

UPDATING REPLICATED FILES IN REAL TIME
DISTRIBUTED PROCESSING SYSTEMS

Joseph Hellerstein

1984
Report No. CSD-840030

MASTER COPY

UNIVERSITY OF CALIFORNIA

Los Angeles

Updating Replicated Files in
Real Time Distributed Processing Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Joseph Hellerstein

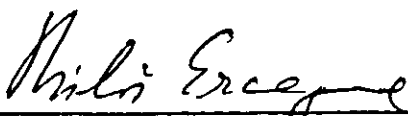
1984

840030
MASTER COPY


© Copyright by
Joseph Hellerstein

1984

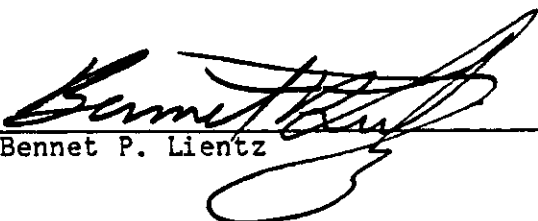
The dissertation of Joseph Hellerstein is approved.



Milos D. Ercegovac




Sheila A. Greibach



Bennet P. Lientz



R. Clay Sprowls



Wesley W. Chu, Committee Chair

University of California, Los Angeles

1984

To

SUZANNE

Table of Contents

	page
LIST OF NOTATION	vii
ABSTRACT	xiv
1 INTRODUCTION	1
1.1 CHARACTERISTICS OF EXISTING PROTOCOLS	8
1.2 CONTRIBUTIONS OF THIS DISSERTATION	17
2 THE EXCLUSIVE-WRITER PROTOCOL (EWP)	19
2.1 EWP OPERATION	19
2.2 EWP APPLICATIONS AREAS	26
2.3 EWP EXTENSIONS	27
2.4 IMPLEMENTATION CONSIDERATIONS	31
2.4.1 UPDATE SEQUENCE NUMBER	31
2.4.2 DEGREE OF FILE REPLICATION	31
2.4.3 DESIGNATING EXCLUSIVE-WRITERS	32
2.4.4 ASSIGNING TRANSACTIONS TO SITES	32
2.4.5 OPERATING SYSTEM DESIGN	33
2.4.6 REDUCING MESSAGE VOLUME	34
2.5 SUMMARY	35
3 THE EXCLUSIVE-WRITER PROTOCOL WITH LOCKING OPTION (EWL)	36
3.1 EWL OPERATION	36
3.2 EWL EXTENSIONS	44
3.3 DYNAMIC SWITCHING BETWEEN PSL AND EWL	46
3.4 IMPLEMENTATION CONSIDERATIONS	48
3.6 SUMMARY	50
4 RESPONSE TIME ANALYSIS	52
4.1 SYSTEM MODELED	54
4.2 APPROACH TO ANALYTIC MODELS	58
4.3 PSL MODEL	60
4.4 EWL MODEL	78
4.5 OTS MODEL	98
4.6 NUMERICAL STUDIES	107
4.7 DISCUSSION	143
5 CONCLUSIONS AND FUTURE RESEARCH	149
REFERENCES	153

List of Figures

	page
Figure 1-1: Example of a Real Time Distributed Processing System	3
Figure 1-2: Need for Consistency Control in Distributed Systems	5
Figure 1-3: Timing Diagram For Primary Site Locking (PSL)	9
Figure 1-4: Timing Diagram For Optimistic Timestamp (OTS)	11
Figure 2-1: The Exclusive-Writer Approach	20
Figure 2-2: EWP Timing Diagram	23
Figure 2-3: EWP Operation at Site i for file F_k	24
Figure 3-1: EWL Timing Diagram	38
Figure 3-2: EWL Operation at site i for file F_k	41
Figure 3-2 continued	42
Figure 3-3: Concurrent Use of EWL and PSL for the same shared file	49
Figure 4.1-1: System Modeled	55
Figure 4.3-1: Timing Diagram For PSL Response Times	61
Figure 4.3-2: Queueing System for Site i Under PSL	64
Figure 4.4-1: Timing Diagram For EWL Response Times	77
Figure 4.4-2: SN Conflict With The First Execution of a Non-EW Site Transaction	89
Figure 4.4-3: SN Conflict with the Second Execution of A Non-EW Site Transaction	92
Figure 4.4-4: SN Conflict with the First Execution of an EW Site Transaction	94
Figure 4.4-5: SN Conflict with the Second Execution of an EW Site Transaction	96
Figure 4.5-1: Timing Diagram For OTS Response Times	99
Figure 4.5-2: Estimating OTS Restart Probabilities	105
Figure 4.6-1: Values For Parameters In Baseline	110
Figure 4.6-2: T_E For Baseline	111
Figure 4.6-3: Components of PSL Response Time For Baseline	113
Figure 4.6-4: Site and Lock Queue Utilizations For Baseline	114
Figure 4.6-5: Number of Transaction Restarts For Baseline	115
Figure 4.6-6: Probability of Transaction Restarts For Baseline	116
Figure 4.6-7: T_U for Baseline	118

Figure 4.6-8: Components of OTS T_U for Baseline	119
Figure 4.6-9: Components of EWL T_U for Baseline	122
Figure 4.6-10: T_E for Background Load	123
Figure 4.6-11: Site Utilizations for Background Load	124
Figure 4.6-12: Lock Queue Utilizations for Background Load	125
Figure 4.6-13: Probability of a Transaction Restart for Background Load (.6,.1)	126
Figure 4.6-14: T_U for Background Load	128
Figure 4.6-15: Response Times for Transaction Execution Times	130
Figure 4.6-16: Lock Queue Utilizations for Transaction Execution Time	131
Figure 4.6-17: Response Times for Update Processing Times	132
Figure 4.6-18: Response Times for Control Processing Times	133
Figure 4.6-19: Response Times for Network Communication Times	135
Figure 4.6-20: Response Times for Lock-grant and Lock-release Times	136
Figure 4.6-21: Response Times for Update Log Processing	138
Figure 4.6-22: Response Times for Number of Sites	139
Figure 4.6-23: Response Times for Partitioning of Transaction Load	141
Figure 4.6-24: Effect on Response Times of Increasing Input Parameter Values	144

LIST OF NOTATION

Roman Capitals

AP = Period during which lock queue arrivals can occur

C = Network communication for an update message

C' = Network communication for a control message

CCP = Consistency control protocol

DR = Database rollback

EA = Time from completing transaction's execution until all update acknowledgements have been received

ER = Time from transaction completing its execution until update-request processing has been completed

EW = Exclusive-writer

\overline{EW} = Non exclusive-writer

F = File

GR = Time from lock-grant to lock-release

H = High priority consistency control work

H' = High priority background work

I = Number of sites

K = Number of files

L = Low priority consistency control work

L' = Low priority background work

LE = Lock-release

LG = Lock-grant

LQ = lock-queue

LR = Lock-request at a primary/exclusive-writer site

LR' = Lock-request message formatting and transmission

N_{rs} = Number of restarts

PL = Pre-lock processing

PS = Primary site

RP = Remaining period after the period during which arrivals are possible at the

lock queue

RS = Restart (for a transaction)

\overline{RS} = No restart (for a transaction)

RTDPS = Real time distributed processing system

T = Time period that includes both service and queue waits

T_E = Transaction execution response time

T_U = Update Confirmation response time

TM = Transaction module

UA = Update acknowledgement processing

UI = Update input processing

UM = Update log maintenance

UO = Update output processing

UR = Update request processing

W = Queue wait

W_0 = Queue wait due to remaining service time

X = Service time

Small Roman

ew_k = Exclusive-writer site for F_k

i = Site index

j = Site index

k = File index

l = File index

$p(LQ;k,i)$ = Probability that F_k is locked by a transaction that executes at site i

$p(TM;k,i)$ = Probability of executing at site i a transaction for F_k

ps_k = Primary site site for F_k

$q(k,i)$ = Probability of restarting a transaction for F_k that executes at site i

Greek

$\delta_{k,i}$ = 1 if site i is the exclusive-writer's/primary site for F_k ; 0 otherwise

λ = Externally generated arrival

$\lambda(TM;k)$ = External arrival rate of F_k transactions

$\lambda(L';i)$ = Arrival rate of background low priority work at site i

$\lambda(H^p; i) =$ Arrival rate of background high priority work at site i

$\gamma =$ Internally generated arrival

$\gamma(LQ; k) =$ Arrival of requests at F_k 's lock queue

$\Lambda(TM; k) =$ Arrival rate of F_k transactions that were externally requested or were restarted

$\rho(i) =$ Utilization of site i

$\rho(LQ; k) =$ Utilization of F_k 's lock queue

Miscellaneous

$\hat{t} =$ Average residual life of the random variable t

$t'(s) =$ LaPlace Transform for the random variable t

ACKNOWLEDGEMENTS

This work was supported in part by U.S. Army Contract No. DASG60-79-C-0087 and subsequent renewals of the same contract.

I would like to express special thanks to my committee chairman, Professor Wesley W. Chu, for his guidance, friendship, encouragement, and insightful comments. I also thank Professor Hector Garcia-Molina for thought provoking ideas on the exclusive-writer protocol and the members of my dissertation committee: Professors Milos Ercegovac, Sheila Greibach, Bennet Lientz, and Clay Sprows. Thanks also are due to Jung Min An, Joseph Bannister, Leslie Holloway, Min-Tsung Lan, and Kin Kwong Leung for their stimulating discussions as well as Wendy Hagar and Aeri Lee for their secretarial assistance with preparing the figures.

Additionally, I would like to thank my parents, Edith and Stan, and my in-laws, Pat and Hersh, for their love and emotional support. Finally, I express special thanks to Suzanne whose love, sensitivity, and devotion kept my motivation from waning and, having completed the Ph.D., she makes it all worthwhile.

VITA

August 24, 1952--Born, Indianapolis, Indiana

1974--B.A., University of Michigan

1974-1978--Software Engineer, Honeywell Inc.

1978-1979--Systems Programmer, Office of Academic Computing, University of California, Los Angeles

1978-1979--M.S., University of California, Los Angeles

1979-1980--Programmer, ARPA Packet Radio Contract, University of California, Los Angeles

1981-1984--Research Assistant, Computer Science Department, University of California, Los Angeles

PUBLICATIONS

Hellerstein, J. and W. W. Chu. "Some Potential Deadlocks in Layered Communications Architectures", *Proceedings of the National Computer Conference*, May 1981, pp. 137-140

Chu, W. W., J. Hellerstein, and M. T. Lan. "The Exclusive-Writer Protocol: A Low Cost Approach For Updating Replicated Files In Distributed Real Time Systems", *Third International Conference On Distributed Computing Systems*, October 1982, pp. 269-277

Chu, W. W., M. T. Lan, and J. Hellerstein. "Estimation of Intermodule Communication (IMC) and Its Application in Distributed Processing Systems", accepted for publication in *IEEE Transactions on Computers*

ABSTRACT OF THE DISSERTATION

Updating Replicated Files in
Real Time Distributed Processing Systems

by

Joseph Hellerstein

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1984

Wesley W. Chu, Chair

Consistency control protocols either check for conflicting file accesses before (early checking) or after (late checking) transactions reference shared files. Early checking protocols (e.g., primary site locking - PSL) delay completing a transaction's execution, or execution response time (T_E), until messages are exchanged for conflict checking. Doing so causes *inter-computer synchronization delays (ICSD)*. Late checking protocols avoid such delays for T_E . But existing protocols (e.g., optimistic timestamps - OTS) repeatedly restart a transaction until it executes without conflict, which can (1) cause long ICSD to finalize a transaction's update (T_U) and (2) saturate the computers and interconnection network.

We present two new protocols that have no ICSD for T_E and avoid repeated transaction restarts. The *exclusive-writer protocol (EWP)* is a late checking protocol with no transaction restarts, database rollbacks, or deadlock due to shared data access. EWP discards update-requests that lose a conflict, which is acceptable for some real time applications, but restricts EWP's usage. The *exclusive-writer protocol with locking option (EWL)* uses EWP when there is no conflict and restarts a transaction under PSL if it loses a conflict. EWL has no database rollbacks and restarts a transaction at most once (if the files read and written by the transaction do not change when it is restarted). To further reduce restarts, sites can dynamically switch between EWL and PSL without additional messages or ICSD.

To study the response times (i.e., T_E and T_U) of PSL, OTS, and EWL, analytical queueing models are developed. Our studies show that OTS is undesirable unless there are small costs for update log maintenance, database rollbacks, distributed update validation, and repeated transaction restarts. EWL is preferred to PSL for T_E , since PSL has ICSD for T_E , but EWL does not. EWL's T_U is lower than PSL's when:

1. Updates are smaller, since EWL includes the proposed update in the update-request message.
2. The cost and/or frequency of transaction restarts is lower, since EWL has restarts but PSL does not; and
3. The cost of locking is higher, since PSL always requires locking but EWL requires it only if a transaction is restarted.

EWP always has lower response times than EWL.

CHAPTER 1

INTRODUCTION

In real time systems, a program (or transaction) must finish within a specified time period if the system is to operate properly (e.g., detect and destroy incoming ballistic missiles). To meet real time constraints, it is desirable to split the processing workload among several computers. However this technique, referred to as *distributed processing*, creates problems with file consistency. For example, if two or more computers simultaneously update the same file, the logical relationships among file data, or *internal consistency*, must be preserved. Additionally when copies of a file are replicated at multiple computers (to reduce read access times and to improve reliability), these copies should be identical once all updates have been finalized. This is referred to as *mutual consistency*.

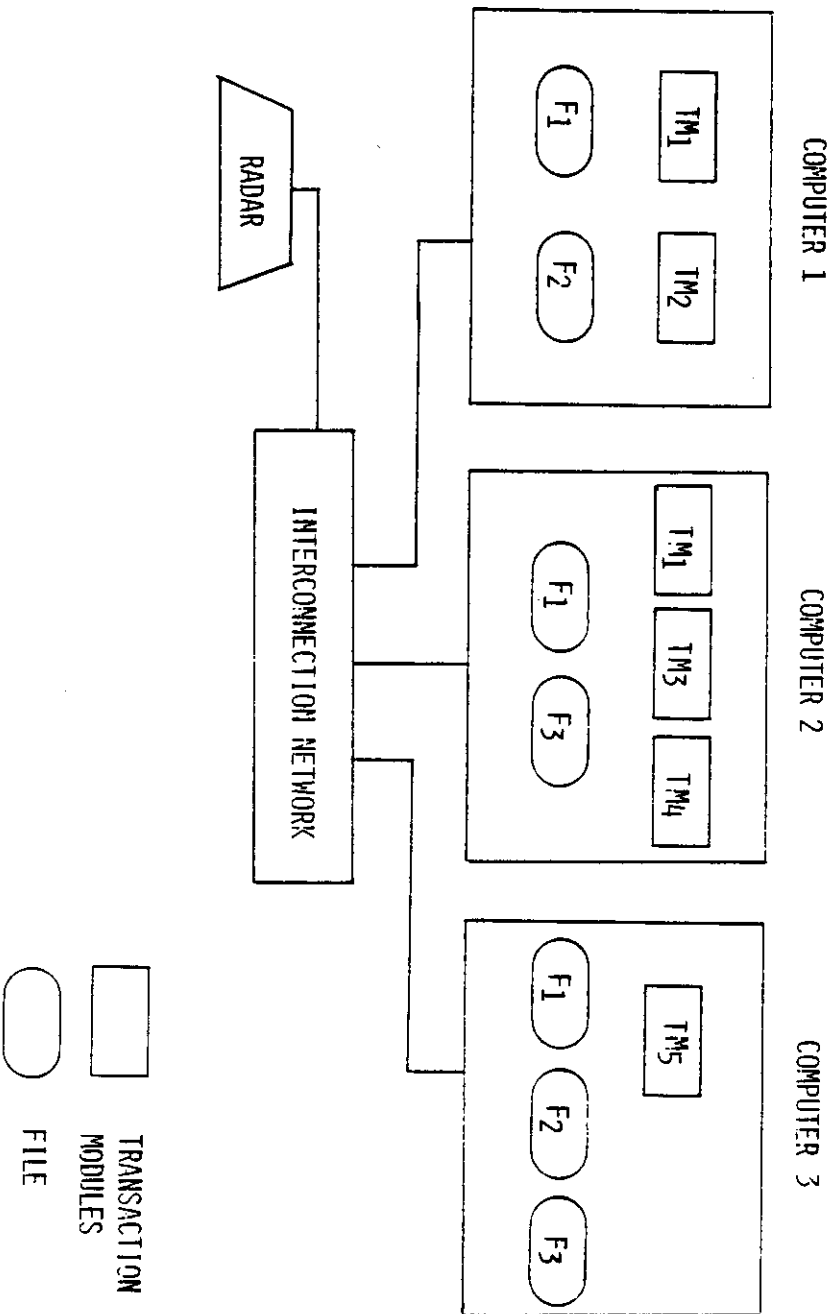
Consistency control protocols (CCPs) are needed to preserve the internal and mutual consistency of shared files. Commonly used techniques for consistency control include locking and timestamp protocols [BERN81]. While the processing time, message volume, and queueing delays of these techniques can be substantial, such costs may not be significant for general purpose database management systems in which shared files reside on secondary storage. However, for *real time distributed processing systems (RTDPS)* in which files consist of data in RAM, the cost of locking or

timestamps may be prohibitive. This has motivated us to introduce two new low cost protocols: the *exclusive-writer protocol (EWP)* and the *exclusive-writer protocol with locking option (EWL)*. In this dissertation, we specify the operating characteristics of EWP and EWL as well as study their performance in terms of response times.

Herein, we consider the type of RTDPS illustrated in figure 1-1 (e.g., the Distributed Processing Architecture Design (DPAD) System [GREE80]). *Transaction modules (TM)*, or just transactions, are assigned to computers (or sites) where they perform time critical functions, such as radar signal processing. More specifically, we consider a transaction to be the execution of a transaction program required to satisfy a request for that program.¹ The files a transaction reads are called its *readset* and those written are its *writeset*. A transaction's *baseset* is the union of its readset and writeset. When updates are made to a file, they are propagated to replicated copies by sending update messages through the interconnection network. (We assume there is no shared memory.) Files reside in RAM and are typically accessed by the transactions without using a high level data model (e.g., relational, network, or hierarchical models). We assume that no computer fails and the interconnection network guarantees that messages are eventually transmitted without error (although they may arrive out of sequence).

¹ For example, TM_1 may be requested to process a record in file F_1 . Because of conflicting accesses to this record, TM_1 may be reexecuted (restarted) one or more times. However, all of these TM_1 executions are part of the same transaction. On the other hand, a second TM_1 request to process a F_1 record is a different transaction.

FIGURE 1-1: EXAMPLE OF REAL TIME DISTRIBUTED PROCESSING SYSTEM



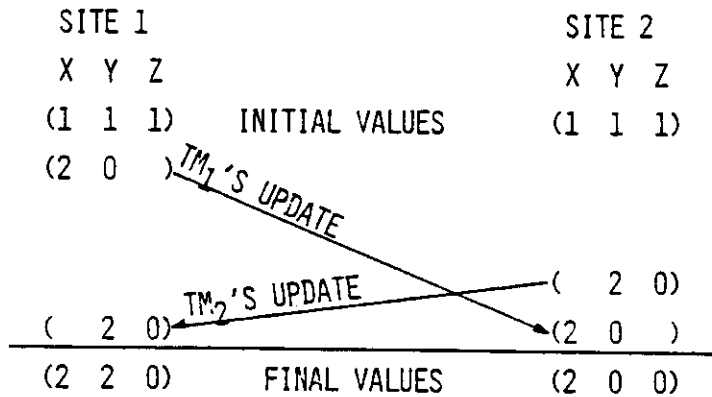
CCPs are required when concurrently executing transactions make conflicting accesses to shared files. Transactions execute concurrently if at least part of their operations overlap. Their file accesses conflict if some transaction writes (updates) a file that another transaction reads or writes. The goal of a CCP is to preserve the internal and mutual consistency of shared files. For example, in figure 1-2 files X , Y , and Z are replicated at sites 1 and 2 with the internal consistency constraint that $X_i + Y_i + Z_i = 3$. (X_i is the value of file X at site i .) For mutual consistency, we require that once all updates cease, $X_1 = X_2$, $Y_1 = Y_2$, and $Z_1 = Z_2$. Initially, $X_i = Y_i = Z_i = 1$, so both internal and mutual consistency are present. If transaction TM_1 at site 1 sets X_1 to 2 and Y_1 to 0, then internal consistency is still preserved at that site (i.e., $X_1 + Y_1 + Z_1 = 2 + 0 + 1 = 3$). Similarly, if TM_2 at site 2 sets Y_2 to 2 and Z_2 to 0, site 2's internal consistency is not violated. However, when TM_1 and TM_2 execute concurrently without using a CCP, their updates can be merged in a manner that violates both internal and mutual consistency.

For many applications CCPs must preserve *serializability*, which is defined as guaranteeing that the result of concurrent transaction executions is the same as if the transactions had executed in some serial order without concurrency [ESWA76]¹. Intuitively, a CCP that ensures serializability guarantees that the effect of doing computations with multiple computers is the same as if only a single monoprogrammed computer had been used. Serializability implies internal consistency, since if the database is initially internally consistent and each transaction preserves internal

¹ Indeed, some authors define a transaction as a program whose database operations have the same the effect as a serial schedule.

FIGURE 1-2: NEED FOR CONSISTENCY CONTROL IN DISTRIBUTED SYSTEMS

- COPIES OF FILES X, Y, AND Z RESIDE AT SITES 1 AND 2.
- CONSTRAINTS
 - INTERNAL CONSISTENCY: $X_1 + Y_1 + Z_1 = 3$
 - MUTUAL CONSISTENCY: $X_1 = X_2, Y_1 = Y_2, Z_1 = Z_2$
- EFFECT OF CONCURRENT UPDATES WITHOUT CONSISTENCY CONTROL



- INCONSISTENCIES
 - INTERNAL: $X_1 + Y_1 + Z_1 \neq 3, X_2 + Y_2 + Z_2 \neq 3$
 - MUTUAL: $Y_1 \neq Y_2$

consistency, then a serial execution of transactions guarantees that the database will remain internally consistent.

Real time systems require CCPs with low delays. In distributed computer systems, the largest delays are usually a result of inter-computer synchronization. For example, the primary site locking (PSL) protocol requires that a transaction's execution be delayed until: (1) a lock-request message has been sent to the primary site, (2) the primary site determines that the transaction can access its files (i.e., there are no conflicting accesses); and (3) the primary site has replied with a lock-grant message. In general, inter-computer synchronization delays have one or more of the following components:

1. Communicating a request (e.g., lock-request) to another site. This requires time for
 - a. Formatting the inter-computer message
 - b. Transmitting the message
 - c. Waiting for processing at the destination
 - d. Processing the request at the destination
2. Waiting for an acceptance condition (e.g., no conflicting file accesses) to be met. This typically requires that one or more additional messages be sent.
3. Communicating the acceptance of a request (e.g., lock-grant) to the requesting

site. This has the same delays as item 1.

Inter-computer synchronization delays can be quite large. For example when the DPAD System is moderately loaded, the time to communicate a request and receive an acceptance is greater than the average execution time of a DPAD transaction!

Here we consider two measures of transaction response time. In RTDPS, time critical functions are often performed by a sequence of transactions. Thus, to meet real time constraints, we need to minimize the time from a transaction's input files being available until the transaction's file updates are available to its successor(s). In some applications, these file updates need only be tentative (e.g., airline reservations in which future scheduling changes will cause flight reassignments or DPAD radar tracking in which object positions are continuously changing). So, we use

T_E: Execution Response Time

Definition: time from the transaction's arrival (i.e., its input files are available) until it has completed its first execution (thereby making available tentative file updates).

For other applications (e.g., banking or DPAD final intercept planning), tentative updates are not sufficient. A transaction must be assured that updates to its input files are finalized (i.e., will not be undone due to database rollbacks) before the files can be used. Here we use,

T_U: Update Confirmation Response Time

Definition: time from the transaction's arrival until its update is known to be

finalized at some site.¹

1.1 CHARACTERISTICS OF EXISTING PROTOCOLS

Consistency control protocols can be classified based on:

1. when inter-computer checking for conflicting file accesses is performed
2. the technique used for serializing conflicting file accesses.

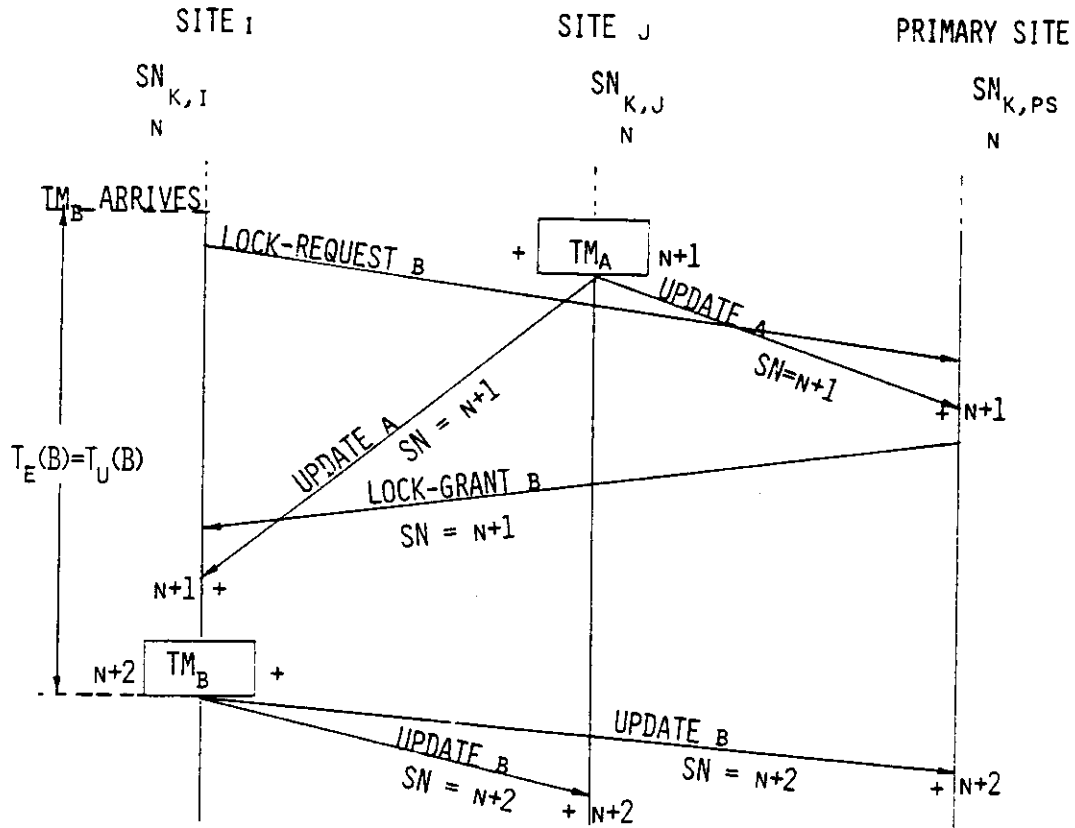
Conflict checking can be performed either before or after transactions access shared files. The former is called *early checking* while the latter is referred to as *late checking*. Late checking protocols are also called *optimistic* since they are optimistic that conflicts will not occur. Conversely, early checking protocols are also called *pessimistic*. Commonly used techniques for serialization include: (1) locking, in which the presence of a lock indicates a reservation for the type of access indicated and (2) timestamps, in which globally unique timestamps are assigned to transactions and file conflicts are resolved based on the transaction's timestamp (e.g., larger (younger) timestamps always lose to smaller (older) ones).

Several authors have surveyed CCPs for distributed processing systems (e.g., [BERN81], [BERN82], [GALL82], and [LIN83]). Rather than doing still another survey of CCPs, we focus on two contrasting approaches that have had more widespread acceptance: primary site locking (an early checking protocol that uses locking) and optimistic timestamps (a late checking protocol that uses timestamps).

¹ Note that update confirmation response time has assignment implications, since it is preferable to assign a successor transaction to the site that will first know that its predecessor's update has been finalized.

FIGURE 1-3: TIMING DIAGRAM FOR PRIMARY SITE LOCKING (PSL)

• TM_A AND TM_B ONLY ACCESS FILE F_K



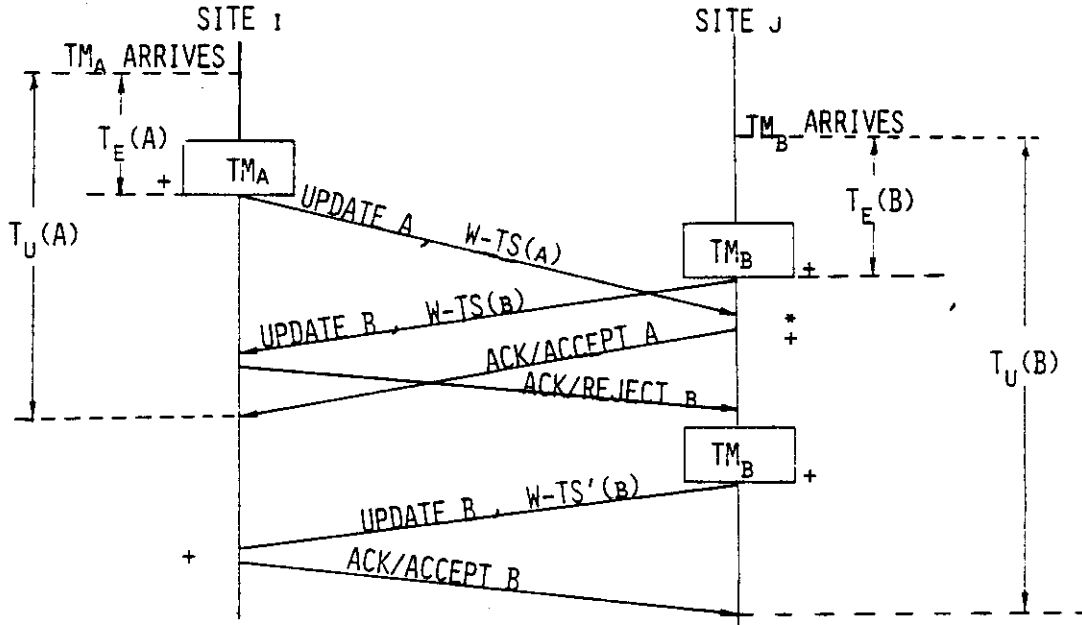
- $SN_{K,I}$ = UPDATE SEQUENCE NUMBER FOR THE COPY OF FILE K AT SITE I
 - = TRANSACTION (TM) EXECUTION
 - T_E = TRANSACTION EXECUTION RESPONSE TIME
 - T_U = UPDATE CONFIRMATION RESPONSE TIME
 - +
- = UPDATE IS WRITTEN

Primary site locking (PSL) has been proposed by [STON79] for distributed INGRES and a variant of PSL is being used by LOCUS [WALK83]. PSL preserves a file's consistency by requiring that transactions get permission from the file's primary site before accessing the file. A timing diagram for PSL is shown in figure 1-3, in which transaction TM_b updates file F_k . Before executing TM_b , site i sends to PS_k (the primary site of F_k) a lock-request message. PS_k puts the lock-request in F_k 's lock queue. Once F_k is available (i.e., no other transaction has a conflicting lock for F_k), PS_k locks F_k for TM_b and sends a F_k lock-grant message which contains the update sequence number of the last update to F_k . If site i has not written this update to its copy of F_k , site i waits until all outstanding F_k updates have been written before TM_b is executed. Once TM_b completes its execution, site i increments the update sequence number for F_k (so that TM_b 's update will be uniquely identified), writes TM_b 's update to its copy of F_k , and broadcasts the update to the other sites. PS_k treats a F_k update as an implicit lock-release by removing TM_b 's lock-request from F_k 's lock queue and unlocking F_k . Note that TM_b encounters inter-computer synchronization delays for both T_E and T_U . These delays begin when site i sends the lock-request message and do not end until i receives the lock-grant message.

The *optimistic timestamp (OTS)* protocol has been discussed in [BERN81] (who uses the term nonconservative timestamps) and [LIN83] (who uses the term basic timestamps). OTS has two sources of appeal: (1) it is relatively simple for a late checking protocol; and (2) [LIN83]'s simulation studies indicate that OTS performs at least as well as other late checking protocols such as WOUND/WAIT [ROSE78] and

FIGURE 1-4: TIMING DIAGRAM FOR OPTIMISTIC TIMESTAMP (OTS)

- TM_A AND TM_B ONLY ACCESS FILE F_K



- $W-TS(A) < W-TS(B) < W-TS'(B)$

- = TRANSACTION EXECUTION
- T_E = TRANSACTION EXECUTION RESPONSE TIME
- T_U = UPDATE CONFIRMATION RESPONSE TIME
- +
- *
- W-TS = WRITE TIMESTAMP

optimistic locking [LIN83]. As with other timestamp protocols, OTS requires that transactions and files have associated timestamps. OTS preserves consistency by ensuring that updates are written in the order prescribed by the timestamp of the updating transactions. If an update violates this ordering, it is removed from the database (or rolled back) and the updating transaction is restarted. Figure 1-4 contains a timing diagram for OTS in which transactions TM_a and TM_b update file F_k . When a transaction is selected for execution, it is assigned a globally unique write timestamp (i.e., $w-ts(a)$).¹ When TM_a first accesses $F_{k,i}$ (the copy of F_k at site i), the current value of $F_{k,i}$'s timestamp ($ts(k,i)$) is saved as TM_a 's read timestamp ($r-ts(a)$) to indicate the version of F_k that TM_a read.² The same is done for TM_b at site j . After execution, the transaction's updates are written to its local file copy (e.g., $F_{k,i}$ for TM_a), the timestamp of its local file copy is changed to the transaction's write timestamp, and a record of these modifications is placed in an update log. (The update log will be used to remove a transaction's updates if a conflict occurs.) Then, messages containing the transaction's updates, its read timestamp, and its write timestamp are sent to the other sites. Let TM be either TM_a or TM_b in figure 1-4, $r-ts$ denote TM 's read timestamp, and $w-ts$ its write timestamp. Since $r-ts < w-ts$ ³, four possibilities exist when site i receives this message.

¹ This can be done by using the current time according to site's local clock and appending to it the site's identifier.

² In general, each F_k copy will have both a read and a write timestamp. However in this example, it suffices to consider a single timestamp for each file copy.

³ This will be the case if during the execution of a transaction at site i no update is written to any file accessed by the transaction.

1. $ts(k,i) = r-ts$

$F_{k,i}$ is identical to the copy of F_k that TM read. So TM 's update is accepted (e.g., TM_a in figure 1-4).

2. $r-ts < ts(k,i) < w-ts$

A conflict occurred and TM lost, since $F_{k,i}$ has been updated by another transaction with a smaller (older) timestamp than $w-ts$. So, TM 's update is rejected (e.g., TM_b in figure 1-4).

3. $ts(k,i) < r-ts$

TM may have read a version of F_k to which was written an update that lost a conflict and hence must be rolled back.¹ So, the TM 's update is rejected.

4. $w-ts < ts(k,i)$

A transaction with a timestamp larger (younger) than $w-ts$ updated site $F_{k,i}$. Thus, a database rollback is required to remove the out of sequence update.

The rollback continues until one of 1, 2, or 3 above is true.

If site i accepts TM 's update, it sends TM 's execution site an update-acknowledgement message indicating acceptance. If site i rejects the update, it sends a message indicating rejection. If all sites accept TM 's update, TM 's execution site removes the update log entry for TM 's update. If any site rejects the update, TM 's execution site rolls back its database to remove TM 's update (if it has not already done so), and TM is restarted in the same manner as it was executed initially. If this restart is rejected,

¹ Another possibility is that site i has not received one or more of the updates that were accepted for $F_{k,i}$. However, if the network is not too slow this will be rare. So, we simplify OTS by assuming that a conflict occurred.

then TM is again restarted. Note that OTS has no inter-computer synchronization delay for T_E . However, inter-computer synchronization delays are required for T_U , since the transaction's execution site must wait for update-acknowledgement messages from the other sites with a copy of F_k .

The foregoing discussion of PSL and OTS provides a basis for characterizing existing CCPs. First, they almost all ensure mutual consistency and internal consistency, as well as serializability (e.g., PSL and OTS). In the case of locking protocols, there is concern about deadlocks.¹ However in real time systems, deadlocks involving file accesses may not be too troublesome since a transaction's file requirements are often known in advance of its execution (e.g., DPAD). This permits using low cost deadlock avoidance techniques. Timestamp protocols require periodic clock synchronization. The extent of this overhead depends on the degree of synchronization required (e.g., [LAMP78]), which is an open research area.

Early checking protocols avoid transaction restarts due to conflicting file accesses², but they require inter-computer synchronization delays for both T_E and T_U , since inter-computer messages must be exchanged to do conflict checking (e.g., PSL lock-request and lock-grant). Late checking protocols can avoid inter-computer synchronization delays for T_E , since conflict checking need not be performed until after

¹ A deadlock occurs when there is a circular wait among transactions waiting for file locks. For example, suppose that TM_a and TM_b both update files F_1 and F_2 , TM_a has locked F_1 , and TM_b has locked F_2 . If TM_a does not release F_1 before it locks F_2 and TM_b does not release F_2 before it locks F_1 , a deadlock is present.

² Even with an early checking protocol, a transaction could be restarted if deadlocks are resolved by using deadlock detection.

the transaction finishes executing. However if a conflict occurs, the transaction must be restarted, which increases T_U .

Since T_E suffices for many real time applications, late checking protocols have appeal. However, such protocols must:

1. prevent conflicting updates from remaining in the database; and
2. ensure that transactions eventually execute without conflict.

The former is achieved by either (a) keeping an update log and rolling back updates that lose a conflict (e.g., OTS) or (b) deferring update writing until the update is confirmed (e.g., the Thomas Majority Vote algorithm [THOM78] or the optimistic protocol proposed in [CERI82]). Approach (a) requires update log maintenance and, when conflicts occur, database rollbacks. *Update log maintenance* consists of saving data values in an *update log* before they are modified. Doing so requires additional time for buffer management, protocol control processing, and, if the update log grows too large, secondary storage accesses. A *database rollback* takes place when an update that loses a conflict is removed from the database. The update log is read to determine the data values present prior to writing the update that lost the conflict and these values are reinstated. A database rollback involves the same types of overhead as update log maintenance. Approach (b), or *deferred update writing*, introduces additional overhead since transactions can not modify files during their execution. Instead, modifications are deferred until the transaction's updates have been validated (i.e., shown not to cause a conflict), thereby requiring additional processing to write the

validated updates.

To ensure that transactions eventually execute without conflict, existing late checking protocols repeatedly restart transactions until they execute without conflict¹ (e.g., OTS). Referred to as *repeated transaction restarts*, this technique is costly when conflicts are frequent (which is when real time constraints are most likely to be violated).² For example consider a late checking timestamp protocol like OTS. If TM_1, \dots, TM_n with $w-ts(1) < \dots < w-ts(n)$ all concurrently read and write F_k , then only TM_1 will succeed³; the other transactions will be restarted (since TM_1 has the smallest timestamp). So, for n concurrently executing and mutually conflicting transactions, $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ transaction executions may be required for all n to succeed. (Even more may be required if new conflicting transactions arrive.) Thus, when conflicts are frequent, repeated transaction restarts can saturate site processing capabilities, cause queue overflows, and overload the interconnection network.

¹ For timestamp protocols, there are two variations depending on how timestamps are selected for the restarted transaction. Either (1) a new timestamp is assigned (e.g., OTS), which permits shorter transactions to complete first to maximize throughput and minimize average response time, or (2) the transaction's original timestamp is retained (e.g., WOUND/WAIT [ROSE78]) which reduces response time variability.

² Actually, there are two cases when existing late checking protocols can avoid transaction restarts: transactions that write but do not read any shared file and transactions that read but do not write any shared file. However, we are concerned with transactions that both read and write shared files.

³ Note that this differs from the case of broadcast communication channels using contention access (e.g., Ethernet under CSMA/CD) in which there is no success when a conflict occurs.

1.2 CONTRIBUTIONS OF THIS DISSERTATION

The goal of this research is to find approaches to consistency control appropriate for real time distributed processing systems. Existing consistency control protocols are unappealing since either: (1) they have inter-computer synchronization delays for execution response times (T_E); or (2) when conflicts are frequent, they have repeated transaction restarts which cause long delays for update confirmation response times (T_U) and can saturate the computers and interconnection network. Our contributions lie in two areas. First, we present *two new protocols that have no inter-computer synchronization delays for T_E and avoid repeated transaction restarts*: the exclusive-writer protocol (EWP) and the exclusive-writer protocol with a locking option (EWL). EWP (see chapter 2) has no database rollbacks, no transaction restarts, no inter-computer synchronization delays for T_E , and small inter-computer synchronization delays for T_U even when conflicts are frequent. However, EWP ensures only a limited form of serializability. EWL (see chapter 3) is a fully serializable extension to EWP that guarantees a transaction will be restarted at most once (if its base set does not change when it is restarted). Performance can be further improved by dynamically switching to primary site locking (PSL) when conflicts are frequent, since PSL does not restart transactions due to conflicting file accesses. Doing so requires no additional messages or inter-computer synchronization delays.

A second area of contribution is our *response times studies of PSL, optimistic timestamps (OTS), and EWL*. While PSL and OTS response times have been studied previously, only simulation techniques were used. In chapter 4, we develop and

validate analytical models for PSL, OTS, and EWL. One application of these models is to determine the criteria for dynamic switching between EWL and PSL.

CHAPTER 2

THE EXCLUSIVE-WRITER PROTOCOL (EWP)

The *exclusive-writer protocol (EWP)* is a new late checking consistency control protocol (CCP) that has no database rollbacks or transaction restarts and has small inter-computer synchronization delays even when conflicts are frequent. EWP ensures mutual and internal consistency, but it provides only a limited form a serializability. Section 2.1 describes EWP's operation, and section 2.2 considers EWP application areas. In section 2.3, we consider some extensions to the algorithm presented in section 2.1. Section 2.4 investigates system design issues related to EWP implementations.

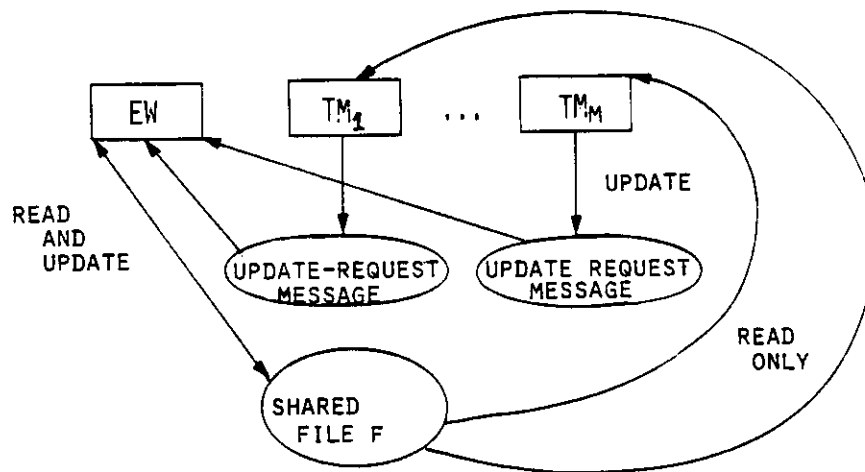
2.1 EWP OPERATION

In the exclusive-writer protocol (EWP), each shared file has a single designated *exclusive-writer (EW)*. (All copies of a file have the same EW.) We define EW_k as the *exclusive-writer for file F_k* . The EW is used to: (1) validate updates after a transaction executes and (2) perform update distribution. This is illustrated in figure 2-1. Transactions can read file copies at any time. When a non-EW transaction wishes to update a file, it sends an update-request message (containing the proposed update) to the file's EW. If the update is accepted, it is distributed by the EW. Such an approach reduces message communication since: (1) only the EW is required to validate an update-request (unlike the Thomas Majority Vote algorithm which requires $\frac{I+1}{2}$ of

FIGURE 2-1: THE EXCLUSIVE-WRITER APPROACH

● DEFINITION

- EACH SHARED FILE HAS A DESIGNATED EXCLUSIVE-WRITER (EW)
(ALL COPIES OF A SHARED FILE HAVE THE SAME EW)
- ONLY A FILE'S EW CAN DISTRIBUTE UPDATES TO THE FILE



- EW IS THE EXCLUSIVE-WRITER OF F.
- TM₁, ..., TM_m SEND THEIR UPDATE-REQUESTS TO EW
- UPDATE-REQUESTS INCLUDE THE PROPOSED UPDATE

the I sites) and (2) the EW's site, rather than the transaction's site, distributes the update (unlike [CERI82] which requires sending an update accepted control message from the validating site to the transaction's execution site). In addition, database rollbacks are avoided since an update is not written to a file copy until the file's EW has validated the update.

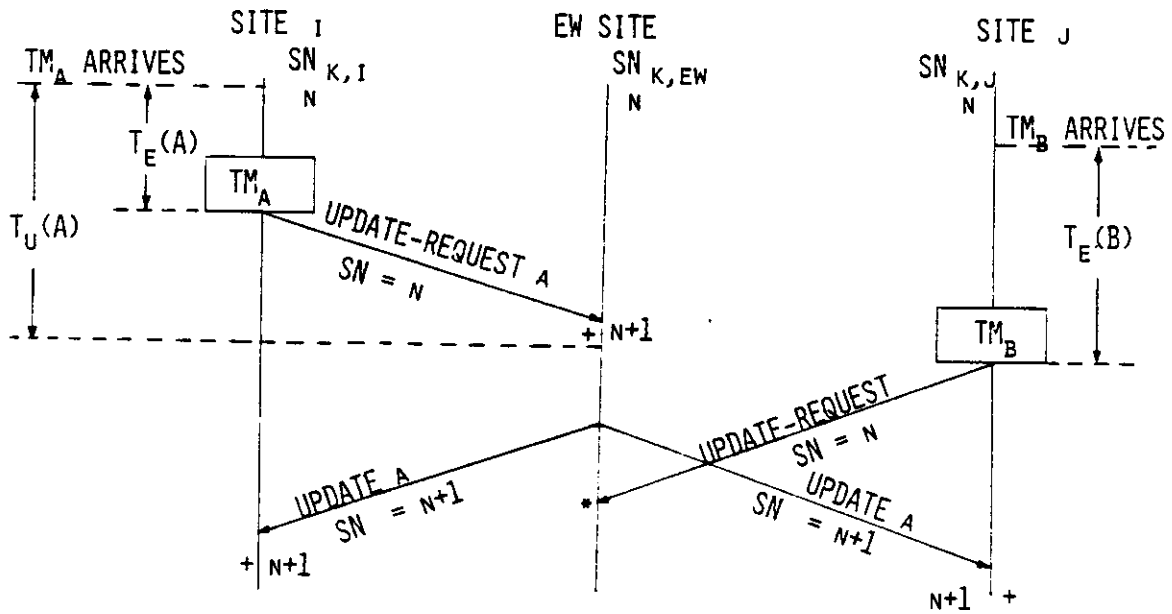
To perform update validation, *update sequence numbers (SNs)* are used to indicate which updates have been written to file copies. An SN is attached to each file copy, and update-request messages include the SN of the file copy read. If an update-request for F_k contains an SN identical to the one at EW_k 's site, the requesting transaction read the most current version of F_k ; so, the update-request is accepted. Then, EW_k (1) assigns a new update version to F_k by incrementing its SN and (2) sends update messages (with the new SN) to the other sites. For example in figure 2-2, $F_{k,i}$, $F_{k,j}$, and $F_{k,ew}$ are, respectively, the copies of F_k at site i , j , and ew (F_k 's EW site); $SN_{k,i}$, $SN_{k,j}$, and $SN_{k,ew}$ are the SNs for these file copies. TM_a reads $F_{k,i}$ with $SN_{k,i} = n$. When TM_a 's update-request arrives at the EW's site, $SN_{k,ew} = n$; so EW_k sets $SN_{k,ew}$ to $n+1$ and sends TM_a 's updates to the other sites. However, if the SN in the update-request differs from its corresponding SN at the EW's site, a conflict occurred (since the transaction read an old version of F_k) and the requesting transaction lost the conflict. In figure 2-2, TM_b at site j reads $F_{k,j}$ with $SN_{k,j} = n$. But when TM_b 's update-request is processed by EW_k , $SN_{k,ew} = n+1 \neq n$, since EW_k had just distributed TM_a 's updates. *EWs discard update-requests that lose a conflict.*

Figure 2-3 presents an algorithm for EWP operation at site i for file F_k . An update-request message is denoted by $RMSG$. We use the programming language notation for records to indicate fields in messages: $RMSG.SN$ is the update sequence number field in the update-request message and $RMSG.UP$ is the the proposed update. $UMSG$ is defined similarly for an update message. The symbol ':=' is used to indicate a programming language assignment statement, and text enclosed in '(* ... *)' are comments. The algorithm consists of several routines, each of which is invoked by a specific event. The algorithm operates under the following assumptions:

1. A transaction's baseset consists only of F_k which is both read and written.
2. Site i and EW_k 's site have a copy of file F_k .
3. At each site, transaction and protocol control processing are performed atomically (i.e., conflicting accesses to shared data are prevented by using local consistency control techniques such as semaphores).
4. Incrementing an update sequence number never causes it to return to 0 (wrap-around).
5. There are no site failures, and the interconnection network guarantees that messages are eventually received without error.
6. When the system begins operation, copies of the same file are identical and their SNs have the same value.

FIGURE 2-2: TIMING DIAGRAM FOR EXCLUSIVE-WRITER PROTOCOL (EWP)

• TM_A AND TM_B ONLY ACCESS FILE F_K



$SN_{K,I}$ = UPDATE SEQUENCE NUMBER FOR THE COPY OF FILE F_K AT SITE I

- = TRANSACTION EXECUTION
- T_E = TRANSACTION EXECUTION RESPONSE TIME
- T_U = UPDATE CONFIRMATION RESPONSE TIME
- + = UPDATE IS WRITTEN
- * = UPDATE IS DISCARDED

Figure 2-3: EWP Operation at Site i for file F_k

TRANSACTION TM HAS JUST COMPLETED ITS EXECUTION

1. $RMSG.SN := SN_{k,i}$
2. $RMSG.UP := TM$'s update
3. send $RMSG$ to EW_k

AN UPDATE MESSAGE ($UMSG$) WAS RECEIVED

1. once $UMSG.SN = SN_{k,i} + 1$
(* Wait until in sequence *)
 - a. $SN_{k,i} := UMSG.SN$
 - b. update $F_{k,i}$ based on $UMSG.UP$

AN UPDATE-REQUEST ($RMSG$) MESSAGE WAS RECEIVED

1. if $RMSG.SN = SN_{k,i}$ (* No Conflict *)
 - a. $SN_{k,i} := SN_{k,i} + 1$
 - b. update $F_{k,i}$ based on $RMSG.UP$
 - c. $UMSG.SN := SN_{k,i}$
 - d. $UMSG.UP := RMSG.UP$
 - e. broadcast $UMSG$ to all other sites with a copy of F_k
2. otherwise (* Conflict *)
 - a. discard $RMSG$

EWP preserves mutual consistency and the serializability of transactions whose update-requests are not discarded. The latter property is called *limited serializability*, since it is limited to transactions that do not lose a conflict. Limited serializability implies internal consistency, since only *successful transactions* (those whose update-requests are not discarded) affect the database, and successful transactions are serializable. We first make a key observation: consecutive updates to F_k are assigned consecutive SNs. This is due to EW_k always incrementing its SN for F_k before a F_k update is broadcast. Mutual consistency follows from the fact that consecutive F_k update messages receive consecutive SNs, and updates are written in order of their SN. Hence, the same updates are written in the same order to all copies of F_k .

To establish limited serializability, we present a serial order for executing successful transactions which results in the same values for F_k as when transactions execute under EWP. Let $TM(m)$ be the m^{th} successful transaction for F_k . We show that $TM(m+1)$ reads a copy of F_k to which $TM(m)$'s updates were the last to be written, or $TM(m+1)$ reads from $TM(m)$. (This is equivalent to a serial execution in which $TM(m)$ executed to completion then $TM(m+1)$ began execution and read $TM(m)$'s updates.) Since $UMSG(m+1)$ ($TM(m+1)$'s update message) is the $(m+1)^{\text{st}}$ successful F_k update, $UMSG(m+1).SN = m+1$. So, $TM(m+1)$'s update-request message ($RMSG(m+1)$) must have had $RMSG(m+1).SN = m$. Note that: (1) a copy of F_k with an SN of m was last written by $TM(m)$, and (2) $RMSG(m+1).SN$ is the SN of the file copy read by $TM(m+1)$. So, $TM(m+1)$ read from $TM(m)$. Hence, EWP preserves limited serializability.

EWP has much appeal for real time systems. It is simple to implement, preserves internal and mutual consistency, and has no database rollbacks. Also unlike other late checking CCPs, *EWP has no transaction restarts*. Since EWP is a late conflict checking protocol, it has no inter-computer synchronization delays for transaction execution response time (T_E). In addition, *EWP's update confirmation response time (T_U) is small even when conflicts are frequent*, since: (1) the only source of inter-computer synchronization delay for T_U is update validation (which is done at the EW's site), and (2) T_U never includes a wait for restarting transactions that lose a conflict (see $T_U(a)$ and $T_U(b)$ in figure 2-2). Also since EWP does not use locking, *EWP avoids deadlocks due to shared data access*. However, caution is required when employing EWP, since it discards update-requests that lose a conflict.

2.2 EWP APPLICATIONS AREAS

For applications such as banking and DPAD final intercept planning, discarding update-requests that lose a conflict is probably not acceptable. However, EWP has much appeal for applications such as:

1. real time feedback control in which conflicts only occur when a real time constraint has been violated;
2. data collection in which occasionally discarding an update-request causes little problem.

Because EWP usage is application dependent, we describe a radar tracking application in which EWP is used (i.e., the DPAD System). A record is maintained in the track file for each object detected (by radar). Track file records are updated only as a result of radar returns for the object. The *real time constraint* of concern here is that *updates to an object's track record must be completed before a new radar beam is sent for the same object*. This constraint ensures that the radar scheduler directs the beam based on the object's most recent sighting so that the beam encounters the correct object.

We observe that a conflict occurs only when the real time constraint is violated (e.g., track processing for record k was not completed prior to scheduling a radar beam for the object associated with record k). By discarding track updates that lose a conflict, system load is reduced which facilitates meeting real time constraints. Of course, information is lost when a track file update is discarded. However, this information can be recovered from future radar returns which, hopefully, will arrive when the system is not so heavily loaded. Thus, under the very extreme conditions of violating a real time constraint, it may be appealing to discard update-requests that lose a conflict.

2.3 EWP EXTENSIONS

So far, we have restricted a transaction's baseset (files it reads and writes) to a single file. Here, we remove this restriction in two stages. First, we consider basesets with multiple files but assume that all files have the same exclusive-writer. Next, we

discuss basesets with multiple exclusive-writers.

Having multiple files in a baseset affects:

1. the contents of update-request messages
2. the criteria for detecting a conflict

Update-requests must include the readset, the writeset, updates to the writeset, and SNs for all files in the baseset. The criteria for detecting a conflict must be generalized as follows: a conflict exists if there is a file in the transaction's baseset whose SN in the update-request differs from its corresponding SN at the EW's site.

If there are multiple EWs for a transaction's baseset, then each EW must accept the transaction's update-request before any EW distributes an update for the transaction. One approach is to have EWs communicate among themselves about whether to accept or reject an update-request. The following is an algorithm using this approach. The update-requesting transaction is TM .

1. TM sends an update-request to the EW of each file in its baseset. The message sent to EW_k contains: TM 's baseset, the SN of the copy of F_k read by TM , and TM 's proposed update to F_k (if F_k is in TM 's writeset).
2. If the SN in the update-request for F_k is identical to the SN of EW_k 's file copy,
 - a. EW_k sends an accept message to the EW of each file in TM 's baseset.
 - b. EW_k remembers that update-request checking is in progress.

Otherwise,

- a. The update-request is discarded.
 - b. Reject messages are sent to the EW of each file in TM 's baseset.
3. If EW_k receives an accept message from the EW of each file in TM 's baseset,
- a. EW_k increments the SN of its file copy.
 - b. TM 's update to F_k is distributed (with the new SN).
 - c. Update-request checking is no longer in progress.
4. If EW_k receives a reject message,
- a. The update-request is discarded.
 - b. Update-request checking is no longer in progress.
5. If EW_k receives an update-request and update-request checking is in progress, the update-request is discarded.
6. File updates are written in order of their SNs.

We now present an argument for why this algorithm preserves limited serializability. Let $TM(i)$ and $TM(j)$ be two transactions whose updates are distributed. Suppose that $TM(j)$ writes file F_k and $TM(i)$ reads F_k . We denote the SN in $TM(j)$'s update message for F_k by $UMSG(k,j).SN$, and the SN in $TM(i)$'s update-

request message for F_k by $RMSG(k,i).SN$. $TM(i)$ reads from $TM(j)$ (denoted by $TM(j) \rightarrow TM(i)$) if and only if

$$UMSG(k,j).SN = RMSG(k,i).SN$$

Further, preserving serializability implies that the following does *not* occur:

$$TM(i) \rightarrow \dots \rightarrow TM(j) \rightarrow TM(i)$$

That is, there is no cycle in the serializability graph [BERN82]. Under the algorithm, $TM(j) \rightarrow TM(i)$ implies that $TM(j)$'s updates were validated and distributed before $TM(i)$ read them. Thus,

$$t_D(j) < t_R(i,j)$$

where:

$t_D(j)$ = Time at which the first of $TM(j)$'s updates was distributed

$t_R(i,j)$ = Time at which $TM(i)$ first read $TM(j)$'s update

Also, $TM(i)$ must finish executing before its updates can be validated by the EWs. So,

$$t_R(i,j) < t_D(i)$$

Hence,

$$t_D(j) < t_R(i,j) < t_D(i)$$

Now, if a cycle exists, then the following must hold.

$$t_D(i) < \dots < t_D(j) < t_R(i,j) < t_D(i)$$

which implies the contradiction that $t_D(i) < t_D(i)$. So, the algorithm preserves limited serializability.

2.4 IMPLEMENTATION CONSIDERATIONS

2.4.1 UPDATE SEQUENCE NUMBER

EWP correctness requires that F_k updates be assigned distinct SNs. Since the number of bits allocated for an SN field will be finite, incrementing a file's SN will eventually cause it to return to 0. Thus, SNs must have a sufficient number of bits so that an update will be written at all sites before the same SN is assigned to another update.

2.4.2 DEGREE OF FILE REPLICATION

So far we have assumed that a copy of a file is present where ever it is accessed. However, this may not be desirable due to the cost of storing and updating replicated file copies. When a transaction accesses a file that is not replicated at its execution site, the site is responsible for obtaining a copy of that file. Along with the file data, the site should also obtain the file copy's SN. The SN of the remote copy is inserted in the transaction's update-request in the same manner as if the file copy had been local.

2.4.3 DESIGNATING EXCLUSIVE-WRITERS

Designating EWs for files affects the frequency of transactions having basesets with multiple EWs. For example, suppose that in some application a transaction's baseset either consists of files F_1 and F_2 or files F_3 and F_4 . Furthermore, suppose that EW_a is the EW for F_1 and F_3 and EW_b for F_2 and F_4 . Then, validating a transaction's update-request requires: (1) sending update-request messages to both EW_a and EW_b , (2) EW_a and EW_b sending one another accept and/or reject messages and (3) broadcasting the update messages. However, if F_1 and F_2 had EW_a as their EW and F_3 and F_4 had EW_b . Then, there would be only one update-request message per transaction execution, and there would be no accept or reject message.

2.4.4 ASSIGNING TRANSACTIONS TO SITES

Three factors affect how transactions should be assigned to sites (referred to as task assignment in [CHU80]): (1) transaction execution time, (2) inter-transaction (or intermodule) communication, and (3) precedence relationships among transactions as a result of inter-transaction control flow. Assigning EW transactions to sites has the following implications:

1. Site utilizations for transaction executions are increased by having an EW assigned to it.
2. Communication time is affected in two opposing ways. When EW_k is assigned to a site where there are transactions that update F_k , communication time can

be decreased since these transactions will not have to send F_k update-request message. However, communication time can be increased by this assignment, since transactions executing at EW_k 's site read the most current copy of F_k . Thus, their update-requests are never discarded, and so their updates are always distributed.

3. A precedence relationship exists if TM_a must execute before TM_b can execute. Suppose that this is the case and that TM_b can execute only when TM_a 's updates have been finalized (i.e., known to not be in conflict). For simplicity, assume that EW is the exclusive-writer for all files in TM_a 's baseset, and EW resides at site ew . Let j be TM_b 's site. If $j \neq ew$, EW must send TM_a 's updates and j must receive and write them before TM_b can begin its execution. However if $j = ew$, then TM_b can be scheduled for execution immediately after TM_a 's update is validated by EW.

2.4.5 OPERATING SYSTEM DESIGN

Operating system design issues include the effect of deferred update writing and the impact of interrupts on local reads to shared files. In EWP, non-EW transactions can not directly modify files. Instead, updates are deferred until the EW has validated them. Deferred update writing requires: (1) providing transactions with temporary file copies to modify during their execution and (2) writing updates once they have been validated. Deferring file updates can be accomplished with little additional overhead if shadow file copies are used for site fault tolerance. With this technique, the operating

system creates a temporary file copy when ever a transaction first writes a file. After the transaction finishes executing, the temporary copy is installed as the finalized copy. Deferred update writing can be achieved by not finalizing the temporary copy until the EW has validated the proposed update.

The impact of interrupts on EWP operation relates to a transaction's ability to read a file copy at any time. This is possible only if writing a copy of file F_k at site i does not occur concurrently with a site i transaction reading F_k . In the DPAD system, this problem is resolved by using a cyclic dispatcher which ensures non-preemptive execution of transactions. Systems that permit preemption require synchronization techniques such as semaphores.

2.4.6 REDUCING MESSAGE VOLUME

For point-to-point networks, EWP message volume can be reduced as follows. When site i sends to EW_k an update-request message with $SN_k = n+1$, the site retains a copy of the message. If EW_k accepts the request, an update-accepted *control message* is sent to site i (instead of an update message), and site i writes the update based on its saved copy. Thus, message volume is reduced since update messages should always be larger than control messages. Site i detects the rejection of its update-request by receiving an update message with $SN_k = n+1$, in which case the site discards its saved message and writes the update message received.

A similar approach can be used for broadcast networks. Update-request messages are broadcast to all sites, and each site saves the update-requests it receives until the update-request is accepted or rejected. When an EW accepts an update-request, it broadcasts an update-accepted control message. Sites use the saved update-request message to write their local file copy.

2.5 SUMMARY

EWP is simple to implement, ensures mutual and internal without database rollbacks, and avoids deadlocks due to shared data access. Unlike other late conflict checking protocols, EWP has no transaction restarts and has minimal inter-computer synchronization delays for T_U even when conflicts are frequent. However, EWP usage is restricted since it discards update-request that lose a conflict.

CHAPTER 3

THE EXCLUSIVE-WRITER PROTOCOL WITH LOCKING OPTION (EWL)

While EWP has good performance characteristics, its usage is restricted since it discards update-requests that lose a conflict and so does not guarantee full serializability. To achieve full serializability with good performance, we want to use a late checking (optimistic) protocol when there is no conflict and an early checking (pessimistic) protocol when there are conflicts. Our approach is to initially execute transactions under EWP; if a conflict occurs, the transaction is restarted under primary site locking (PSL). This hybrid protocol is called the *exclusive-writer protocol with a locking option (EWL)*. Section 3.1 describes EWL's operation, and section 3.2 discusses some extensions to EWL. Section 3.3 describes how performance can be further improved by dynamically switching between EWL and PSL. Section 3.4 discusses implementation considerations.

3.1 EWL OPERATION

The *exclusive-writer protocol with a locking option (EWL)* is an extension of EWP in which an update-request is not discarded if it loses a conflict. Rather, the update-request is treated as a primary site locking (PSL) lock-request message, and the transaction is restarted under PSL. In the following discussion, we use the example in figure 3-1 where TM_a executes at site i and TM_b at site j . TM_a and TM_b both read and

write file F_k , and the EW resides at site ew . A copy of F_k is present at sites i , j , and ew . These copies are denoted by $F_{k,i}$, $F_{k,j}$ and $F_{k,ew}$, respectively. $SN_{k,i}$, $SN_{k,j}$, and $SN_{k,ew}$ are their associated SNs. As in chapter 2, we define EW_k as the exclusive-writer for file F_k .

EWL's operation is identical to EWP's when a transaction arrives (i.e., its execution is requested). In figure 3-1, after TM_a arrives, site i schedules it for execution. After TM_a executes, an update-request message is sent to EW. In addition if there is no conflict, EW's operation is the same for both EWP and EWL. $SN_{k,ew}$ is incremented and TM_a 's updates are broadcast to all other sites with a copy of F_k .

However, EWL differs from EWP when a conflict occurs. For example in figure 3-1, TM_b 's update-request contains an SN of n , but $SN_{k,ew} = n+1$; so a conflict occurred, and TM_b lost. EW puts the update-request in F_k 's lock queue. At this point, EW's role changes. Recall that under EWP, an exclusive-writer

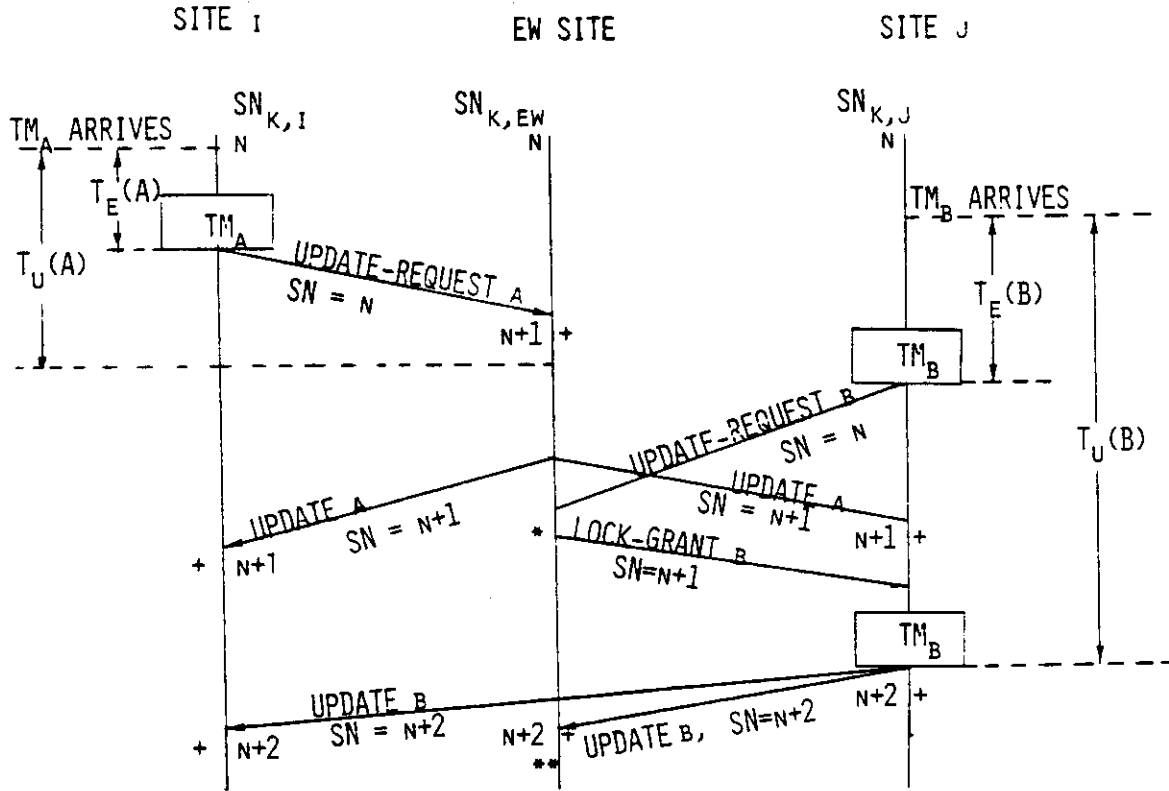
1. detects conflicts.
2. distributes updates (i.e., increments F_k 's SN and broadcasts F_k 's modifications with the new SN value). This is referred to as *update distribution authority (UDA)*.

However, under EWL when F_k 's lock queue becomes non-empty and until its is once again empty, the EW

1. considers arriving update-requests to be in conflict;

FIGURE 3-1: TIMING DIAGRAM FOR EXCLUSIVE WRITER PROTOCOL WITH LOCKING OPTION (EWL)

• TM_A AND TM_B ONLY ACCESS FILE F_K



$SN_{K,I}$ = UPDATE SEQUENCE NUMBER FOR THE COPY OF F_K AT SITE I



= TRANSACTION EXECUTION

T_E = TRANSACTION EXECUTION RESPONSE TIME

T_U = UPDATE CONFIRMATION RESPONSE TIME

+ = UPDATE IS WRITTEN

* = FILE IS LOCKED

** = FILE IS UNLOCKED

2. selects a lock queue entry and delegates UDA to the associated transaction until EW receives the transaction's updates.

The mechanism for delegating UDA is the same as that used by PSL to grant file access. Referring to figure 3-1, TM_b lost a conflict¹, so EW locks F_k for TM_b , and sends a lock-grant message to site j . Included in the lock-grant is the SN for the most current version of F_k , $n+1$. Once $SN_{k,j} = n+1$, site j can restart TM_b . Note that *the transaction is assured of not being restarted more than once*, since it is not restarted until its baseset has been locked. After TM_b executes, site j distributes TM_b 's updates. When EW's site receives TM_b 's update, EW removes TM_b 's entry from F_k 's lock-queue. EW can now assign F_k UDA to another transaction. However, since F_k 's lock queue is empty, EW acquires F_k UDA.

Figure 3-2 contains an algorithm for EWL operation at site i for file F_k . An update-request message is denoted by *RMSG*. We use the programming language notation for records to denote message fields: *RMSG.SN* is the field for the update-request message SN and *RMSG.UP* is the field for the proposed update. (*UMSG* and *LMSG* are defined similarly for an update message and a lock-grant message.) The symbol ':=' is used to indicate a programming language assignment statement, and text enclosed in '(* ... *)' are comments. The algorithm consists of several routines, each of which is invoked by a specific event. The algorithm operates under the following assumptions:

¹ Site j learns that TM_b lost a conflict by receiving an update with $SN = n+1$ that was not made by TM_b .

1. A transaction's baseset consists of F_k which is both read and written.
2. Site i and EW_k 's site have a copy of file F_k .
3. At each site, transaction and protocol control processing are performed atomically (i.e., conflicting accesses to shared data are prevented by using local consistency control techniques such as semaphores).
4. Incrementing an update sequence number never causes it to return to 0 (wrap-around).
5. There are no site failures, and the interconnection network guarantees that messages are eventually received without error.
6. When the system begins operation, copies of the same file are identical and their SNs have the same value.
7. A transaction's baseset does not change when it is restarted.

We now show that EWL preserves mutual consistency and full serializability (hereafter, just serializability). First note that consecutive updates to F_k are assigned consecutive update sequence numbers. This is a consequence of: (1) at most one transaction has UDA at any instant, and this transaction keeps UDA until its updates have been received by EW_k ; (2) a transaction with UDA does not execute until its copy of F_k (and hence SN_k) is current (i.e., identical to EW_k 's copy); and (3) F_k 's SN is always incremented before its updates are distributed. Mutual consistency follows

Figure 3-2: EWL Operation at Site i for file F_k

TRANSACTION TM HAS JUST COMPLETED ITS EXECUTION

1. if TM was restarted
 - a. $SN_{k,i} := SN_{k,i} + 1$
 - b. update $F_{k,i}$ based on TM 's update
 - c. $UMSG.SN := SN_{k,i}$
 - d. $UMSG.UP := TM$'s update
 - e. broadcast $UMSG$ to all other sites with a copy of F_k
2. otherwise
 - a. $RMSG.SN := SN_{k,i}$
 - b. $RMSG.UP := TM$'s update
 - c. send $RMSG$ to F_k 's EW

AN UPDATE MESSAGE ($UMSG$) WAS RECEIVED

1. once $UMSG.SN = SN_{k,i} + 1$
(* Wait until in sequence *)
 - a. $SN_{k,i} := UMSG.SN$
 - b. update $F_{k,i}$ based on $UMSG.UP$
2. if F_k 's EW resides at site i and
 F_k 's lock queue is not empty
 - a. remove head entry in F_k 's lock queue and unlock F_k
 - b. if F_k 's lock queue is not empty
(* Delegate Update Distribution Authority *)
 - i. lock F_k
 - ii. $LMSG.SN := SN_{k,i}$
 - iii. send $LMSG$ to the transaction
with the first entry in F_k 's lock queue

(Figure 3-2: Continued)

AN UPDATE-REQUEST (*RMSG*) MESSAGE WAS RECEIVED

1. if $RMSG.SN = SN_{k,i}$ and
 F_k 's lock is empty
(* No Conflict *)
 - a. $SN_{k,i} := SN_{k,i} + 1$
 - b. update $F_{k,i}$ based on $RMSG.UP$
 - c. $UMSG.SN := SN_{k,i}$
 - d. $UMSG.UP := RMSG.UP$
 - e. broadcast $UMSG$ to all other sites with a copy of F_k
2. otherwise (* Conflict *)
 - a. put $RMSG$ at the tail of F_k 's lock queue
 - b. if $RMSG$ is at the head of F_k 's lock queue
(* Delegate Update Distribution Authority *)
 - i. lock F_k
 - ii. $LMSG.SN := SN_{k,i}$
 - iii. send $LMSG$ to the transaction
with the first entry in F_k 's lock queue

A LOCK-GRANT MESSAGE (*LMSG*) WAS RECEIVED FOR TRANSACTION TM

1. once $LMSG.SN = SN_{k,i}$
(* Wait until $F_{k,i}$ is current *)
 - a. schedule TM for execution

since consecutive F_k updates are assigned consecutive update sequence numbers and updates are written in order of their SN. Thus, all file copies write the same updates in the same order.

Establishing serializability requires a more elaborate argument. Let $TM(m)$ be the transaction that produced the m^{th} consecutive update to F_k under EWL. We show that the same value for F_k would have been obtained by a serial execution of the transactions in the order in which their updates were distributed. Specifically, $TM(m+1)$ read a copy of F_k to which $TM(m)$'s updates were the last to be written, or $TM(m+1)$ reads from $TM(m)$. (This is equivalent to a serial execution in which $TM(m)$ executed to completion then $TM(m+1)$ began execution and read $TM(m)$'s updates.) Let $RMSG(m+1)$ be $TM(m+1)$'s update-request message, $UMSG(m+1)$ be its update message, and i be its site of execution. Note that $UMSG(m+1).SN = m+1$, since $UMSG(m+1)$ is the $(m+1)^{th}$ consecutive update to F_k . If $TM(m+1)$ was not restarted, EW_k distributed $UMSG(m+1)$. So,

$$RMSG(m+1).SN = UMSG(m+1).SN - 1 = m$$

A copy of F_k with an SN equal to m was last updated by $TM(m)$, and $RMSG(m+1).SN$ is the SN of the file copy $TM(m+1)$ read. So, $TM(m+1)$ reads from $TM(m)$. Now suppose that $TM(m+1)$ was restarted. Let $SN_{k,i,t}$ be $SN_{k,i}$ when $TM(m+1)$ was scheduled for execution. Since $SN_{k,i}$ is incremented before $UMSG(m+1)$ is distributed,

$$m + 1 = UMSG(m+1).SN = SN_{k,i,t} + 1$$

Hence, $SN_{k,i,t} = m$. So, $TM(m+1)$ reads from $TM(m)$.

EWL is more difficult to implement than EWP since EWL requires that both EWP and PSL be implemented. However unlike EWP, EWL does not discard update-requests that lose a conflict, and this is achieved without requiring database rollbacks. As with other late checking protocols, EWL has no inter-computer synchronization delay for transaction execution response time (T_E). For update confirmation response time (T_U), EWL's inter-computer synchronization delays are similar to EWP's, if no conflict occurs, and are like PSL's when there is a conflict. For example in figure 3-1, $T_U(a)$ for TM_a is the same as under EWP in figure 2-2; the only inter-computer synchronization delays result from EW update validation. But TM_b is restarted, so $T_U(b)$ has the additional inter-computer synchronization delay of waiting for the F_k lock-grant message. Finally, *unlike existing late checking serializable protocols, EWL guarantees that a transaction will be restarted at most once*, if its baseset does not change.

3.2 EWL EXTENSIONS

In section 3.1, we assumed that a transaction's baseset consists of a single file and that its baseset does not change if the transaction is restarted. Here, we remove these restrictions by considering (1) basesets with multiple files and a single EW, (2) basesets with multiple EWs, and (3) basesets that change when a transaction is restarted.

The following aspects of EWL operation are affected by having basesets with multiple files:

1. contents of update-request messages
2. criteria for conflict detection
3. criteria for granting locks
4. criteria for restarting a transaction

Item 1 is the same as for EWP (see section 2.3). As to conflict detection (item 2), an EW rejects an update-request for a file in the transaction's baseset whose: (1) SN in the update-request is not identical to the corresponding SN at the EW's site or (2) lock queue is not empty. With multiple files, the criteria for granting a lock (item 3) must consider the possibility of deadlock. Thus, some form of deadlock management is required (i.e., avoidance, prevention, detection). As to the criteria for restarting a transaction (item 4), a transaction may not be restarted until: (1) it has received a lock-grant message for all files in its baseset; and (2) all outstanding updates to these files have been written.

Extending EWL to basesets with multiple EWs involves the same considerations as for EWP (see section 2.3). However instead of being discarded, update-requests that lose a conflict are placed in lock-queues for the files they request. Also, transactions must send a lock-release message to the EW of each file locked in read-only mode.

In general, a transaction's baseset may be different when it is restarted, since file access patterns can be data dependent. For example, when transaction *TM* at site *i*

is initially executed, its baseset may consist of file F_1 . If a conflict occurred, F_1 's EW will send TM a lock-grant for F_1 , and TM will be restarted. However, during its second execution TM might update file F_2 rather than F_1 . Thus, site i can not distribute TM 's updates, since TM was not granted a lock for F_2 . This problem can be resolved by having site i ensure that TM has the proper permissions for all files it accesses before its updates are distributed. If it does not, site i : (1) sends a lock-release message to all EWs from which TM had received a lock-grant; and (2) sends update-request messages to all EWs for files in TM 's new baseset.

Whether or not a transaction's baseset changes when the transaction is restarted depends on the application. Thus, it is difficult to make a general statement on this issue. However, with specific reference to DPAD track update transactions, the basesets of these transactions do not change when they are restarted, since they will still access the same record in the track file.

3.3 DYNAMIC SWITCHING BETWEEN PSL AND EWL

As with all serializable late checking CCPs, EWL's performance degrades when conflicts are frequent as a result of transaction restarts (which increase load and lengthen update confirmation response times). EWL improves on existing late checking protocols by guaranteeing that transactions are restarted at most once if their basesets do not change. However, when conflicts are frequent, early checking protocols are preferred since they have no transaction restarts. Ideally, we would like to use a late checking protocol when conflicts are rare and not too costly and an early checking

protocol when conflicts are frequent and/or costly. A very appealing aspect of EWL is that sites can independently and dynamically choose between EWL and a low cost early checking protocol - PSL.

Dynamic switching between CCPs has been proposed by [BERN78] and [MILE79]. In [BERN78]'s SDD-1, four early checking timestamp protocols (P1, P2, P3, and P4) were proposed which were dynamically selected based on a pre-analysis of a transaction's baseset. However, this approach is unappealing since (1) protocol selection is not based on dynamic system behavior; and (2) with the exception of P1, the SDD-1 protocols have high overhead due to inter-computer synchronization delays and message exchanges required for dynamic protocol selection. [MILE79] suggested dynamic switching between the SDD-1/P3 protocol and optimistic timestamps (an early checking protocol). Unfortunately, [MILE79]'s scheme is unattractive since (1) SDD-1/P3 has high overhead and (2) additional messages and inter-computer synchronization delays are required to switch between SDD-1/P3 and OTS.

The appeal of dynamic switching between EWL and PSL is that: (1) both protocols are relatively low overhead, (2) their performance is optimal in different regions of conflict rates (EWL for low conflict rates and PSL for high conflict rates), and (3) for each transaction execution requested, *sites can independently choose whether EWL or PSL should be used for consistency control, without additional messages or inter-computer synchronization delays.* This last feature follows because EWL and PSL can be used concurrently for the same shared file (see figure 3-3). Implementing this capability only requires treating a PSL lock-request as a rejected

EWL update-request. (Recall that EWL already requires implementing PSL.) Specifically, a lock-request received by EW_k is placed in F_k 's lock queue until EW_k selects it. Then, EW_k sends a lock-grant to the requesting transaction, and the transaction executes. EW_k treats the transaction's update as an implicit lock-release.

Establishing the precise criteria for dynamic switching between EWL and PSL is beyond the scope of this dissertation. However, our intuitive criteria are that EWL should be used when the real time constraint depends on execution response time (since EWL has no inter-computer synchronization delay for T_E) or when conflicts are infrequent and not costly (e.g., the system is lightly loaded). Otherwise, PSL should be used. The response time studies in chapter 4 give further insights as to when each protocol is preferred.

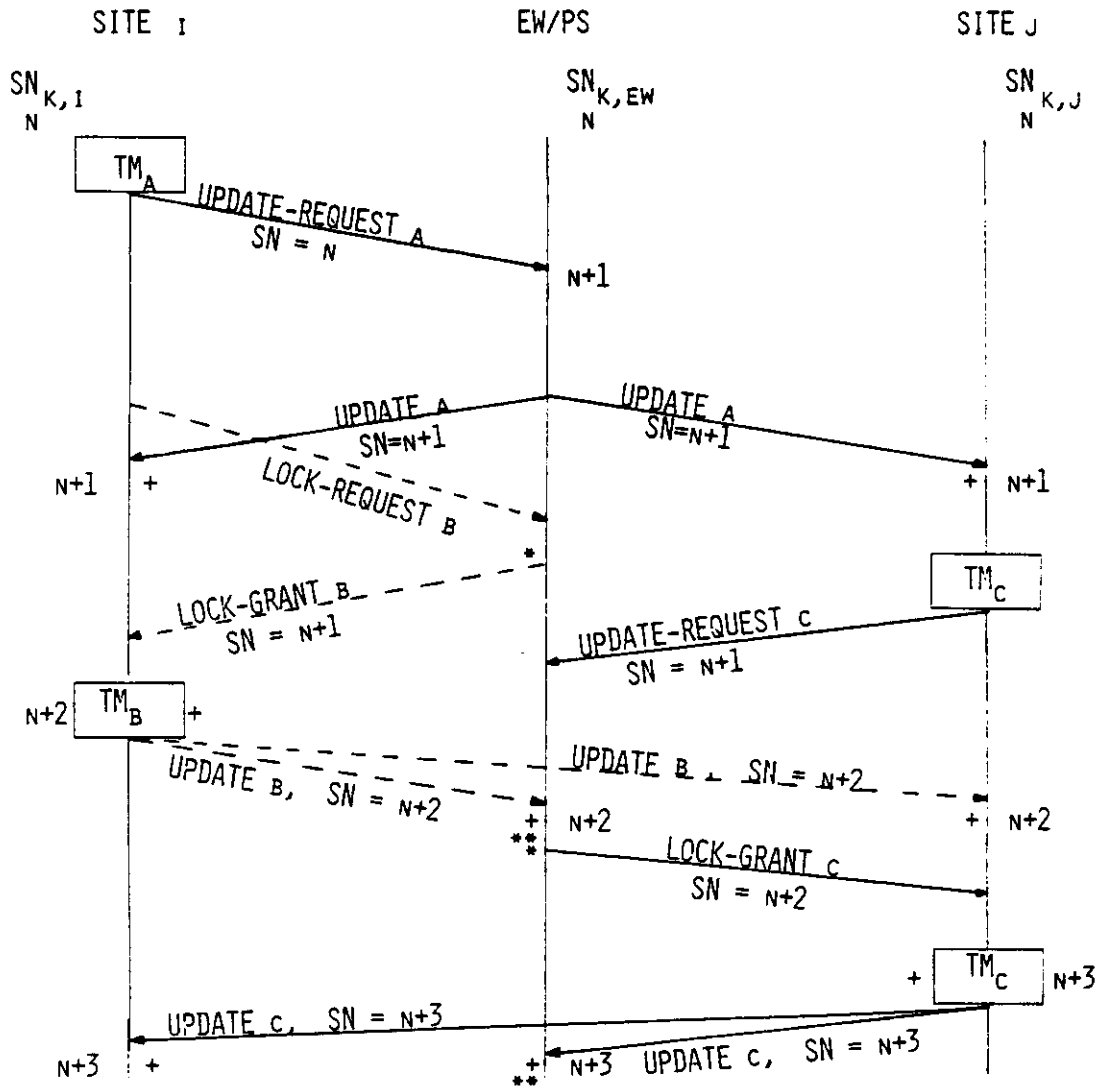
3.4 IMPLEMENTATION CONSIDERATIONS

The same system design issues mentioned in section 3.5 for EWP (i.e., SN field size, degree of file replication, designating exclusive-writers, assigning transactions to sites, operating system design, and reducing message volume) also apply to EWL. In addition, using EWL has implications on transaction scheduling. However, discussing these implications requires some background.

There are two situations in which a site knows a conflict will occur if the transaction is immediately executed. Suppose that site i chooses to execute transaction TM_a which modifies F_k . The following *known conflict situations* can occur depending on where EW_k is assigned:

FIGURE 3-3: CONCURRENT USE OF EWL AND PSL FOR THE SAME SHARED FILE

• TM_A , TM_B , AND TM_C ONLY ACCESS F_K



$SN_{K,I}$ = UPDATE SEQUENCE NUMBER FOR THE COPY OF F_K AT SITE I

□ = TRANSACTION EXECUTION

+ = UPDATE IS WRITTEN

* = FILE IS LOCKED

** = FILE IS UNLOCKED

———— = EWL

----- = PSL

1. EW_k is not assigned to site i .

Suppose that TM_b has recently executed at site i and has proposed modifications to F_k , but TM_b 's updates have not been received at site i . Then, immediately executing TM_a guarantees that it reads an old version of F_k and thus will be restarted.

2. EW_k is assigned to site i

If the lock-queue for F_k is not empty, then immediately executing TM_a guarantees that it will be restarted.

The desirability of executing transactions when a known conflict situation exists depends on: (1) whether T_E or T_U is required for the real time constraint; and (2) system load. If only T_E is required and system load is not too heavy, the cost of additional restarts may be more than offset by avoiding the inter-computer synchronization delays incurred by waiting for the known conflict situation to end. On the other hand, if T_U is required and/or load is heavy, a transaction should not be executed if a known conflict situation exists for it.

3.6 SUMMARY

EWL ensures mutual consistency, internal consistency, and serializability. While deadlocks are possible, EWL has no database rollbacks. Also unlike existing late checking serializable protocols, EWL guarantees that a transaction is restarted at most once if its baseset does not change when it is restarted. Performance can be further improved by dynamically switching to PSL when conflicts are frequent, since PSL has

no transaction restarts due to conflicting file accesses. Doing so requires no additional messages or inter-computer synchronization delays. (Both types of overhead are incurred in existing proposals for dynamic protocol switching.)

CHAPTER 4

RESPONSE TIME ANALYSIS

A key performance measure for real time distributed processing systems is response time. Here, we study the response times of PSL, OTS, and EWL. Since EWP is a simplification of EWL, EWP's response times can be estimated from EWL's. We use the following response time measures:

T_E: Execution Response Time

Definition: Time from the transaction's arrival until its first execution has been completed.

T_U: Update Confirmation Response Time

Definition: Time from the transaction's arrival until its update is known to be finalized at some site.

T_E is useful for real time functions that can operate on data which change frequently, and hence updates can be tentative rather than finalized. (For example in DPAD radar tracking, objects are continuously changing positions.) T_U is needed for real time constraints which require that updates be finalized (e.g., launching a missile).

Very little work has been done on the response time of consistency control protocols (CCP) in distributed processing systems. [RIES79] presented simulation

studies for several variants of primary site locking in distributed INGRES, and came up with conclusions specific to that primary site locking. [LIN81], [LIN82], and [LIN83] simulated a large number of CCP (e.g., primary site locking, primary copy locking, WOUND/WAIT, basic locking), but the system being simulated was never described (e.g., how transactions are assigned to sites, what transmission delays are required), so the results are difficult to interpret. Additionally, the simulation data were not validated (e.g., by an analytical model). [LEE80] developed and validated analytic models for three CCP: network semaphores, hopping permits, and adaptive hopping permits. But [LEE80]'s models did not consider contention for physical resources, such as computers. Probably the most thorough modeling studies have been done by [DANTE80] and [GARC78]. Both developed analytic and simulation models that included contention for computers and disk. [DANT80] studied the Thomas Majority Vote Algorithm proposed by [THOM78]) and [GARC79] studied this algorithm and a centralized locking protocol. However, their models are not directly applicable here since they were developed for different protocols.

The remainder of this chapter is organized as follows. Section 4.1 describes the system for which we developed performance models, and section 4.2 discusses our approach to modeling PSL, EWL, and OTS. Then, analytic models are presented for these three protocols: PSL in section 4.3, EWL in 4.4, and OTS in 4.5. Section 4.6 contains numerical studies based on these models as well as simulation validations. Section 4.7 discusses our results.

4.1 SYSTEM MODELED

The environment in which we model the protocols is a characterization of the DPAD system for track update transactions (see figure 4.1-1). The system has the following characteristics: there are I sites each consisting of a single computer (same as DPAD); there are K files; each site has a copy of all K files and files reside in RAM (same as DPAD); a transaction reads and writes a single file, and the file it accesses does not change if the transaction is restarted (same as DPAD track update transactions); there is both low (L') and high (H') priority background load to account for other system processing; the site scheduling discipline is head-of-line non-preemptive (same as DPAD); high priority work consists of control message processing, update message processing, and high priority background work (same as DPAD); low priority work consists of transaction executions and low priority background work (same as DPAD); shadow file copies are used for fault tolerance of site databases; and externally generated arrivals consist of requests for transaction executions and background load (good approximation to DPAD). Below we list the input parameters for the model:

1. *External Arrivals*

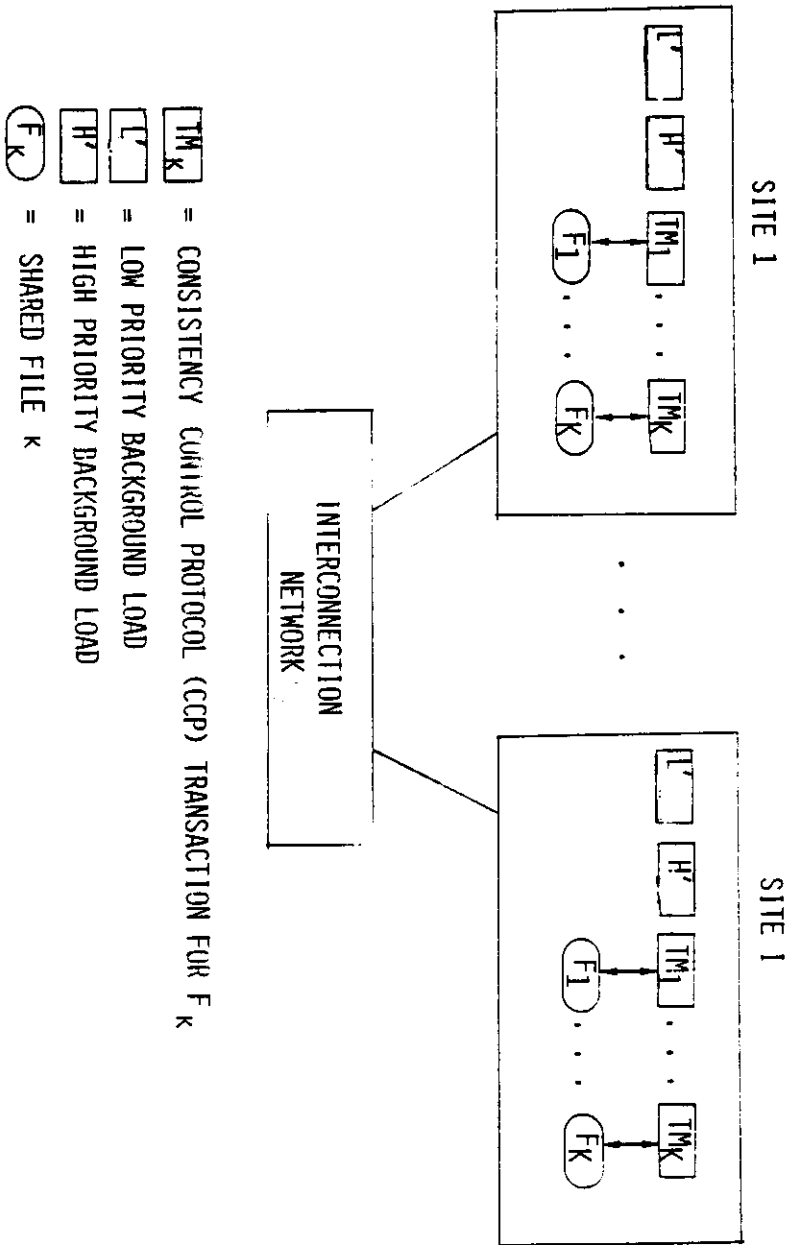
$\lambda(H' ; i)$ - Rate of external arrivals for high priority background work at site i ;

poisson arrival process

$\lambda(L' ; i)$ - Rate of external arrivals for low priority background work at site i ;

poisson arrival process

FIGURE 4.1-1: SYSTEM MODELLED



$\lambda(TM;k)$ - Rate of external arrivals of requests for transactions that access file

F_k ; poisson arrival process

2. *System Size*

I - Number of sites in the system

K - Number of files

3. *Task Assignment*

$p(TM;k,i)$ - Probability that a F_k transaction executes at site i

ps_k - Primary site for file F_k (only for PSL)

ew_k - Exclusive-writer site for file F_k (only for EWL)

$$\delta_{k,i} = \begin{cases} 1 & \text{if } ps_k=i \text{ or } ew_k=i \\ 0 & \text{otherwise} \end{cases}$$

4. *Service Times*

(Assume that mean, second moment, and LaPlace transform are available)

$X(DR;k)$ - Service time for database rollback of F_k

$X(H^i ;i)$ - Service time for high priority background work at site i

$X(L^i ;i)$ - Service time for low priority background work at site i

$X(LE)$ - Service time for lock-release processing

$X(LG)$ - Service time for lock-grant processing

$X(LR)$ - Service time for lock-request processing at the primary/exclusive-writer
site

$X(LR')$ - Service time for formatting a lock-request message and operating
system output processing

$X(C)$ - Service time for network communication of a control message; constant

$X(C;k)$ - Service time for network communication of an F_k update message;
constant

$X(TM;k)$ - Service time for executing a transaction for F_k

$X(UA)$ - Service time for processing update acknowledgements (OTS)

$X(UC)$ - Service time for checking an update to see if it conflicts (OTS)

$X(UI;k)$ - Service time in operating system input routines for receiving and
writing a F_k update message

$X(UM;k)$ - Service time for maintaining the update log for file F_k

$X(UO;k)$ - Service time in operating system output routines for sending a F_k
update message

$X(UR)$ - Service time for update request processing (EWL)

The system handles transaction arrivals, transaction executions, and file updates as follows. When a transaction arrives, it has either: (1) a high priority wait for site lock-request processing (PSL) or (2) a low priority wait for transaction processing (EWL and OTS). When a transaction is selected for execution, an average CPU time of $X(TM;k)$ is required.

$X(TM;k)$ = Average time for F_k transactions to: make a temporary copy of F_k (part of site fault tolerance); execute the transaction; and defer, write, or discard the updates to F_k

Immediately following the transaction's service is operating system update processing, $X(UO;k)$.

$X(UO;k)$ = Average time in operating system output routines to format and send an update or update-request message for F_k

When an F_k update message is received or an F_k update-request message is accepted, update input processing, $X(UI;k)$, is required.

$X(UI;k)$ = Average time in operating system input routines to receive an update message for F_k and apply it to the site's database

4.2 APPROACH TO ANALYTIC MODELS

The models herein developed estimate $T_E(k)$ and $T_U(k)$, which are the response times averaged over all transactions that access F_k . Thus,

$$T_E(k) = \sum_{i=1}^I p(TM;k,i) T_E(k,i) \quad (4.2-1)$$

where:

$T_E(k)$ = Average execution response time of transactions for F_k

$p(TM;k,i)$ = Probability of a transaction for F_k executing at site i

I = Number of sites

$T_E(k,i)$ = Average execution response time of F_k transactions that execute at site i

$$T_U(k) = \sum_{i=1}^I p(TM;k,i) T_U(k,i) \quad (4.2-2)$$

where:

$T_U(k,i)$ = Average update confirmation response time of F_k transactions

$T_U(k,i)$ = Average update confirmation response time of F_k transactions that execute at site i

For tractability, the models make a two approximations: (1) internally generated arrivals (e.g., update messages) are poisson; and (2) each site is an

independent queueing system. The validity of these approximations is indicated by our simulation results.

4.3 PSL MODEL

PSL response times have three components (see figure 4.3-1): pre-lock processing which begins with the transaction's arrival and ends when its lock-request is placed in the lock queue; wait in the lock queue for file access; and time beginning with the lock-grant until the transaction has completed its execution. So,

$$T_E(k,i) = T(PL;k,i) + W(LQ;k) + T(GE;k,i) \quad (4.3-1)$$

where:

$T(PL;k,i)$ = Average time for pre-lock processing of a transaction that accesses F_k and executes at site i

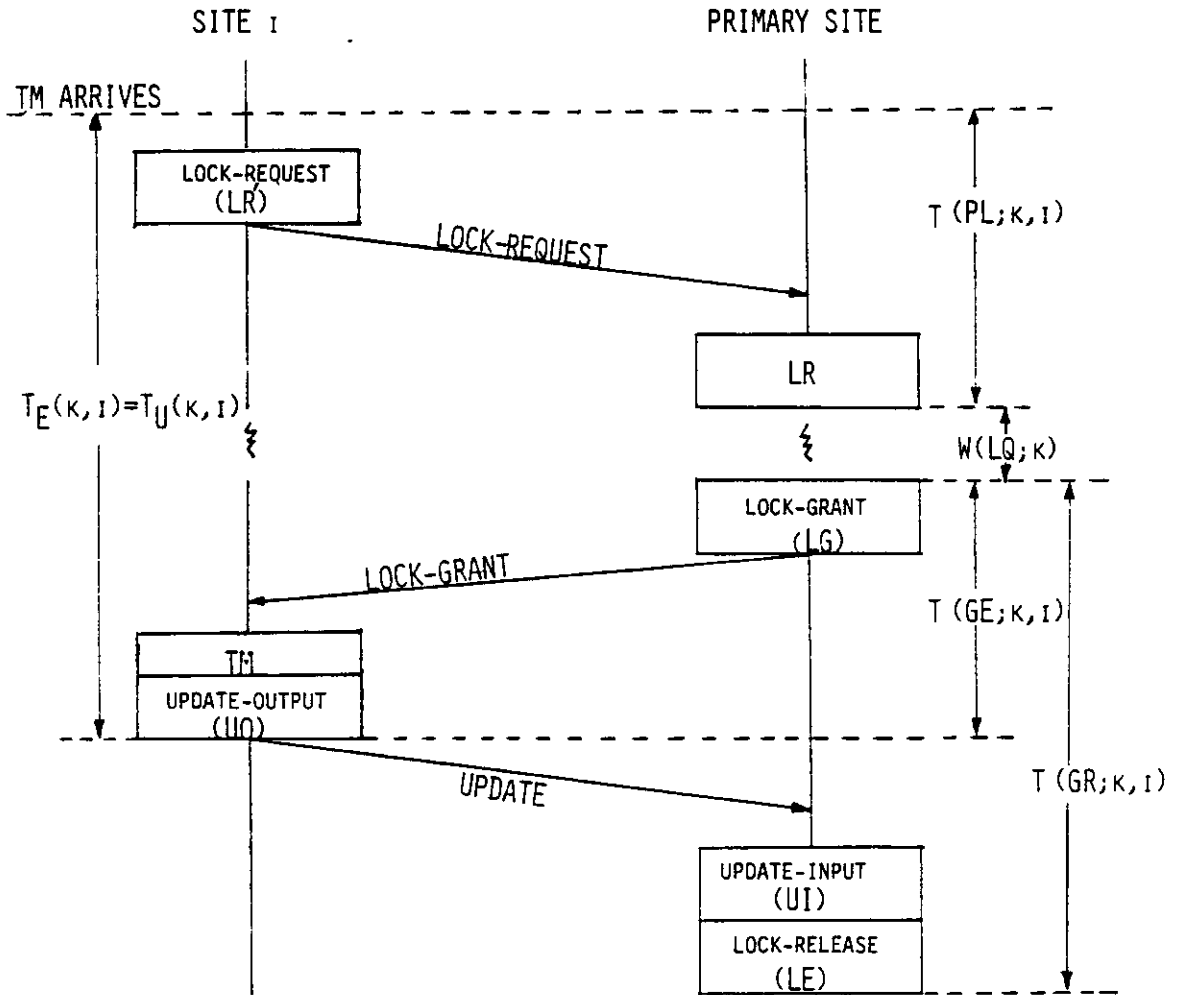
$W(LQ;k)$ = Average wait in the lock queue for a transaction that accesses F_k

$T(GE;k,i)$ = Average time from the lock-grant to completing the execution and update output processing of a transaction that accesses F_k and executes at site i

Furthermore, when a transaction executes under PSL, it is assured that there will be no conflict and hence its update will be finalized (since PSL is an early checking protocol). So,

FIGURE 4.3-1: TIMING DIAGRAM FOR PSL RESPONSE TIMES

- TM ONLY ACCESSES FILE F_k



- $T_E(k,i) = T_U(k,i) = T(PL;k,i) + W(LQ;k) + T(GE;k,i)$

$T(PL;k,i)$ = TIME FOR PRE-LOCK PROCESSING

$W(LQ;k)$ = WAIT IN LOCK-QUEUE

$T(GE;k,i)$ = TIME FROM LOCK-GRANT THROUGH COMPLETING TM'S EXECUTION

$$T_U(k,i) = T_E(k,i) \quad (4.3-2)$$

Pre-lock processing times depend on whether the transaction executes at its file's primary site (*PS transaction*) or it does not execute at its primary site (*non-PS transaction*). Referring to figure 4.3-1, PS transactions have a high priority wait for lock-request processing at the PS plus the lock-request processing itself. Non-PS transactions additionally require a high priority wait to format a lock-request message, time to do the formatting, and network communication of the lock-request message.

$$T(PL;k,i) = W(H;ps_k) + X(LR) + (1 - \delta_{k,i})(W(H;i) + X(LR') + X(C')) \quad (4.3-3)$$

where:

ps_k = Primary site for file F_k

$$\delta_{k,i} = \begin{cases} 1 & \text{if } ps_k = i \\ 0 & \text{otherwise} \end{cases}$$

$X(LR)$ = Average time for lock-request processing at the primary site

$W(H;i)$ = Average wait of high priority work for execution at site i

$X(LR')$ = Average time for formatting a lock-request message

$X(C')$ = Average time for network communication of a control message

For $T(GE;k,i)$, there is time required for the lock-grant processing, network communication of the lock-grant message for non-PS transactions, low priority wait for

transaction execution (which is conditioned on the transaction having locked its file), and update output processing.

$$T(GE;k,i) = X(LG) + (1 - \delta_{k,i})X(C') + W(L;i|k) + X(TM;k) + X(UO;k) \quad (4.3-4)$$

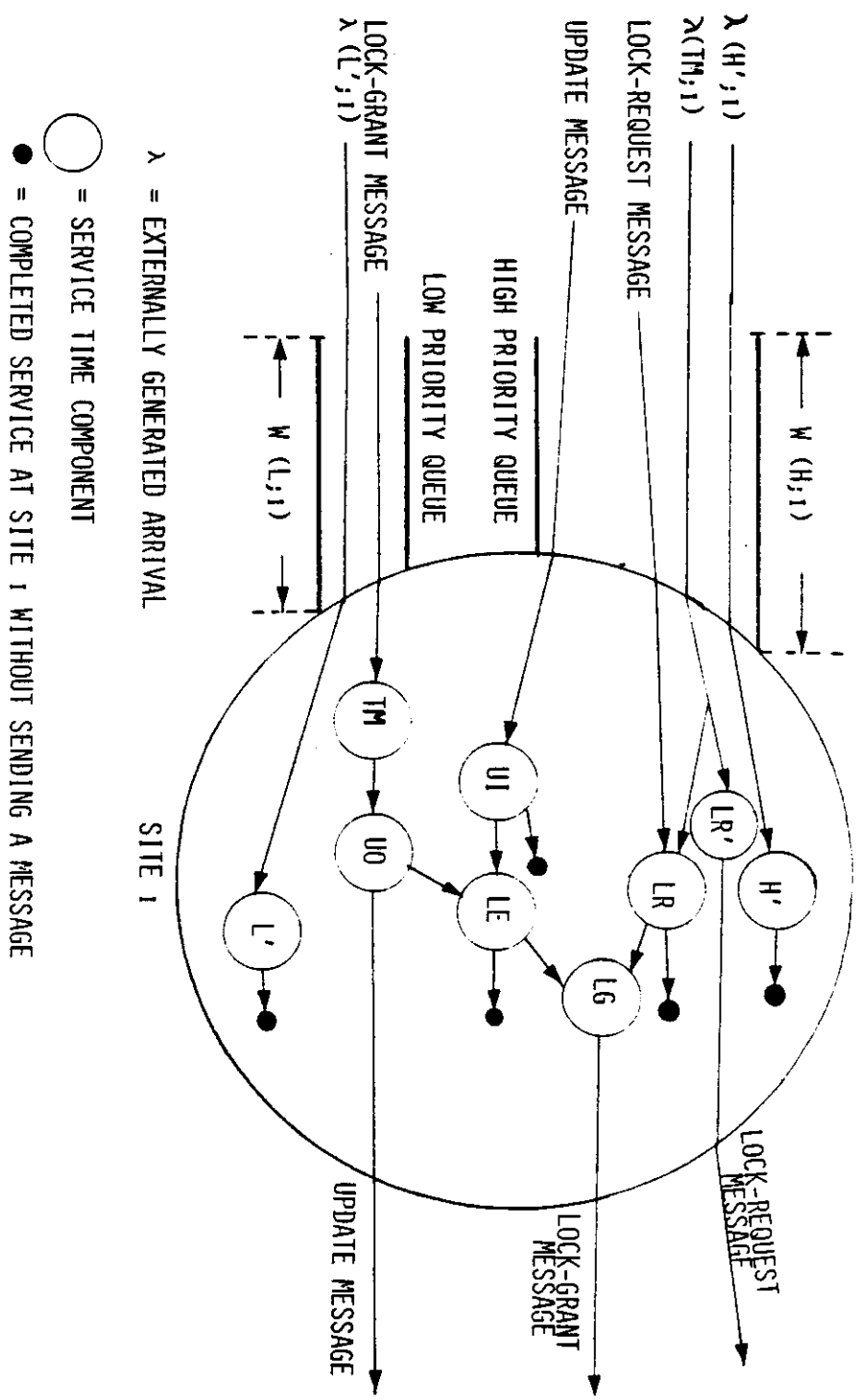
where:

$X(LG)$ = Average service time for lock-grant processing

$W(L;i|k)$ = Average low priority wait at site i conditioned on the waiting work being performed for a transaction that has locked F_k

Only the site and lock queue waiting times are unknown. Since we assume all arrivals are independent poisson processes, estimating the former only requires determining the arrival rates and service times for each type of work at every site. Figure 4.3-3 depicts a queueing system for site i which operates as follows. Background work (H' and L') is queued for execution and executes at its priority level. When a transaction arrives and site i is not its primary site, there is a high priority wait for formatting and transmitting a lock-request message (LR'). If site i is the primary site, lock-request processing (LR) is done, and if the lock queue is non-empty, the request is placed in the lock queue. Otherwise if the lock queue is empty, lock-grant processing (LG) is performed, and a lock-grant message is sent to the transaction's execution site. When a lock-grant message arrives, the locking transaction incurs a low priority wait for execution. After the transaction executes (TM), update output processing (UO) is performed. If site i is the primary site, lock-release (LE) processing is done, and if the

FIGURE 4.3-2: QUEUING SYSTEM FOR SITE 1 UNDER PSL



lock queue is not empty, then lock-grant processing (*LG*) follows. In any case, the updates are distributed. When site *i* receives an update message, update input processing (*UI*) is performed to write the update to the data base. If site *i* is the primary site, lock-release and possibly lock-grant processing follow.

We now consider the transaction's wait in the site execution queue. Since this wait occurs after the transaction has received a lock-grant, delays due to some types of processing will not occur.

1. There should be no other F_k transaction in the execution queue at site *i*. (If there were another F_k transaction, the primary site would have given two concurrent transactions conflicting access to the same file.)
2. If site *i* is the primary site for F_k , then no lock-grant or lock-release processing for F_k will be done during the transaction's wait for execution.
3. Unless the system is heavily loaded, there should be no F_k file update waiting to be processed at site *i*.

Thus, we condition the transaction's wait for execution on having locked its file. Recalling that transactions are low priority work and we assume all arrivals to be poisson, we can apply the formula for M/G/1 head-of-line priority queue systems [KLEI76],

$$W(L; i|k) \leq \frac{W_0(i|k)}{(1 - \rho(H; i) - \rho(H^P; i))(1 - \rho(i))} \quad (4.3-5)$$

where:

$W_0(i|k)$ = Average remaining service time at site i given that the waiting work is performed for a transaction that has locked F_k

$\rho(H;i)$ = Utilization of site i due to high priority consistency control protocol work

$\rho(H^* ;i)$ = Utilization of site i due to high priority background work

$\rho(i)$ = Utilization of site i

This is an upper bound since utilizations are not conditioned on F_k being locked. Similarly for high priority work,

$$W(H;i|k) \leq \frac{W_0(i|k)}{1 - \rho(H;i) - \rho(H^* ;i)} \quad (4.3-6)$$

where:

$W(H;i|k)$ = Average wait of high priority work at site i given that the waiting work is performed for a transaction that has locked F_k

In some cases, site waits should not be conditioned on a file being locked (e.g., wait for lock-request processing). Such waits can be viewed as being conditioned on having locked a non-existent file, since there will be no transaction, file updates, lock-grants, or lock-releases for a non-existent file; and hence, waits are the same as not having locked any file. For low priority work, this is

$$W(L;i) = W(L;i|k), \text{ for } k > K \quad (4.3-7)$$

where:

$W(L;i)$ = Average wait of low priority work at site i

For high priority work, this is

$$W(H;i) = W(H;i|k), \text{ for } k > K \quad (4.3-8)$$

where:

$W(H;i)$ = Average wait of high priority work at site i

To compute the average remaining service time at site i , we must consider all types of work at site i , calculate the average remaining service for each, and then sum the terms ([KLEI76]). For a M/G/1 queueing system, if a type of work has arrival rate λ and second moment $E[y^2]$, its average remaining service time is $\frac{1}{2}\lambda E[y^2]$ ([KLEI75]).

So,

$$W_0(i|k) = \frac{1}{2}(\lambda(H^* ;i)E[X^2(H^* ;i)] + \lambda(L' ;i)E[X^2(L' ;i)]) \\ + W_0(H;i|k) + W_0(L;i|k) \quad (4.3-9)$$

where:

$W_0(H;i|k)$ = Average remaining service time at site i due to high priority consistency control work given that the waiting work is performed for

a transaction that has locked F_k

$W_0(L;i|k)$ = Average remaining service time due to low priority consistency control work at site i given that the waiting work is performed for a transaction that has locked F_k

The first two terms are delays due to background work and the last two terms are delays due to consistency control protocol work (e.g., transactions, file updates).

Utilizations can be calculated by considering low and high priority work for both consistency control and background.

$$\rho(i) = \rho(H;i) + \rho(H' ;i) + \rho(L;i) + \rho(L' ;i) \quad (4.3-10)$$

where:

$\rho(L;i)$ = Utilization of site i for low priority work consistency control work

$\rho(L' ;i)$ = Utilization of site i for low priority work background work

For background utilization, we have

$$\rho(H' ;i) = \lambda(H' ;i)X(H' ;i) \quad (4.3-11)$$

$$\rho(L' ;i) = \lambda(L' ;i)X(L' ;i) \quad (4.3-12)$$

Thus, site waiting times can be computed once we solve for $W_0(L;i|k)$, $W_0(H;i|k)$, $\rho(L;i)$, and $\rho(H;i)$. For average remaining service time, we use the decomposition technique described previously. Since a site may be the primary site for

some files and not for others, we consider separately work for PS operations (e.g., lock-grants) and non-PS operations (e.g., formatting lock-request messages).

$$W_0(H; i | k) = W_0(H, PS; i | k) + W_0(H, \overline{PS}; i | k) \quad (4.3-13)$$

where:

$W_0(H, PS; i | k)$ = Average remaining service time due to high priority processing when site i is the primary site given that the waiting work is performed for a transaction that has locked F_k

$W_0(H, \overline{PS}; i | k)$ = Average remaining service time due to high priority processing for those files for which site i is not the primary site given that the waiting work is performed for a transaction that has locked F_k

We consider each term.

$$W_0(H, PS; i | k) \geq \frac{1}{2} \sum_{l \neq k}^K \lambda(TM; l) \delta_{l,i} \left\{ E[X^2(LR) + E[X^2(LG)] \right. \\ \left. + (1-p(TM; l, i)) E[(X(UI; l) + X(LE))^2] \right\} + \frac{1}{2} \lambda(TM; k) \delta_{k,i} E[X^2(LR)] \quad (4.3-14)$$

where:

K = Number of files

$\lambda(TM; k)$ = Arrival rate of transactions for F_k

$X(LE)$ = Average service time for lock-release processing

Since we are conditioning on F_k being locked, we sum over all files $F_l \neq F_k$. The arrival rate of transactions that update file F_l is $\lambda(TM;l)$. If site i is the primary site for F_l (i.e., $\delta_{l,i} = 1$), then site i performs lock-request (LR) and lock-grant (LG) processing for the transaction. For non-PS transactions, site i does lock-release processing (LE) immediately after receiving a F_l update. Site i receives a file update for all transactions that do not execute at site i ; so the probability of receiving a F_l file update is $1 - p(TM;l,i)$. Since the lock-release immediately follows the update, the second moment of service time is $E[(X(UI;l) + X(LE))^2]$. The last term accounts for F_k lock-request processing, which occurs even though F_k is locked. This equation is a lower bound since it does not consider that lock-grant processing will immediately follow either lock-request or lock-release processing.

Average remaining service time due to high priority processing when site i is not the primary site is

$$\begin{aligned}
 W_0(H, \overline{PS}; i|k) &= \frac{1}{2} \sum_{l=1}^K \lambda(TM;l)(1 - \delta_{l,i})(p(TM;l,i)E[X^2(LR')]) \\
 &+ \frac{1}{2} \sum_{l \neq k}^K \lambda(TM;l)(1 - p(TM;l,i))E[X^2(UI;l)]
 \end{aligned} \tag{4.3-15}$$

The arrival rate of transactions for F_l at site i when $ps \neq i$ is $\lambda(TM;l)(1 - \delta_{l,i})p(TM;l,i)$. For each such transaction, a lock-request message must be formatted (LR'). There also is update input processing for each update message received, and an update message is

received for each transaction that does not execute at site i . So, the arrival rate of update messages is $\lambda(TM;l)(1 - \delta_{l,i})(1 - p(TM;l,i))$. Since we condition on the waiting work being for a F_k transaction that has locked F_k , there should be no update input processing for that file.

Low priority work consists of transaction executions and update output processing. At the primary site, this is immediately followed by lock release processing. Since we condition on the waiting work being for a F_k transaction that has locked F_k , there should be no remaining service time due to a F_k transaction executing.

$$W_0(L;i|k) \geq \frac{1}{2} \sum_{l \neq k}^K \lambda(TM;l)p(TM;l,i) \left\{ \delta_{l,i} \left[E[(X(TM;l) + X(UO;l) + X(LE))^2] \right] \right. \\ \left. + (1 - \delta_{l,i}) E[(X(TM;l) + X(UO;l))^2] \right\} \quad (4.3-16)$$

This equation is a lower bound since it does not consider that lock grant processing may immediately follow lock-release processing.

Utilizations can be computed in a similar manner to remaining service times, only second moments are not used. For high priority work, we consider utilization due to formatting lock-request messages, lock-request processing, lock-grant and lock-release processing for non-PS transactions (since this is done immediately after update input processing which is high priority), and update input processing.¹

¹ Actually this is an approximation, since it assumes that the frequency of high priority lock-grant processing will be the same as the frequency of high priority lock-release processing.

$$\begin{aligned} \rho(H;i) = & \sum_{k=1}^K \lambda(TM;k) \left((1 - \delta_{k,i}) p(TM;k,i) X(LR^i) + \delta_{k,i} X(LR) \right) \\ & + \delta_{k,i} (1 - p(TM;ps_k)) (X(LG) + X(LE)) + (1 - p(TM;k,i)) X(UI;k) \end{aligned} \quad (4.3-17)$$

For low priority work (the previous footnote applies here), we consider transaction executions, update output processing, and lock-grant and lock-release processing for PS transactions (since this is done immediately following the transaction's execution, which has low priority).

$$\rho(L;i) = \sum_{k=1}^K \lambda(TM;k) p(TM;k,i) \left(X(TM;k) + X(UO;k) + \delta_{k,i} (X(LG) + X(LE)) \right) \quad (4.3-18)$$

To estimate $W(LQ;k)$, we view the lock queue as a single server queueing system with First-Come-First-Served (FCFS) service. A service time is the period during which a transaction holds a file lock. This starts when the lock is granted and does not end until the lock is released. So, assuming poisson arrivals at the lock queues, we can apply the formula for M/G/1 waiting times with FCFS service [KLEI75].

$$W(LQ;k) = \frac{W_0(LQ;k)}{1 - \rho(LQ;k)} \quad (4.3-19)$$

where:

$W_0(LQ;k)$ = Average wait in F_k 's lock queue due to the remaining service time of the current lock holder

$\rho(LQ;k)$ = Utilization of the lock queue for F_k

And lock queue utilization is

$$\rho(LQ;k) = \gamma(LQ;k)T(GR;k) \quad (4.3-20)$$

where:

$\gamma(LQ;k)$ = Arrival rate of lock-requests at F_k 's lock queue

$T(GR;k)$ = Average holding time of a lock for F_k (i.e., time starting with F_k lock-grant processing and ending with the completion of F_k lock-release processing)

Note that $\gamma(LQ;k) = \lambda(TM;k)$, since all F_k transactions must lock F_k . Also,

$$T(GR;k) = \sum_{i=1}^I p(TM;k,i)T(GR;k,i) \quad (4.3-21)$$

where:

$T(GR;k,i)$ = Average time starting with F_k lock-grant processing for a transaction that executes at site i and ending with the completion of F_k lock-release processing for that transaction

From [KLEI75], the average remaining service is

$$W_0(LQ;k) = \sum_{i=1}^I p(LQ;k,i)\hat{T}(GR;k,i) \quad (4.3-22)$$

where:

$p(LQ;k,i)$ = Probability that a F_k is locked by a transaction that executes at

site i

$\hat{T}(GR;k,i)$ = Average residual life of $T(GR;k,i)$

Difficulties arise, however, when solving for $p(LQ;k,i)$ and $\hat{T}(GR;k,i)$. The issue is that arrivals at F_k 's lock queue are not truly poisson since an arrival can not occur until lock grant processing has been performed at the PS. So, arrivals are precluded during those portions of $T(GR;k)$ that require PS processing (e.g., $X(LG)$). To correct for this, $T(GR;k,i)$ is decomposed as follows:

$$T(GR;k,i) = X(LG) + T(AP;k,i) + T(RP;k,i) \quad (4.3-23)$$

where:

$T(AP;k,i)$ = Average time when arrivals are possible during the lock holding period of a F_k transaction that executes at site i

$T(RP;k,i)$ = Average time remaining in $T(GR;k,i)$ after $T(AP;k,i)$

Thus,

$$\hat{T}(GR;k,i) = \hat{T}(AP;k,i) + T(RP;k,i) \quad (4.3-24)$$

where:

$\hat{T}(AP;k,i)$ = Average residual life of $T(AP;k,i)$

Assuming that arrivals are poisson during $T(AP;k,i)$ and from [KLEI75],

$$\hat{T}(AP;k,i) = \frac{E[T^2(AP;k,i)]}{2T(AP;k,i)} \quad (4.3-25)$$

To compute $p(LQ;k,i)$, we again assume poisson arrivals during $T(AP;k,i)$. The probability of a poisson arrival occurring during an interval is the same as the fraction of time that interval occurs, or its utilization. So,

$$p(LQ;k,i) = \gamma(AP;k,i)T(AP;k,i) \quad (4.3-26)$$

where:

$$\gamma(AP;k,i) = \text{Arrival rate of } F_k \text{ lock-requests during } T(AP;k,i)$$

And,

$$\gamma(AP;k,i) = \lambda(TM;k)p(TM;k,i) \left(\frac{T(GR;k)}{\sum_{j=1}^I p(TM;k,j)T(AP;k,j)} \right) \quad (4.3-27)$$

$\lambda(TM;k)p(TM;k,i)$ is the arrival rate of lock-requests for F_k made by transactions that execute at site i . Since these requests arrive at F_k 's primary site at any time during $T(GR;k)$ (not just during $T(AP;k,i)$), the adjustment factor in (\dots) is included.

$T(AP;k,i)$ depends on whether the transaction that currently holds the lock for F_k is a PS transaction or a non-PS transaction. If the current lock holder is a non-PS transaction, then arrivals at the lock-queue can occur during the period starting after lock grant processing has been completed and ending once update input processing begins for the lock holder's update. Specifically, this consists of transmission time for the lock-grant, transaction wait for execution, transaction execution time, update

output processing, transmission of the file update, and wait for update input processing. If the current lock holder is a PS transaction, arrivals can only occur during the lock holder's wait for execution.

$$\begin{aligned}
 T(AP;k,i) = & (1 - \delta_{k,i})(X(C) + X(TM;k) + X(UO;k)) \\
 & + X(C;k) + W(H;ps_k[k]) + W(L;i)
 \end{aligned}
 \tag{4.3-28}$$

The remaining period for non-PS transactions is update input processing and lock-release processing; for PS transactions, it is the transaction's execution, update output processing, and lock release processing.

$$T(RP;k,i) = (1 - \delta_{k,i})X(UI;k) + \delta_{k,i}\{X(TM;k) + X(UO;k)\} + X(LE)
 \tag{4.3-29}$$

4.4 EWL MODEL

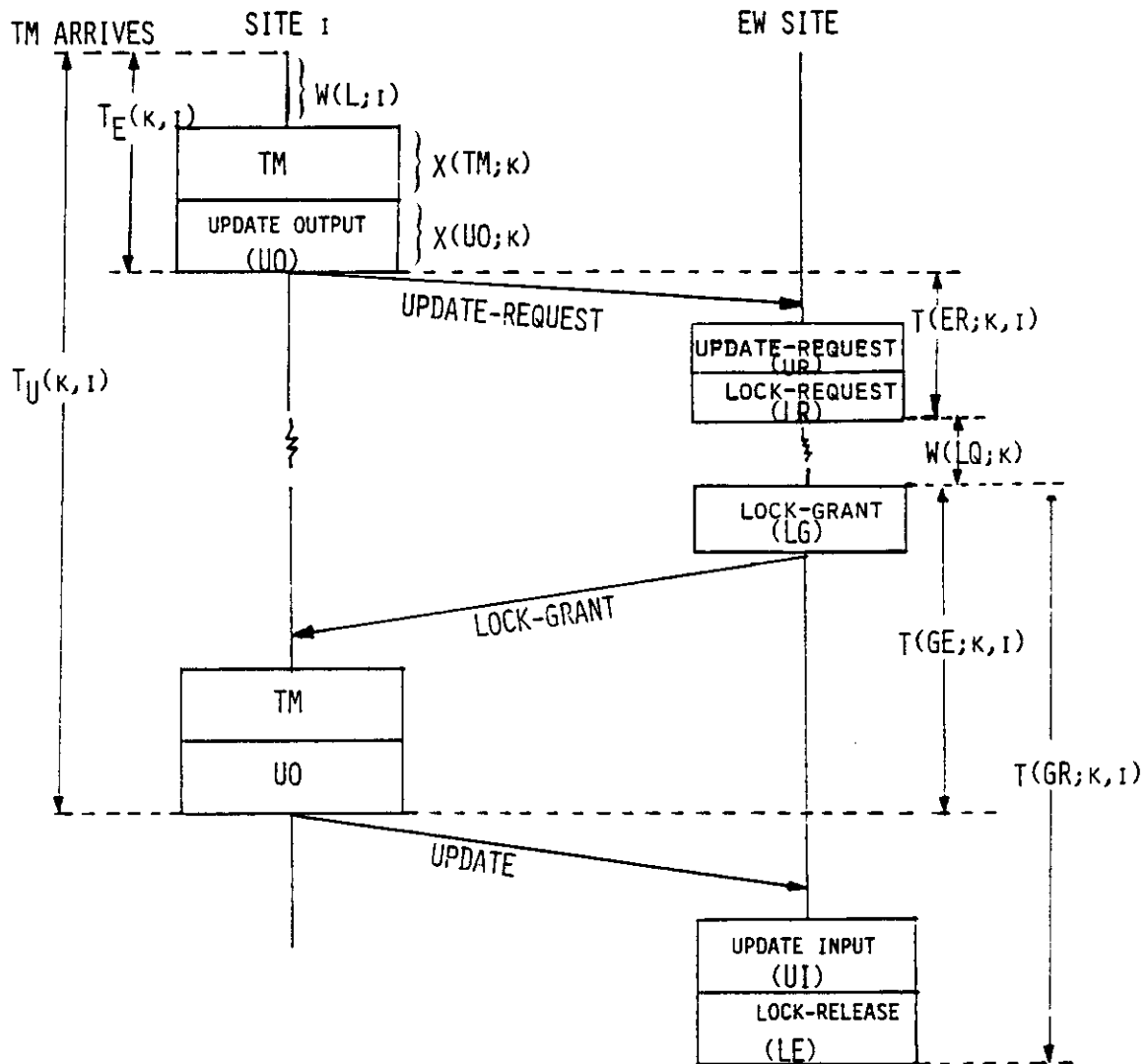
In the following, we refer to transaction's that execute at their EW's site as *EW site transactions* and those that do not execute at their EW site as *non-EW site transactions*.

The components of EWL response times are illustrated for the case of a conflict in figure 4.4-1. T_E consists of the transaction's wait for execution, execution time, and update output processing time.¹

¹ Transactions execute even when a known conflict situation exists. See section 3.4. Also, we assume that if a F_k transaction executes at its EW site and is restarted, the transaction still incurs update output processing ($X(UO;k)$) costs on its initial execution.

FIGURE 4.4-1: TIMING DIAGRAM FOR EWL RESPONSE TIMES

- RESTART CASE FOR TM ACCESSING ONLY FILE F_k



$$\bullet T_E(k, I) = W(L; I) + X(TM; k) + X(UO; k)$$

$$\bullet T_U(k, I) = T_E(k, I) + T(ER; k, I) + q(k, I) [W(LQ; k) + T(GE; k, I)]$$

$T(ER; k, I)$ = TIME FROM EXECUTING A F_k TRANSACTION AT SITE 1
UNTIL COMPLETING ITS UR PROCESSING

$q(k, I)$ = PR [RESTART F_k TRANSACTION THAT EXECUTES AT SITE 1]

$$T_E(k,i) = W(L;i) + X(TM;k) + X(UO;k) \quad (4.4-1)$$

T_U depends on whether or not a conflict occurs. If there is no conflict, then a F_k update-request is known to be finalized once the EW has completed update-request processing and written the update at the EW's site. However, if there is a conflict, the transaction is restarted under PSL.

$$T_U(k,i) = T_E(k,i) + T(ER;k,i) + q(k,i)\{W(LQ;k) + T(GE;k,i)\} \quad (4.4-2)$$

where:

$T(ER;k,i)$ = Average time from completing the execution (and update output processing) until finishing update-request processing for a F_k transaction executing at site i

$q(k,i)$ = Probability of restarting a F_k transaction that executes at site i

The first term is time to complete the transaction's first execution (i.e., execution response time), and the second is time from finishing its first execution until completing update-request processing. The last two terms are included only when there is a conflict, in which case the transaction waits in the lock queue and once the lock is granted, there is the period from the lock-grant until the transaction's second execution has been completed.

$T(ER;k,i)$ includes delays for: (1) network communication of the update-request message, (2) wait for update-request processing at the EW's site, (3) update-request processing, and (4) if no conflict is detected, update input processing, and otherwise

lock-request processing. For EW site transactions, update-request processing is performed immediately after update output processing. So, time is required for update-request processing, and if a conflict occurs, lock-request processing.

$$T(ER;k,i) = (1 - \delta_{k,i})(X(C;k) + W(H;ew_k) + (1 - q(k,i))X(UI;k)) + X(UR) + q(k,i)X(LR) \quad (4.4-3)$$

where:

$X(UR)$ = Average service time for update-request processing which includes checking for conflicts and update distribution if there is no conflict¹

$W(LQ;k)$ and $T(GE;k,i)$ have the same meaning as for PSL and are computed in the same manner as for PSL, with the following two exceptions: (1) EWL values for site waiting times are used; (2) ew_k is used instead of ps_k ;

where:

ew_k = EW site for file F_k

and (3) the arrival rate of lock-requests is computed differently. Recall that locking is required for EWL only when a conflict occurs. For a F_k transaction executing at site i , this occurs with probability $q(k,i)$. So the arrival rate of lock-requests from F_k transactions at site i , is $\lambda(TM;k)p(TM;k,i)q(k,i)$. Summing over all sites,

¹ Very little output processing is required when EWs distribute updates, since the same buffer in which the update-request was received can be used for update transmission. So, the cost of EW update distribution is included in update request (UR) processing.

$$\gamma(LQ;k) = \sum_{i=1}^I \lambda(TM;k) p(TM;k,i) q(k,i) \quad (4.4-4)$$

We use the approach taken for PSL in section 4.3 to solve for site waiting times. Thus, to complete this model we must compute $W_0(H;i|k)$, $W_0(L;i|k)$, $\rho(H;i)$, $\rho(L;i)$, and $q(k,i)$.

The decomposition techniques of section 4.3 are used to compute average remaining service time.

$$W_0(H;i|k) = W_0(H,EW,\overline{RS};i|k) + W_0(H,EW,RS;i|k) + W_0(H,\overline{EW};i|k) \quad (4.4-5)$$

where:

$W_0(H,EW,\overline{RS};i|k) =$ Average remaining service time at site i due to high priority work for EW site processing of transactions that are not restarted given that the waiting work is performed for a transaction that has locked F_k

$W_0(H,EW,RS;i|k) =$ Average remaining service time at site i due to high priority work for EW site processing of transactions that are restarted given that the waiting work is performed for a transaction that has locked F_k

$W_0(H,\overline{EW};i|k) =$ Average remaining service time at site i due to high priority work for non-EW site processing given that the waiting work is performed for a transaction that has locked F_k

These terms reflect average remaining service time when the waiting transaction has locked F_k . Since a site may be an EW site for some files but not for others, we consider separately its EW and non-EW processing. The first term is EW site processing for transactions that are not restarted, the second is EW site processing for transactions that are restarted, and the third is non-EW site processing.

High priority EW site processing for transactions that are not restarted consists of update-request processing followed by update input processing.

$$W_0(H,EW,\overline{RS};i|k) = \frac{1}{2} \sum_{l \neq k}^K \lambda(TM;l) \delta_{l,i} p(\overline{RS};l) E[(X(UR) + X(UI;l))^2] \quad (4.4-6)$$

where:

$$p(\overline{RS};l) = \text{Probability of not restarting a non-EW site transaction for file } F_l$$

For transactions that are restarted, high priority EW site processing consists of update-request processing followed by lock-request processing (applies to F_k transactions since we condition on F_k being locked). When the update is received, there is update input processing, followed by lock-release processing, and when the lock queue is not empty, lock-grant processing.

$$W_0(H,EW,RS;i|k) \leq \frac{1}{2} \sum_{l \neq k}^K \lambda(TM;l) \delta_{l,i} p(RS;l) \left\{ E[(X(UR) + X(LR))^2] + E[(X(UI;l) + X(LE) + X(LG))^2] \right\} + \delta_k \lambda(TM;k) E[(X(UR) + X(LR))^2] \quad (4.4-7)$$

where:

$p(RS;l)$ = Probability of restarting a non-EW site transaction for file F_l EW site

This is an upper bound since it assumes that there is always a lock-grant after a lock-release (i.e., the lock queue is never empty), thereby increasing residual life.

High priority non-EW site processing consists solely of update input processing.

$$W_0(H, \overline{EW}; i|k) = \frac{1}{2} \sum_{l \neq k}^K (\lambda(TM;l)(1 - \delta_{l,i})(1 - p(TM;l,i)q(l,i)) E[X^2(UI;l,i)] \quad (4.4-8)$$

When site i is not the transaction's EW site, update input processing is required at site i except when the transaction is restarted. $p(TM;l,i)q(l,i)$ is the probability of a F_l transaction executing at site i and being restarted. So, the term in (\dots) is the arrival rate of transactions that send update messages to site i .

The average remaining service time due to low priority work results from the first and second execution of transactions.

$$W_0(L,i|k) = W_0(L,1;i) + W_0(L,2;i|k) \quad (4.4-9)$$

where:

$W_0(L,1;i)$ = Average remaining service time due to the first execution of transactions at site i

$W_0(L,2;i|k)$ = Average remaining service time at site i due to the second

execution of transactions given that the waiting work is performed for a F_k transaction that has locked F_k

For $W_0(L,1;i)$, we consider both EW site and non-EW site transactions. For the former, there is the update-request, immediately followed by the transaction's execution, immediately followed by update output processing, which is immediately followed by lock-request processing if the transaction is restarted. For non-EW site transactions, there is the transaction execution, followed by update output processing.

$$W_0(L,1;i) = \frac{1}{2} \sum_{l=1}^K \lambda(TM;l) p(TM;l,i) \left\{ \delta_{l,i} E[(X(UR) + X(TM;l) + X(UO;l) + q(l,i)X(LR))^2] + (1 - \delta_{l,i}) E[(X(TM;l) + X(UO;l))^2] \right\} \quad (4.4-10)$$

For $W_0(L,2;i|k)$, EW site and non-EW site transactions are again considered separately.

$$W_0(L,2;i|k) \leq \frac{1}{2} \sum_{l \neq k}^K \lambda(TM;l) p(TM;l,i) q(l,i) \times \left\{ \delta_{l,i} E[(X(TM;l) + X(UO;l) + X(LE) + X(LG))^2] + (1 - \delta_{l,i}) E[(X(TM;l) + X(UO;l))^2] \right\} \quad (4.4-11)$$

Since we condition on F_k being locked, the locking transaction will not be delayed by

the remaining service time of the second execution of another F_k transaction (since the latter transaction could not execute until it had locked F_k). The arrival rate of transactions that lose a conflict for F_i at site i is $\lambda(TM; l)p(TM; l, i)q(l, i)$. When a transaction is restarted at the EW site, it executes, has update output processing, then releases its lock, and if the lock-queue is not empty, there is a lock-grant that immediately follows. For non-EW site transactions, there is only the transaction's execution and update output processing. This equation is an upper bound since it assumes that the lock queue is never empty.

Computing the probability of restarting or not restarting a non-EW site transaction only requires summing the probabilities of transactions at each site being or not being restarted.

$$p(\overline{RS}; k) = \sum_{i=1}^I (1 - \delta_{k,i}) p(TM; k, i) (1 - q(k, i)) \quad (4.4-12)$$

$$p(RS; k) = \sum_{i=1}^I (1 - \delta_{k,i}) p(TM; k, i) q(k, i) \quad (4.4-13)$$

To compute utilizations, we first consider the arrival rate at site i of update input processing work for F_k .

$$\begin{aligned} \gamma(UI; k, i) = & \lambda(TM; k) (\delta_{k,i} (1 - p(TM; k, i)) \\ & + (1 - \delta_{k,i}) (1 - p(TM; k, i) q(k, i))) \end{aligned} \quad (4.4-14)$$

The first term is for an EW site, since update input processing is required whenever a

non-EW site transaction executes. The second term is for non-EW site transactions, since update input processing is required at their execution site except when the transaction is restarted (since updates will be distributed by the execution site). Thus¹,

$$\begin{aligned} \rho(H;i) = & \sum_{k=1}^K \gamma(UI;k,i)X(UI;k) \\ & + \delta_{k,i}\lambda(TM;k)p(RS;k)[X(LR) + X(LG) + X(LE)] \\ & + \delta_{k,i}\lambda(TM;k)(1-p(TM;k,i))X(UR) \end{aligned} \quad (4.4-15)$$

The first term is utilization due to update input processing, the second is lock request, grant and release processing (occurs with high priority for non-EW site transactions), and the third is update-request processing. For low priority work,

$$\begin{aligned} \rho(L;i) = & \sum_{k=1}^K \lambda(TM;k)p(TM;k,i) \left\{ (1 + q(k,i))(X(TM;k) + X(UO;k)) \right. \\ & \left. + \delta_{k,i}(X(UR) + q(k,i)(X(LR) + X(LG) + X(LE))) \right\} \end{aligned} \quad (4.4-16)$$

The first term in (···) is due to the first and restart executions of transactions as well as their update output processing. The second term is update-request, lock-request, lock-grant, and lock-release processing for EW site transactions (since transactions are low priority work, and these types of processing occur at the transaction's priority level for EW site transactions).

¹ Actually the equation for $\rho(H;i)$ ($\rho(L;i)$) is an approximation since it assumes that the frequency of high (low) priority lock-grant processing will be the same as the frequency of high (low) priority lock-release processing.

What remains to estimate is $q(k,i)$ - the probability of restarting a F_k transaction that executes at site i . Under EWL, there are two reasons for restarting a F_k transaction.

1. *SN CONFLICT.*

The transaction read an old copy of F_k . Thus, the EW will detect that the update sequence (SN) in the transaction's update-request differs from the SN of F_k copy at the EW's site.

2. *LOCK QUEUE CONFLICT.*

When the EW processed the update-request, F_k 's lock queue was not empty.

Assuming that the occurrence of SN and lock queue conflicts are independent events,

$$q(k,i) = 1 - (1 - q(LQ;k))(1 - q(SN;k,i)) \quad (4.4-17)$$

where:

$q(LQ;k)$ = Probability of a lock queue conflict for a F_k transaction

$q(SN;k,i)$ = Probability of an SN conflict for a F_k transaction executing at site

i

Additionally, if we assume (as in the PSL model) that arrivals at F_k 's lock queue are poisson, the probability of a lock queue conflict is the same as the lock queue's utilization. So,

$$q(LQ;k) = \rho(LQ;k) \quad (4.4-18)$$

To compute $q(SN;k,i)$, $q(SN;k,ew_k) = 0$, since transactions executing at F_k 's EW site always read the most current copy of F_k . An SN conflict occurs when the update-request arrives at the EW too soon after an update has been distributed; hence the transaction's execution site did not receive the update before the transaction executed. The cases to consider for SN conflicts are determined by the situations in which updates are distributed. There are four such situations, and they are characterized by: (a) whether or not the transaction executed at its EW site and (b) whether it was the transaction's first or second execution. (Under EWL, transaction's are restarted at most once if their basesets do not change, which is the case for the system we are studying.) Thus, if $i \neq ew_k$ and assuming independence,

$$\begin{aligned}
q(SN;k,i) &= 1 - (1 - q(\overline{EW},1;k))(1 - q(\overline{EW},2;k,i)) \\
&\quad \times (1 - q(EW,1;k))(1 - q(EW,2;k))
\end{aligned}
\tag{4.4-19}$$

where:

$q(\overline{EW},1;k)$ = Probability of a F_k transaction losing an SN conflict due to the first execution of a non-EW site transaction

$q(\overline{EW},2;k)$ = Probability of a F_k transaction losing an SN conflict due to the second execution of a non-EW site transaction

$q(EW,1;k)$ = Probability of a F_k transaction losing an SN conflict due to the first execution of an EW site transaction

$q(EW,2;k)$ = Probability of a F_k transaction losing an SN conflict due to the

second execution of an EW site transaction

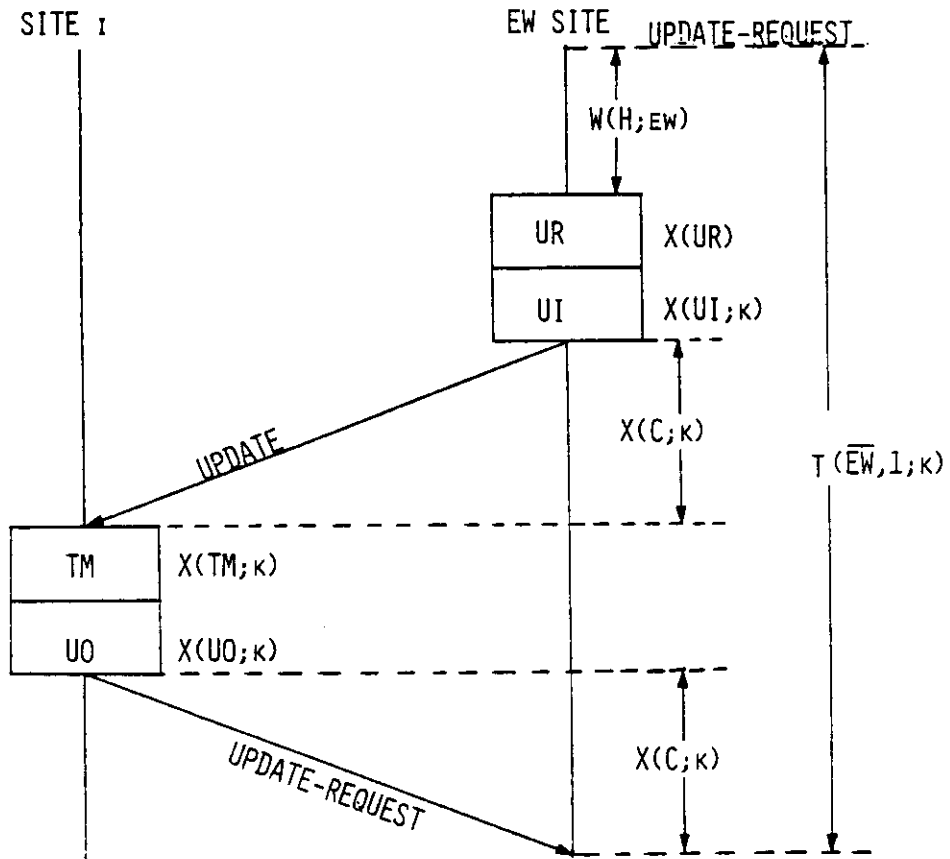
We now estimate the probability of each type of SN conflict. First consider $q(\overline{EW},1;k)$ for a specific transaction, TM, that executes at site $i \neq ew_k$. TM will lose an SN conflict if its update-request arrives at the EW site too soon after the EW has accepted an update-request. Let $T(\overline{EW},1;k)$ be the period of time prior to the arrival of TM's update-request during which the arrival of an update-request that is accepted by the EW will cause TM to lose an SN conflict. Figure 4.4-2 illustrates the components of $T(\overline{EW},1;k)$. Such an conflict causing update-request must, after its arrival, wait for processing at the EW site, perform request and update input processing, and the update must be transmitted. If the update arrives at site i after TM has begun execution, the update will not be written to the site's copy of F_k ($F_{k,i}$) until TM has completed its execution, since scheduling is non-preemptive. However, if the update had arrived just prior to TM being scheduled for execution, $F_{k,i}$ would have been written, since update input processing has higher priority than transaction executions. The last component of $T(\overline{EW},1;k)$ is the transmission time for the update-request. So, $q(\overline{EW},1;k)$ is the probability that there is no arrival during $T(\overline{EW},1;k)$ of an update-request that will be accepted by F_k 's EW. Assuming that such update-requests form a poisson arrival process with rate $\gamma(\overline{EW},1;k)$ and $t = T(\overline{EW},1;k)$ is a constant,

$$q(\overline{EW},1;k) = 1 - \exp[-\gamma(\overline{EW},1;k)t] \tag{4.4-20}$$

where:

$$\gamma(\overline{EW},1;k) = \text{Arrival rate of update-requests made by non-EW site } F_k$$

FIGURE 4.4-2: SN CONFLICT WITH THE FIRST EXECUTION OF A NON-EW SITE TRANSACTION



- TM LOSES AN SN CONFLICT IF AN UPDATE-REQUEST THAT IS ACCEPTED ARRIVES DURING $T(\overline{EW}, 1; k)$

transactions whose update is distributed after the first execution

Integrating over all possible values of t ,

$$q(\overline{EW},1;k) = 1 - \int_0^{\infty} \exp[-\gamma(\overline{EW},1;k)t] T(\overline{EW},1;k,t) \partial t \quad (4.4-21)$$

where:

$T(\overline{EW},1;k,t)$ = Probability density function of $T(\overline{EW},1;k)$ evaluated at t

This integral is the LaPlace transform for $T(\overline{EW},1;k)$ evaluated at $\gamma(\overline{EW},1;k)$. Letting $T^*(\overline{EW},1;k,s)$ be this LaPlace transform,

$$q(\overline{EW},1;k) = 1 - T^*[\overline{EW},1;k,\gamma(\overline{EW},1;k)] \quad (4.4-22)$$

where:

$$T^*[\overline{EW},1;k,\gamma(\overline{EW},1;k)] = \text{LaPlace transform of } T(\overline{EW},1;k) \text{ evaluated at } \gamma(\overline{EW},1;k)$$

With

$$\gamma(\overline{EW},1;k) = \lambda(TM;k) \sum_{j=1}^I (1 - \delta_{k,j}) p(TM;k,j) (1 - q(k,j)) \quad (4.4-23)$$

In general, $T^*(\cdot)$ is the LaPlace transform of $T(\cdot)$. So, from figure 4.4-2 and the fact that the LaPlace transform of the sum of independent random variables is the product of the LaPlace transforms of each random variable

$$\begin{aligned}
T^*(\overline{EW},1;k,s) &= W^*(H;ew_k,s)X^*(UR;s)X^*(UI;k,s) \\
&\times X^*(C;k,s)X^*(TM;k,s)X^*(UO;k,s)X^*(C;k,s)
\end{aligned}
\tag{4.4-24}$$

where:

$W^*(H;ew_k,s)$ = LaPlace transform of $W(H;ew_k)$ evaluated at s

$X^*(UR;s)$ = LaPlace transform of $X(UR)$ evaluated at s

$X^*(UI;k,s)$ = LaPlace transform of $X(UI;k)$ evaluated at s

$X^*(C;k,s)$ = LaPlace transform of $T(C;k)$ evaluated at s

$X^*(TM;k,s)$ = LaPlace transform of $X(TM;k)$ evaluated at s

$X^*(UO;k,s)$ = LaPlace transform of $X(UO;k)$ evaluated at s

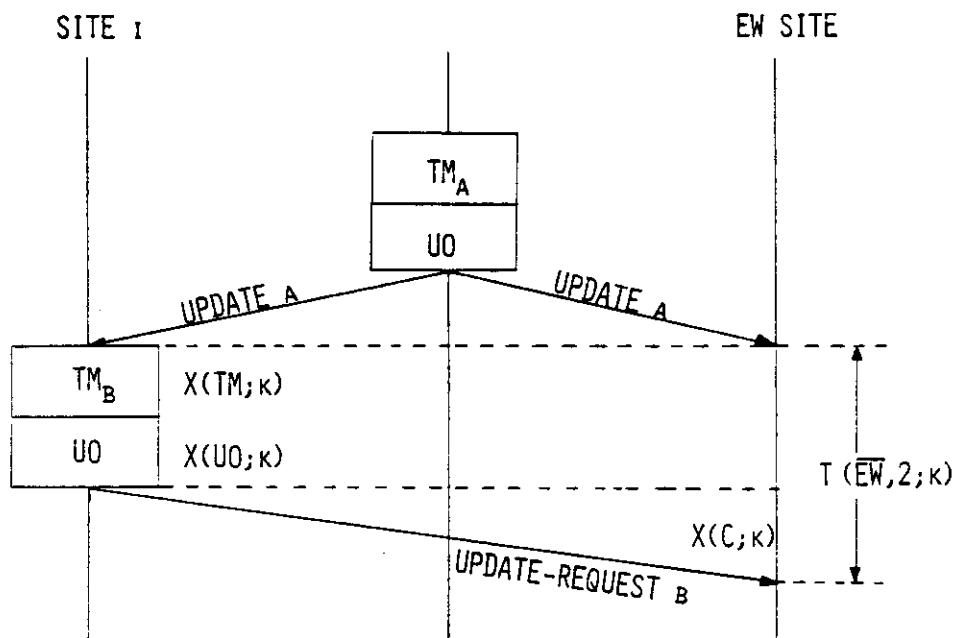
Only $W^*(H;ew_k,s)$ is not an input parameter to the performance model, and this equation has been presented elsewhere ([KLEI76]).

The other three cases for SN conflicts can be derived in a similar manner. Figure 4.4-3 is a timing diagram for SN conflicts caused by the second execution of non-EW site transactions. We use the above described approach to estimate $q(\overline{EW},2;k,i)$.

$$q(\overline{EW},2;k,i) = 1 - T^*(\overline{EW},2;k,i,\gamma(\overline{EW},2;k,i))
\tag{4.4-25}$$

where:

FIGURE 4.4-3: SN CONFLICT WITH THE SECOND EXECUTION OF A NON-EW SITE TRANSACTION



- TM_A IS RESTARTED AND EXECUTES AT A NON-EW SITE
- TM_B LOSES AN SN CONFLICT IF TM_A 'S UPDATE ARRIVES AT THE EW'S SITE DURING $T(\overline{EW}, 2; k)$

$\gamma(\overline{EW},2;k,i)$ = Arrival rate at F_k 's EW of updates to F_k made by the second execution of transactions that did not execute at either their EW's site or site i

$T^*(\overline{EW},2;k,i,\gamma(\overline{EW},2;k,i))$ = LaPlace transform evaluated at $\gamma(\overline{EW},2;k,i)$ of the period during which the second execution of a non-EW site transaction for F_k will cause an SN conflict for a transaction executing at site i

$$\gamma(\overline{EW},2;k,i) = \lambda(TM;k) \sum_{j \neq i}^I (1 - \delta_{k,j}) p(TM;k,j) q(k,j) \quad (4.4-26)$$

This equation excludes transactions that restart at site i , since the transaction that loses the conflict executes at i and thus would read the most current copy of F_k . From figure 4.4-3,

$$T^*(\overline{EW},2;k,s) = X^*(TM;k,s) X^*(UO;k,s) X^*(C;k,s) \quad (4.4-27)$$

Figure 4.4-4 shows a timing diagram for an SN conflict resulting from the first execution of an EW site transaction.

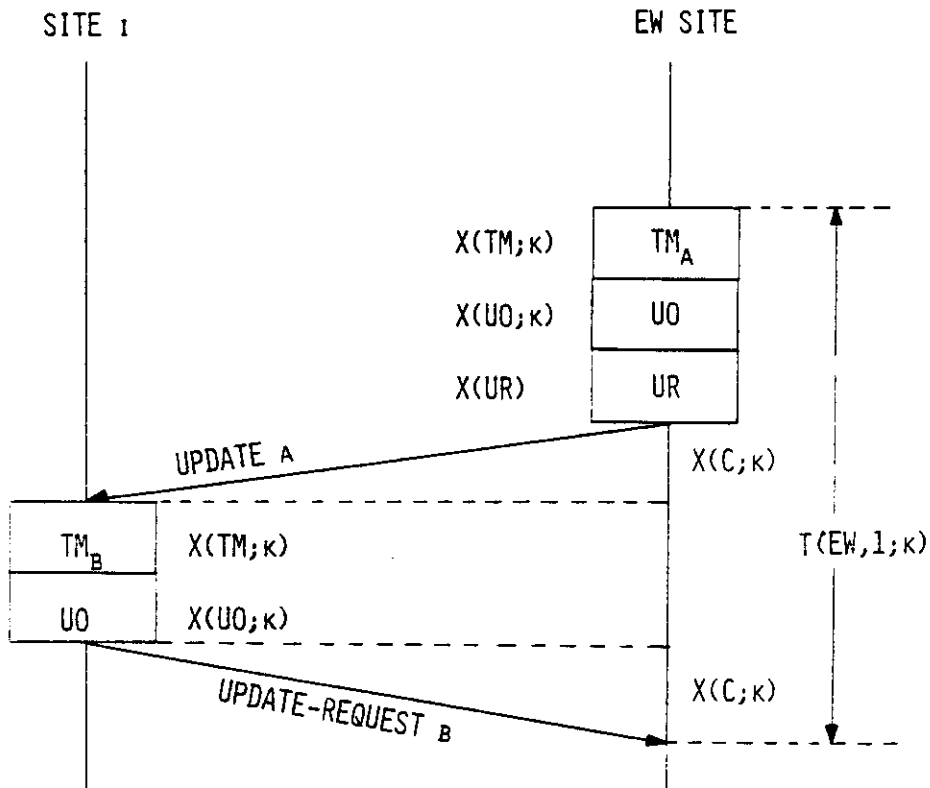
$$q(EW,1;k) = 1 - T^*(EW,1;k,\gamma(EW,1;k)) \quad (4.4-28)$$

where:

$\gamma(EW,1;k)$ = Arrival rate EW site transactions for F_k whose update is distributed after the first execution

$T^*(EW,1;k,\gamma(EW,1;k))$ = LaPlace transform evaluated at $\gamma(EW,1;k)$ of the

FIGURE 4.4-4: SN CONFLICT WITH THE FIRST EXECUTION OF AN EW SITE TRANSACTION



- TM_A 'S UPDATE IS ACCEPTED BY THE EW
- TM_B LOSES AN SN CONFLICT IF TM_A BEGINS EXECUTION DURING $T(EW, 1; \kappa)$

period during which the first execution of an EW site transaction for F_k will cause an SN conflict

Thus,

$$\gamma(EW,1;k) = \lambda(TM;k)p(TM;k,ew_k)(1 - q(k,ew_k)) \quad (4.4-29)$$

And from figure 4.4-4,

$$\begin{aligned} T^*(EW,1;k,s) &= X^*(TM;k,s)X^*(UO;k,s)X^*(UR;s)X^*(C;k,s) \\ &\quad \times X^*(TM;k,s)X^*(UO;k,s)X^*(C;k,s) \end{aligned} \quad (4.4-30)$$

Figure 4.5-5 shows a timing diagram for an SN conflict resulting from the second execution of an EW site transaction.

$$q(EW,2;k) = 1 - T^*(EW,2;k,\gamma(EW,2;k)) \quad (4.4-31)$$

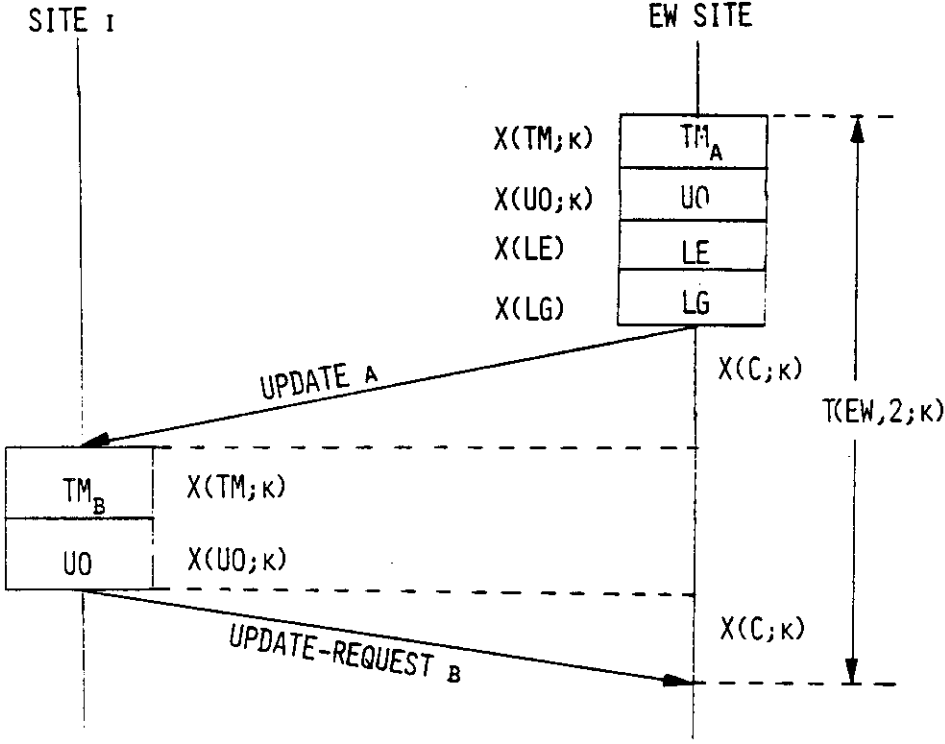
where:

$\gamma(EW,2;k)$ = Arrival rate of EW site transactions for F_k that are executed a second time

$T^*(EW,2;k,i,\gamma(EW,2;k,i))$ = LaPlace transform evaluated at $\gamma(EW,2;k,i)$ of the period during which the second execution of an EW site transaction for F_k will cause an SN conflict

So,

FIGURE 4.4-5: SN CONFLICT WITH THE SECOND EXECUTION OF AN EW SITE TRANSACTION



- TM_A IS RESTARTED AT THE EW'S SITE
- TM_B LOSES AN SN CONFLICT IF TM_A BEGINS EXECUTION DURING $T(EW, 2; k)$

$$\gamma(EW,2;k) = \lambda(TM;k)p(TM;k,ew_k)q(k,ew_k) \quad (4.4-32)$$

And from figure 4.4-5,

$$\begin{aligned} T'(EW,2;k,s) &\geq X'(TM;k,s)X'(UO;k,s)X'(LE;s)X'(LG;s) \\ &\times X'(C;k,s)X'(TM;k,s)X'(UO;k,s)X'(C;k,s) \end{aligned} \quad (4.4-33)$$

where:

$X'(LE;s)$ = LaPlace transform for lock-release processing evaluated at s

$X'(LG;s)$ = LaPlace transform for lock-grant processing evaluated at s

This equation is a lower bound since it assumes there is always a lock-grant after a lock-release (i.e., the lock queue is never empty). By using a lower bound for $T'(EW,2;k)$, we get an upper bound for $q(SN;k,i)$ and hence an upper bound for $q(k,i)$.

Finally, note that the above equations are recursive in that $q(k,i)$ is expressed as a function of $\{q(k,1), \dots, q(k,l)\}$. We resolve this problem by using iterative approximations. Initially, we let $q(k,i) = q(prv;k,i) = 0$, for $i = 1, \dots, l$. Using the above equations, we estimate $q(new;k,i)$ in terms of $q(prv;k,1), \dots, q(prv;k,l)$. If

$$\max_i \{ |q(new;k,i) - q(prv;k,i)| \} < \epsilon \quad (4.4-34)$$

where:

$q(new;k,i)$ = New value of $q(k,i)$ estimated by the iterative approximation algorithm

$q(\text{prev};k,i)$ = Previous value of $q(k,i)$ estimated by the iterative approximation algorithm

the iteration stops, and response times are calculated. Otherwise, the iteration continues. In the numerical studies, $\epsilon = .001$.

4.5 OTS MODEL

Because accurately estimating OTS response times is complicated, we make several assumptions all of which cause our model to be a *lower bound for OTS response times*:

1. Updates that must be rolled back are never written at a site other than the transaction's execution site.
2. There are no cascading rollbacks (i.e., transactions never read a file copy which is rolled back).
3. The update log has infinite size, so there is no cost for removing log entries.

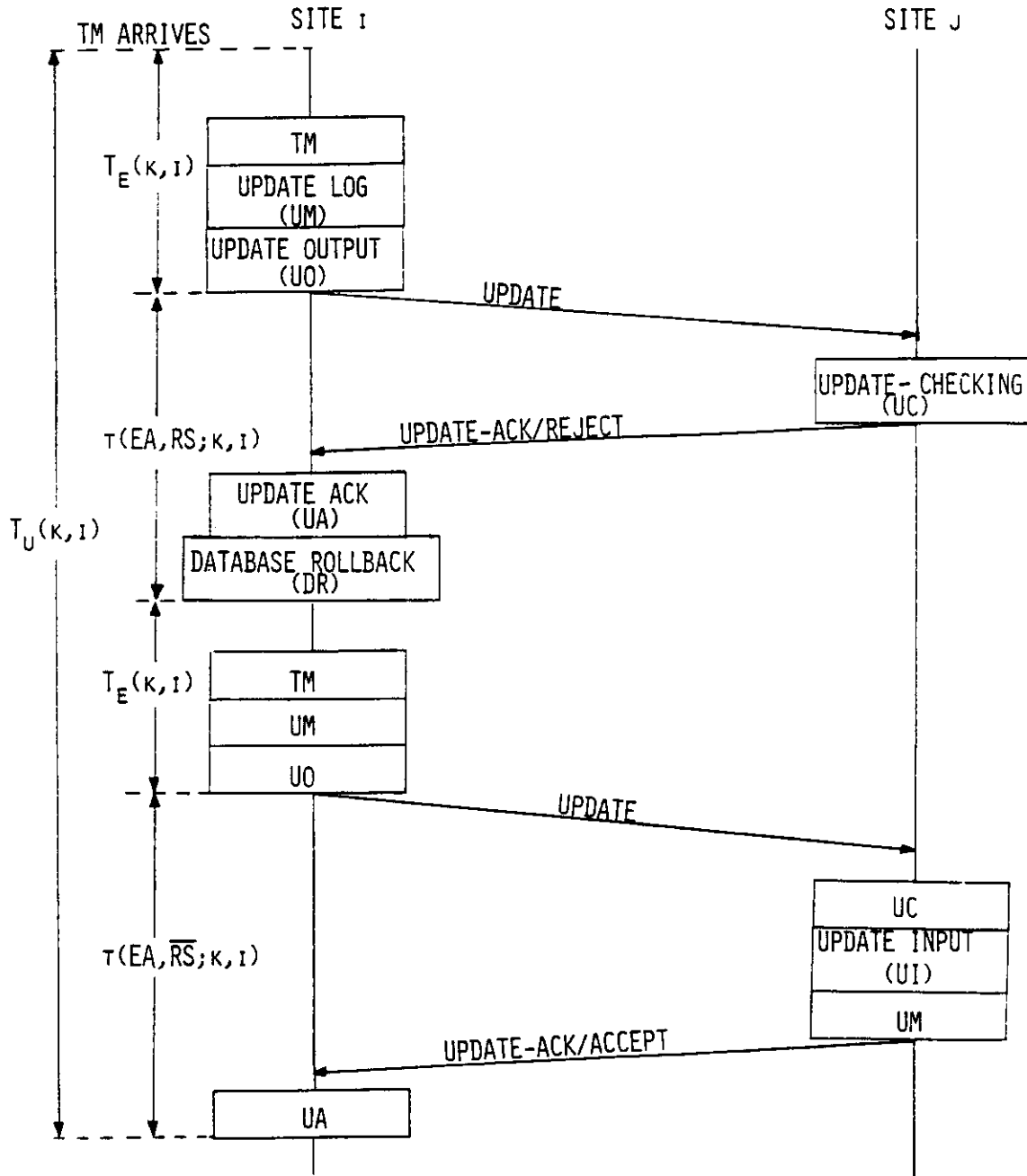
Figure 4.5-1 shows a timing diagram for OTS with its components of response time. For T_E , there is no inter-computer synchronization delay, only the transaction's wait for execution, transaction execution, update output processing, and update log maintenance.

$$T_E(k,i) = W(L;i) + X(TM;k) + X(UO;k) + X(UM;k) \quad (4.5-1)$$

where:

FIGURE 4.5-1: TIMING DIAGRAM FOR OTS RESPONSE TIMES

• TM ONLY ACCESSES FILE F_k



$X(UM;k)$ = Average service time for update log maintenance of file F_k

T_U includes T_E and the time to receive update acknowledgement messages from all other sites.¹ Additionally, there may be one or more transaction restart. A restart occurs for a F_k transaction at site i with probability $q(k,i)$. With probability $1 - q(k,i)$, a F_k transaction is not restarted. Thus, such transactions are restarted exactly n times with probability $(1 - q(k,i))q^n(k,i)$, which is a geometric distribution and has a mean of $\frac{q(k,i)}{1 - q(k,i)}$. So,

$$N_{rs}(k,i) = \frac{q(k,i)}{1 - q(k,i)} \quad (4.5-2)$$

where:

$N_{rs}(k,i)$ = Average number of restarts for a F_k transaction that executes at site i

And the update confirmation response time is

$$T_U(k,i) = N_{rs}(k,i)(T_E(k,i) + T(EA,RS;k,i)) + T_E(k,i) + T(EA,\overline{RS};k,i) \quad (4.5-3)$$

where:

$q(k,i)$ = Probability of restarting a F_k transaction that executes at site i

$T(EA,\overline{RS};k,i)$ = Average time from completing a F_k transaction's execution at site i until the transaction's last update acknowledgement has been

¹ The response time model for OTS assumes that transactions are not restarted until update acknowledgements have been received from all other sites. Doing so prevents a transaction from being restarted until all conflicting updates have been written.

processed when the transaction is not restarted

$T(EA,RS;k,i)$ = Average time from completing a F_k transaction's execution at site i until the transaction's last update acknowledgement has been processed when the transaction is restarted

Referring to figure 4.5-1, $T(EA,\overline{RS};k,i)$ is

$$\begin{aligned}
 T(EA,\overline{RS};k,i) \geq & X(C;k) + \max_{j \neq i} \{W(H;j)\} + X(UC) + X(UI;k) + X(UM;k) \\
 & + X(C') + \max\{(I-2)X(UA), W(H;i)\} + X(UA)
 \end{aligned}
 \tag{4.5-4}$$

where:

$X(UC)$ = Average service time to check an update to see if it conflicts

These terms are time for transmitting the update, waiting for update-checking at each of the other sites, update-checking, update input processing, update log maintenance, transmitting the update-acknowledgement control message, waiting for update-acknowledgement processing at site i (which requires a delay no less than the time to process $I-2$ update acknowledgements or a high priority wait for control processing), and update-acknowledgement processing at site i . The equation is a lower bound since it uses the maximum of means instead of the mean of maximums.

Similarly, $T(EA,RS;k,i)$ is

$$\begin{aligned}
T(EA,RS;k,i) \geq & X(C;k) + \max_{j \neq i} \{W(H;j)\} + X(UC) + \\
& + X(C^*) + \max\{(I-2)X(UA) + X(DR;k), W(H;i)\} + X(UA)
\end{aligned}
\tag{4.5-5}$$

where:

$$X(DR;k) = \text{Average service time for database rollback of file } F_k$$

These terms reflect time for transmitting the update, waiting for update-checking at each of the other sites, update-checking, transmitting the update-acknowledgement control message, waiting for update-acknowledgement processing at site i (which requires a delay no less than the time to process $I-2$ update acknowledgements plus a database rollback or a high priority wait for control processing), and update-acknowledgement processing at site i .

To complete this model, we must estimate site waiting times (i.e., $W(H;i)$ and $W(L;i)$) and the probability of a transaction being restarted - $q(k,i)$. Site waiting times are calculated use the approach of section 4.3, which requires estimating $W_0(H;i)$, $W_0(L;i)$, $\rho(H;i)$, and $\rho(L;i)$. We begin with $W_0(H;i)$ and using the decomposition approach of section 4.3.

$$\begin{aligned}
W_0(H;i) \geq & \frac{1}{2} \sum_{k=1}^K \left\{ \lambda(TM;k,i)(I-1)E[X^2(UA)] \right. \\
& \left. + \lambda(TM;k)(1 - p(TM;k,i))E[(X(UI;k) + X(UM;k))^2] \right\}
\end{aligned}$$

$$+ \gamma(DR;k,i)E[X^2(DR;k)] + \sum_{j \neq i}^I \Lambda(TM;k,j)E[X^2(UC)] \} \quad (4.5-6)$$

where:

$\Lambda(TM;k,i)$ = Arrival rate at site i of externally generated and restarted transactions for F_k

$\gamma(DR;k,i)$ = Arrival rate at site i of database rollbacks for F_k

Each term reflects average remaining service time for a different type of work: (1) update-acknowledgement processing, since F_k transactions (including restarts) arrive at site i with a rate of $\Lambda(TM;k,i)$ and each execution requires processing $I-1$ update acknowledgements; (2) accepted update messages which have an arrival rate at site i of $\lambda(TM;k)(1-p(TM;k,i))$ (site i accepts one update message for each transaction that executes on another site) and requires update input processing immediately followed by update log maintenance; (4) database rollbacks; and (5) update-checking which is required for each update message received (i.e., each non-site i transaction execution).

The number of times a F_k transactions executes at site i is one plus the average number of times a transaction is restarted. Thus,

$$\Lambda(TM;k,i) = \lambda(TM;k)p(TM;k,i)(N_{r,k}(k,i) + 1) \quad (4.5-7)$$

A database rollbacks occurs whenever a transaction is restarted. So,

$$\gamma(DR;k,i) = q(k,i)\Lambda(TM;k,i) \quad (4.5-8)$$

The only low priority consistency control work is transaction executions (which are immediately followed by update output processing and update log maintenance).

So,

$$W_0(L;i) = \frac{1}{2} \sum_{k=1}^K \Lambda(TM;k,i) E[(X(TM;k) + X(UO;k) + X(UM;k))^2] \quad (4.5-9)$$

Utilizations are estimated in a similar manner. The terms of these equations have the same interpretation as those for average remaining service time.

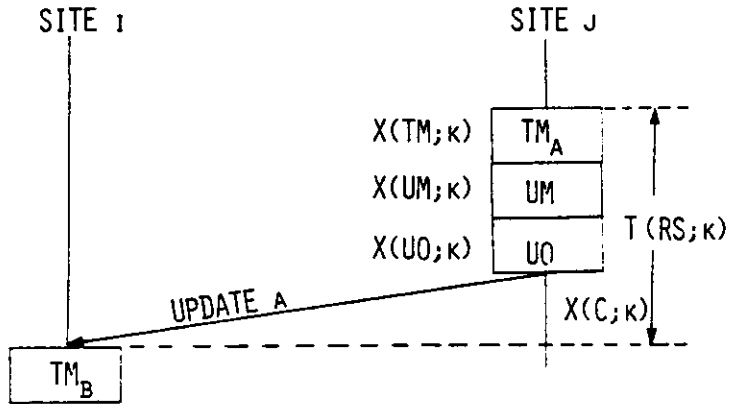
$$\begin{aligned} \rho(H;i) = & \sum_{k=1}^K \left\{ \Lambda(TM;k,i)(I-1)X(UA) \right. \\ & + \lambda(TM;k)(1 - \rho(TM;k,i))(X(UI;k) + X(UM;k)) \\ & \left. + \gamma(DR;k,i)X(DR;k) + \sum_{j \neq i}^I \Lambda(TM;k,j)X(UC) \right\} \end{aligned} \quad (4.5-10)$$

$$\rho(L;i) = \sum_{k=1}^K \Lambda(TM;k,i) (X(TM;k) + X(UO;k) + X(UM;k)) \quad (4.5-11)$$

We complete this model by estimating $q(k,i)$. As shown in figure 4.5-2, transaction TM_b at site i will be restarted if site i has not received an update made by transaction TM_a , since TM_a has a lower valued timestamp than TM_b .¹ Since a transaction's timestamp is assigned when it begins execution, TM_b will be restarted if it begins execution too soon after TM_a has begun execution and if TM_b does not execute

¹ Actually, only those transactions with lower valued timestamps and whose updates are written at site i will cause TM_b to be restarted. We simplify the model by assuming that all lower valued timestamps can cause a restart and justify our simplification through simulation.

FIGURE 4.5-2: ESTIMATING OTS RESTART PROBABILITIES



- TM_B WILL BE RESTARTED IF TM_A EXECUTES DURING $T(RS;k)$

at the same site as TM_a (since TM_b would read TM_a 's update). Let $T(RS;k)$ be the period of time starting when a F_k transaction begins execution and ending when the other sites have received the transaction's update. The arrival rate of F_k transactions at site j is $\Lambda(TM;k,j)$. Assuming poisson arrivals, $q(k,i) = 1 - P_r[\text{no transaction begins execution at site } j \neq i \text{ during } T(RS;k)]$. Using the same transform technique as in the EWL model, $T^*(RS;k,\Lambda(TM;k,j))$ is the probability of no conflict causing transaction executing at site j during $T(RS;k)$. So, the product of these probabilities for $j \neq i$ is the probability of no conflict. Hence,

$$q(k,i) = 1 - \prod_{j \neq i} T^*(RS;k,\Lambda(TM;k,j)) \quad (4.5-12)$$

where:

$T^*(RS;k,\Lambda(TM;k,j)) =$ LaPlace transform evaluated at $\Lambda(TM;k,j)$ of the period of time following the execution of a F_k transaction during which executing another F_k transaction will cause the latter to be restarted

This LaPlace transform can be calculated as follows.

$$T^*(RS;k,s) = X^*(TM;k,s)X^*(UO;k,s)X^*(UM;k,s)X^*(C;k,s) \quad (4.5-13)$$

where:

$X^*(UM;k,s) =$ LaPlace transform of the service time for a F_k database rollback

Note that as with the EWL model, $q(k,i)$ is recursively defined (e.g., $q(k,i)$ is a function of $\Lambda(TM;k,i)$ which is a function of $q(k,i)$). As with EWL, we use iterative approximations with $\epsilon = .001$ for the numerical studies.

4.8 NUMERICAL STUDIES

Here, we use the analytical models for PSL, EWL, and OTS to study the effect of the input parameters on response times (i.e., T_E and T_U). Ideally, we would indicate which protocol is preferred for each region of input parameter values. However, this goal is difficult to achieve since there are a large number of input parameters, many of which affect two or more of the protocols in a complex way. Instead, we show how input parameters affect the protocols. We reduce the number of parameters by grouping similar parameters together. Specifically, we perform the following numerical studies by changing the values of the input parameters indicated.

1. Baseline.

Input parameters are set to values reasonable for the DPAD System. In the other studies, parameters values that are not explicitly mentioned have their baseline value.

2. Background Load.

$\lambda(H';i), \lambda(L';i)$

3. Transaction Execution Times.

$X(TM)$

4. Update Processing Times.
 $X(UO), X(UI)$
5. Control Processing Times.
 $X(LR), X(LR'), X(UA), X(UC), X(UR)$
6. Network Communication Times.
 $X(C), X(C')$
7. Lock-grants and Lock-release Processing Times.
 $X(LG), X(LE)$
8. Update Log Maintenance and Database Rollbacks.
 $X(UM), X(DR)$
9. Number of Sites.
 I
10. Transaction Load Partitioning.
 $p(TM;i)$

Throughout this section we assume there is only a single shared file, since (under the model assumptions) the effect of multiple shared files is the same as having background load. Thus, we do not index parameters by file.

Utilization is an important parameter for understanding the behavior of queueing systems. In the system we study, there is queueing for only two types of resources: sites (for program execution) and files (for write access). We define

utilization measures for each.

$\rho(i)$ = Utilization of site i

$\rho(LQ)$ = Utilization of the file's lock queue

For sites, we are concerned with the one that is most heavily utilized, referred to as the bottleneck site.

$$\rho = \max_i \{\rho(i)\} \quad (4.6-1)$$

where:

ρ = Utilization of the bottleneck site

We begin with the baseline study, in which parameters are set to values that are reasonable for the DPAD system (see figure 4.6-1). Figure 4.6-2 plots T_E against the external arrival rate of transactions ($\lambda(TM)$). Since the response time models are approximations, simulation models of the protocols were implemented in PAWS [BERR82], and simulation studies were performed. In figure 4.6-2, results from these studies are presented as 90% confidence intervals (bars).

PSL has the largest T_E over a wide range of $\lambda(TM)$. This is due to the *cost of locking*, which consists of processing lock-requests, lock-grants, and lock-releases as well as inter-computer synchronization delays due to waits in the lock queue. To better illustrate this point, we decompose PSL response times into pre-lock processing, waits in the lock queue, and time from the lock-grant through the transaction's execution. (For details, see section 4.3.) Since PSL is an early checking protocol, $T_E = T_U$. So,

Figure 4.6-1: Baseline Values For Parameters

$\lambda(TM;k)$; poisson arrival process

$\lambda(H^i ;i) = 0$; poisson arrival process

$\lambda(L^i ;i) = 0$; poisson arrival process

$I = 5$

$K = 1$

$\rho(TM;k,i) = \frac{1}{I}$

$X(DR;k) = .3 \text{ msec}$; exponential

$X(H^i ;i) = 1. \text{ msec}$; exponential

$X(L^i ;i) = 1. \text{ msec}$; exponential

$X(LE) = .25 \text{ msec}$; constant

$X(LG) = .25 \text{ msec}$; constant

$X(LR) = .25 \text{ msec}$; constant

$X(LR^i) = .05 \text{ msec}$; constant

$X(C^k ;k) = .01 \text{ msec}$; constant

$X(C;k) = .01 \text{ msec}$; constant

$X(TM;k) = 2.5 \text{ msec}$; exponential

$X(UA) = .05 \text{ msec}$; constant

$X(UC) = .05 \text{ msec}$; constant

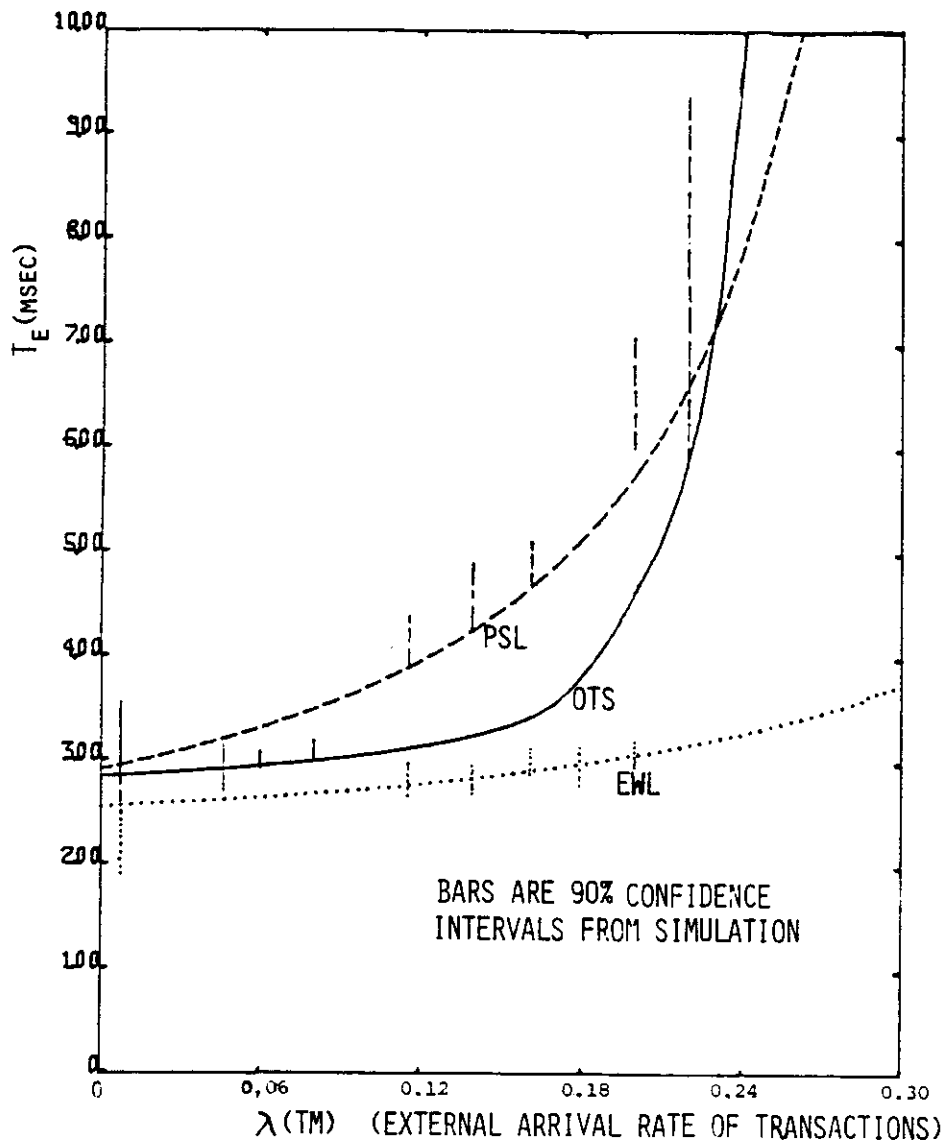
$X(UI;k) = .06 \text{ msec}$; exponential

$X(UM;k) = .3 \text{ msec}$; exponential

$X(UO;k) = .06 \text{ msec}$; exponential

$X(UR) = .05 \text{ msec}$; constant

FIGURE 4.6-2: T_E FOR BASELINE



$$T_U(PSL) = T_E(PSL) = T(PL) + W(LQ,PSL) + T(GE,PSL) \quad (4.6-2)$$

where:

$T(PL)$ = Average time for pre-lock processing

$W(LQ,PSL)$ = Average wait in the lock queue for PSL

$T(GE,PSL)$ = Average time for PSL from the lock-grant through the transaction's execution

Figure 4.6-3 plots these components. When $\lambda(TM)$ is small, locking increases response times because of processing for lock-requests, lock-grants, and lock-releases; this is included in $T(PL)$ and $T(GE,PSL)$. As $\lambda(TM)$ increases, $W(LQ,PSL)$ dominates PSL response times. In fact, *PSL does not saturate due to site utilizations but due to lock queue utilizations* (see figure 4.6-4).

$T_E(OTS) > T_E(EWL)$ when $\lambda(TM)$ is small due to OTS update log maintenance. As $\lambda(TM)$ increases, OTS has transaction restarts (see figure 4.6-5). Under OTS, transactions are repeatedly restarted until they execute without conflict, referred to as *repeated transaction restarts*. Thus, the number of times an OTS transaction is restarted, $N_{r,r}(OTS)$, is

$$N_{r,r}(OTS) = \frac{q(OTS)}{1 - q(OTS)} \quad (4.6-3)$$

where:

$N_{r,r}(OTS)$ = Average number of times an OTS transaction is restarted

FIGURE 4.6-3: COMPONENTS OF PSL RESPONSE TIME FOR BASELINE

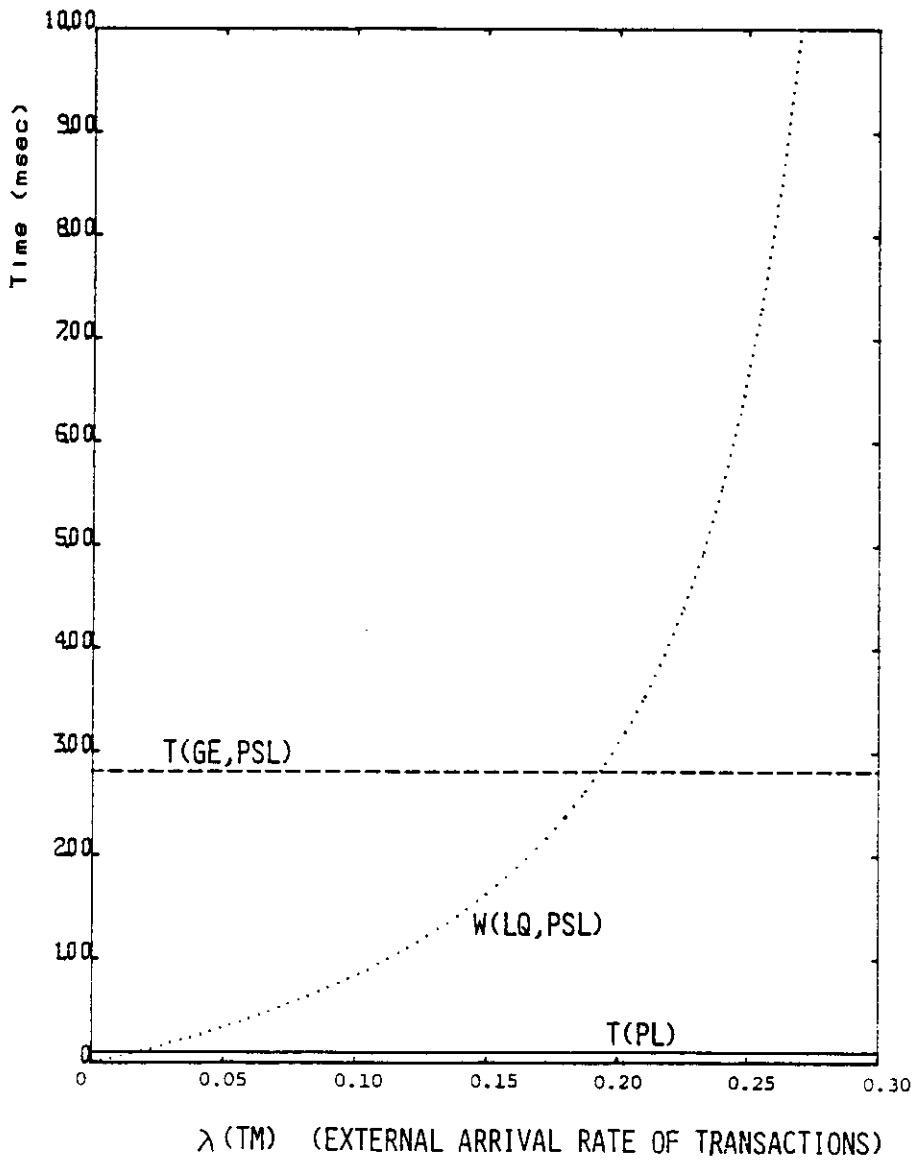


FIGURE 4.6-4: SITE AND LOCK QUEUE UTILIZATIONS FOR BASELINE

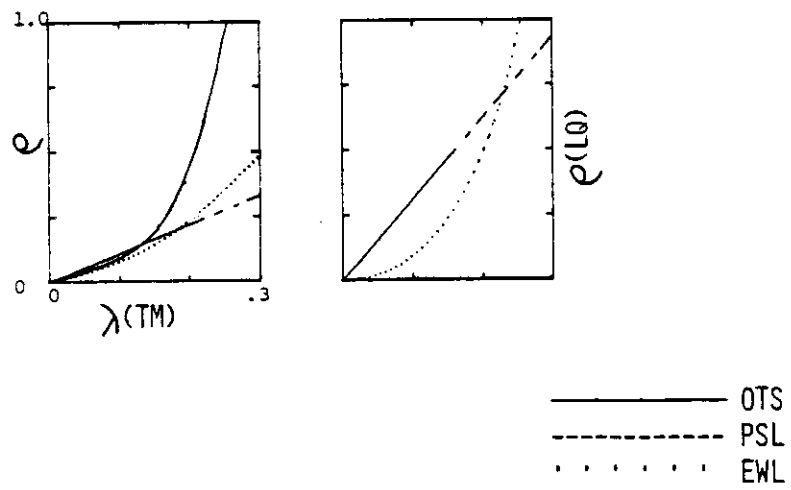


FIGURE 4.6-5: NUMBER OF TRANSACTION RESTARTS FOR BASELINE

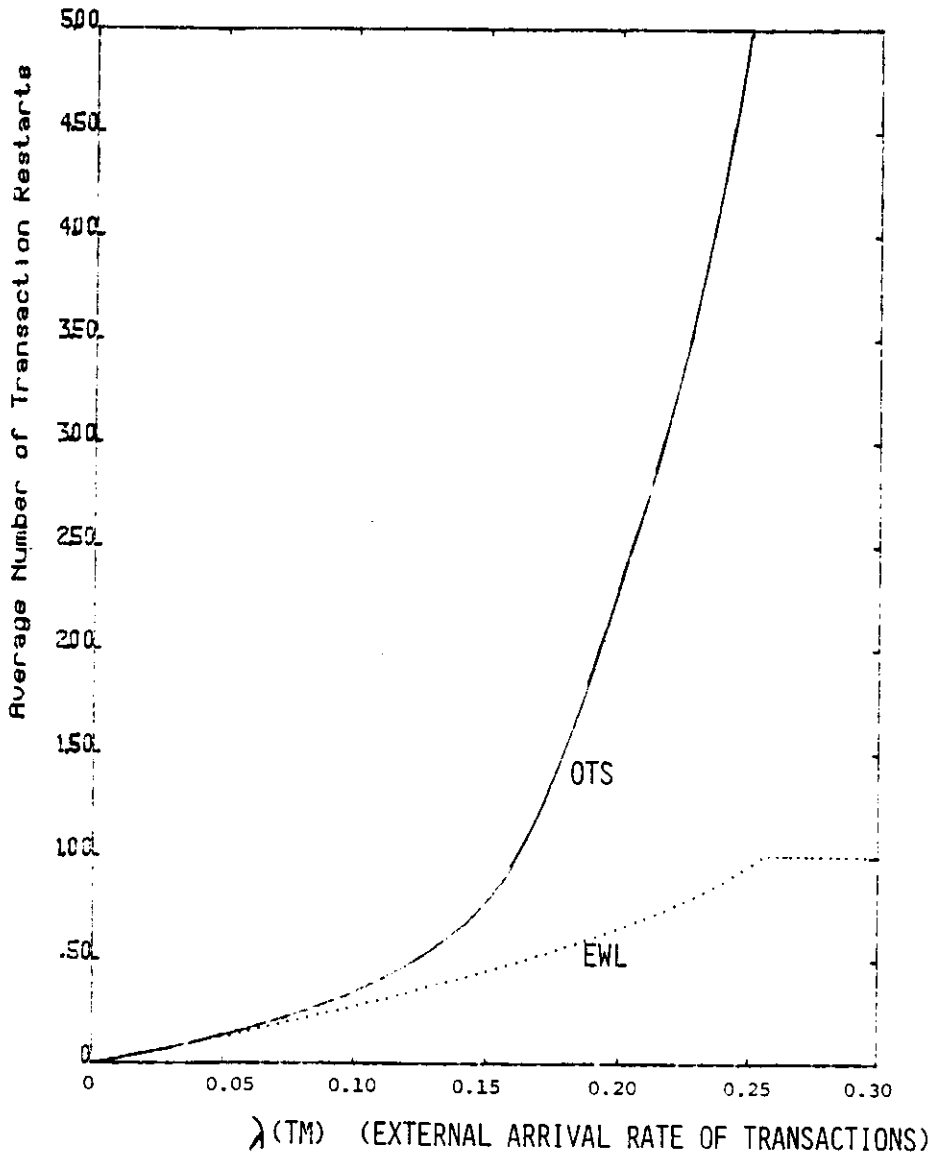
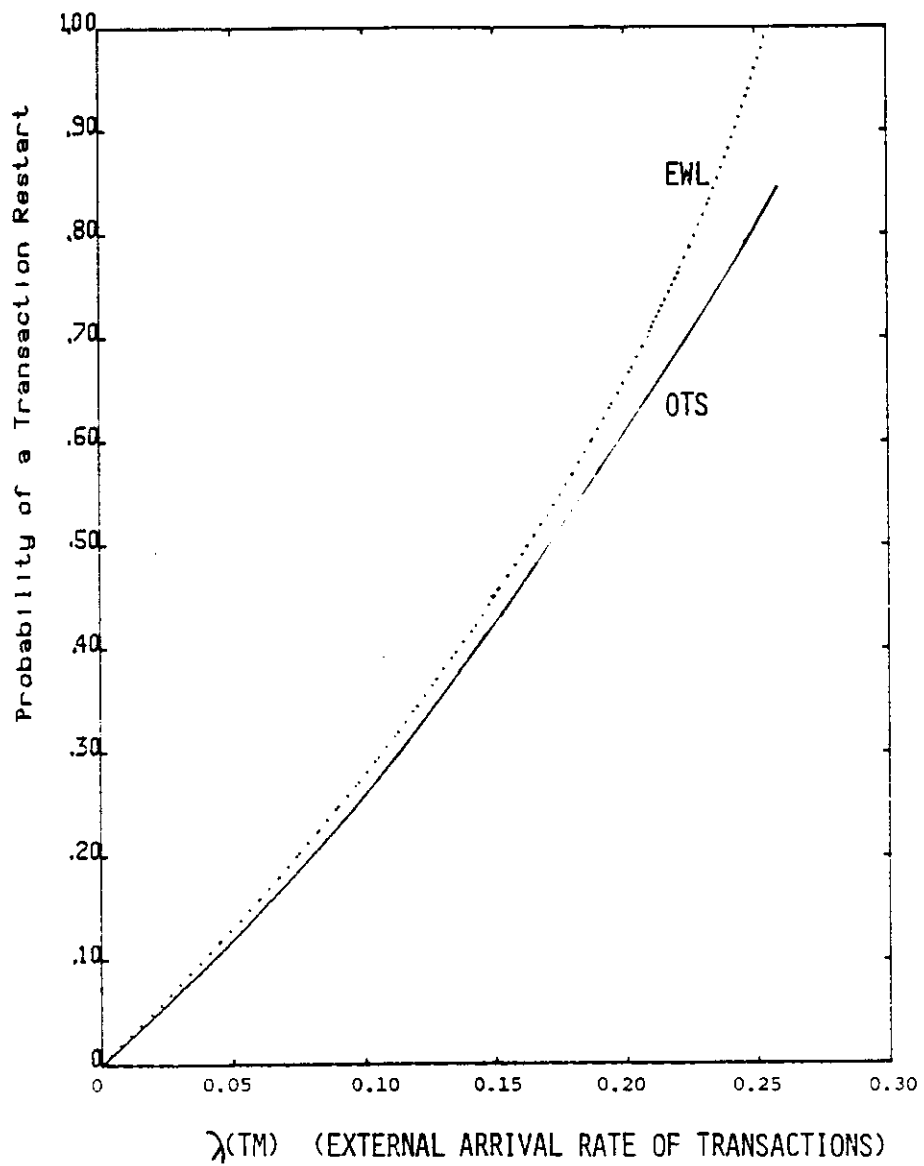


FIGURE 4.6-6: PROBABILITY OF A TRANSACTION RESTART FOR BASELINE



$q(OTS) =$ Probability of an OTS transaction being restarted

It is due to repeated transaction restarts that OTS is so sensitive to $q(OTS)$. Specifically, as $q(OTS)$ approaches 1, $N_{rd}(OTS)$ becomes unbounded (see figure 4.6-6), and the sites saturate (see ρ in figure 4.6-4).

EWL has no locking for T_E and hence no delays due to lock queue waits. Also, EWL restarts a transaction at most once if (as is true here) the transaction reads and writes the same files when it is restarted. Thus, $T_E(EWL)$ is relatively small for the entire range shown in figure 4.6-2.

Figure 4.6-7 plots T_U for the three protocols. Note that OTS has the largest T_U . To understand why, we decompose $T_U(OTS)$ into time for execution response time and update validation; the effect of restarts is considered separately. (See section 4.5 for details.)

$$\begin{aligned} T_U(OTS) = & T_E(OTS) + T(EA, \overline{RS}) + N_{rd}(OTS)T_E(OTS) \\ & + N_{rd}(OTS)T(EA, RS) \end{aligned} \tag{4.6-4}$$

where:

$T(EA, \overline{RS}) =$ Average time beginning after a transaction has executed and ending when all update-acknowledgments have been processed when the transaction is not restarted

$T(EA, RS) =$ Average time beginning after a transaction has executed and

FIGURE 4.6-7: T_U FOR BASELINE

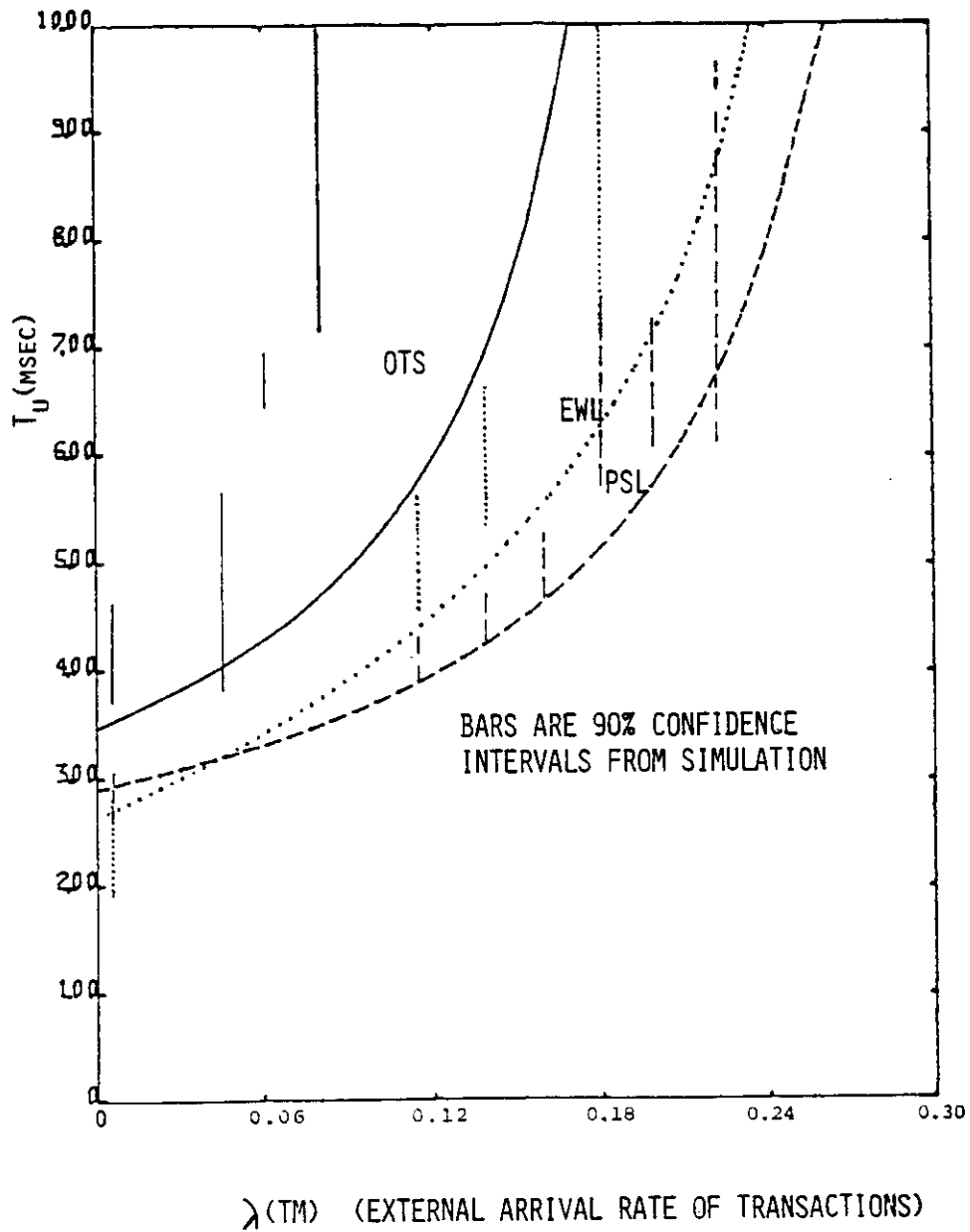
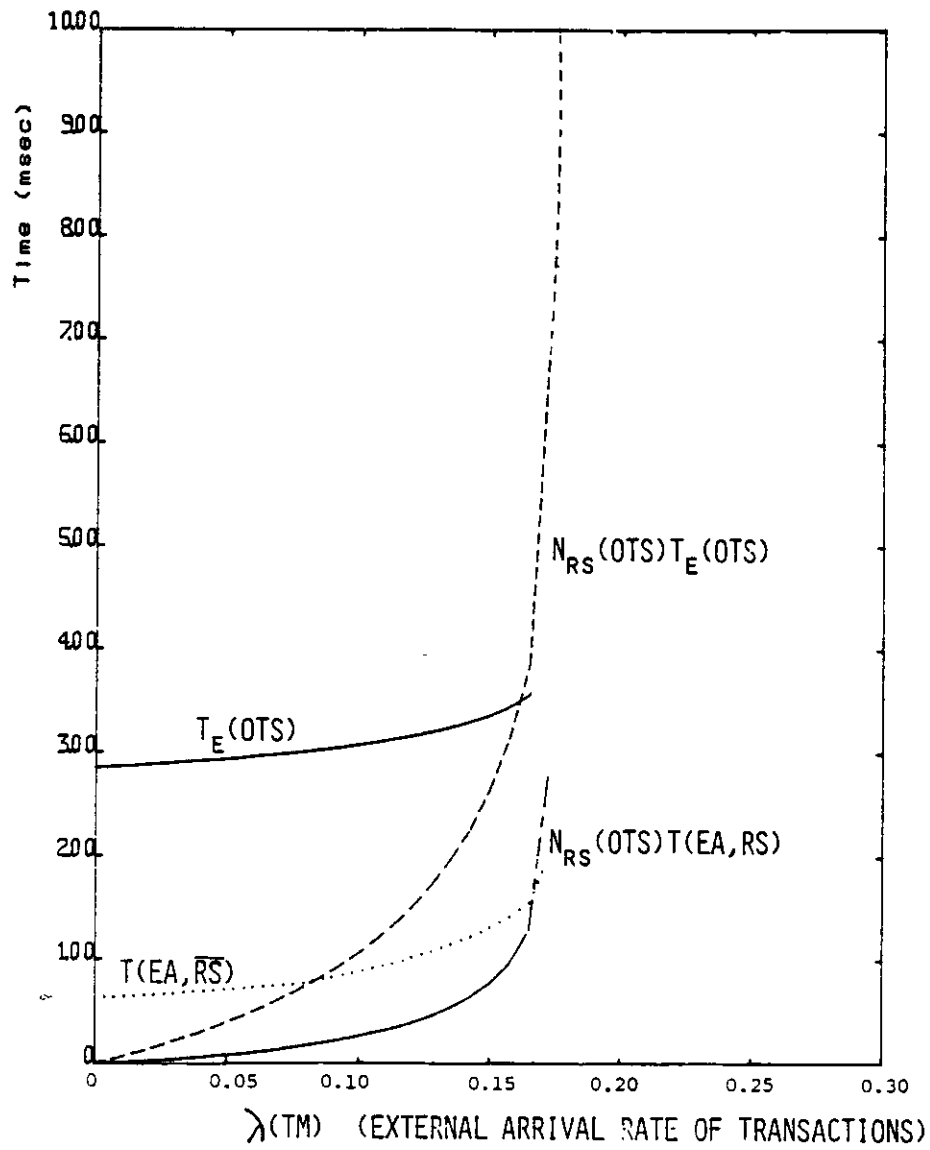


FIGURE 4.6-8: COMPONENTS OF OTS T_U FOR BASELINE



ending when all update-acknowledgments have been processed when the transaction is restarted

Under OTS, a transaction's update is not known to be confirmed until all sites have validated it. Referred to as *distributed update validation*, this approach introduces delays as indicated by $T(EA, \overline{RS})$ and $N_{ra}(OTS)T(EA, RS)$ in figure 4.6-8. In addition, when $\lambda(TM)$ increases there is a substantial cost due to repeated transaction restarts as indicated by $N_{ra}(OTS)T_E(OTS)$ and $N_{ra}(OTS)T(EA, RS)$.

Figure 4.6-7 shows a crossover point for $T_U(EWL)$ and $T_U(PSL)$. When $\lambda(TM)$ is small, restarts are rare, so EWL incurs small costs for locking; hence, $T_U(EWL) < T_U(PSL)$. However, $q(EWL)$ increases with $\lambda(TM)$ (see figure 4.6-5). When a transaction is restarted, T_U increases due to locking. Also when restarts are frequent, ρ increases, and hence site waits increase. To illustrate the foregoing, $T_U(EWL)$ is decomposed into execution response time, time from completing execution through completing update-request processing, wait in the lock queue, and time from the lock-grant through the transaction's execution. (See section 4.4 for details).

$$\begin{aligned}
 T_U(EWL) = & T_E(EWL) + T(ER) + q(EWL)W(LQ, EWL) \\
 & + q(EWL)T(GE, EWL)
 \end{aligned}
 \tag{4.6-5}$$

where:

$T(ER)$ = Average time from completing the transaction's execution until its update-request processing has been completed

Figure 4.6-9 plots the components of EWL response times. When $\lambda(TM)$ is small, $T_U(EWL)$ is dominated by $T_E(EWL)$. As $\lambda(TM)$ (and hence $q(EWL)$) increases, then $T_U(EWL)$ increases due to the cost of locking (i.e., $W(LQ,EWL)$ and $T(GE,EWL)$).

Next we consider the effect on response times of background load. The utilization of site i due to high priority background load is $\rho(H^* ;i)$ and is changed by varying $\lambda(H^* ;i)$, the arrival rate of high priority background load at site i . Similarly, the utilization of site i due to low priority background load is $\rho(L' ;i)$ and is changed by varying $\lambda(L' ;i)$, the arrival rate of low priority background load at site i . Figure 4.6-10 plots T_E for four combinations of $\rho(H^* ;i)$ and $\rho(L' ;i)$. $T_E(PSL)$ increases with background load due to increases in all three PSL response time components (i.e., $T(PL)$, $W(LQ,PSL)$, and $T(GE,PSL)$), but lock queue waits increase most dramatically. Recall that a service time in the PSL lock queue, $T(GR,PSL)$, is defined as

$T(GR,PSL)$ = Average time beginning with the lock-grant and ending with the
lock-release when PSL is used

This period includes waits for transaction executions and writing file updates. Such waits increase with background load, thereby increasing $T(GR,PSL)$. $W(LQ,PSL)$ increases with $T(GR,PSL)$ due to increases in:

1. lock queue utilizations (see figure 4.6-12)
2. average remaining service time for lock queue service, which depends on the second moment of $T(GR,PSL)$.

FIGURE 4.6-9: COMPONENTS OF EWL T_U FOR BASELINE

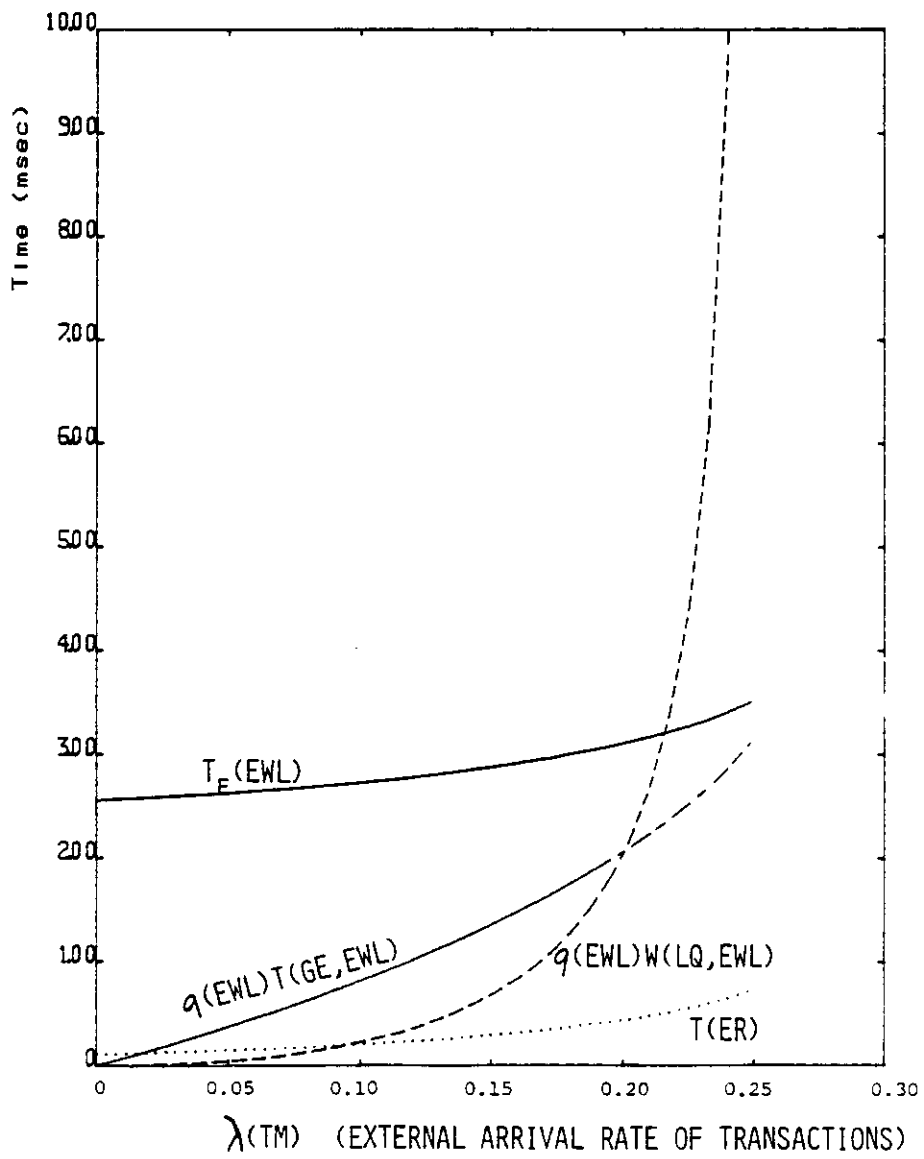


FIGURE 4.6-10: T_E FOR BACKGROUND LOAD

$(\rho L'; 0, \rho H'; 0)$

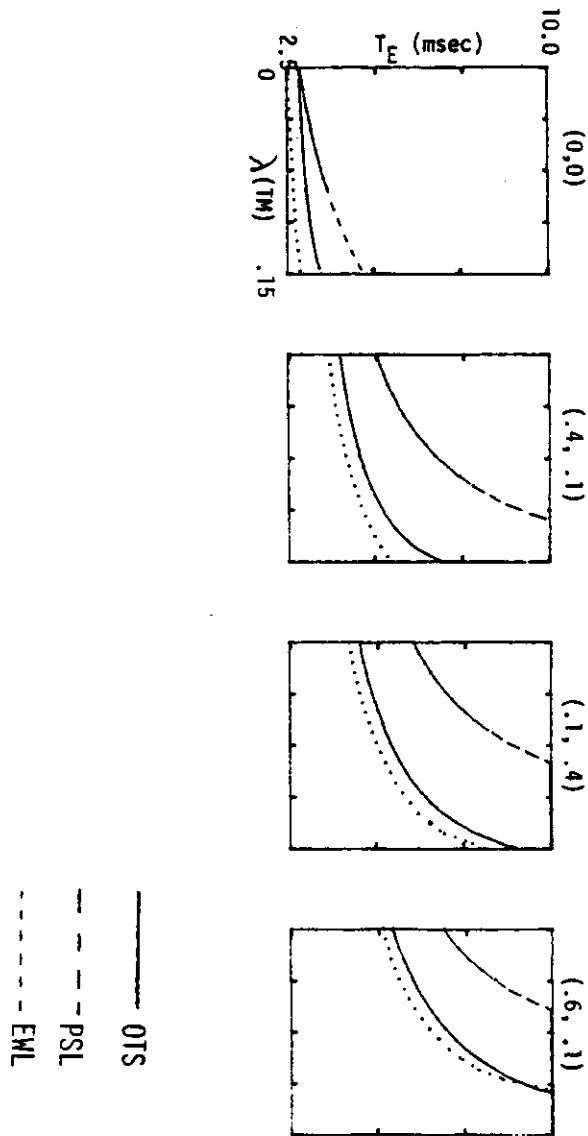


FIGURE 4.6-11: BOTTLENECK SITE UTILIZATIONS FOR BACKGROUND LOAD

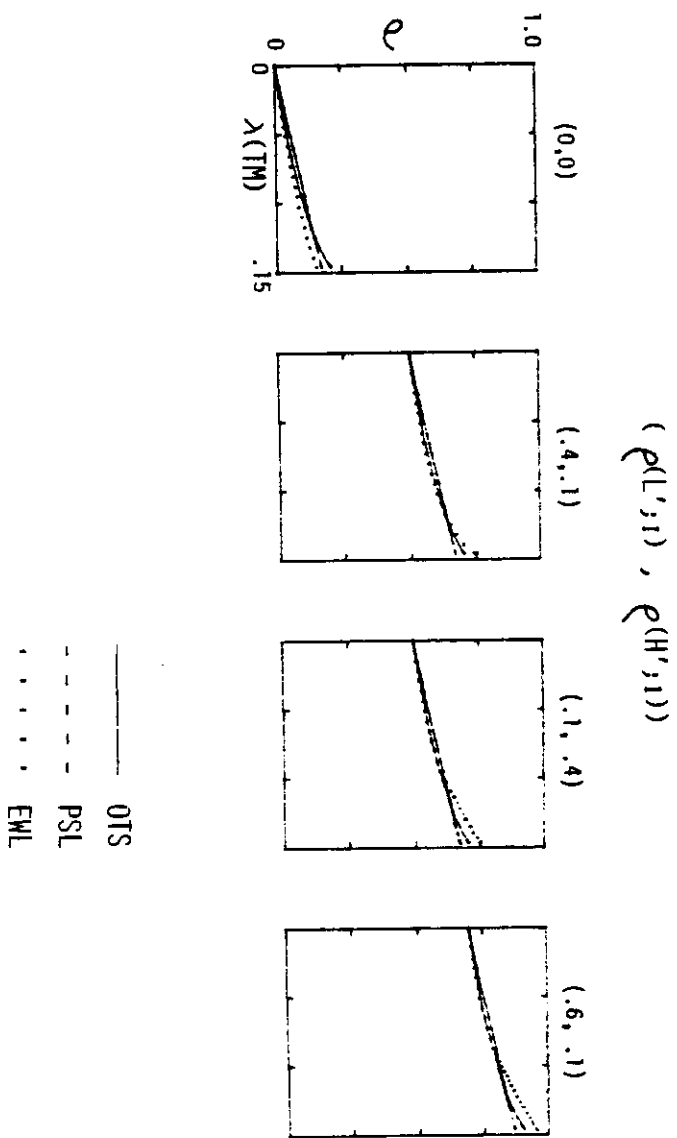


FIGURE 4.6-12: LOCK QUEUE UTILIZATIONS FOR BACKGROUND LOAD

$(\rho(L;1), \rho(H;1))$

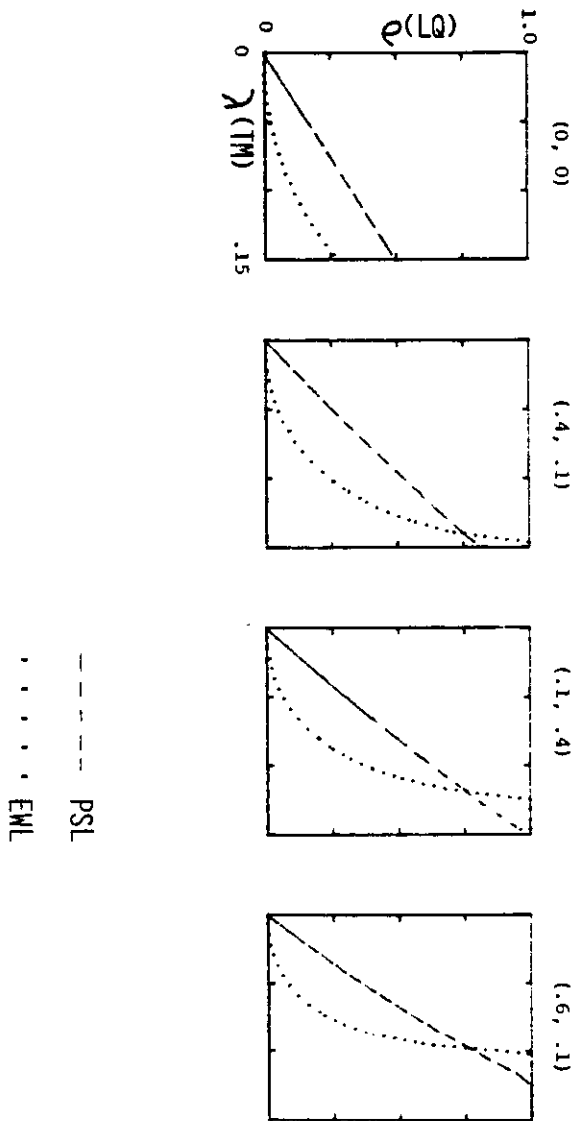
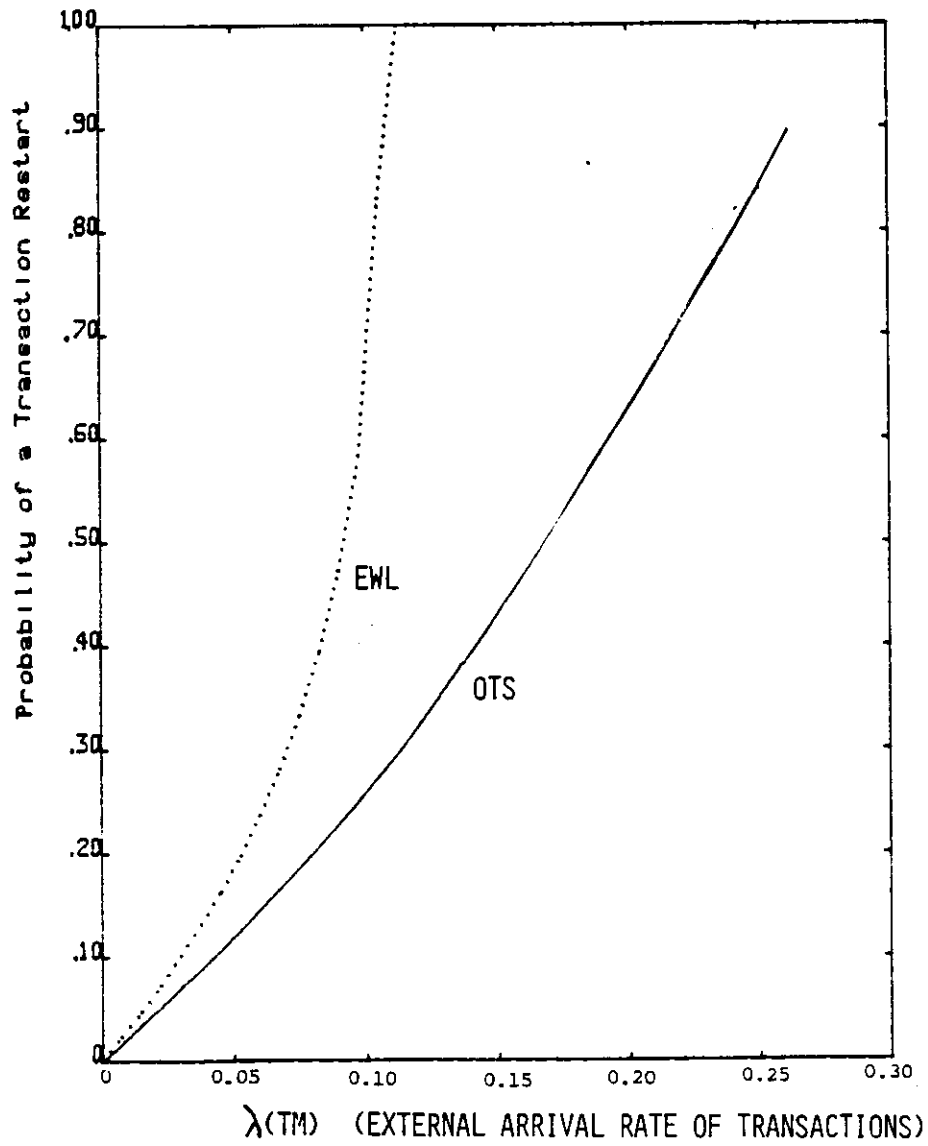


FIGURE 4.6-13: PROBABILITY OF A TRANSACTION RESTART FOR BACKGROUND LOAD (.6 , .1)



As background load increases, $T_E(OTS)$ approaches $T_E(EWL)$. As shown in figure 4.6-13, this is a consequence of $q(OTS)$ being independent of background load, but $q(EWL)$ increasing with such load due to lock queue conflicts (i.e., the lock queue was not empty when an update-request arrived). From section 4.5, the probability of a lock queue conflict is $\rho(LQ,EWL)$. Figure 4.6-12 demonstrates that $\rho(LQ,EWL)$ increases with background load.

Figure 4.6-14 plots T_U for the background load study. $T_U(OTS)$ increases with background load due to:

1. distributed update validation, which requires site waits for update checking and update acknowledgement processing
2. repeated transaction restarts, which increase the number of times site waits are required to perform update confirmation.

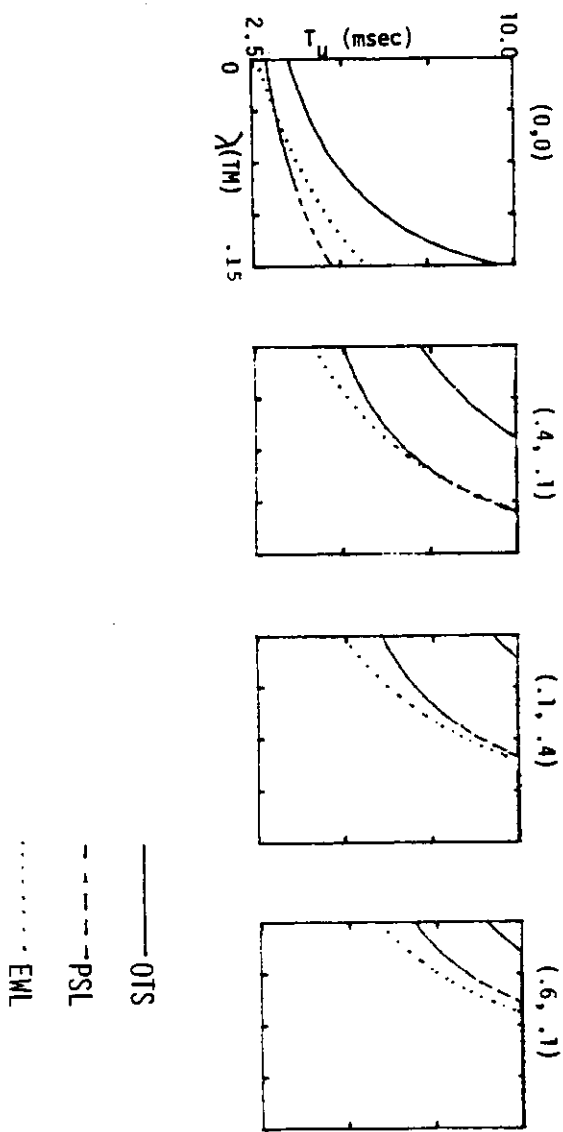
For EWL and PSL, the figure shows a shift in their crossover point: as background load increases, EWL is preferred to PSL for larger values of $\lambda(TM)$. This is a consequence of:

1. Lock queue waits grow rapidly with background load;
2. PSL always requires locking; and
3. EWL only uses locking when a conflict occurs.

However, $q(EWL)$ increases with background load, which in turn increases $T_U(EWL)$. So, in general to choose between EWL and PSL we must consider the net effect on

FIGURE 4.6-14: T_U FOR BACKGROUND LOAD

($\rho(L'; 0), \rho(H'; 0)$)



their response times when background load is changed.

Figure 4.6-15 shows that response times increase with transaction execution times ($X(TM)$). For PSL, this is primarily due to $W(LQ,PSL)$ (as evidenced by $\rho(LQ,PSL)$ in figure 4.6-16), since lock queue service times increase with $X(TM)$. For OTS, the cost and frequency of transaction restarts increases with $X(TM)$, since: (1) the period during which conflict causing transactions can arrive is lengthened; and (2) re-executing the transaction takes longer. EWL response times increase with $X(TM)$ due to both the cost of locking and the cost/frequency of transaction restarts.

We study the effect of update processing times (i.e., $X(UI)$, $X(UO)$) in figure 4.6-17. Larger update processing times increase T_E and T_U for all three protocols, since these costs are required whenever transactions execute or updates are received. However in the figure, OTS and EWL response times increase faster than those for PSL. For OTS, this is due to increasing the probability of a transaction restart, since this probability increases with $X(UO)$. EWL is affected in two ways. First, restarts increase with both $X(UO)$ and $X(UI)$, since: (1) utilizations and hence site waits increase with update processing times, which in turn increases $\rho(LQ)$ and hence lock queue conflicts; and (2) the period during which an update sequence number conflict can occur is lengthened. Secondly, under EWL writing a transaction's update at its execution site is deferred until the EW has validated the update (referred to as *deferred updated writing*). So instead of transaction's being able to modify during execution the data they write, additional update input processing is required to write the update once the EW has validated it.

FIGURE 4.6-15: RESPONSE TIMES FOR TRANSACTION EXECUTION TIMES

VALUES FOR $X(TM)$

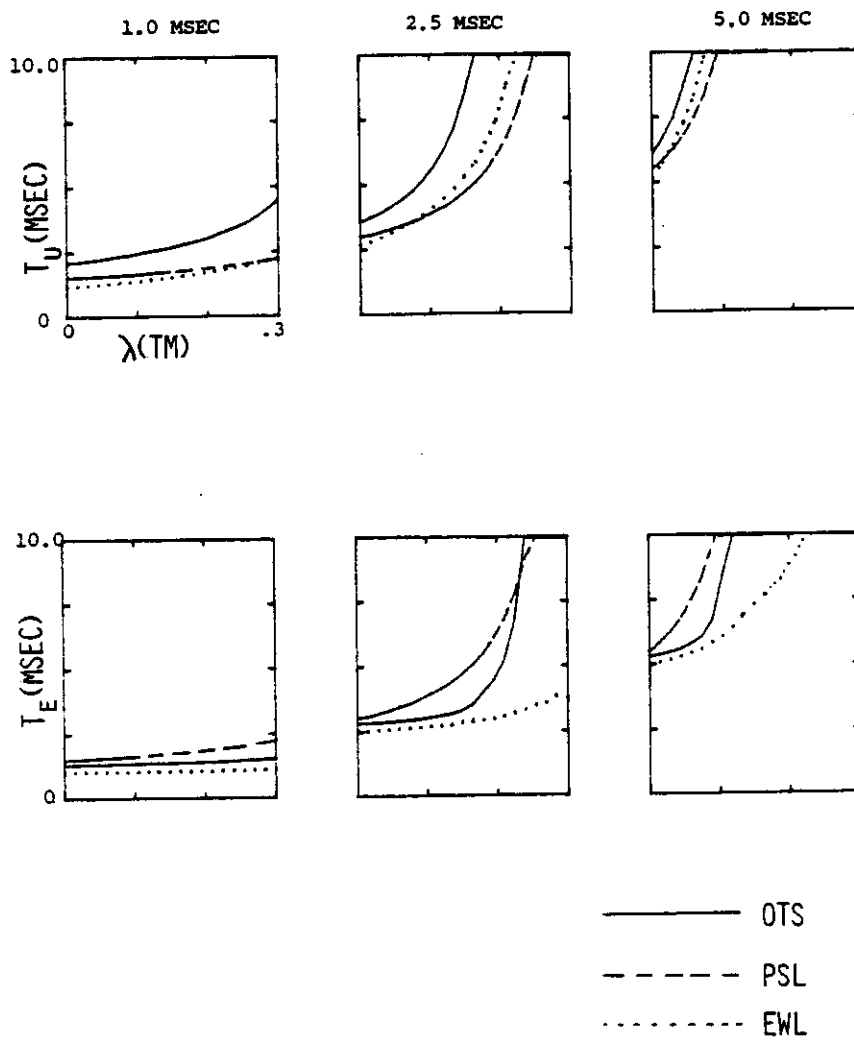


FIGURE 4.6-16: LOCK QUEUE UTILIZATIONS FOR TRANSACTION EXECUTION TIMES

VALUES FOR $X(TM)$

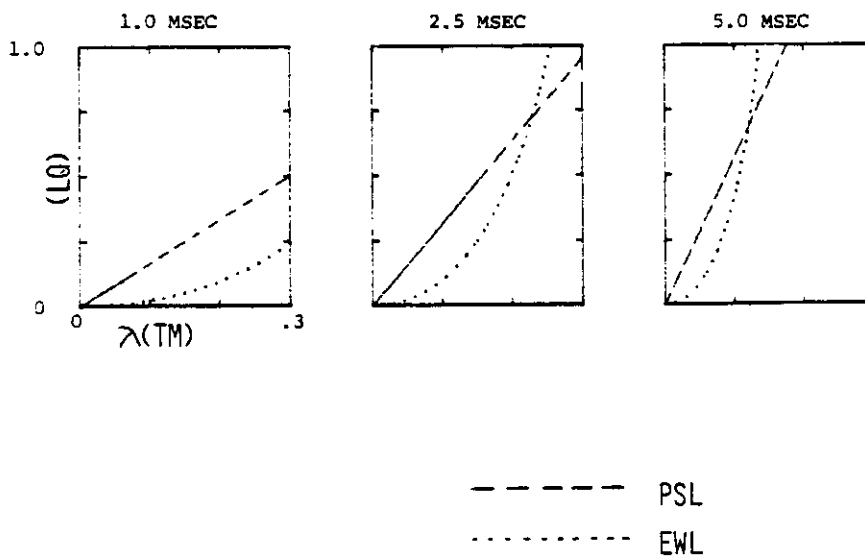


FIGURE 4.6-17: RESPONSE TIMES FOR UPDATE PROCESSING TIMES

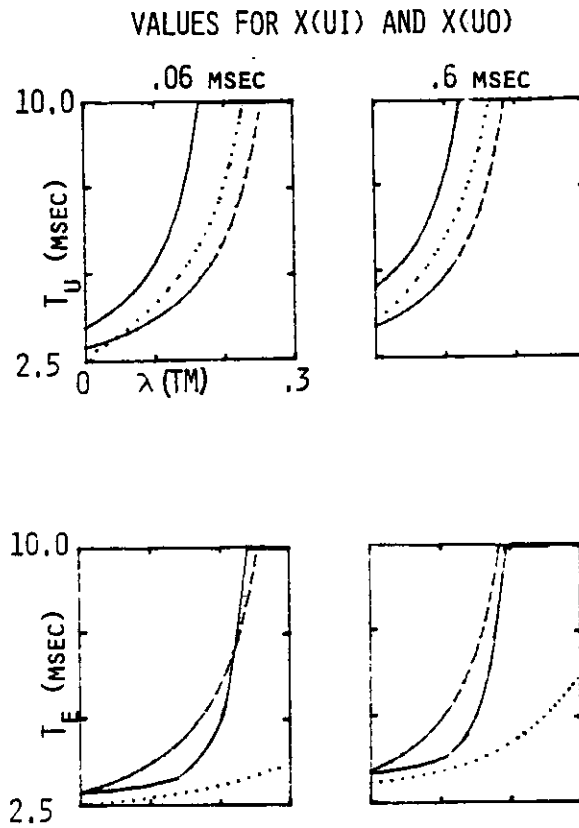


FIGURE 4.6-18: RESPONSE TIMES FOR CONTROL PROCESSING TIMES

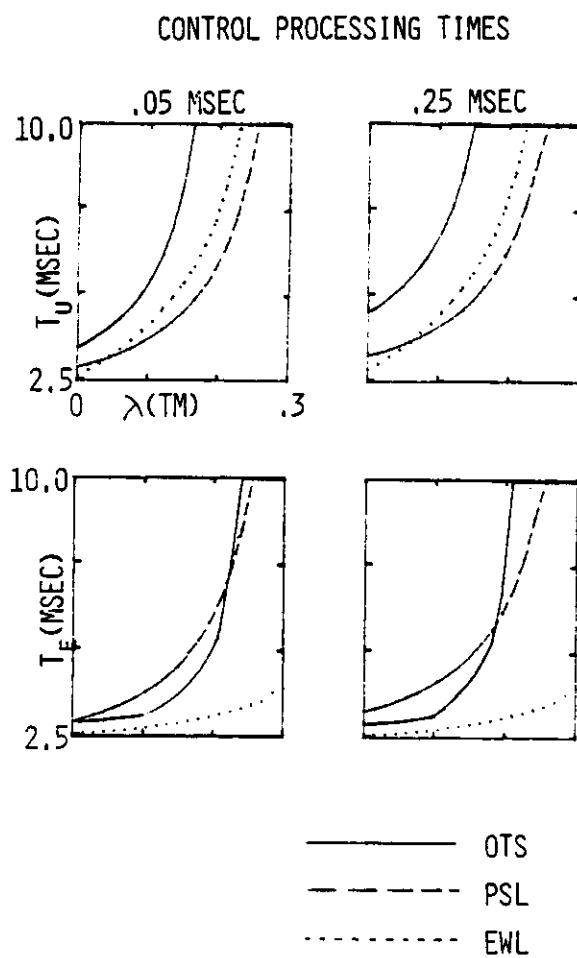


Figure 4.6-18 plots the effect on response times of control processing times (i.e., $X(LR')$, $X(LR)$, $X(UA)$, $X(UC)$, and $X(UR)$). OTS response times increase most rapidly due to distributed update validation which requires $I-1$ instances of update-acknowledgement processing and update checking for each transaction execution. (I is the number of sites.) PSL is affected more than EWL, since PSL has two instances of control processing (i.e., $X(LR')$ and $X(LR)$) while EWL has only one (i.e., $X(UR)$). However for EWL, increasing $X(UR)$ increases the probability of an update sequence number conflict due to the first execution of a transaction distributing its update (i.e., $q(EW,1)$ and $q(\overline{EW},1)$ in section 4.4). This increases the probability of a restart and hence can increase response times.

Figure 4.6-19 plots the effect on response times of network communication times for update ($X(C)$) and control ($X(C')$) messages. Response times increase with communication times for all three protocols. $T_E(OTS)$ increases since $q(OTS)$ increases with $X(C)$, which causes repeated transaction restarts that can saturate the system at lower values of $\lambda(TM)$. $T_U(OTS)$ is additionally affected by distributed update validation which requires that $(I-1)(N_{ra}(OTS)+1)$ update messages be sent and the same number of update acknowledgements be received before the update is confirmed. PSL response times increase due to PSL's requiring two message transmissions (i.e., lock-request and lock-grant) before update confirmation is achieved. EWL only requires one message transmission (i.e., update-request) if the transaction is not restarted; if it is restarted, a lock-grant is also needed. However, for EWL both update sequence number and lock queue conflicts increase with communication times.

FIGURE 4.6-19: RESPONSE TIMES FOR NETWORK COMMUNICATION TIMES

VALUES FOR $X(C)$ AND $X(C')$

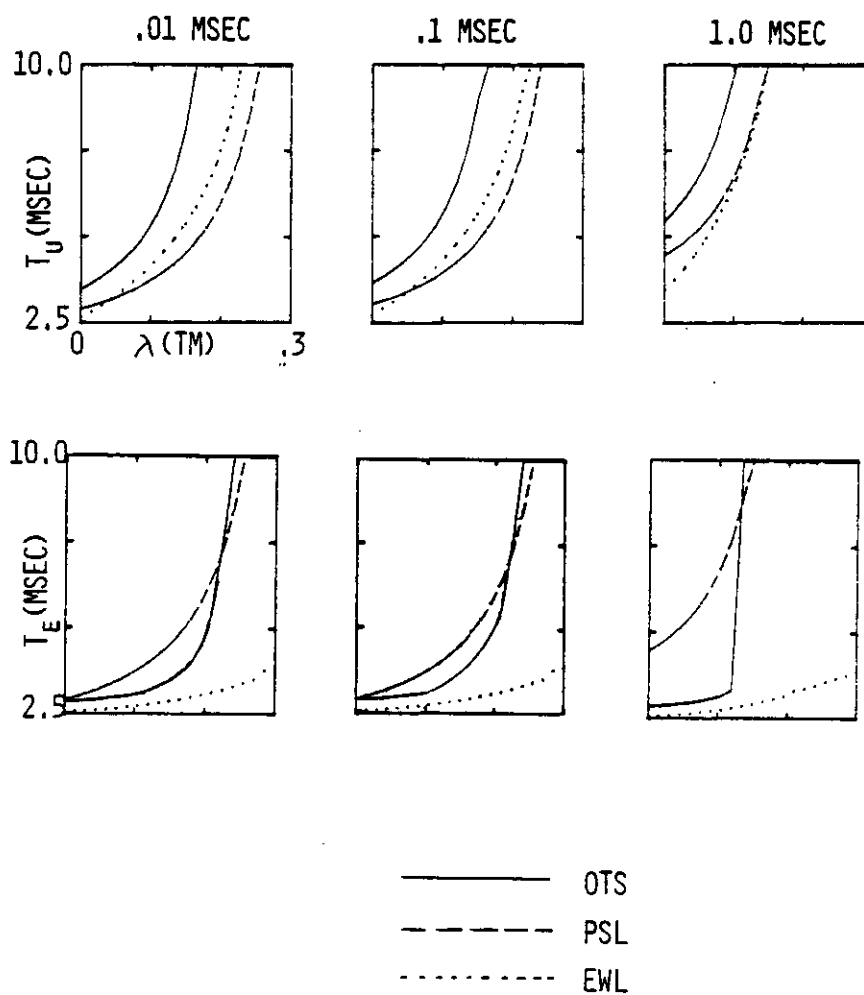


FIGURE 4.6-20: RESPONSE TIMES FOR LOCK-GRANT
AND LOCK-RELEASE TIMES
VALUES FOR X(LG) AND X(LE)

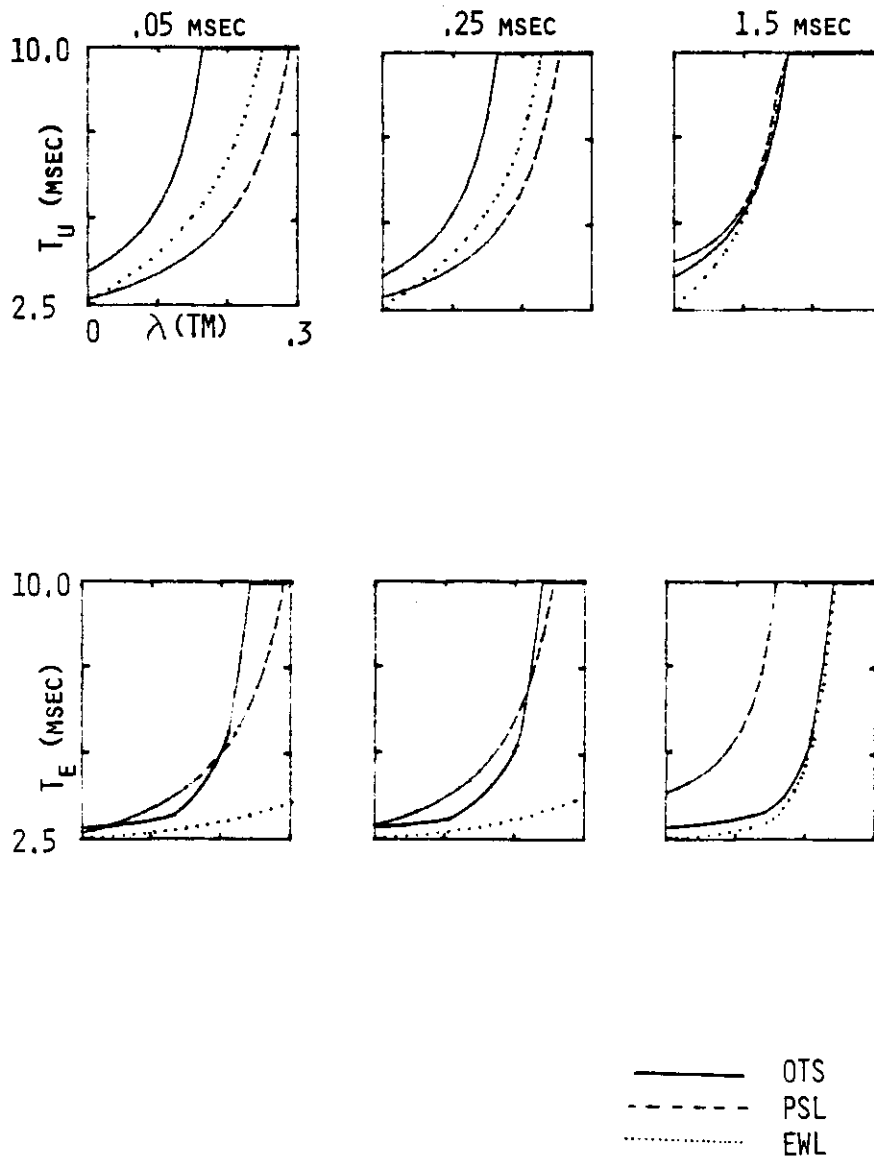


Figure 4.6-20 shows the impact on response times of lock-grant ($X(LG)$) and lock-release ($X(LE)$) processing times. Since OTS does not use locking, its response times are unaffected. Response times for both PSL and EWL increase with lock queue processing times. In the figure, PSL response times increase more quickly than those for EWL since all PSL transactions have lock queue costs, but only restarted EWL transactions incur them. However, $q(EWL)$ increases with lock queue costs, since lock queue conflicts ($\rho(LQ,EWL)$) increase with these costs. Thus, the choice between EWL and PSL depends on the relative effect on their response times of increasing $X(LG)$ and $X(LE)$.

Figure 4.6-21 indicates the impact of update log processing (i.e., $X(UM)$ and $X(DR)$) on response times. Only OTS is affected since EWL and PSL do not use update logs or have database rollbacks. OTS is affected in two ways. First utilizations increase due to update log processing, which in turn increases site waits and hence response times. Second, $q(OTS)$ increases with $X(UM)$ by lengthening the period during which a conflict causing transaction could execute. If $X(UM)$ and $X(DR)$ are very small (e.g., 0), then OTS compares favorably to EWL, when $\lambda(TM)$ is small. However, if these costs are large, OTS response times are very large.

Figure 4.6-22 plots the effect on response times of the number of sites (I). For PSL, increasing I can increase response times if the fraction of primary site transactions decreases, since such transactions have smaller lock queue service times. However, PSL response times decrease with I if delays at the primary site are reduced. $T_E(EWL)$ and $T_E(OTS)$ decrease as I increases, since ρ decreases due to partitioning the transaction

FIGURE 4.6-21: RESPONSE TIMES FOR UPDATE LOG PROCESSING
 VALUES FOR X(UM) AND X(DR)

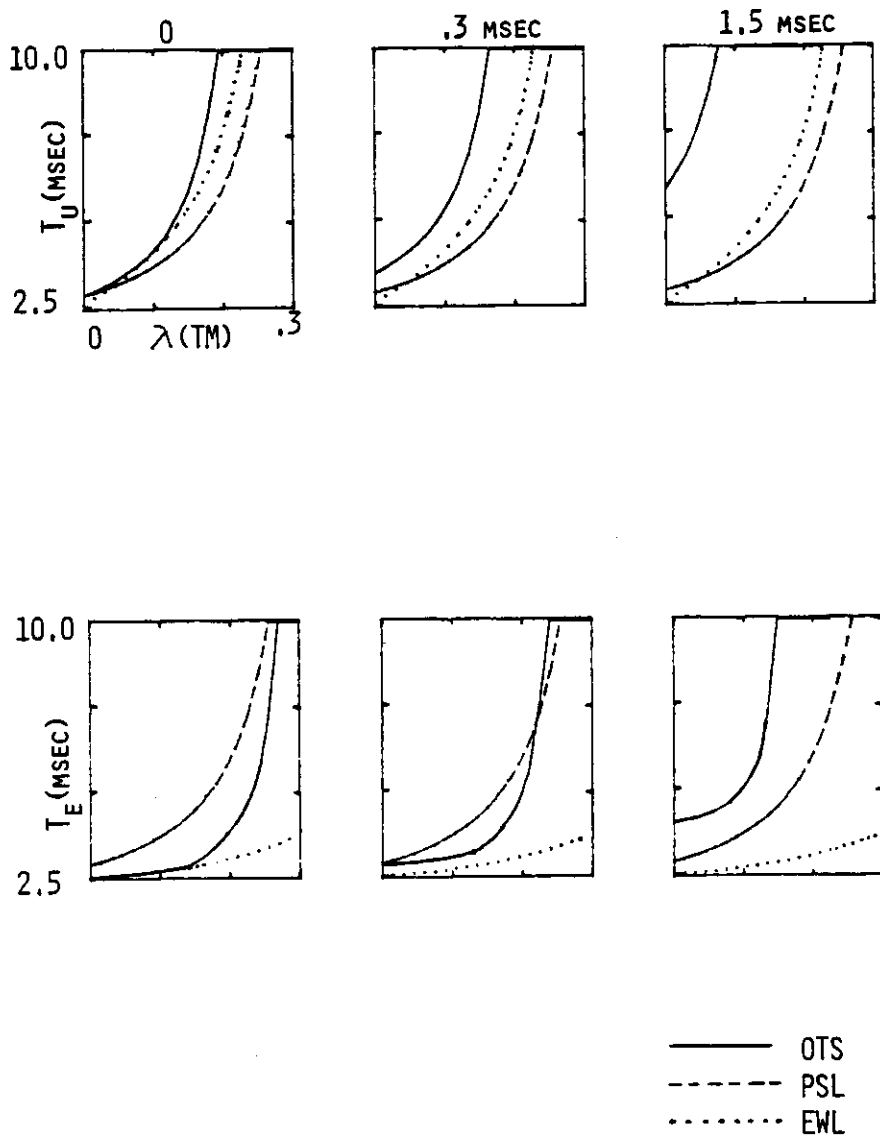
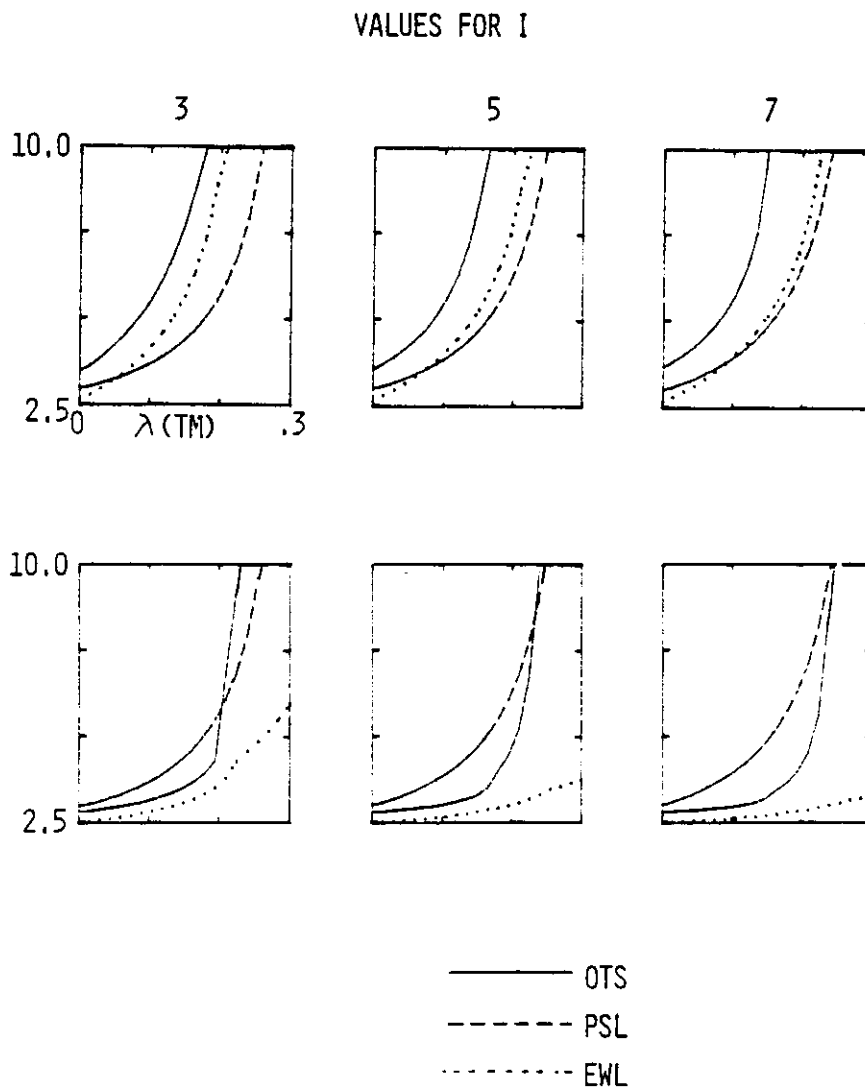


FIGURE 4.6-22: RESPONSE TIMES FOR NUMBER OF SITES



load among more sites. However, $T_I(OTS)$ increases with I due to

1. distributed update validation which increases control processing in proportion with I ; and
2. repeated transaction restarts which increase when it is less frequent that transactions execute at the same site as the last transaction whose update was accepted (see section 4.5).

Increasing I affects $q(EWL)$ in two opposing ways:

1. Site waits decrease, which decreases $\rho(LQ, EWL)$ and hence $q(EWL)$.
2. The frequency of non-EW site transactions increases which increases update sequence number conflicts and hence $q(EWL)$.

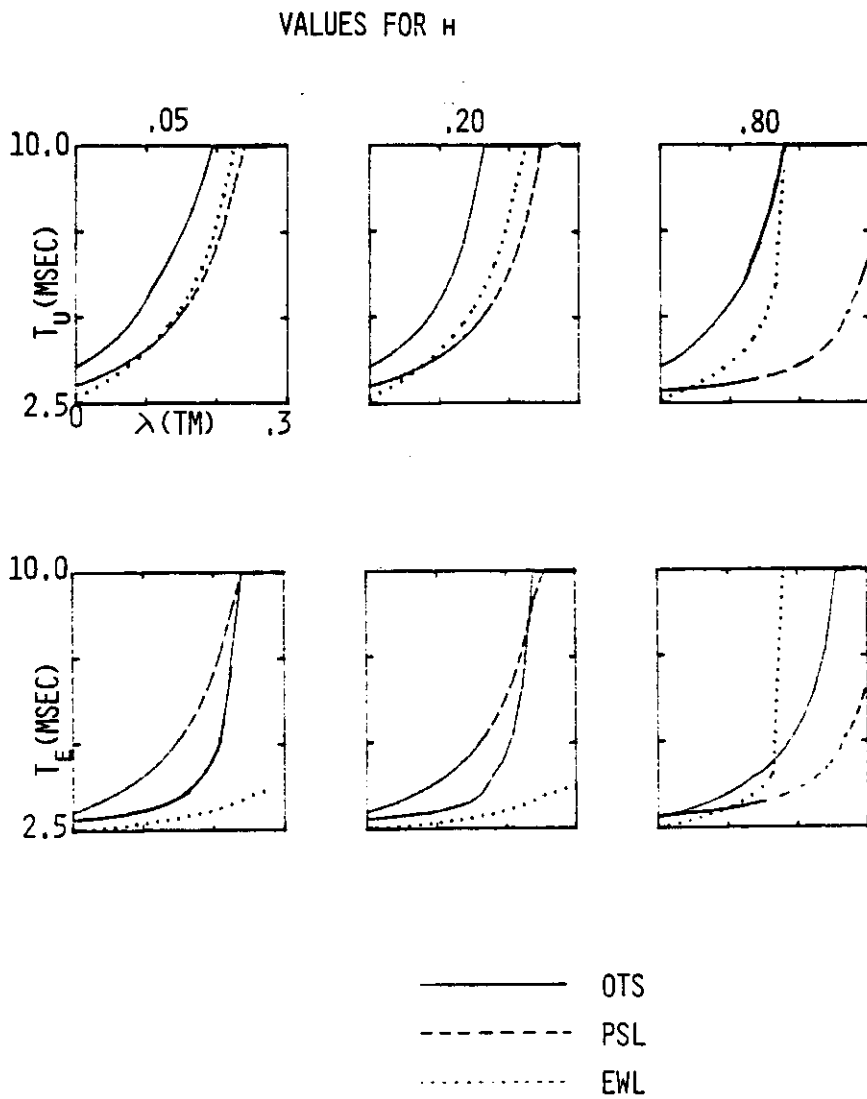
The issue of partitioning transaction load is complex. Here, we consider a situation in which there is a "designated" site. We define the fraction of external transaction arrivals processed by the designated site as h ,

$h =$ Probability of a transaction executing at the designated site

The remaining external transaction arrivals are equally split among the non-designated sites. For EWL, the designated site is the EW site; for PSL, it is the primary site.

Figure 4.6-23 plots the effect on response times of varying h . When $h \neq \frac{1}{I}$, response times can increase due to the bottleneck site being more heavily utilized. However, OTS response times decrease due to reducing the frequency of transaction restarts. By having a small value for h , the number of sites is effectively reduced by one, and hence

FIGURE 4.6-23: RESPONSE TIMES FOR PARTITIONING OF TRANSACTION LOAD



it is more probable that transactions execute at the same site as the last transaction whose update was accepted. By having a large value for h , the number of sites effectively becomes one; when h approaches 1, $q(OTS)$ goes to 0. PSL response times decrease as h grows, since primary site transactions become more frequent, thereby decreasing all components of PSL response times. EWL is affected in two opposing ways. Increasing h can

1. increase EWL response times due to increasing lock queue conflicts due to longer waits at the primary site (i.e., $\rho(LQ, EWL)$ becomes larger as h increases).
2. decrease EWL response times since
 - a. locking costs due to inter-site communication are decreased; and
 - b. increasing h increases the frequency of EW site transactions, and update sequence number conflicts do not occur for EW site transactions.

In this section, we provided insight into how PSL, OTS, and EWL are affected by the input parameters. Figure 4.6-24 summarizes the effect of input parameters on protocol response times. Our studies indicate that the key key costs affecting the response times of these protocols are:

OTS

1. Repeated transaction restarts
2. Update log maintenance and database rollbacks

3. Distributed update validation

PSL

1. Locking

EWL

1. Transaction restarts
2. Deferred update writing
3. Locking

4.7 DISCUSSION

We were motivated to develop EWP and EWL because of performance problems with existing consistency control protocols. Approaches to consistency control protocols can be classified depending on whether checking for conflicting file accesses is done before (early checking) or after (late checking) transactions reference shared files. Early checking protocols have inter-computer synchronization delays for a transaction's execution response time (T_E). Existing late checking protocols repeatedly restart a transaction until it executes without conflict. So when conflicts are frequent, late checking protocols have: (1) long delays for update confirmation response time (T_U) and saturate the computers and interconnection network.

FIGURE 4.6-24: EFFECT ON RESPONSE TIMES OF INCREASING INPUT PARAMETER VALUES

INPUT PARAMETERS	PROTOCOL RESPONSE TIMES		
	OTS	PSL	EWL
$\lambda(TM)$	+	+	+
$\lambda(H';I)$, $\lambda(L';I)$	+	+	+
$X(TM)$	+	+	+
$X(UI)$, $X(UO)$	+	+	+
$X(UA)$, $X(UC)$, $X(LR')$, $X(LR)$, $X(UR)$	+	+	+
$X(C)$, $X(C')$	+	+	+
$X(LG)$, $X(LE)$	0	+	+
$X(UM)$, $X(DR)$	+	0	0
I	+ , -	+ , -	+ , -
H	+ , -	+ , -	+ , -

- = DECREASES RESPONSE TIME
- 0 = DOES NOT AFFECT RESPONSE TIME
- + = INCREASES RESPONSE TIME
- + , - = SOME RESPONSE TIME COMPONENTS INCREASE
OTHERS DECREASE

PSL is an example of an early checking protocol. Transactions must acquire permission from a primary site before accessing a shared file. Thus, T_E includes inter-computer synchronization delays in the form of lock queue waits, which increase rapidly with transaction arrival rate and background load.

For some real time applications, updates need not be finalized before they are used by time critical functions. In such cases, late checking protocols have much appeal. OTS is a late checking protocol, since no inter-computer message need be exchanged before a transaction executes (hence there is no inter-computer synchronization delay for T_E). However, to preserve consistency in case of conflicting file updates, OTS requires that an update log be maintained. Additionally, when conflicts do occur, database rollbacks are required. Thus, OTS response times increase with these costs. Furthermore, OTS update confirmation response times increase with the degree of file replication due to inter-computer synchronization delays for distributed update validation (i.e., all sites with a copy of F_k must validate each update to F_k). Finally, OTS is very sensitive to the transaction arrival rate. When transactions arrive too frequently, there is a higher probability of a conflict and hence a restart. Since OTS repeatedly restarts transactions, each conflict creates one or more new (restarted) transactions, which further increase the transaction arrival rate. Repeated transaction restarts can saturate the computers and the interconnection network.

EWL is a late checking protocol, so it has no inter-computer synchronization delays for T_E . EWL avoids the delays caused by distributed update validation by designating a single exclusive-writer for each shared file. A file's EW controls update validation and distribution for the file. Because updates are not written until they are validated, EWL avoids the cost of update log maintenance and database rollbacks. Additionally, EWL guarantees that a transaction is restarted at most once if the files it reads and writes do not change. Thus, for a wide range of parameters, $T_E(EWL)$ is lower than $T_E(OTS)$ or $T_E(PSL)$. When the real time constraint depends on T_U , our studies suggest that OTS is undesirable for the abovementioned reasons. We have identified the following conditions under which EWL is superior to PSL for T_U :

1. The size and update processing times for updates are smaller so as to reduce:
(a) the effect of not writing updates until the EW has validated them, and (b) the communication cost of including the proposed update in the update-request message.
2. The cost and/or frequency of transaction restarts is lower, since EWL has restarts and PSL does not. The cost of transaction restarts increases with input parameters such as background load, transaction execution time, and update processing times. The frequency of restarts increases with these parameters (because of update sequence number and lock queue conflicts) as well as transaction arrival rate (which increases the probability of conflicting transactions executing concurrently).

3. The cost of locking is higher, since PSL incurs these costs for all transactions but EWL only incurs them for transactions that are restarted. Locking costs increase with input parameters such as the time for processing lock-grants and lock-releases as well as background load (which lengthens lock queue service times and hence lock queue waits).

Otherwise, PSL is preferred.

As to EWP, its response times and utilizations are always lower than EWL's. However, like EWL, EWP costs increase with update size and update processing times.

We conclude by considering the impact on our studies if files reside on disk rather than in RAM. Doing so has two implications. First, disk access takes significantly longer than access to RAM. Since update log maintenance and database rollbacks would require disk accesses, OTS response times would increase. Secondly, to optimize disk utilization, existing disk based systems use multiprogramming in which program executions are preempted by disk interrupts. Multiprogramming increases the cost for scheduling transaction executions (since the scheduler becomes more complicated), thereby making restarts more costly. However once a transaction has been preempted, other high priority processing could be performed, such as conflict detection. Thus, transactions that lose a conflict could be aborted prior to finishing their execution, thereby reducing the cost of conflicts. So, a disk based system affects the cost of conflicts in two opposing ways:

1. Conflicts become more costly due to increased scheduling costs.

2. Conflicts become less costly since transactions that lose a conflict can be aborted before completing their execution.

If the net effect of a disk based system is to reduce the cost of conflicts, EWL would be more desirable; if a disk based system increases the cost of conflicts, PSL would be more desirable.

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH

The motivation for this research was to find approaches to consistency control appropriate for real time distributed processing systems. We found existing consistency control protocols unappealing since either: (1) they have inter-computer synchronization delays for execution response times (T_E); or (2) when conflicts are frequent, they have repeated transaction restarts which result in long delays for update confirmation response times (T_U) and can saturate the computers and interconnection network. Our contributions lie in two areas. First, we presented *two new protocols that have no inter-computer synchronization delays for T_E and avoid repeated transaction restarts*: the exclusive-writer protocol (EWP) and the exclusive-writer protocol with a locking option (EWL). EWP is simple to implement, has no database rollbacks, and no transaction restarts. Since it does not use locking, EWP avoids deadlocks due to shared data access. EWP has no inter-computer synchronization delays for T_E and its intercomputer synchronization delays for T_U are small, even when conflicts are frequent. However, EWP ensures only a limited form of serializability, since it discards update-requests that lose a conflict.

EWL is a fully serializable extension to EWP in which transactions that lose a conflict are restated under primary site locking (PSL). EWL has no database rollbacks.

Also, EWL restarts a transaction at most once if the files the transaction reads and writes do not change when it is restarted. Performance can be further improved by dynamically switching to primary site locking (PSL) when conflicts are frequent, since PSL has no transaction restarts due to conflicting file accesses. Doing so requires no additional messages or inter-computer synchronization delays. EWL has no inter-computer synchronization delays for T_E . For T_U , these delays are similar to EWP's when there is no conflict, and like PSL's if the transaction loses a conflict.

A second area of contribution is our response times studies of PSL, optimistic timestamps (OTS), and EWL. While PSL and OTS response times have been studied previously (e.g., [RIES78] and [LIN83]), only simulation techniques were used. In chapter 4, we developed analytical models for PSL, OTS, and EWL. From the models, we conclude that OTS is undesirable since: (1) it uses distributed update validation which causes long inter-computer synchronization delays for T_U unless the degree of file replication is low; (2) it requires maintaining an update log which creates additional processing overhead; and (3) it has repeated transaction restarts which saturate the sites and the interconnection network when conflicts are frequent.

Additionally, our models provide insight into the parameters to consider for dynamically switching between EWL and PSL. EWL is preferred to PSL for T_E , since PSL has inter-computer synchronization delays for T_E , but EWL does not. EWL's T_U is lower than PSL's when:

1. Updates and/or update processing times are smaller, since EWL

- a. includes the proposed update in the update-request message
 - b. incurs additional update processing overhead due to sites not finalizing a file's updates until it has been validated by the file's EW;
2. The cost and/or frequency of transaction restarts is lower, since EWL has restarts but PSL does not; and
 3. The cost of locking is higher, since PSL always requires locking but EWL requires it only if a transaction is restarted.

Otherwise, PSL is superior to EWL.

There are several research topics created by this dissertation: (1) selecting a policy for dynamically switching between EWL and PSL; (2) making EWP and EWL resilient to hardware failures; and (3) incorporating consistency control protocols into a general model for task response times. While we have indicated the key parameters to consider for dynamically switching between EWL and PSL, we have not discussed an actual policy for performing such switching. One approach is to have sites use information on conflicts incurred by their transactions (indicated by the receipt of lock-grant messages). Over a period of time t_1 , sites could count the number of conflicts that occur for each file. If this count at site i for file F_k is too large, then transactions at this site could use PSL to access F_k for the next t_2 seconds, after which EWL would again be used. Alternatively, PSL could be used until F_k 's primary/exclusive-writer site broadcasts an update with an indicator that the lock queue is empty.

In our presentation of EWP and EWL, we assumed that computers did not fail and that the interconnection network was reliable. If these assumptions are not valid, then EWP and EWL need to be made fault tolerant. One approach is to designate for each EW site another site which is its backup. The backup would be responsible for detecting EW failures as well as recovering from them. To make EWL fault tolerant, the fault tolerant EWP would be merged with an existing fault tolerant PSL protocol (e.g., [WALK83]).

In real time systems, time critical functions (or tasks) often consist of several transactions which are executed according to a control flow graph. Such graphs can include forks, joins, and branching. The response time models of chapter 4 only considered transaction response times, not the response time of the entire task. Our models could be used as part of a more elaborate algorithm for computing task response times that include consistency control.

REFERENCES

- BERN81 Bernstein, Philip A., James B. Rothnie, Nathan Goodman, and Christos A. Papadimitriou. "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)," *IEEE Transactions on Computers*, Vol. SE-4, NO. 3, May 1978, pp. 154-168.
- BERN78 Bernstein, Philip A. and Nathan Goodman. "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, Vol. 13, No. 2, June 1981, pp. 185-222.
- BERN82 Bernstein, Philip A. and Nathan Goodman. "A sophisticate's introduction to distributed database concurrency control," *Proceedings of the Eighth International Conference on Very Large Data Bases*, September 8-10, 1982, pp. 62-76.
- BERR82 Berry, Robert, K. Mani Chandy, Jay Misra, and Doug Neuse. *PAWS 2.0 Performance Analyst's Workbench System User Manual*, Information Research Associates, 1982.
- CERI82 Ceri, Stefano and Susan Owicki. "On the Use of Optimistic Methods for Concurrency Control in Distributed Databases," *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks*, February 16-19, 1982, pp. 117-129.
- CHU80 Chu, Wesley W., Leslie J. Holloway, Min-Tsung Lan, and Kemal Efe, "Task allocation in distributed data processing," *IEEE Computer*, November 1980, pp. 57-69.
- DANT80 Dantas, Joao. "Performance Analysis of Distributed Database Systems," Ph.D. Dissertation, UCLA Computer Science Dept., 1980.
- ESWA76 Eswaran, K.P., J.N. Gray, R.A. Lorie, and I.L. Traiger, "The notions of consistency and predicate locks in a database system," *Commun. Ass. Comput. Mach.*, Vol. 19, November 1976.

- GALL82 Galler, Bruce. "Concurrency control performance issues," University of Toronto Technical Report CSRG-147, September 1982.
- GARC78 Garcia-Molina, Hector. "Performance comparison of two update algorithms for distributed databases," *Third Berkeley Workshop on Distributed Data Management and Computer Networks*, August 29-31, 1978, pp. 108-119.
- GREE80 Green, Michael L. et al., "A Distributed Real Time Operating System," *Proceedings of the Symposium on Distributed Data Acquisition, Computing, and Control*, December 1980.
- KLEI75 Kleinrock, Leonard. *Queuing Systems Volume I: Theory*, John Wiley & Sons, 1975
- KLEI76 Kleinrock, Leonard. *Queuing Systems Volume II: Computer Applications*, John Wiley & Sons, 1976
- KOHL81 Kohler, Walter. "A survey of techniques for synchronization and recovery in decentralized computer systems," *ACM Computing Surveys*, Vol. 13, No. 2, June 1981, pp. 149-185.
- LAMP78 Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system," *Commun. Ass. Comput. Mach.*, Vol. 7, July 1978, pp. 558-565.
- LEE80 Lee, Chin-Hwa. "Queueing analysis of global locking synchronization schemes for multicopy databases," *IEEE Transactions on Computers*, Vol. c-29, No. 5, May 1980, pp. 371-384.
- LIN81 Lin, Wen-Te. "Performance evaluation of two concurrency control mechanisms in a distributed database system," *Proceedings of ACM International Conference on Management of Data*, April 29 - May 1, 1981, pp. 84-92.
- LIN82 Lin, Wen-Te and Jerry Nolte. "Performance of concurrency control algorithms using timestamps," *Eighth International Conference on Very Large Database Systems*, September 8-10 1982.
- LIN83 Lin, Wen-Te, Jerry Nolte, Philip Bernstein, and Nathan Goodman. "Distributed database system designer handbook," Computer Corporation of America, contract number F30602-81-C-0028.
- MILE81 Milenkovic, Milan. *Update Synchronization in Multiaccess Systems*, University of Michigan Research Press, 1981.

- RIES79 Ries, Daniel. "The effects of concurrency control on the performance of a distributed management system," *Fourth Berkeley Conference on Distributed Data Management and Computer Networks*, August 28-30, 1979, pp. 75-112.
- ROSE78 Rosenkrantz, Daniel, Richard Stearns, and Philip Lewis. "System level concurrency control for distributed database systems," *ACM Transactions on Database Systems*, Vol. 3, No. 2, June 1978, pp. 178-198.
- STON79 Stonebraker, Michael, "Concurrency Control and Consistency of Multiple Copies of Data In Distributed INGRES," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 3, May 1979, pp. 188-194.
- THOM78 Thomas, Robert. "A solution to the concurrent control problem for multiple copy data bases," *COMPCON78*, February 28 - March 3, 1978, pp. 56-62.
- WALK83 Walker, Bruce, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. "The LOCUS distributed operating system," *Proceedings of the 9th Symposium in operating system principles*, October 10-13, 1983, pp. 49-70.