldots + w_{kM} = \overline{0}\)

Computation of \(Q(N_k, M)\) requires \(\binom{M}{N_k} \cdot M\) operations. By using the technique of the Appendix, we can compute \(Q(N_k, M)\) in only \(4N_kM\) operations. Note that a sum in the denominator of equation (3) is \(Q(n_i, M), 1 \leq i \leq t\).

Computation of \(A(N_k, n_{k+1}, M)\) requires \(\binom{M}{n_{k+1}} \cdot \binom{M-n_{k+1}}{N_k} \cdot 2M\) operations.

By using the technique of the Appendix, we can compute \(A(N_k, n_{k+1}, M)\) in
5N_kn_{k+1}M operations.

The Algorithm

The probability that at least a pair out of the 't' transactions (with their request vectors as RV_1 (= \overline{a_1}), RV_2 (= \overline{a_2}), ..., RV_t (= \overline{a_t})) conflicts is:

\[
P(\text{at least a pair conflicts}) = \sum_{\forall RV_1 \lor RV_2 \lor \cdots \lor RV_t} P(RV_1) \cdot P(RV_2) \cdot \cdots \cdot P(RV_t)
\]

(43)

Computation of the denominator of the above equation is straightforward (it is equal to Q(n_1, M) \cdot Q(n_2, M) \cdot \cdots \cdot Q(n_t, M)). However, computation of the numerator is non-trivial because the state space of the summation is extremely large due to combinatorial explosion. We handle this explosion by splitting the state space. We decompose numerator \(H^{(t)}(n_1, n_2, ..., n_t)\) of equation (43) (the superscript of \(H\) denotes the number of the transactions involved and the arguments denote the size of the transactions) into two functions \(E^{(t)}(n_1, n_2, ..., n_t)\) and \(F^{(t)}(n_1, n_2, ..., n_t)\) by splitting the state space defined by the condition of the summation of \(H^{(t)}(n_1, n_2, ..., n_t)\) into two smaller disjoints spaces: first where \(\overline{a_1} \land \overline{a_2} = \overline{0}\) and second where \(\overline{a_1} \land \overline{a_2} = \overline{0}\) and \((\overline{a_1} \land \overline{a_j} = \overline{0}\) or \(\overline{a_2} \land \overline{a_j} = \overline{0}\) for \(3 \leq j \leq t\).

\[
H^{(t)}(n_1, n_2, ..., n_t) = E^{(t)}(n_1, n_2, ..., n_t) + F^{(t)}(n_1, n_2, ..., n_t)
\]

(44)

where
\[ E^{(t)}(n_1, n_2, \ldots, n_t) = \left( \sum_{\forall RV_1, \forall RV_2, \ldots, \forall RV_t} P(RV_1) \cdot P(RV_2) \cdot \ldots \cdot P(RV_t) \right) \quad (45) \]

\[ F^{(t)}(n_1, n_2, \ldots, n_t) = \left( \sum_{\forall RV_1, \forall RV_2, \ldots, \forall RV_t} P(RV_1) \cdot P(RV_2) \cdot \ldots \cdot P(RV_t) \right) \quad (46) \]

Since the summation in \( E^{(t)}(\ldots) \) does not impose any condition on the request vectors \( RV_3, \ldots, RV_t \), we can write \( E^{(t)}(\ldots) \) as

\[ E^{(t)} = \left( \sum_{\forall RV_1, \forall RV_2, \ldots, \forall RV_t} P(RV_1) \cdot P(RV_2) \cdot \ldots \cdot P(RV_t) \right) \quad (47) \]

The first term on the right hand side is \( Q(n_1, M) \cdot Q(n_2, M) \cdot \ldots \cdot Q(n_t, M) \), and the second term on the right hand can be written as

\[ \left( \sum_{\forall RV_1, \forall RV_2} P(RV_1) \cdot P(RV_2) - \sum_{\forall RV_1, \forall RV_2} P(RV_1) \cdot P(RV_2) \right) \quad (48) \]

\[ \left[ Q(n_1, M) \cdot Q(n_2, M) - A(n_1, n_2, M) \right] \]

Therefore, we can write

\[ E^{(t)}(n_1, n_2, \ldots, n_t) = \prod_{i=3}^{t} Q(n_i, M) \cdot [Q(n_1, M) \cdot Q(n_2, M) \cdot A(n_1, n_2, M)] \quad (49) \]

Next, we examine the function \( F^{(t)}(n_1, n_2, \ldots, n_t) \).

\[ F^{(t)} = \sum_{\forall RV_1, \forall RV_2} P(RV_1) \cdot P(RV_2) \sum_{\forall RV_3, \forall RV_t, \ldots, \forall RV_t} P(RV_3) \cdot \ldots \cdot P(RV_t) \quad (50) \]

Note that \( F^{(t)}(n_1, n_2, \ldots, n_t) \) is the summation of the product \( P(RV_1) \cdot P(RV_2) \cdot \ldots \cdot P(RV_t) \) over the domain where \( \overline{a}_1 \) and \( \overline{a}_2 \) do not conflict but
a_k, 3≤k≤t, does conflict with a_1 or a_2. To simplify the computation of F^{(i)}(n_1, n_2, ..., n_t), we perform a transformation on the domain of the summation. We lump a_1 and a_2 into w_2 = (w_{21}, w_{22}, ..., w_{2M}) such that w_{2i} = a_{1i} OR a_{2i} for every 1≤i≤M; consequently, the request vector W_2 = w_2 denotes the request vectors a_1 and a_2 combined and \[ \sum_{1≤i≤M} w_{2i} = n_1 + n_2 \equiv N_2 \] because a_1 and a_2 are disjoint and non-conflicting. As a result, F^{(i)}(n_1, n_2, ..., n_t) becomes the summation of the product P(RV_1)*P(RV_2)*...*P(RV_t) over the domain where W_2 and a_k, 3≤k≤t, conflict.

P(W_2) denotes the probability of choosing data objects according to the request vector w_2. Therefore, P(W_2) = \prod_{w_2=1} \prod_{w_2=0} (1-p_i). However, it should be noted that

\[
\sum_{\forall RV_1, \forall RV_2} P(RV_1)*P(RV_2) \neq \sum_{\forall W_2} P(W_2)
\] (51)

This is because of two reasons:

First, we should note that P(RV_1)*P(RV_2)≠P(W_2). For an explanation, let us next examine both sides of this inequality:

\[
P(RV_1)*P(RV_2) = \prod_{a_1=1} \prod_{a_1=0} (1-p_i) \prod_{a_2=0} (1-p_i)
\] (52)

\[
P(W_2) = \prod_{w_2=1} \prod_{w_2=0} (1-p_i)
\] (53)

The first two terms of P(RV_1)*P(RV_2) give the first term of P(W_2); however, the remaining two terms of P(RV_1)*P(RV_2) fail to give the second term of P(W_2).

This is because P(W_2) falls short of P(RV_1)*P(RV_2) by \[ \prod_{i=1}^{i=M} (1-p_i) \]. Therefore,
\[
P(RV_1) \cdot P(RV_2) = P(W_2) \prod_{i=1}^{i=M} (1-p_i)
\]

(54)

Second, there are several possible pairs of \(\overline{a}_1\) and \(\overline{a}_2\) which can give the same \(\overline{w}_2\). Given that \(\overline{w}_2\) has \(n_1 + n_2\) Is, \(\overline{a}_1\) can choose its \(n_1\) Is from those \(n_1 + n_2\) in \(\binom{n_1 + n_2}{n_1}\) ways. Therefore, there are \(\binom{n_1 + n_2}{n_1}\) pairs of \(\overline{a}_1\) and \(\overline{a}_2\) which will result in the same \(\overline{w}_2\). With these two observations, we can claim the following:

\[
\sum_{\forall RV_1 \in RV_1} \sum_{\forall RV_2} P(RV_1) \cdot P(RV_2) = \binom{N_2}{n_1} \cdot \prod_{i=1}^{i=M} (1-p_i) \cdot \sum_{\forall W_2} P(W_2)
\]

(55)

With accesses \(\overline{a}_1\) and \(\overline{a}_2\) lumped together as an access \(\overline{w}_2\), we can rewrite equation (50) for \(F^{(i)}(n_1, n_2, ..., n_i)\) as follows:

\[
F^{(i)} = \binom{N_2}{n_1} \cdot \prod_{i=1}^{i=M} (1-p_i) \cdot \sum_{\forall W_2} P(W_2) \cdot \sum_{\forall RV_3 \in RV_3, ..., \forall RV_i} P(RV_3) \cdot \ldots \cdot P(RV_i)
\]

(56)

Note that the rightmost term on the right side of the above equation is nothing but \(H^{(i-1)}(N_2, n_3, ..., n_i)\), that is,

\[
F^{(i)}(n_1, n_2, ..., n_i) = \binom{N_2}{n_1} \cdot \prod_{i=1}^{i=M} (1-p_i) \cdot H^{(i-1)}(N_2, n_3, ..., n_i)
\]

(57)

If we denote \(\binom{N_k}{n_k} \cdot \prod_{i=1}^{i=M} (1-p_i)\) by \(C(N_k, n_k)\), then we obtain a recursive relation for \(F\) as follows:
\[ F^{(t)}(n_1, n_2, ..., n_t) = C(N_2, n_2)[E^{(t-1)}(N_2, ..., n_t) + F^{(t-1)}(N_2, ..., n_t)] \] (58)

where \( E^{(t-1)}(N_2, ..., n_t) \) is given by the following equation:

\[ E^{(t-1)}(N_2, ..., n_t) = \prod_{i=4}^{t-1} Q(n_i, M) \cdot Q(n_2, M) \cdot Q(n_3, M) - A(N_2, n_3, M) \] (59)

Thus, we can represent the numerator of equation (43) recursively in terms of itself such that every level of recursion reduces the number of transactions by one but the size of a transaction increases by the size of the reduced transaction. The algorithm can be represented by the following set of recursive equations:

\[ F^{(t-k)}(N_{k+1}, n_{k+2}, ..., n_t) = C(N_{k+2}, n_{k+2})[E^{(t-k-1)}(N_{k+2}, n_{k+3}, ..., n_t) \]

\[ + F^{(t-k-1)}(N_{k+2}, n_{k+3}, ..., n_t)] \quad 0 \leq k \leq 4 \] (60)

where:

\[ E^{(t-k)}(N_{k+1}, n_{k+2}, ..., n_t) \]

\[ = \prod_{i=k+3}^{t} Q(n_i, M) \cdot Q(n_{k+1}, M) \cdot Q(n_{k+2}, M) \]

\[ - A(N_{k+1}, n_{k+2}, M)] \quad 0 \leq k \leq 3 \] (61)

**Boundary Condition**

The recursion stops when we are left with three transactions of size \( N_{t-2}, n_{t-1}, \) and \( n_t. \) Next, we derive the value of the boundary condition, i.e., evaluate \( F^{(3)}(N_{t-2}, n_{t-1}, n_t). \)

\[ F^{(3)}(N_{t-2D}, n_{t-1}, n_t) = \left\{ \begin{array}{l}
\sum_{\forall \bar{W}_{t-2} \forall \bar{RV}_{t-1} \forall \bar{RV}_t}
\bar{P}(W_{t-2}) \cdot \bar{P}(RV_{t-1}) \cdot \bar{P}(RV_t)
\text{such that } \bar{w}_{t-2} \bar{A}_{t-1} \bar{d}
\text{and } \bar{w}_{t-1} \bar{A}_{t-1} = \bar{d} \text{ or } \bar{w}_{t-1} = \bar{d}
\end{array} \right\} = \frac{1}{n_t} \] (62)
At this point, we lump $\bar{w}_{t-2}$ and $\bar{n}_{t-1}$ into $\bar{w}_{t-1}$ such that $w_{(t-1)i} = w_{(t-2)i}$ OR $a_{(t-1)i}$ for every $1 \leq i \leq M$; consequently, the request vector $\bar{w}_{t-1}$ denotes the request vectors $\bar{w}_{t-2}$ and $\bar{a}_{t-1}$ combined and $\sum_{1 \leq i \leq M} w_{(t-1)i} = n_1 + n_2 + \ldots + n_{t-1} = N_{t-1}$. As a result, $F^{(3)}(N_{t-2}, n_{t-1}, n_t)$ becomes the summation of the product $P(W_{t-2})P(RV_{t-1})P(RV_t)$ over the domain where $\bar{w}_{t-1}$ and $\bar{a}_t$ conflict. With accesses $\bar{w}_{t-2}$ and $\bar{a}_{t-1}$ lumped together as an access $\bar{w}_{t-1}$, we can write the first term on the right hand side of equation (62) in the following way:

$$\sum_{\forall W_{t-2} \forall RV_{t-1}} P(W_{t-2})P(RV_{t-1}) = \left[ \frac{N_{t-1}}{n_{t-1}} \right] \times \prod_{i=1}^{M} (1-p_i) \times \sum_{\forall W_{t-1}} P(W_{t-1}) \sum_{\forall RV_i} P(RV_i)$$

Consequently, we can rewrite equation (62) for $F^{(3)}(N_{t-2}, n_{t-1}, n_t)$ as follows:

$$F^{(3)}(N_{t-2}, n_{t-1}, n_t) = \left[ \frac{N_{t-1}}{n_{t-1}} \right] \times \prod_{i=1}^{M} (1-p_i) \times \left[ \sum_{\forall W_{t-1} \forall RV_i} P(W_{t-1})P(RV_i) \right]$$

Using the notations developed, we can express $F^{(3)}(N_{t-2}, n_{t-1}, n_t)$ as,
which gives the boundary condition for the recursion.

\[ F^{(3)}(l_{l-2}, n_{l-1}, n_l) = C(l_{l-1}, n_{l-1}) + \left[ Q(l_{l-1}, M) Q(n_l, M) - A(l_{l-1}, n_l, M) \right] \tag{65} \]

Verification for Uniform Case

We prove that our method for computing the probability of conflict is correct for the special case of uniform data access distribution. The proof is by induction on the number \( t \) of transactions.

Base Step

The base of the induction is the case \( t=2 \). For this case, the probability of conflict is computed by the method presented in Section 3.2. For \( k \) data objects, and transactions of sizes \( i \) and \( j \), the computed probability of no conflict is \( \frac{P(i,j,k)}{Q(i,k)Q(i,k)} \).

In Section 3.1.1 we have shown that in the uniform case, the probability of no conflict is equal to \( U(i,j,k)/(V(i,k)V(j,k)) \), where

\[ U(i,j,k) = \binom{k}{i} \binom{k-i}{j} p^i(1-p)^{k-i} p^j(1-p)^{k-j} \tag{66} \]

and:

\[ V(i,k) = \binom{k}{i} p^i(1-p)^{k-i} \tag{67} \]

It remains to prove that in the uniform case \( U(i,j,k)=P(i,j,k) \) and \( V(i,k)=Q(i,k) \). First we show by induction on \( k \) that \( V(i,k)=Q(i,k) \). The base of the induction is implied by boundary conditions in equation (14). Suppose that \( V(i,k-1)=Q(i,k-1) \) for every \( i \). We will prove that \( V(i,k)=Q(i,k) \) for every \( i \). Using the induction hypothesis, equation (13), and the Pascal triangle identity \( \binom{a-1}{b-1} + \binom{a-1}{b} = \binom{a}{b} \), we get:
\[ Q(i, k) = (1-p)Q(i, k-1) + pQ(i-1, k-1) = (1-p)V(i, k-1) + pV(i-1, k-1) \]  (68)

\[ = (1-p) \left[ \binom{k-1}{i} p^i (1-p)^{k-1-i} + p \binom{k-1}{i-1} p^{i-1} (1-p)^{k-1-(i-1)} \right] \]

\[ = p^i (1-p)^{k-i} \left[ \binom{k-1}{i} + \binom{k-1}{i-1} \right] = p^i (1-p)^{k-i} \binom{k}{i} = V(i, k) \]

The next step is proving by induction on \( k \) that \( U(i, j, k) = P(i, j, k) \). The base of the induction is implied by the boundary conditions in equation (12). Suppose \( U(i, j, k-1) = P(i, j, k-1) \) for every \( i \) and \( j \). We prove that \( U(i, j, k) = P(i, j, k) \). Using the induction hypothesis, equation (11), and the Pascal triangle identity, we get:

\[ P(i, j, k) = (1-p)^2 P(i, j, k-1) + p(1-p)P(i-1, j, k-1) + (1-p)pP(i, j-1, k-1) \]  (69)

\[ = (1-p)^2 U(i, j, k-1) + p(1-p)U(i-1, j, k-1) + (1-p)pU(i, j-1, k-1) \]

\[ = p^i (1-p)^{k-i} \left[ \binom{k}{i} \left( \binom{k-1}{i} \binom{k-1-i}{j} + \binom{k-1}{i-1} \binom{k-1}{j} \right) \right] \]

\[ = p^i (1-p)^{k-i} \left[ \binom{k-1}{i} \binom{k-1-i}{j} + \binom{k-1}{i-1} \binom{k-1}{j} \right] \]

\[ = p^i (1-p)^{k-i} \left[ \binom{k-1}{j} \left( \binom{k}{i} \binom{k-1-i}{j} + \binom{k-1}{i-1} \binom{k-1}{j} \right) \right] \]

\[ = p^i (1-p)^{k-i} \left[ \binom{k}{i} \binom{k-1}{j} \right] \]

Induction Step

Now we prove the correctness for \( t \) transactions, by induction on \( t \). The base of the induction is the case \( t=2 \), which was proved above. Let \( d_n = p^n (1-p)^{M-n} \). In our proof for two transactions, we already showed that \( Q(M, n_1) = \binom{M}{n_1} d_n \). Therefore, the
denominator of equation (25) is equal to:

\[ Q = \prod_{i=1}^{I} \left( \frac{M}{n_i} \right) d_n = \prod_{i=1}^{I} \left( \frac{M}{n_i} \right) \prod d_n \]  

Equation (5) gives the probability of no conflict for the uniform case. Therefore, the probability of conflict is

\[ 1 - \frac{\prod_{i=1}^{I} \left( \frac{M-N_{i-1}}{n_i} \right)}{\prod_{i=1}^{I} \left( \frac{M}{n_i} \right)} = \frac{\prod_{i=1}^{I} \left( \frac{M}{n_i} \right) - \prod_{i=1}^{I} \left( \frac{M-N_{i-1}}{n_i} \right)}{\prod_{i=1}^{I} \left( \frac{M}{n_i} \right)} \]  

where \( N_0 = 0 \) and \( N_1 = n_1 \).

We denote \( E^{(l-k)}(N_{k+1}, n_{k+2}, \ldots, n_l) \) by \( E^{(l-k)} \), \( F^{(l-k)}(N_{k+1}, n_{k+2}, \ldots, n_l) \) by \( F^{(l-k)} \), and \( C(N_k, n_k) \) by \( C_k \). In the uniform case, \( C_k = (1-p)^M \left( \frac{N_k}{n_k} \right) \). Let

\[ H^{(l-k)} = H^{(l-k)}(N_{k+1}, n_{k+2}, \ldots, n_l) \]

\[ = d_n \prod_{i=k+2}^{l} d_n \left( \frac{M}{n_{k+1}} \right) \prod_{i=k+2}^{l} \left( \frac{M}{n_i} \right) - \prod_{i=k+2}^{l} \left( \frac{M-N_{i-1}}{n_i} \right) \]

It remains to be shown that \( H^{(l-k)} = E^{(l-k)} + F^{(l-k)} \). We prove by induction on \( k \) that \( H^{(l-k)} = E^{(l-k)} + F^{(l-k)} \). The induction hypothesis is: \( H^{(l-k-1)} = E^{(l-k-1)} + F^{(l-k-1)} \). Using the induction hypothesis and equation (60):

\[ F^{(l-k)} = C_{k+2} H^{(l-k-1)} \]

Therefore:

\[ F^{(l-k)} + E^{(l-k)} = C_{k+2} H^{(l-k-1)} + E^{(l-k)} \]

It remains to show that \( H^{(l-k)} \) is equal to the right hand side of equation (74). We have the following derivation:
\[ C_{k+2}H^{[t-k-1]} + E^{[t-k]} = \sum_{i=k+2}^{n+3} \binom{N_{k+1}}{n_{k+2}} \left( (1-p)^{M-N_{k+1}} (1-p)^{M-N_{k+2}} \prod_{i=k+3}^{t} p^n (1-p)^{M-n} \right) + E^{[t-k]} \]  

Using our proof for the case \( t=2 \):

\[ E^{[t-k]} = \prod_{i=k+3}^{t} Q(n_i, M) [Q(N_{k+1}, M) Q(n_{k+2}, M) - A(N_{k+1}, n_{k+2}, M)] \]

\[ = \prod_{i=k+3}^{t} \binom{M}{n_i} d_{n_i}, i \left( \prod_{i=k+3}^{t} \binom{M}{n_i} \right) \left( \binom{M}{n_{k+2}} d_{n_{k+2}} - d_{n_{k+2}}, n_{k+2} \right) \left[ \binom{M}{N_{k+1}} \left( \binom{M}{n_{k+2}} \right) - \binom{M}{n_{k+2}} \right] \]

We use the following derivation:

\[ d_{n_{i+1}, n_{k+2}, i} \]  

Also, using the identity \( \binom{a}{b} \binom{b}{c} = \binom{a-b}{b-c} \):

\[ \binom{M}{N_{k+2}} d_{n_{k+2}} + d_{n_{k+2}, n_{k+2}} = \binom{M}{N_{k+2}} d_{n_{k+2}} - d_{n_{k+2}, n_{k+2}} \]

Therefore, equation (75) becomes:

\[ C_{k+2}H^{[t-k-1]} + E^{[t-k]} = d_{n_{i+1}} (\prod_{i=k+2}^{t} d_i) (B+C) \]

where:

\[ B = \binom{M}{N_{k+1}} \left( \binom{M-N_{k+1}}{n_{k+2}} \prod_{i=k+3}^{t} \binom{M}{n_i} - \prod_{i=k+3}^{t} \binom{M-N_{i-1}}{n_i} \right) \]
It remains to show that \( B + C = E \), where:

\[
E = \prod_{i=k+3}^{t} \left[ \binom{M}{n_i} \right] \left[ \binom{M}{n_{k+2}} \right] - \left( \binom{M}{n_{k+1}} \right) \left( \binom{M-n_{k+1}}{n_{k+2}} \right) \tag{81}
\]

This is true because \( E \) is the number of possibilities of simultaneously choosing \( t-k \) subsets of a set of \( M \) elements, such that the sizes of the subsets are \( N_{k+1}, n_{k+2}, \ldots, n_i \), and such that the subsets are not pairwise disjoint. This number can be partitioned into two parts. The first part is the number of such choices, with the additional constraint that the first two subsets are disjoint. The second part is the number of such choices, with the additional constraint that the first two subsets are not disjoint. Clearly, the first part is equal to \( B \), and the second part is equal to \( C \).

**Verification for b-c Access Model**

Now we prove that the our computational method correctly computes the probability of conflict for the special case of the b-c data access model. The probability of no conflict for this model is given by equation (10). In Section 3.1.1 we prove that, for the b-c model, this equation is equivalent to equation (4). Equation (43) for the probability of conflict is equivalent to equation (4) (which gives the probability of conflict). This is clear, since the sum of these probabilities is 1. Using the above equivalences, it is enough to work with equation (43). In other words, we will prove that our computational method correctly computes equation (43). Our proof is by induction on the number of transactions.
Base Step

The base case is the case of two transactions. The computational method for this case is presented in Section 3.2. This computational method consists of the recurrence relations (11), (13), and the boundary conditions (12), (14).

We prove the correctness of this method by induction on the number k of data objects. Let RV\(^i\) denote an access vector of size i. Let D\(^k\) denote the vector (1,...,1) with k components. Using equation (4), the probability of no conflict for two transactions of sizes i and j is equal to \(U(i,j,k)/(V(i,k)V(j,k))\), where:

\[
U(i,j,k) = \sum_{RV \subseteq D^i} \sum_{RW \subseteq D^j-RV^i} P(RV^i)P(RW^j) 
\]

\[
V(i,k) = \sum_{RV \subseteq D^k} P(RV^i) 
\]

We have to show that \(U(i,j,k)=P(i,j,k)\) and that \(V(i,k)=Q(i,k)\), where \(P(i,j,k)\) is computed according to equations (11), (12), and \(Q(i,k)\) is computed according to equations (13), (14).

We first prove by induction on k that \(V(i,k)=Q(i,k)\). For \(k<0\), \(V(i,k)=0\). Also, \(Q(i,k)=0\), using the boundary conditions. Also, by definition of \(V(i,k)\), \(V(i,k)=1\) if \(i=k=0\), and \(V(i,k)=0\) if \(i>k\). Using the boundary conditions, the same is true for \(Q(i,k)\).

For the induction step, assume that \(V(i,k-1)=Q(i,k-1)\) for every i. We show that \(V(i,k)=Q(i,k)\) for every i. Using equation (13) and the induction hypothesis:

\[
Q(i,k) = (1-p_k)Q(i,k-1) + p_kQ(i-1,k-1) 
\]
\[(1-p_k)V(i,k-1)+p_kV(i-1,k-1)\]

\[=\sum_{RV \in D^{i-1}} P(RV^i) + p_k \sum_{RV \in D^{i-1}} P(RV^{i-1})\]

\[=\sum_{RV \in D^i, RV^i=0} P(RV^i) + \sum_{RV \in D^i, RV^i=1} P(RV^{i-1})\]

Next, we prove by induction on \(k\) that \(U(i,j,k) = \text{P}(i,j,k)\). As before, the base case follows from the boundary conditions. We now assume that \(U(i,j,k-1) = \text{P}(i,j,k-1)\) for every \(i,j\). We will prove that \(U(i,j,k) = \text{P}(i,j,k)\). Using equation (11) and the induction hypothesis:

\[P(i,j,k) = (1-p_k)^2 P(i,j,k-1) + p_k(1-p_k)P(i-1,j,k-1) + (1-p_k)p_k P(i,j-1,k)\]  \hspace{1cm} (86)

\[= (1-p_k)^2 U(i,j,k-1) + p_k(1-p_k)U(i-1,j,k-1) + (1-p_k)p_k U(i,j-1,k-1)\]

\[+ \sum_{RV \in D^i, RV^i=0} P(RV^i) P(RW^i)\]

\[+ \sum_{RV \in D^i, RV^i=1} P(RV^{i-1}) P(RW^i)\]

\[+ (1-p_k)p_k \sum_{RV \in D^i, RV^i=0} P(RV^i) P(RW^i)\]

Using the definitions of \(P(RV^i)\), \(P(RV^{i-1})\), \(P(RW^i)\), \(P(RW^{i-1})\), the above equation yields:

\[\sum_{RV \in D^i, RV^i=0} \sum_{RW \in D^i, RW^i=0} P(RV^i) P(RW^i)\]

\[+ \sum_{RV \in D^i, RV^i=1} \sum_{RW \in D^i, RW^i=0} P(RV^i) P(RW^i)\]

\[+ \sum_{RV \in D^i, RV^i=0} \sum_{RW \in D^i, RW^i=1} P(RV^i) P(RW^i)\]
\[ I(S(P(RV')P(RW))=U(i,j,k) \]

In the above derivation we used the observation that the set of \((RV',RW)\) where
\[ RW \subseteq D^k-RV \] and \(RV_k=RW_k=1\) is empty. This is true because \(RW \subseteq D^k-RV\)
implies: for every \(r\), \(RV_r=1\) implies \(RW_r=0\).

**Induction Step**

We now prove the correctness of the method for \(t\) transactions. Using the proof
for \(t=2\), the denominator of equation (43) is computed correctly. We will prove that
the numerator of equation (43) is computed correctly. We define \(N_1=n_1, N_0=0\). Let:

\[ Z_k = \{ P(RW_{k+1}) \cdots P(RV_{t}) \}_{\subseteq D^k-RV_k} \]

\(Z_k\) is the set of all \(RW_{k+1},RV_{k+2},...,RV_t\) that are not pairwise disjoint. \(RW_{k+1}\)
has size \(N_{k+1}\). \(RV_i\) has size \(n_i\).

In particular, for \(k=0\), \(H^{(0)}(n_1,n_2,...,n_t)\) is equal to \(H^{(0)}(n_1,n_2,...,n_t)\), which is the
numerator of equation (43). According to equation (43), we have to show that

\[ E^{(t-k)}(n_1,...,n_t)+F^{(t-k)}(n_1,...,n_t)=H^{(t-k)}(n_1,...,n_t) \]

where \(E^{(t-k)}(n_1,...,n_t)\) and \(F^{(t-k)}(n_1,...,n_t)\) are computed according to equations (60),
(61). We denote \(E^{(t-k)}(N_{k+1},n_{k+2},...,n_t)\) by \(E^{[t-k]}\), \(F^{(t-k)}(N_{k+1},n_{k+2},...,n_t)\) by \(F^{[t-k]}\),
\(H^{(t-k)}(N_{k+1},n_{k+2},...,n_t)\) by \(H^{[t-k]}\), and \(C(N_k,n_k)\) by \(C_k\). By induction on \(k\), we will
prove that

\[ E^{[t-k]}+F^{[t-k]}=H^{[t-k]} \]

The base case of two transactions was established above. This case corresponds to
\(t-k=2\). We assume that equation (90) is true for \(t-k-1\), and prove that it is true for \(t-k\)
The induction hypothesis is \( H^{[t-k-1]} = E^{[t-k-1]} + F^{[t-k-1]} \). Using the induction hypothesis and equation (60):

\[
F^{[t-k]} = C_{k+2} H^{[t-k-1]} \tag{91}
\]

Therefore:

\[
F^{[t-k]} + E^{[t-k]} = C_{k+2} H^{[t-k-1]} + E^{[t-k]} \tag{92}
\]

Therefore, in order to establish equation (90) it is enough to show:

\[
H^{[t-k]} = C_{k+2} H^{[t-k-1]} + E^{[t-k]} \tag{93}
\]

We complete the proof by proving equation (93). In equation (88), we split the summation into two parts. The first part is the summation of products \( P(RW_{k+1}) \cdot P(RV_{k+2}) \cdot \ldots \cdot P(RV_t) \) where \( RW_{k+1} \) and \( RV_{k+2} \) intersect. By equations (48), (49), this part is equal to \( E^{[t-k]} \).

The second part is the summation of products \( P(RW_{k+1}) \cdot P(RV_{k+2}) \cdot \ldots \cdot P(RV_t) \) where \( RW_{k+1} \) and \( RV_{k+2} \) are disjoint, but \( RV_{k+3}, \ldots, RV_t \) are not pairwise disjoint. We denote this sum by \( S \). It remains to show that \( S \) is equal to \( C_{k+2} H^{[t-k-1]} \). Let \( RW_{k+2} \) be the union of \( RW_{k+1} \) and \( RV_{k+2} \). Since \( RW_{k+1} \) and \( RV_{k+2} \) are disjoint, the size of their union \( RW_{k+2} \) is \( N_{k+2} = N_{k+1} + n_{k+2} \). We write \( S \) as:

\[
S = \sum_{RW_{k+2}} S[RW_{k+2}] \tag{94}
\]

where \( RW_{k+2} \) ranges over all vectors of size \( N_{k+2} \), and \( S[RW_{k+2}] \) is the part of \( S \) which consists of those products \( P(RW_{k+1}) \cdot P(RV_{k+2}) \cdot \ldots \cdot P(RV_t) \) in \( S \) where the union of \( RW_{k+1} \) and \( RV_{k+2} \) is equal to \( RW_{k+2} \). There are \( \binom{N_{k+2}}{n_{k+2}} \) possibilities for choosing \( RW_{k+1}, RV_{k+2} \), of sizes \( N_{k+1}, n_{k+2} \), respectively, such that their union is
equal to \( RW_{k+2} \). Let \( RW_{k+1} = (a_1, \ldots, a_M) \), \( RV_{k+2} = (b_1, \ldots, b_M) \), \( RW_{k+2} = (c_1, \ldots, c_M) \). Also:

\[
P(RW_{k+1})P(RV_{k+2}) = \prod_{a=1}^{p} \prod_{a=0}^{n} (1-p_i) \prod_{b=1}^{q} \prod_{b=0}^{m} (1-p_i)
\]

Because \( RW_{k+1} \) and \( RV_{k+2} \) are disjoint, the above equation is equal to:

\[
= \prod_{a=1}^{p} \prod_{a=0}^{n} (1-p_i) \prod_{c=1}^{r} \prod_{c=0}^{r} (1-p_i)
\]

Consequently:

\[
S[RW_{k+2}] = \left[ \frac{N_{k+2}^{\text{all } i}}{n_{k+2}} \right] (\prod_{a=1}^{p} \prod_{a=0}^{n} (1-p_i))P(RW_{k+2})
\]

\[
= \sum_{RV_{k+1}, \ldots, \text{RV}_t \text{ not pairwise disjoint}} P(RV_{k+3}) \cdots P(RV_t)
\]

Therefore, by definition of \( H^{[l-k-1]} \)

\[
S = C_{k+2} H^{[l-k-1]}
\]

and the proof is completed.

4.1.3 Complexity Analysis

Apparently the derivation of the algorithm (i.e., the set of equations for \( F^{(l)}(\ldots) \)) \( E^{(l)}(\ldots) \) is fairly intricate; nevertheless, the algorithm permits very fast computation of the numerator of equation (43). (The denominator of equation (43) is \( Q'(n, M) \)
which is automatically computed as part of the numerator.) The algorithm uses the
functions \( C(N_k, n_k) \) for \( 2 \leq k \leq (t-1) \), and \( Q(N_k, M) \) and \( A(N_k, n_{k+1}, M) \) for \( 1 \leq k \leq (t-1) \).
All of the functions \( C(N_k, n_k) \) for \( 2 \leq k \leq (t-1) \) can be calculated in about \( 4N_{t-2} \)
operations. Likewise, by using the technique of the Appendix we can show that all
the functions \( Q(N_k, M) \) and \( A(N_k, n_{k+1}, M) \) for \( 1 \leq k \leq (t-1) \) can be computed and
stored in arrays in about \( 10n_{t-2}^2 \) operations and can be used during the
computation of the functions \( F^{(t)}(...) \) and \( E^{(t)}(...) \). If \( CC[f] \) denotes the computation
complexity of a function \( f \), then the complexity of the numerator of equation (43) is:

\[
CC[H^{(t)}(n_1, n_2, ..., n_t)] + \max(CC[Q(...)], CC[A(...)]) =
\]

\[
CC[E^{(t)}(n_1, n_2, ..., n_t)] + CC[F^{(t)}(n_1, n_2, ..., n_t)] + \max(CC[Q(...)], CC[A(...)])
\]

Since:

\[
CC[F^{(t-k)}(N_{k+1}, n_{k+2}, ..., n_t)] =
\]

\[
CC(C(N_{k+2}, n_{k+2})) + CC[E^{(t-k-1)}(N_{k+2}, n_{k+3}, ..., n_t)]
\]

\[
+ CC[F^{(t-k-1)}(N_{k+2}, n_{k+3}, ..., n_t)],
\]

\( 0 \leq k \leq t-4 \)

we can write the above equation as:

\[
= \sum_{i=3}^{t} CC[E^{(i)}(...)] + \sum_{i=2}^{t-2} CC[C(N_i, n_i)]
\]

\[
+ CC[F^{(3)}(...)] + \max(CC[Q(...)], CC[A(...)])
\]

\[
= \sum_{i=3}^{t} (i) + \sum_{i=2}^{t-2} (4n_i)
\]
+ [4n_{t-1} + 4] + 10n_{\text{max}}^2(t-1)M

=(t-2)(t+3)/2 + 4(N_{t-2} - n_t) + [4n_{t-1} + 4] + 10n_{\text{max}}^2(t-1)M

=(t-2)(t+3)/2 + 4[N_{t-1} - n_t + 1] + 10n_{\text{max}}^2(t-1)M

Therefore, the order of complexity is \(O(t^2) + O(n_{\text{max}}^2(t-1)M)\), which is much lower than the exponential complexity we had before.

### 4.2 Extension to Dynamic Lock Acquisition Policy

Thus far, our analysis is based on static locking policy; that is, a transaction acquires locks on all the needed data objects before it starts execution. In this section, we extend our results to dynamic lock acquisition policy (where locks on data objects are acquired as and when needed during the transaction execution).

A transaction is termed *active* if it is running (i.e., has all the data objects needed for execution) or is blocked due to unavailability of a needed data object and holds lock on at least one data object. (Thus, we do not count transactions which are blocked on their first data request.) In dynamic lock acquisition policy, conflict or blocking can arise due to two reasons: First, a newly arrived transaction finds that a data object requested by it is already taken by active transactions, and second, a running transaction makes a request for a data object which is already taken by another active transaction. (Conflict and blocking are synonymous because blocking occurs if and only if there is a conflict.) We show that both types of blocking can be handled by the algorithm presented in Section 4.1.2.
Consider a state of the system where 't' transactions $T_1, T_2, ..., T_t$ are active with their current request vectors $RV_1 (= a_1), RV_2 (= a_2), ..., RV_t (= a_t)$, respectively, and their sizes are $n_1, n_2, ..., n_t$, respectively. A conflict will arise if a newly arrived transaction $T_{t+1}$ requests a data object which is locked by one of the 't' active transactions or if a running transaction requests a new data object which is already blocked by any of the other 't-1' active transactions. Although both types of conflict arise if a newly requested data object happens to be one of the already locked ones, the probability of conflict and blocking is different for both types (in the same system state). This is because an already running transaction cannot request a data object which is already locked by it, while a newly arrived transaction can request any of the 'M' data objects. Next, we analyze the probability of conflict and blocking for both types of requests in the state described above (in fact, we analyze the probability of no conflict and no blocking).

**The Probability of no Conflict for the First Type of Requests**

The probability that a newly arrived transaction $T_{t+1}$ is immediately executed (i.e., does not block) when 't' transactions are already active is given by the following equation:

$$P(\text{An arriving trans. does not block} \mid \text{t trans. of sizes } n_1, n_2, ..., n_t \text{ are already active})$$

$$= \frac{\text{Probability that } '(t+1)\text{ transactions of sizes } n_1, n_2, ..., n_t \text{ run concurrently}}{\text{Probability that } 't\text{ transactions of sizes } n_1, n_2, ..., n_t \text{ run concurrently}}$$ (102)
The Probability of no Conflict for the Second Type of Requests

The probability that a request of an already running transaction $T_i$, $1 \leq i \leq t$, does not get blocked when it is running with 't-1' active transactions is given by the following equation:

$$P(\text{Request of a transaction } T_i \text{ does not block } I \text{ 't-1' trans. of sizes } n_1, n_2, ..., n_{t-1}, n_{t+1}, ..., n_t \text{ are running with it})$$

$$= \frac{\text{Probability that 't' trans. of sizes } n_1, n_2, ..., n_{t+1}, ..., n_t \text{ run concurrently}}{\text{Probability that 't' trans. of sizes } n_1, n_2, ..., n_t \text{ run concurrently}}$$ (103)

Note that equations (102) and (103) are similar to equation (3) and can be computed in the same way (i.e., by computing their numerators and denominators by using the algorithm in Section 4.1.2). Therefore, conflicts and blocking due to both types of requests can be treated uniformly by using the algorithm discussed in Section 4.1.2. Analysis of the probability of transaction blocking in dynamic lock acquisition environments is required in the performance analysis of dynamic locking algorithms [8] as well as in the analysis of the frequency of deadlocks [6] in environments of non-uniform data access distribution.

4.3 Numerical Examples

In this section, we present numerical results to illustrate the algorithm presented in Section 4.1.2 for computing the probability of conflict under non-uniform data access distribution. We assume $M=1000$, $n=8$, and that data access is distributed according to a truncated Normal distribution (that is, $p_i = C \cdot e^{-\frac{1}{2}(i-\mu)^2}$, $1 \leq i \leq M$,)$.
where $C$ is a constant such that $\sum_{i} p_i = \text{transaction size over all } i$. We compute the probability of conflict among transactions and the probability that an arriving transaction does not get blocked due to conflict using the algorithm of Section 4.1.2 and equation (3), respectively, for various values of the number of concurrent transactions ($t$) and variance of the Normal distribution ($\sigma$). To elucidate the difference in the probability of conflict and the probability of transaction blocking which results due to non-uniform data access distribution, we also compute these probabilities for uniform access distribution. Results of the numerical study are compiled in Tables 4 and 5.

**Table 4. The Probability of Conflict among 't' Transactions (for $\mu=500$)**

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>$t=4$</th>
<th>$t=5$</th>
<th>$t=6$</th>
<th>$t=7$</th>
<th>$t=8$</th>
<th>$t=9$</th>
<th>$t=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.4816</td>
<td>0.6665</td>
<td>0.8084</td>
<td>0.9017</td>
<td>0.9550</td>
<td>0.9817</td>
<td>0.9933</td>
</tr>
<tr>
<td>300</td>
<td>0.3858</td>
<td>0.5571</td>
<td>0.7062</td>
<td>0.8208</td>
<td>0.8995</td>
<td>0.9483</td>
<td>0.9755</td>
</tr>
<tr>
<td>400</td>
<td>0.3508</td>
<td>0.5141</td>
<td>0.6623</td>
<td>0.7821</td>
<td>0.8695</td>
<td>0.9276</td>
<td>0.9627</td>
</tr>
<tr>
<td>500</td>
<td>0.3365</td>
<td>0.4962</td>
<td>0.6434</td>
<td>0.7648</td>
<td>0.8556</td>
<td>0.9175</td>
<td>0.9561</td>
</tr>
<tr>
<td>$\infty$(unif.)</td>
<td>0.1236</td>
<td>0.1979</td>
<td>0.2823</td>
<td>0.3722</td>
<td>0.4634</td>
<td>0.5519</td>
<td>0.6344</td>
</tr>
</tbody>
</table>
Table 5. The Probability of a Transaction not Blocking (for \( \mu = 500 \))

<table>
<thead>
<tr>
<th>( \sigma )</th>
<th>( t=4 )</th>
<th>( t=5 )</th>
<th>( t=6 )</th>
<th>( t=7 )</th>
<th>( t=8 )</th>
<th>( t=9 )</th>
<th>( t=10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.6432</td>
<td>0.5746</td>
<td>0.5128</td>
<td>0.4572</td>
<td>0.4071</td>
<td>0.3625</td>
<td>0.3213</td>
</tr>
<tr>
<td>300</td>
<td>0.7210</td>
<td>0.6634</td>
<td>0.6099</td>
<td>0.5605</td>
<td>0.5146</td>
<td>0.4723</td>
<td>0.4330</td>
</tr>
<tr>
<td>400</td>
<td>0.7484</td>
<td>0.6950</td>
<td>0.6634</td>
<td>0.5605</td>
<td>0.5549</td>
<td>0.5141</td>
<td>0.4759</td>
</tr>
<tr>
<td>500</td>
<td>0.7593</td>
<td>0.7078</td>
<td>0.6594</td>
<td>0.6140</td>
<td>0.5714</td>
<td>0.5314</td>
<td>0.4941</td>
</tr>
<tr>
<td>( \infty )(unif.)</td>
<td>0.9152</td>
<td>0.8948</td>
<td>0.8746</td>
<td>0.8547</td>
<td>0.8352</td>
<td>0.8159</td>
<td>0.7968</td>
</tr>
</tbody>
</table>

4.3.1 Discussion of the Numerical Results

The numerical results confirm that the assumption of uniform data access distribution does underestimate the probability of conflict and the probability of transaction blocking.

The results also indicate that an increase in the number of transactions (\( t \)) results in a higher probability of conflict and a higher probability of blocking — the larger the number of transactions, the larger the number of data objects locked by the transactions, resulting in higher chances of overlap in data access. The probability of a transaction not blocking is more sensitive to the number of concurrently running transactions when \( \sigma \) is small as compared to when \( \sigma \) is large. When \( \sigma \) is smaller, the fall in the probability of not blocking is much steeper as a function of \( 't' \). For example, when \( 't' \) varies from 4 to 10, the probability of not blocking falls by 50
percents for $\sigma=200$ as compared to 35 percents when $\sigma=500$ and 13 percents when $\sigma=\infty$ (Uniform).

### 4.4 Summary

Past performance analyses of concurrency control algorithms have assumed that data access distribution of transactions is uniformly distributed across the entire database because the amount of computation required to accurately capture any arbitrary data distribution was not justified due to high computation cost. This is because computation of the probability of blocking of a transaction for an arbitrary data access distribution is very expensive even for a database of small size (due to combinatorial explosion).

Since the assumption of uniform data access gives a lower bound on the probability of conflict among transactions [31] and since lower probability of conflict among transactions leads to performance enhancement (because less transactions get blocked and are delayed), the assumption of uniform data access distribution has led past performance studies to compute optimistic bounds on the performance of concurrency control algorithms (e.g., an upper bound on the system throughput and a lower bound on the transaction response time).

Since in practice transactions exhibit locality of data reference, elimination of the assumption of uniform data access distribution is essential for more accurate performance analysis of real-life databases. In this chapter, we have presented an
efficient algorithm for computing the probability of transaction blocking under arbitrary data access distribution which makes it possible to accurately reflect any data access distribution in the performance analysis of concurrency control algorithms at a moderate cost of computation.

We have illustrated the applicability of the algorithm by using numerical examples where we have computed the probability of conflict among ‘t’ transactions and the probability that an arriving transaction does not get blocked due to conflict. The examples confirm that the assumption of uniform data access distribution indeed underestimates the probability of conflict and the probability of blocking, and in some cases the error can be substantial. The results given in the examples also show that the probability of a transaction not blocking is more sensitive to the number of concurrently running transactions when σ is small as compared to when σ is large.

We have also analyzed the probability of transaction blocking in the environments of dynamic lock acquisition under non-uniform data access distribution. We have shown that the behavior of a transaction under dynamic lock acquisition environments can be analyzed relatively inexpensively (and quickly) by using the algorithm for static lock acquisition policy (Section 4.1.2). Fast analysis of transaction blocking in dynamic locking environments permits the performance study of dynamic locking algorithms as well as the analysis of the frequency of deadlocks in non-uniform data access environments at a moderate cost of computation.
Concurrency control mechanisms based on the notion of locking were among the first to be reported in the literature [8] and are among the most commonly used. A locking mechanism can be static or dynamic. The basic principle of static locking is that a transaction must lock all the data objects before it starts execution. If one or more requested locks are held by another transaction, then none of the requested locks are granted. Otherwise, the transaction receives all the locks and proceeds with its execution. The locks must be exclusive if a write is to be performed, otherwise shared locks may be used.

The performance of static locking algorithms has been studied by a number of researchers [9, 17, 19, 27, 35], using both analytic and simulation techniques. Ries and Stonebraker have reported extensive simulations on static locking to study the effect of granularity on locking. In their study they considered exclusive locking, both static and dynamic, in centralized and distributed systems. Their studies concluded that dynamic locking is better than static locking when transactions are short, and worse when they are long. Also dynamic locking has higher throughput than static locking when the workload is light, and lower when it is heavy.
The first analytical model for static locking was introduced by Potier and Leblanc [25]. They applied the hierarchical decomposition approach to analysis of database systems with static locking. The decomposition separates the data contention and resource contention into two modules. The authors used combinatorial analysis on the first module, and queueing analysis on the second. The main contribution of their study is the separation of the data and resource contention issues. Another queueing network model of static locking (except that by Potier and Leblanc), which incorporates data and resource contention, was proposed by Galler [9]. However, the usefulness of his model is limited. The model produces accurate results only if the waiting time for a transaction request is deterministic, and the transaction size is one.

One of the most successful works in the area of the performance of static locking was done by Thomasian [39]. His model is powerful and at the same time easy to understand. A recent study of locking mechanisms, performed by Pun and Belford [26], also deserves credit. By using a combinatorial argument and probability theory, they derive analytical expressions for quantities such as the probability of a lock being granted, blocking delay, and the probability of deadlock. However, their analysis uses several approximations and oversimplified assumptions, which make their model not very useful.

Although there have been a number of performance studies of static locking, just very few incorporated a non-uniform data access distribution in their analysis [24]. This was because analysis of concurrency control algorithms (locking in particular) is intractable under the assumption of arbitrary data access distribution.
However, with the help of numerical examples, we have shown in this dissertation that the assumption of uniform data access distribution gives us lower and upper bounds on the estimate of such important system parameters as response time and throughput.

In order to analyze the behavior of locking mechanisms under more realistic conditions, we present in this chapter performance analysis of static locking mechanism under non-uniform data access distribution. We present a simple, yet very powerful queueing network model of locking mechanism under an arbitrary data access distribution. We show that the expressions for quantities such as the probability of lock being granted, the mean wait time for a blocked request (blocking delay) and throughput can be derived analytically. The numerical results based on the analytical model show that the assumption of uniform data access introduces significant error in the performance analysis of the static locking mechanism.

5.1 Notation, Assumptions and Performance Measures

We begin with a discussion of notations. The following are the definitions of the parameters and notations for our model:

1) \( N \): The degree of multiprogramming, i.e., the number of transactions in the system.

2) \( N_a \): The number of active transactions in the system.
3) \( N_b \): The number of blocked transactions in the system.

4) \( M \): Database size, i.e., the total number of data objects in the database.

5) \( K \): Transaction size.

6) \( S(N_a) \): Service rate of the system (msec of work/msec elapsed time) when there are \( N_a \) active transactions in the system.

7) \( W_a \): Average total time needed for an active transaction to complete its execution.

8) \( W_b \): Average time needed for a blocked transaction to obtain its locks (blocking delay).

9) \( \bar{p} = (p_1, p_2, p_3, \ldots p_M) \): is an \( M \)-dimensional vector which represents the probability distribution of data access.

The performance of static locking can be judged on the basis of several performance measures, e.g., system throughput, transaction response time, probability of conflict, blocking delays and so on. In this performance study, we will compute the following performance measures:

**R**: Transaction response time, which is the time interval between

the instant when a transaction is submitted by a user and the instant

when the transaction is completely executed.
t: System throughput, which is the rate of the completion of transactions.

W_b: Blocking delay, which is the time that the blocked transaction needs to wait to acquire its locks.

P: Probability of not blocking.

Response time is an important measure from the point of view of user satisfaction because it reflects how long a user has to wait for the execution of its transaction. Throughput is important for a database manager who would like to know at what rate his system can execute transactions. The probability of not blocking and the blocking delays are important because they play a crucial role in determining the values for throughput and response time.

5.2 Queueing Network Model of the System

Transaction behavior obeying static locking is depicted in Figure 6. When a transaction enters the system, it begins execution by requesting all its locks at once. If at least one of the locks requested is already held by some other transaction, then none of them will be granted and the requesting transaction is blocked and waits until its locks can be granted. Otherwise, all the locks requested will be granted and the transaction is allowed to access the data objects.

We assume a closed system, i.e., there are N transactions in the system. When a completion occurs, a transaction is replaced by a statistically identical transaction.
Whenever a transaction departs, all blocked transactions are checked in FCFS (first come first serve) order to see if they can become runnable. Since the number of transactions in the system is constant, \( N = N_a + N_b \). We aggregate the CPU and disk complex by a load-dependent server. The Markov chain of this server system is shown in Figure 7, where a state \( N_a \) of the Markov chain represents that there are \( N_a \) transactions in the load-dependent server complex. \( S(N_a) \) is the rate at which the load dependent server serves the transactions when there are \( N_a \) transactions at this server, \( 1 \leq N_a \leq N \). The Markov chain of Figure 7 can be analyzed using the technique
described in [10,15,22]. The steady state distribution of the Markov chain, \( \{P_0, P_1, \ldots, P_k\} \) is given by the following equation:

\[
P_i(k) = \left( \frac{\mu_2}{\mu_1} \right)^i P_0(k) \quad i=1,\ldots,k
\]

where

\[
P_0(k) = \frac{1}{1 + \left( \frac{\mu_2}{\mu_1} \right) + \left( \frac{\mu_2}{\mu_1} \right)^2 + \ldots + \left( \frac{\mu_2}{\mu_1} \right)^k}
\]

Therefore, the rate at which the load-dependent server serves the transactions can be computed using the following equation:

\[
S(N_tM_l- P_0(N_t) = \mu_1
\]

The values of \( \mu_1, \mu_2 \) are the speeds of CPU and disk, respectively.

![Figure 7. The Markov Chain](image)
5.3 Performance Analysis

Due to the complex nature of the static locking algorithm, it is very difficult to accurately analyze its performance mathematically. We reduce the complexity of the analysis by analyzing a single transaction in isolation rather than analyzing the whole system. We reflect the presence of other transactions on the transaction by the probability that a conflicting access to a data object is made. In our performance study, we assume that a transaction sees the average state of the system and all the transactions exhibit the average state behavior.

A crucial step in analyzing the model is to obtain an expression for the probability of not blocking (P). P represents the probability that an arriving transaction (let's call it T) becomes active (i.e., locks are granted) given that there are already several active transactions in the system. An efficient technique for computing the probability of not blocking under non-uniform data access distribution is given in Section 4.1.2.

In our model, we keep track of the number of blocked transactions, and the number of active transactions. Since we have a closed system, we expect that when a transaction finishes its execution, a blocked transaction will become active. Thus, the number of active transactions in the system remains the same for the new transaction which immediately enters. In case the transaction gets blocked, it will join the blocked queue. The expected blocked queue length is $N-N_a$, or in other words the number of blocked transactions in the system at any moment can be computed as
\[ N_b = N - N_a \quad (107) \]

On the other hand, the number of blocked transactions depends on the probability of blocking, blocking delay and system throughput. Therefore,

\[ N_b = W_b \cdot t \cdot (1 - P) \quad (108) \]

Using Little’s Law [20] the equation for the throughput can be given by the following formula:

\[ t = \frac{N}{W_a + W_b} \quad (109) \]

Thus the value of \( W_b \) can be computed using the following equation:

\[ W_b = \frac{N}{t \cdot (1 - P)} = \frac{N - (t \cdot W_a)}{t} \quad (110) \]

The value for \( W_a \) (the average time needed for an active transaction to complete its execution) is given by

\[ W_a = (K + 1) \cdot S(N_a) \quad (111) \]

K steps are required to execute K requested locks and one step is required for transaction completion. However, the performance measures we are mainly interested in are \( W_b \) (blocking delay), \( t \) (throughput of the system), and response time. But how do we determine them? Here we get into an iterative situation. Examining equations (107)-(111) we can conclude that the choice for \( t \) (throughput) determines the value of \( W_b \) (blocking delay), which when substituted into the model, generates a new computed value for the system throughput. We define percent difference between "old value of throughput" of the previous iteration and "new value of throughput" of the current iteration to be 0.01. In our analysis, we terminate
the iterative process when the percentage difference is less than 1. We choose the starting throughput for the iteration using the following equation,

\[ t = \frac{N}{(K+1) \cdot S(N_a)} \]  \hspace{1cm} (112)

which is the total processing time requested of a transaction without taking blocking delay into consideration [20]. In all the cases, we found that the process terminates after no more than six iterations. A sample output of the iterative process is presented in Table 6.

5.4 Numerical Results

In this section, we present numerical results to illustrate the analysis of the static locking algorithm. We assume that the database size is \( M \) (\( M=1000 \) or \( M=500 \)). We

<table>
<thead>
<tr>
<th>Iteration</th>
<th>old t</th>
<th>new t</th>
<th>percent difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01399</td>
<td>0.01233</td>
<td>11.89</td>
</tr>
<tr>
<td>2</td>
<td>0.01233</td>
<td>0.01093</td>
<td>11.35</td>
</tr>
<tr>
<td>3</td>
<td>0.01093</td>
<td>0.01062</td>
<td>2.83</td>
</tr>
<tr>
<td>4</td>
<td>0.01062</td>
<td>0.01049</td>
<td>1.22</td>
</tr>
<tr>
<td>5</td>
<td>0.01049</td>
<td>0.01041</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Table 6. Convergence for Throughput
vary transactions sizes between 4 and 14 (K=4,6,8,...,14) and vary the number of active transactions in the system between 4 and 14 (tr=4,6,8,...,14). The data access is distributed according to truncated Normal distribution (that is, \( p_i = C e^{-\frac{1}{2}((i-\mu_1)\sigma)^2}, 1 \leq i \leq M \), where \( C \) is a constant such that \( \sum p_i = \text{transaction size} \). The values for \( S(N_a) \) are derived using the technique described in Section 5.2 and \( \mu_1=10\text{msec} \), and \( \mu_2=25\text{msec} \).

We compute the probability that an arriving transaction does conflict with already running transactions in the system using the technique of Section 4.1.2 for various values of the number of concurrent transactions (tr) and variance of the Normal distribution (\( \sigma \)). Results of the analytical study are compiled in Table 7. The numerical results indicate that an increase in the number of transactions (tr) results in

<table>
<thead>
<tr>
<th>tr</th>
<th>( \sigma=200 )</th>
<th>( \sigma=300 )</th>
<th>( \sigma=400 )</th>
<th>( \sigma=\infty ) (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.4816</td>
<td>0.3858</td>
<td>0.3508</td>
<td>0.1236</td>
</tr>
<tr>
<td>5</td>
<td>0.6665</td>
<td>0.5571</td>
<td>0.5141</td>
<td>0.1979</td>
</tr>
<tr>
<td>6</td>
<td>0.8084</td>
<td>0.7062</td>
<td>0.6623</td>
<td>0.2823</td>
</tr>
<tr>
<td>7</td>
<td>0.9017</td>
<td>0.8208</td>
<td>0.7821</td>
<td>0.3722</td>
</tr>
<tr>
<td>8</td>
<td>0.9550</td>
<td>0.8995</td>
<td>0.8695</td>
<td>0.4634</td>
</tr>
<tr>
<td>9</td>
<td>0.9817</td>
<td>0.9483</td>
<td>0.9276</td>
<td>0.5519</td>
</tr>
<tr>
<td>10</td>
<td>0.9933</td>
<td>0.9755</td>
<td>0.9627</td>
<td>0.6344</td>
</tr>
</tbody>
</table>
a higher probability of conflict. The larger the number of running transactions - the larger the number of data objects locked by the transactions, which results in higher chances of overlap in data access. We also notice that the probability of conflict is more sensitive to the number of concurrently running transactions when \( \sigma \) is large. When \( \sigma \) is large, the increase in the probability of conflict is steeper with \( \text{tr} \).

Using the method developed in Section 5.3 we compute the values of throughput of the system for various values of the number of concurrent transactions \( \text{tr} \) and variance of the Normal distribution \( \sigma \). The numerical results are compiled in Table 8 and plotted in Figure 8.

As expected, with the increased number of running transactions we are getting higher values for the throughput. However, we notice that the smaller the \( \sigma \) the smaller the values of throughput. And the difference for values of throughput when

<table>
<thead>
<tr>
<th>( \text{tr} )</th>
<th>( \sigma=200 )</th>
<th>( \sigma=300 )</th>
<th>( \sigma=400 )</th>
<th>( \sigma=\infty ) (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.02459</td>
<td>0.0250</td>
<td>0.02631</td>
<td>0.03269</td>
</tr>
<tr>
<td>8</td>
<td>0.03133</td>
<td>0.03162</td>
<td>0.03278</td>
<td>0.03970</td>
</tr>
<tr>
<td>10</td>
<td>0.03846</td>
<td>0.03962</td>
<td>0.04087</td>
<td>0.04778</td>
</tr>
<tr>
<td>12</td>
<td>0.03964</td>
<td>0.04068</td>
<td>0.04192</td>
<td>0.04854</td>
</tr>
<tr>
<td>14</td>
<td>0.03138</td>
<td>0.03169</td>
<td>0.03278</td>
<td>0.04163</td>
</tr>
</tbody>
</table>
\( \sigma = 200 \) as compared to \( \sigma = \infty \) (uniform) can be larger than 30 percents. This results confirm the necessity of incorporation of data access distribution in the performance analysis of the concurrency control algorithms and database systems in general.

Next, we examine the effect that transaction size has on the values of the probability of conflict and throughput. In our analysis we assume that we have 4 concurrently running transactions and that the database has 1000 data objects (\( M = 1000 \)). The values for the probability of conflict with variable transaction sizes are recorded in Table 9. As expected, the larger the transactions are getting in size, the higher the probability of conflict is. Since more data objects are locked, the
Table 9. The probability of Conflict (for $\mu=500$, $M=1000$, $t=4$)

<table>
<thead>
<tr>
<th>$K$</th>
<th>$\sigma=200$</th>
<th>$\sigma=300$</th>
<th>$\sigma=400$</th>
<th>$\sigma=\infty$ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.05928266</td>
<td>0.04437109</td>
<td>0.03942000</td>
<td>0.035552</td>
</tr>
<tr>
<td>8</td>
<td>0.10316290</td>
<td>0.07765577</td>
<td>0.06912923</td>
<td>0.062449</td>
</tr>
<tr>
<td>10</td>
<td>0.15675470</td>
<td>0.11886585</td>
<td>0.10608316</td>
<td>0.096032</td>
</tr>
<tr>
<td>12</td>
<td>0.21811870</td>
<td>0.16689604</td>
<td>0.14968725</td>
<td>0.135565</td>
</tr>
<tr>
<td>14</td>
<td>0.28512185</td>
<td>0.22130661</td>
<td>0.21475087</td>
<td>0.180195</td>
</tr>
</tbody>
</table>

The probability of acquiring the locks for a specific transaction is getting smaller. We also notice that as the distribution drifts away from being uniform ($\sigma=\infty$), the probability of conflict increases drastically. In certain cases the difference can be as large as 60 percents. Table 10 contains the values for throughput for various transaction sizes. These values are plotted in Figure 9. The numerical results confirm that throughput is degraded by increasing the transaction size. We also notice that the data access distribution has similar effect on the values of throughput as we observed in previous studies, i.e., the smaller the $\sigma$ the smaller the values for throughput we get.

For example, when $K=14$ and $\sigma=200$ the value of throughput is smaller by 36 percents as compared to the value of the throughput obtained when $K=14$ and $\sigma=\infty$ (uniform). The values for the response time are recorded in Tables 11, 12, and plotted in Figures 10, 11.
Table 10. System Throughput - II

<table>
<thead>
<tr>
<th>K</th>
<th>( \sigma=200 )</th>
<th>( \sigma=300 )</th>
<th>( \sigma=\infty )(uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.026976</td>
<td>0.027442</td>
<td>0.029073</td>
</tr>
<tr>
<td>8</td>
<td>0.020034</td>
<td>0.020565</td>
<td>0.022145</td>
</tr>
<tr>
<td>10</td>
<td>0.015604</td>
<td>0.016145</td>
<td>0.017678</td>
</tr>
<tr>
<td>12</td>
<td>0.012576</td>
<td>0.013091</td>
<td>0.014575</td>
</tr>
<tr>
<td>14</td>
<td>0.010414</td>
<td>0.010874</td>
<td>0.013115</td>
</tr>
</tbody>
</table>

Figure 9. System Throughput - II
Table 11. Response time (K=8, μ=500, M=500)

<table>
<thead>
<tr>
<th>tr</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=400</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>244.00</td>
<td>240.00</td>
<td>228.05</td>
<td>183.54</td>
</tr>
<tr>
<td>8</td>
<td>255.34</td>
<td>253.00</td>
<td>244.05</td>
<td>201.51</td>
</tr>
<tr>
<td>10</td>
<td>260.01</td>
<td>252.39</td>
<td>244.67</td>
<td>209.29</td>
</tr>
<tr>
<td>12</td>
<td>302.72</td>
<td>294.98</td>
<td>286.25</td>
<td>247.21</td>
</tr>
<tr>
<td>14</td>
<td>446.14</td>
<td>441.77</td>
<td>427.08</td>
<td>291.36</td>
</tr>
</tbody>
</table>

Table 12. Response time (tr=4, μ=500, M=1000)

<table>
<thead>
<tr>
<th>K</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>148.27</td>
<td>145.76</td>
<td>137.58</td>
</tr>
<tr>
<td>8</td>
<td>199.66</td>
<td>194.50</td>
<td>180.62</td>
</tr>
<tr>
<td>10</td>
<td>256.34</td>
<td>247.75</td>
<td>226.26</td>
</tr>
<tr>
<td>12</td>
<td>318.04</td>
<td>305.55</td>
<td>274.42</td>
</tr>
<tr>
<td>14</td>
<td>384.09</td>
<td>367.84</td>
<td>304.98</td>
</tr>
</tbody>
</table>
Figure 10. Response Time - I

Figure 11. Response Time - II
It is obvious, and confirmed by the numerical results, that the more transactions we have in the system and the larger the transaction sizes are - the bigger the values for response time we get. From Tables 11, 12, we can conclude that the more non-uniform the distribution of data access of the transactions is, the longer we have to wait for their executions. Thus, uniform data access distribution gives us a lower bound on the values of the response time.

5.5 Summary

In this chapter we presented the analysis of static locking under non-uniform data access distribution. The main contribution of this analysis is the use of analytic techniques as opposed to simulation techniques to study the performance of static locking. We presented a queueing network model of the static locking algorithm and showed that the values for such system performance measures as throughput, response time and blocking delays can be derived analytically.

The presented analysis is the first one, which incorporated the non-uniformity of data access in their model at the very fine level, i.e., at the level of data object without making any oversimplified assumptions or approximations. The analytical results obtained have confirmed the fact that the incorporation of non-uniformity is essential for more realistic and accurate studies of the performance analysis of concurrency control algorithms.
CHAPTER VI

PERFORMANCE ANALYSIS OF TWO-PHASE LOCKING

Thus far, we have analyzed the performance of static locking policy, in which a transaction acquires locks on all the required data objects before it starts executing. In this chapter, we extend the performance study to two-phase locking acquisition policy (in which locks on data objects are acquired as and when needed during the transaction execution [8]). Static locking mechanism requires locking the data objects for longer periods of time than are needed under two-phase locking. That is why two-phase locking is more popular and in most cases outperforms static locking.

The performance of two-phase locking has been studied analytically by Irani and Lin [14]. They studied the effect of granularity on the performance of two-phase locking, using a queueing network model. However, the numerical results presented in their paper are not accurate. They assume that the mean number of locks held by transactions other than the one making a request is 1, and the waiting time for a lock is independent of the database size and the number of concurrently running transactions.

Munz and Krenz studied the performance of two-phase locking using simulation [24]. They examined two problems concerning deadlock: (i) When deadlock occurs, which transaction should be restarted? (ii) If the state of a transaction is saved
periodically, then a transaction need not restart from the beginning. It can be rolled back to the nearest checkpoint necessary for breaking the deadlock. Does the checkpointing improve the performance? The conclusions of the study are: (i) if the smallest (cheapest by some measure) transaction in deadlock is restarted, this would be considerably better than simply restarting the transaction that caused a deadlock. (ii) transactions should release all locks when they restart, i.e., the need for checkpointing is very small.

The other extensive simulation study of two-phase locking was done by Ries and Stonebraker in [27]. They study the effect of granularity on locking and considered sequential data access, locking hierarchies, and predicate locks. Ries and Stonebraker concluded that, for sequential access, a medium granularity is usually optimum; locking hierarchies should be used if transactions are long and granularity is fine.

The work by Lin and Nolte [19] is an extension of the study by Ries and Stonebraker. However, they chose to factor out resource contention, and just concentrated on the effect of data contention. The former refers to competition for memory, I/O, etc. and the latter to conflicts over data objects. Devor and Carlson [7] tried to incorporate more details about the transactions and database to make their simulations more realistic. They tried to keep the simulation model close to the structure of relations and semantics of the transactions. Their study concluded that static locking is better than dynamic locking under heavy workloads, whereas the reverse is true under light workloads.
The majority of previous performance studies of two-phase locking made an assumption of uniform data access distribution. This was due to the fact that the analysis of two-phase locking is intractable under arbitrary data access distribution. Very few performance studies [24, 35] tried to relax this assumption by using the simple b-c access model. However, the b-c access model is an approximation and captures the non-uniform behavior of data access at a very coarse level. But the results in Chapter 3 of this dissertation show that the assumption of uniform data access distribution gives us upper and lower bounds on estimate of such important system parameters as throughput and response time. That is why it is essential that the assumption of uniform data access will be eliminated from the performance analysis of concurrency control algorithms (two-phase locking in particular).

Next, we present a performance analysis of two-phase locking under the non-uniform data access distribution. We show that the expressions for quantities such as the probability of lock being granted, the mean wait time for a blocked request (blocking delay) and throughput can be derived analytically. In Section 6.3, we also present a technique for the computation of the probability of deadlock.

6.1 Queueing Network Model of the System

The transaction behavior obeying the two-phase locking is represented in Figure 12. Upon arrival in the system, a transaction requests its first lock. In case the request is granted, the transaction proceeds with execution. Otherwise it joins the blocked queue of transactions.
When a transaction acquires its last lock and the last computation is completed, it releases all its locks and leaves the system (i.e., strict two-phase locking). A statistically identical transaction immediately enters the system, keeping the degree of multiprogramming fixed. We assume that for each transaction, the number of computations is equal to its number of lock requests, K.

The model of the system characterizes the hardware and software resources of the database. Hardware resources consist of CPU for processing and disk for storage capability. Execution of a transaction requires service at the CPU for transaction computations. The disk holds the data objects of the database. A transaction requires
disk access for reading data objects and writing values into the database. We aggregate the CPU and disk into a single load-dependent server which serves transactions at rate $S(N_a)$, when there are $N_a$ transactions at the server. A technique to compute $S(N_a)$ has been discussed in Section 5.2.

6.2 Performance Analysis

As in Chapter 5, in our performance study, we assume that a transaction exhibits the average state behavior, which permits us to work with averages rather than with probability distributions.

Throughout the analysis we will use the notation developed in Section 5.1. Initially, we assume that deadlocks do not occur. Absence of deadlocks is a common assumption in the literature [23,36]. According to Gray et al. [13] and Beeri and Obermack [1], deadlock is rare ( < 0.1 percent) in a typical database. In Section 6.3, we derive an expression for the probability of deadlock.

In our model we keep track of the number of blocked transactions and the number of active transactions. A transaction is termed active if it is running (i.e., has all the data objects needed for execution) and it is termed blocked if a needed data object is unavailable. In a two-phase locking acquisition policy, conflict or blocking can arise due to two reasons: First, a newly arrived transaction finds that an object requested by it is already taken by a transactions, and second, a running transaction makes a new request for a data object which is already taken by another transaction.
The probability of no blocking for both types of requests can be computed using the techniques presented in Section 4.2.

Let \( P_1 \) be the probability that a newly arrived transaction will not get blocked on its first request, and let \( P_2 \) be the probability that a running transaction will not get blocked on its next request for a data object. Let \( W_b \) be the average time a blocked transaction needs to wait to obtain the requested lock. The number of transactions which are blocked due to the first reason is denoted by \( N_{b1} \). The number of transactions which are blocked due to the second reason is denoted by \( N_{b2} \). Therefore,

\[
N_b = N_{b1} + N_{b2}
\]  

(113)

The value of \( N_{b1} \) depends on the probability of blocking, the average time a blocked transaction needs to wait to obtain the requested lock, and the throughput of the system. Thus,

\[
N_{b1} = W_b \times (1 - P_1)
\]

(114)

The derivation of the equation for \( N_{b2} \) is more involved, since every transaction during its lifetime can get blocked on several different lock requests. Therefore, it is likely that the majority of transactions that are blocked belong to the second category. The value of \( N_{b2} \) depends on the probability of blocking, the average time a blocked transaction needs to wait to obtain the requested lock, and the throughput of the system. Thus,
Since we have a closed system the alternative equation for \( N_b \) is

\[
N_b = N - N_a
\]  

(116)

Using Little's Law, the value of throughput can be derived using the following equation:

\[
t = \frac{\text{Degree of multiprogramming}}{\text{mean transaction residence time}} = \frac{N}{W_a + K \cdot W_b}
\]

(117)

Therefore, the value of \( W_b \) can be computed using the following equation:

\[
W_b = \frac{N_b}{t \cdot K} = \frac{N-t \cdot W_a}{t \cdot K}
\]

(118)

The value of the average time needed for an active transaction to complete its execution is given by:

\[
W_a = (K+1) \cdot S(N_a)
\]

(119)

\( K \) steps are needed in order to execute \( K \) lock requests and one additional step is needed for transaction completion. A similar technique for computing the values of \( W_a \) has been used in [35]. Here we have a system of five equations, which we are going to solve by iteration. Examining equations (113)-(119) we can conclude that the choice for \( t \) (throughput) determines the value of \( W_b \) (the average time a blocked transaction waits to acquire a lock), which when substituted into the system of equations, generates a new computed value for the system throughput. We define percent difference between "old value of throughput" of the previous iteration and "new value of throughput" of the current iteration to be 0.01. In our analysis, we terminate the iterative process when the percentage difference is less than 1. We
choose the starting throughput for iteration using the following equation:

\[ t = \frac{N}{(K+1) \cdot S(N_a)} \]  \hspace{1cm} (120)

which is the total resource time requested by each transaction without taking blocking delay into consideration [20]. The value of the response time of the transaction is computed using the following equation:

\[ R = \frac{N_a}{t} \]  \hspace{1cm} (121)

In all cases, we found that the process terminates after no more than eight iterations. A sample output of the iterative process is presented in Table 13.

<table>
<thead>
<tr>
<th>n</th>
<th>old t</th>
<th>new t</th>
<th>percent difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01598</td>
<td>0.01512</td>
<td>5.3</td>
</tr>
<tr>
<td>2</td>
<td>0.01512</td>
<td>0.01441</td>
<td>3.1</td>
</tr>
<tr>
<td>3</td>
<td>0.01441</td>
<td>0.01398</td>
<td>2.98</td>
</tr>
<tr>
<td>4</td>
<td>0.01398</td>
<td>0.01359</td>
<td>2.7</td>
</tr>
<tr>
<td>5</td>
<td>0.01359</td>
<td>0.01328</td>
<td>2.28</td>
</tr>
<tr>
<td>6</td>
<td>0.01328</td>
<td>0.01298</td>
<td>2.18</td>
</tr>
<tr>
<td>7</td>
<td>0.01298</td>
<td>0.01271</td>
<td>2.0</td>
</tr>
<tr>
<td>8</td>
<td>0.01271</td>
<td>0.01244</td>
<td>0.3</td>
</tr>
</tbody>
</table>
6.3 The Probability of Deadlock

A deadlock occurs when a set of processes circularly wait for exclusive access to resources held by other processes in the same set. The probability of deadlock depends upon several factors such as resource request and release patterns, resource holding times, the average number of data objects locked by the process, etc. An approximate analysis of the probability of deadlock was done in [13], and the following conclusions were made: 1) Deadlocks are rare, but they increase linearly with the degree of multiprogramming. 2) Deadlocks rise as the fourth power of the transaction size. 3) Blocking delays rise as the second power of transaction size. 4) Most deadlocks are of length two.

Next, we present an analytical method for computing the probability of deadlock. The analyses presented are in the spirits of [13,34]. However, in our analysis we made a more realistic assumption, such as non-uniform data access distribution. The probability that a transaction encounters a deadlock, denoted by \( P_d \), is the sum of probabilities that the transaction encounters deadlocks of different cycle lengths:

\[
P_d = P_d^{(2)} + P_d^{(3)} + \ldots
\]

However, the simulation study performed in [7,37] concluded that only deadlocks of cycle length two are significant. Therefore, we can conclude that the probability of deadlocks is approximately equal to the probability of deadlocks of cycle two. Based on the results in [13], the probability that a transaction \( T_1 \) has a deadlock of cycle-length two with another transaction is
\[ P_d = \text{Probability} \{ T_1 \text{ conflicts with } T_2 \text{ at lock request} \} \times \text{Probability} \{ T_2 \text{ has been blocked by } T_1 \mid T_2 \text{ is currently blocked} \} \times \text{Probability} \{ T_2 \text{ is currently blocked} \} \times \text{(number of possible candidates for } T_2 \}) \]

\[ = \frac{1-P_{NC}}{N-1} \times \frac{1}{(N-1)} \times P_2 \times (N-1) \]

\[ = \frac{(1-P_{NC}) \times P_2}{N-1} \]

The efficient technique for computing \( P_{NC} \) (the probability of no conflict between two transactions) under the non-uniform data access distribution is developed in Section 3.3. An efficient technique for computing \( P_2 \) (probability of transaction blocking) is given in Section 4.2.

In this section we compute the probability of deadlock when transactions request a fixed or variable number of exclusive locks with non-uniform probabilities. The values of the probability of deadlock are recorded in Table 14. It is clear from Table 14 that the probability of deadlock is extremely small.

In addition to knowing the probability of conflict and deadlock, it is important to determine their impact on the overall system performance. It turns out that transaction blocking due to lock conflicts is the main cause of performance degradation, rather than the wasted processing due to deadlock [28,42]. In fact, as the number of active transactions in a (dynamic) locking system is increased, the system throughput may degrade.
Table 14. Probability of Deadlock (μ=500, K=8, M=500)

<table>
<thead>
<tr>
<th>tr</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=400</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.0040</td>
<td>0.0031</td>
<td>0.0011</td>
<td>0.0003</td>
</tr>
<tr>
<td>5</td>
<td>0.0048</td>
<td>0.0038</td>
<td>0.0034</td>
<td>0.0012</td>
</tr>
<tr>
<td>6</td>
<td>0.0055</td>
<td>0.0044</td>
<td>0.0038</td>
<td>0.0014</td>
</tr>
<tr>
<td>7</td>
<td>0.0062</td>
<td>0.0050</td>
<td>0.0043</td>
<td>0.0018</td>
</tr>
<tr>
<td>8</td>
<td>0.0067</td>
<td>0.0061</td>
<td>0.0048</td>
<td>0.0021</td>
</tr>
<tr>
<td>9</td>
<td>0.0073</td>
<td>0.0066</td>
<td>0.0057</td>
<td>0.0025</td>
</tr>
<tr>
<td>10</td>
<td>0.0076</td>
<td>0.0069</td>
<td>0.0061</td>
<td>0.0029</td>
</tr>
</tbody>
</table>

6.4 Numerical Results

In this section, we present numerical results to illustrate the analysis of the two-phase locking algorithm. We assume that the database size is M (M=1000 or M=500) we vary transactions sizes between 4 and 18 (K=4,6,8,...,18) and we vary the number of active transactions in the system between 4 and 13 (tr=4,6,8,...,13). The data access is distributed according to truncated Normal distribution (that is, $p_i = C\cdot e^{-1/2((i-\mu)/\sigma)^2}$, 1 ≤ i ≤ M, where C is a constant such that $\sum p_i =$ transaction size over all i) and the service rate of the system (S(Na)) is computed using the technique developed in Section 5.2. The values of $\mu_1$ and $\mu_2$ are 10 msec and 25 msec, respectively.
We compute the probability that a running transaction will get blocked on its subsequent request with running transactions in the system using the technique of Section 4.2. We assume various values of the number of concurrent transactions (tr) and variance of the Normal distribution (σ). Results of the analytical study are recorded in Table 15.

The numerical study shows that the increase in the number of transactions (tr) results in higher probability of blocking. The other interesting observation is that the probability of blocking is more sensitive to the number of concurrently running transactions when σ is large.

The values of the system throughput for various values of the number of concurrent transactions (tr) and variance of the Normal distribution (σ) are recorded

<table>
<thead>
<tr>
<th>tr</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=400</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.3568</td>
<td>0.2790</td>
<td>0.2516</td>
<td>0.0848</td>
</tr>
<tr>
<td>5</td>
<td>0.4254</td>
<td>0.3366</td>
<td>0.3050</td>
<td>0.1052</td>
</tr>
<tr>
<td>6</td>
<td>0.4872</td>
<td>0.3901</td>
<td>0.3366</td>
<td>0.1254</td>
</tr>
<tr>
<td>7</td>
<td>0.5428</td>
<td>0.4395</td>
<td>0.4298</td>
<td>0.1453</td>
</tr>
<tr>
<td>8</td>
<td>0.5929</td>
<td>0.4854</td>
<td>0.4451</td>
<td>0.1648</td>
</tr>
<tr>
<td>9</td>
<td>0.6375</td>
<td>0.5277</td>
<td>0.4859</td>
<td>0.1841</td>
</tr>
<tr>
<td>10</td>
<td>0.6787</td>
<td>0.5670</td>
<td>0.5241</td>
<td>0.2032</td>
</tr>
</tbody>
</table>
in Table 16 and plotted in Figure 13.

The larger the number of running transactions, the higher the values for the throughput. We also notice that the smaller the $\sigma$ - the smaller the values of the throughput. The difference for values of throughput when $\sigma=200$ as compared to $\sigma=\infty$ (uniform) can be larger than 50 percents. Therefore, the assumption of uniform data access distribution gives us an upper bound for the values of system throughput.

Next, we analyze the effect that the transaction size has on the values of the probability of blocking and throughput. In our study we assume that we have 4 concurrently running transactions and that the database has 1000 data objects ($M=1000$). The values for the probability of blocking are compiled in Table 17.

Obviously, the bigger the transactions are getting in size, the higher the probability of blocking is. Since more data objects are locked, the probability of

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$\sigma=200$</th>
<th>$\sigma=300$</th>
<th>$\sigma=400$</th>
<th>$\sigma=\infty$ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.00871</td>
<td>0.00947</td>
<td>0.00998</td>
<td>0.01772</td>
</tr>
<tr>
<td>7</td>
<td>0.01022</td>
<td>0.01206</td>
<td>0.01215</td>
<td>0.02051</td>
</tr>
<tr>
<td>9</td>
<td>0.01244</td>
<td>0.01328</td>
<td>0.01369</td>
<td>0.02114</td>
</tr>
<tr>
<td>11</td>
<td>0.01486</td>
<td>0.01546</td>
<td>0.01602</td>
<td>0.02308</td>
</tr>
<tr>
<td>13</td>
<td>0.01289</td>
<td>0.01382</td>
<td>0.01408</td>
<td>0.02219</td>
</tr>
</tbody>
</table>
Figure 13. System Throughput - III

Table 17. The Probability of Blocking (τr=4, μ=500, M=1000)

<table>
<thead>
<tr>
<th>K</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=∞ (uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.4254</td>
<td>0.3366</td>
<td>0.3050</td>
</tr>
<tr>
<td>10</td>
<td>0.5817</td>
<td>0.4753</td>
<td>0.4355</td>
</tr>
<tr>
<td>12</td>
<td>0.6912</td>
<td>0.6102</td>
<td>0.5628</td>
</tr>
<tr>
<td>14</td>
<td>0.8015</td>
<td>0.7265</td>
<td>0.6918</td>
</tr>
<tr>
<td>16</td>
<td>0.8738</td>
<td>0.8092</td>
<td>0.8257</td>
</tr>
<tr>
<td>18</td>
<td>0.9272</td>
<td>0.8688</td>
<td>0.8409</td>
</tr>
</tbody>
</table>
acquiring the lock for a specific transaction is getting smaller. As the distribution drifts away from being uniform (σ=∞), the probability of blocking increases rapidly. In certain cases the difference can be as large as 40 percents. The values for throughput for various transaction sizes are recorded in Table 18 and plotted in Figure 14.

The larger the transaction size - the smaller the values of the throughput. We also notice that the values of throughput are largest when the distribution is uniform. For example, when K=14 and σ=200 the value of throughput is smaller by 50 percents as compared to the value of throughput obtained when K=14 and σ=∞ (uniform).

The values of the response time for various numbers of running transactions and variable transaction sizes are presented in Tables 19-20 and depicted in Figures 15-

Table 18. System Throughput (tr=4, μ=500, M=1000)

<table>
<thead>
<tr>
<th>K</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=∞(uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.00784</td>
<td>0.00838</td>
<td>0.00964</td>
</tr>
<tr>
<td>10</td>
<td>0.00567</td>
<td>0.00591</td>
<td>0.00710</td>
</tr>
<tr>
<td>12</td>
<td>0.00452</td>
<td>0.00461</td>
<td>0.00667</td>
</tr>
<tr>
<td>14</td>
<td>0.00377</td>
<td>0.00381</td>
<td>0.00583</td>
</tr>
<tr>
<td>16</td>
<td>0.00241</td>
<td>0.00261</td>
<td>0.00419</td>
</tr>
</tbody>
</table>
The more running transactions we have in the system and the larger the transaction sizes are - the higher the values for response time we get. Analyzing the data in Tables 19-20, we can also conclude that the uniform data access distribution gives us a lower bound on the values of the response time. The error introduced due

Table 19. The Response Time (tr=4, μ=500, M=1000)

<table>
<thead>
<tr>
<th>K</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=∞(uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>510.20</td>
<td>477.32</td>
<td>414.93</td>
</tr>
<tr>
<td>10</td>
<td>705.46</td>
<td>676.81</td>
<td>563.38</td>
</tr>
<tr>
<td>12</td>
<td>884.95</td>
<td>857.67</td>
<td>599.70</td>
</tr>
<tr>
<td>14</td>
<td>1061.00</td>
<td>1049.86</td>
<td>686.10</td>
</tr>
<tr>
<td>16</td>
<td>1659.75</td>
<td>1532.56</td>
<td>954.65</td>
</tr>
</tbody>
</table>

Table 20. The Response Time (K=8, μ=500, M=500)

<table>
<thead>
<tr>
<th>tr</th>
<th>σ=200</th>
<th>σ=300</th>
<th>σ=400</th>
<th>σ=∞(uniform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>574.05</td>
<td>527.98</td>
<td>501.00</td>
<td>282.16</td>
</tr>
<tr>
<td>7</td>
<td>684.93</td>
<td>580.43</td>
<td>576.13</td>
<td>341.29</td>
</tr>
<tr>
<td>9</td>
<td>723.47</td>
<td>677.71</td>
<td>657.41</td>
<td>425.73</td>
</tr>
<tr>
<td>11</td>
<td>740.24</td>
<td>711.51</td>
<td>686.64</td>
<td>476.60</td>
</tr>
<tr>
<td>13</td>
<td>1008.53</td>
<td>940.66</td>
<td>923.29</td>
<td>585.84</td>
</tr>
</tbody>
</table>
to the assumption of uniform data access distribution can be as large as 60 percents.

6.5 Summary

In this chapter, we presented the analysis of two-phase locking under non-uniform data access distribution. We present a queueing network model for a two-phase locking mechanism and derive analytical equations for quantities such as the probability of lock being granted, the probability of deadlock, mean wait time for a blocked request (blocking delay) and throughput of the system. We provide a discussion of the numerical results and study the effect of non-uniform data access distribution on the performance of two-phase locking and the probability of deadlock. The major contribution of this performance study is the use of analytic techniques and incorporation of non-uniformity of data access into the performance model.
Figure 15. Response Time - III

Figure 16. Response Time - IV
CHAPTER VII
CONCLUSIONS

Most past performance analyses of database systems have assumed that data access distribution of transaction requests is uniformly distributed across the entire database [10,13,21,22,23,38]. Although this assumption simplifies the performance analysis considerably, it is unrealistic and gives optimistic bounds on the performance of concurrency control algorithms. This is because the assumption of uniform access distribution gives the lower bound on the probability of conflict among transactions because their data access is spread over the database as much as possible [31]. Or in other words, non-uniform data access reduces the effective size of the database, causing more conflicts [32]. This assumption has largely been adopted because computation of the probability of conflict under arbitrary data access distribution is very expensive (due to combinatorial explosion) even for a database of small size.

In this dissertation, we have developed a series of efficient analytical techniques, which enables us to capture the non-uniformity of data access at a very fine level, i.e., at the level of data object, at a moderate cost of computation. We have also presented a more realistic model of transaction blocking and examined the effect of non-uniform data access distribution on the performance of static and two-phase locking concurrency control algorithms. We have shown that the assumption of
uniform data access distribution gives us upper and lower bounds on the values of throughput and response time, respectively. Therefore, an important consequence of this research is that the assumption of uniform data access distribution has led past performance studies to compute upper or lower bounds on the performance of concurrency control algorithms. Since in practice transactions exhibit locality of data reference [33,40], actual system throughput is likely to be lower and actual transaction response time is likely to be higher than what is estimated by previous performance studies.

The computational techniques developed are a substantial improvement over the existing techniques, since they capture the non-uniformity of data access much more accurately. We have analyzed the performance of static and two-phase locking algorithms under non-uniform data access distribution. We have analytically derived the values of the following system parameters: throughput, blocking delay, probability of blocking, and probability of deadlock.

The computational techniques developed can be used as an analytical tool, which would help to choose a more suitable concurrency control algorithm for a specific application environment. The two basic philosophies behind concurrency control algorithms are: (i) optimistic approach (the conflicts are rare), and (ii) pessimistic approach (the conflicts are not rare). Therefore, the ability to compute the probability of conflict/blocking under the realistic assumption of non-uniform data access is essential for the selection of a proper underlying concurrency control scheme. It was emphasized in this dissertation that in database performance
evaluation studies, unrealistic modeling of data access distribution may lead to serious errors in the estimated performance.

The correctness of the developed computational techniques has been verified analytically. However, it is a future research project to employ simulation to study the effect that the assumption of independence of transaction accessing has on the value of the probability of conflict.

The results of this dissertation can be used to guide system designers and analysts in their implementation of database management systems. Finally, the development of analytical performance predictor of concurrency control algorithms is an important research project. The major difficulty in the analysis of concurrency control algorithms is that they are not tractable by currently available tools. Research efforts are needed in the direction of modeling and performance evaluation of concurrency control algorithms, especially the development of new tools, and the development of some approximation techniques to solve the models of these algorithms.
APPENDIX

EFFICIENT COMPUTATION OF Q(kn, M) AND A(kn, n, M)

Suppose two independent transactions T1 and T2 have their request vectors RV1 and RV2 defined as RV1 = (a1, a2, ..., aM) and RV2 = (b1, b2, ..., bM), respectively, with \( \sum_{i=1}^{M} a_i = n \) and \( \sum_{i=1}^{M} b_i = kn \). P(RV1) and P(RV2) are given by

\[
P(RV_1) = \prod_{a \in RV_1} p_i^a (1-p_i)^{1-a}
\]

and

\[
P(RV_2) = \prod_{b \in RV_1} b_i (1-p_i)^{1-b}
\]

 respectively. Note that T1 and T2, respectively, indicate a normal transaction and a lumped transaction which is obtained by merging k non-conflicting transactions.

Computation of Q(kn, M)

\[
Q(kn, M) = \sum_{\forall RV_2} P(RV_2) = \sum_{\forall RV_2} \prod_{b \in RV_2} b_i (1-p_i)^{1-b}
\]  

(124)

Computation of Q(kn, M) directly from the above equation requires \( M \times M \) operations. We present a technique which computes Q(kn, M) in \( 4knM \) operations.

Consider the following recursive function:

\[
X(i, j) = (1-p_j)X(i, j-1) + p_jX(i-1, j-1)
\]  

(125)

Function X(i, j) is similar to function F(r, m) of [43]; however, we have
different boundary conditions.

Boundary Conditions:

\[
X(i, j) = \begin{cases} 
1 & \text{if } i=j=0 \\
0 & \text{if } i > j \\
0 & \text{if } i \text{ or } j < 0
\end{cases} 
\quad (126)
\]

For a transaction of size \( i \), \( X(i, j) \) gives the probability that all the data objects accessed by it are concentrated over the first \( j \) data objects of the database, i.e., if \( \Psi \) denotes the collection of all \( RV=(a_1, a_2, \ldots, a_M) \) such that \( a_1+\ldots+a_j=i \) and \( a_{j+1}+\ldots+a_M=0 \), then:

\[
\sum_{RV \in \Psi} P(RV) = X(i, j) 
\quad (127)
\]

Note that \( Q(kn, M) \) is \( X(kn, M) \) which can be computed by using the recursive relation of equation (13) in \( 4knM \) operations rather than in \( \left( \begin{array}{c} M \\ kn \end{array} \right) \) * \( M \) operations. At any time during the computation, we have to keep only those values of \( X(i, j) \) which have the same value for \( j \). Therefore, the space complexity is \( O(n) \).

Computation of \( A(kn, n, M) \)

If \( D = (1, 1, \ldots, 1) \) is a \( M \)-dimensional unit-vector, then the \( A(kn, n, M) \) can be written as:

\[
A(kn, n, M) = \sum_{\forall RV_1} \sum_{\forall RV_2 \in (D-RV_1)} P(RV_1)P(RV_2) = 
\quad (128)
\]

\[
\sum_{\forall RV_1} \sum_{\forall RV_2 \in (D-RV_1)} \prod_{b \in RV_1} p_i^b(1-p_i)^{1-b} \prod_{a \in RV_1} p_i^a(1-p_i)^{1-a}
\]
Note that the complexity of computing $A(kn, n, M)$ directly from the above equation is $\binom{M}{n} \times \binom{M-n}{kn} \times M$. Consider a function $Z(i, j, l)$ defined recursively as follows ($i$, $j$, and $l$ are integers):

$$Z(i, j, l) = (1-p_i)^2 Z(i, j, l-1) + p_i (1-p_i) Z(i-1, j, l-1)$$
$$+ (1-p_i) p_i Z(i, j-1, l-1)$$

**Boundary Conditions:**

$$Z(i, j, l) = \begin{cases} 
0 & \text{if } i+j>l \\
1 & \text{if } i=j=l=0 \\
0 & \text{if } i \text{ or } j \text{ or } l < 0 
\end{cases}$$

For two transactions $T_1$ and $T_2$ of size greater than or equal to $i$ and $j$, respectively, $Z(i, j, l)$ gives the probability that $T_1$ and $T_2$ do not conflict given that $i$ accesses of $T_1$ and $j$ accesses of $T_2$ are concentrated over first $l$ data objects of the database. Note that $A(kn, n, M)$ is nothing but $Z(kn, n, M)$ which can be computed by using recursive relation (130) in about $11Mkn^2$ operations rather than in $\binom{M}{n} \times \binom{M-n}{kn} \times M$ operations.

If we precompute the terms $(1-p_i)^2$ and $(1-p_i) p_i$, $1 \leq l \leq M$, and store them into arrays, then we can use these ready-made values during the computation of $Z(kn, n, M)$ bringing down the number of operations to $5Mkn^2$, less than half of the previous number. Since at any time we need only those elements of $Z(i, j, l)$ which have the same value of $l$, the space complexity is $O(kn^2)$ rather than $O(kMn^2)$. 
Q(kn, M) and A(kn, n, M) can be computed using the following PASCAL like program:

PR: array[1..M] of real; (* stores p_i's *)
X: array[1..n(t-1), 1..M] of real;
Z: array[1..n(t-1), 1..n, 1..M] of real;
Q: array[1..t-1] of real; (* Q[k] holds Q(kn, M) *)
A: array[1..t-1] of real; (* A[k] holds A(kn, n, M) *)

begin

initialize boundary conditions;
read array PR;
for l=1 to M do
for j=1 to n do
for i=1 to (t-1).n do
begin
X[i, l] = (1-PR[l]) X[i, l-1] + PR[l] X[i-1, l-1];
Z[i,j,l] = (1-PR[l])^2 Z[i,j-1, l-1]
PR[l] (1-PR[l]) Z[i-1,j, l-1] + (1-PR[l]) PR[l] Z[i,j-1, l-1]
if l=M and j=n then (* store arrays Q and A *)
if ((i/n)*n-i=0) then
begin
Q[i/n]:= X[i,l]; A[i/n]:= Z[i, j, l];
We can run this program to compute the functions $Q(kn, M)$ and $A(kn, n, M)$ before computing the numerator of equation (25) in approximately $c(t-1)n^2M$ steps ($c = 10$ is a constant which denotes number of operations performed inside the ‘$i$’ loop). and use them subsequently while computing the numerator of equation (25).
BIBLIOGRAPHY


