AN ADAPTIVE LOAD SENSING PRIORITY ASSIGNMENT PROTOCOL FOR DISTRIBUTED REAL-TIME DATABASE SYSTEMS

by

Shah Nahid Mahmud

B.Sc., Jahangirnagar University, Bangladesh, 2003

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTERS OF SCIENCE
IN
MATHEMATICAL, COMPUTER, AND PHYSICAL SCIENCES
(COMPUTER SCIENCE)

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

August 2012

Your file Votre référence
ISBN: 978-0-494-94088-4

Our file Notre référence
ISBN: 978-0-494-94088-4

Abstract

Transaction processing in a distributed real time database system (DRTDBS) is coordinated by a concurrency control protocol (CCP). The performance of a CCP is affected by the load condition of a transaction processing system. For example, the performance of the Adaptive Speculative Locking (ASL) protocol degrades in high load conditions of the system. Priority protocols help a CCP by prioritizing transactions. The performance of the priority protocols is also affected by system load conditions, but they can be optimized by dynamically switching between priority protocols at run time when the system load changes. The objective of this research is to develop a protocol, Adaptive Priority Assignment protocol (APAP), which changes the priority protocol at run time to improve the performance of a CCP in a DRTDBS.

APAP is implemented in a DRTDBS, where ASL is used as the underlying CCP to validate APAP. The performance of APAP was tested under varying system load conditions with various combinations of the database system parameters. Under the scenarios tested, APAP performed better than other priority protocols and demonstrated that dynamic selection of priority protocols during run time is an effective way to improve the performance of a CCP in a DRTDBS.

Table of Contents

List of Tables

List of Figures

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisor, Dr. Waqar Haque for his guidance, patience, and immense knowledge in my research. Throughout this research, his professionalism, attention to detail, and high quality of standards tremendously helped me to find myself in a better academic standing. I could not have achieved my goal without his constant support and generosity.

I also would like to thank my committee members Dr. Alex Aravind and Dr. Balbinder Deo, for their support, encouragement, and insightful comments. I spent a long time in Dr. Alex Aravind's research lab. He gave me valuable suggestions about theses based on his experience with his other students, which helped me a lot.

A special word of gratitude to my friend, Christina Tennant, for her continuous help and enthusiasm to improve my thesis writing. For the last couple of months I have been continuously bugging her to proofread my thesis. She patiently checked every line of my writing and made suggestions to fix any issues. I also would like to thank my friend, Barbara Willmer who helped check my writing many times.

Finally, I would like to thank my family: my parents and my siblings for supporting me throughout my life. I am grateful to them that they kept faith in me in all situations of my life.

Chapter 1

Introduction

A database system provides a systematic and secure way to store information and answer queries in an organized manner. In today's world, almost every business uses a database system. Access to a database system is controlled by transactions, which are combination of read operations that read data from the database systems and write operations that update data in the database systems [1]. According to Berstein and Newcomer [2], a transaction must follow ACID properties: atomicity, consistency, isolation, and durability. Atomicity ensures that partial completion of a transaction is not accepted. Consistency means database changes made by a transaction should not violate consistency of the database. When a number of transactions are running in parallel, isolation ensures that each transaction runs as if it were independent of other transactions. The results of a transaction will be permanent as indicated by durability, even in the event of a failure [2].

A real time database system (RTDBS) is a repository for data, like a conventional database, which supports data retrieval and manipulation. In addition, it ensures "some

degree of confidence in meeting the system's timing requirements" [3] [4]. A RTDBS is evaluated by how many transactions complete their tasks before the deadlines expire. The performance of a RTDBS depends on the number of transactions missing their deadlines, the effects of transactions missing their deadlines, the average 'lateness' or 'tardiness' of late transactions, the present status of the data, and the time interval in which the data in the database was collected from the external world [3].

A distributed database system (DDBS) is a collection of data sites, which contain one or more databases connected by a communication network. For example, in the stock market, information is stored in a geographically distributed database, since stocks are bought and sold from different places. A DDBS supports sharing of data and programs, and load balancing among all sites. It can also be incrementally expanded to any number of sites [5]. In a distributed real time database system (DRTDBS), transactions at each site have explicit timing constraints, which become more challenging to follow because the transactions are distributed and database consistency needs to be maintained through controlled data access.

Concurrency control protocols (CCPs) coordinate concurrent access to data and are considered the core component of database systems. There are several concurrency control approaches used to maintain the consistency of the database while transactions are concurrently accessing data. These approaches can be categorized into two types: aggressive or optimistic where operations are scheduled immediately, and conservative or pessimistic where operations may be delayed [6]. Two-phase locking (2PL) is the most popular concurrency control protocol in commercial products. In 2PL, all data items have locks associated with them. When a transaction accesses a data item, it holds the lock of that data item [6]. If the lock is an exclusive lock, the data item becomes unavailable to other

transactions until the transaction holding the lock completes its execution. Therefore, 2PL increases transaction execution time.

The speculative locking (SL) protocol is an approach where conflicting transactions are allowed to access a data item that is held by another transaction to minimize the transaction execution time, and resolve conflicts later [7]. The adaptive speculative locking (ASL) protocol extends the basic function of the SL protocol in DRTDBS and outperforms it under most conditions by exploiting a variety of techniques: efficient memory management, hyper-threading, and transaction queue management (discussed in Chapter 2) [8] [9]. However, the ASL protocol uses the fixed priority assignment approach, where a given protocol is selected when a transaction is initiated and used for the entire duration until it completes. The fixed approach of choosing priority protocols may not always produce optimum results in a real time system, especially where system conditions change frequently, since there may not be enough time for transactions to complete before their deadlines.

Our hypothesis is that by dynamically switching between various priority protocols in a DRTDBS, the number of transactions that meet their deadlines can be maximized. The goal is thus to develop an adaptive priority protocol approach for the ASL protocol that will allow automatic switching between priority protocols as the system load changes, thus improving overall performance. The remaining part of this chapter explains the necessary components and features of the DRTDBS and provides background knowledge pertaining to our work. This is followed by an outline of the contribution of this research.

1.1 Transaction Processing

In a database system, the basic unit of processing is a transaction, which is a set of read/write operations that can be either local or global [1]. Local transactions deal with data

locally at a single site, while global transactions deal with data at multiple sites and may have a number of sub-transactions. In DRTDBS, most transactions are global, and transaction execution involves running sub-transactions at remote sites. According to the distributed transaction model [10], transactions are controlled by processes, which work at different sites to coordinate between a transaction and its sub-transactions. The process that executes at the site where the transaction originates is called the master. Other processes that execute on behalf of the master are called cohorts, which need to maintain communication with the master for a successful global transaction execution. There are two types of distributed transaction execution models: sequential and parallel [11]. In a sequential execution model, operations from a single cohort are executed sequentially. The cohort can only commit after successful completion of all operations. During the execution of operations, a site may have only one cohort or nothing. In a parallel execution model, all the cohorts are initiated together and execute in parallel without interfering with each other. Therefore, transactions complete earlier than in the sequential execution model.

The lifetime of a transaction can be divided into two phases: a work phase and a commit phase [9]. In its work phase, a transaction reads or manipulates data. The master process informs other participating cohorts about the work to be done at each site. The cohorts then complete the work and confirms with the master about the completed work. In its commit phase, a transaction completes when the master gets confirmation from all the cohorts, and executes a commit protocol which makes the changes permanent, or executes an abort protocol which reverts any changes. Concurrent access of a data item causes inconsistency in the database. Serialization guarantees the serial execution of transactions when they execute concurrently on the same data [6]. To protect data and ensure

serialization, locking and commit protocols are used in the work and commit phase, respectively.

1.2 Data Distribution

Data in a DRTDBS are distributed throughout the sites of a system. In a DRTDBS, the nature of the data distribution with respect to the execution of transactions can severely affect the performance. There are two ways data can be distributed (Figure 1.1): partitioned and replicated [6].

1. In partitioned distribution, there are no intersections of data when data are distributed in different nodes. The partitioned distribution minimizes maintenance cost, but if a single site fails then the data is lost and cannot be recovered. Therefore, the whole system fails.

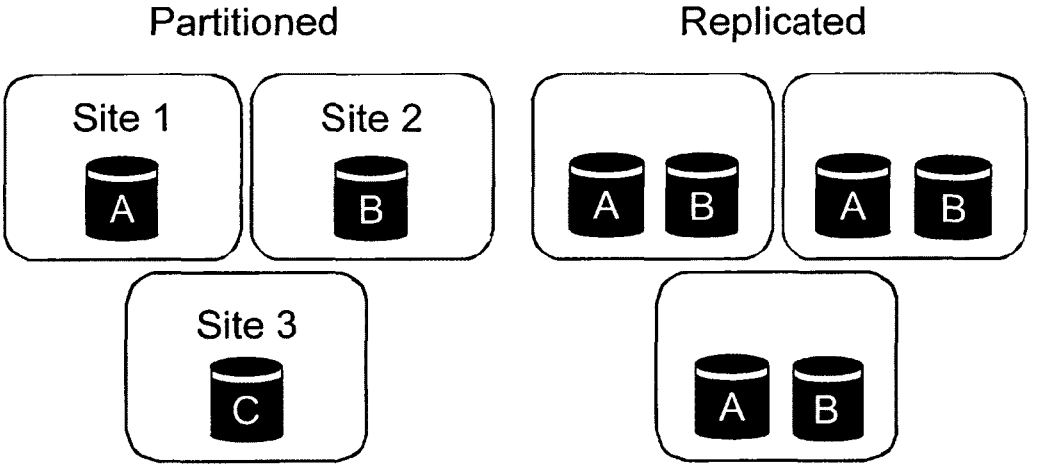

Figure 1.1: Partitioned and replicated data distribution processes

2. In replicated distribution, multiple copies of the same data items are distributed to different sites [6]. This increases availability of the data to transactions when they need it for their operations. In such an environment, the system does not need to stop

operation even when some sites fail because the required data may be available at other sites. However, updating data at one site requires updating all copies at other sites to prevent inconsistencies [12]. CCPs are used to ensure that the "database is a one copy equivalent" [6].

1.3 Deadlines

In a DRTDBS, deadlines represent timing constraints that a transaction must meet to successfully commit. A global transaction requires processing of all associated sub-transactions before it commits, so it requires more time than a local transaction. Deadlines can be categorized as (Figure 1.2): hard, soft, and firm [13].

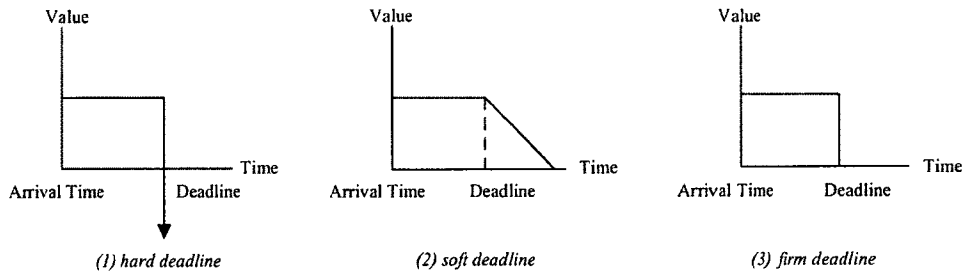

Figure 1.2: Transaction Deadline Model [13]

1. A hard deadline follows strict timing constraints for transactions. If a transaction misses this timing constraint, its value becomes negative and that severely affects the system.

2. A soft deadline provides an extra amount of time for a transaction to finish its work after the deadline. When the deadline expires, the value of the transaction degrades. If the transaction exceeds the extra time then its value becomes zero.

3. A firm deadline is similar to soft deadline, but it does not provide extra time after the deadline. When a transaction misses the deadline, the value becomes zero and the transaction is discarded instantly [14].

1.4 Deadlocks

A deadlock occurs when no transaction can complete due to a circular wait on data requests. For example, a transaction T_1 requests a data lock which is held by another transaction T_2, which may be waiting (either directly or indirectly) for data items which are held by T_1. This circular wait causes a deadlock, where no transactions can complete [14]. Deadlocks can be handled in three different ways [5]: deadlock prevention, deadlock avoidance, or deadlock detection.

1. "Deadlock prevention algorithms ensure deadlock free condition through guaranteeing that at least one of the conditions that cause deadlocks fails to hold" [5]. These algorithms suffer from a high number of transaction restarts.
2. Deadlock avoidance algorithms use prior information about the use of resources to analyze every incoming request, which helps them to predict deadlocks beforehand. These algorithms create lower system overhead than deadlock prevention algorithms. Without enough information these algorithms can fail [5].
3. Deadlock detection algorithms detect a deadlock when it occurs and abort one of the transactions involved in causing it, to resolve the deadlock [5].

In a DRTDBS, deadlock detection requires good coordination among sites. They also need to deliberate a transaction's timing constraints, since a transaction needs to have enough

time to complete if the transaction needs to restart. Deadlock detection in a DRTDBS can be categorized as: centralized, distributed, or hierarchical [15]. In the centralized approach, a site is used as a central coordinator to maintain the resource utilization graph of the entire system. Only the coordinator updates the resource utilization graph and searches it for circular waits. The approach is easy to implement, but it fails if the central coordinator site fails. In the distributed approach, the resource utilization graph is distributed among many sites and requires coordination among the sites to detect deadlocks. Without good coordination between the sites, it is not possible to have exact information about the entire system, making it a complex process. In the hierarchical approach, sites are arranged in a hierarchical order so that deadlock detection involves only some sites, making it simpler than the distributed approach. However, a site can only detect deadlocks in its descendant sites [15].

1.5 Priority Assignment

Priority assignment protocols determine the order of execution of transactions. These protocols also determine which transactions should be blocked or restarted during deadlocks. Therefore, transactions need to be prioritized to avoid unnecessary blockages or delays.

There are three categories of priority assignment techniques: static, dynamic, and hybrid. When the priority of a transaction is "assigned once and for all", these are called static priority protocols [1]. In static priority protocols, priorities of transactions are set before the system executes the transactions and these priorities are not changed at run time. Static priority protocols require complete information about the transactions characteristics and are mostly suitable for small systems [16]. In dynamic priority protocols, the priority of a transaction "changes from request to request" where decisions about scheduling are made at

run-time [1]. Certain important characteristics (i.e. deadlines, slack time) of a transaction change when the system restarts. Therefore, at each request, the characteristic of a transaction is checked to determine the priority of the transaction. In hybrid priority protocols, priorities are fixed for some transactions and varied for others. One use of hybrid priority protocols is making some critical transactions non-preemptive with a static priority protocol during dynamic priority assignment [17]. Some of the popular priority assignment protocols are described below:

1. First Come First Serve (FCFS): In FCFS, the transaction with the earliest arrival time is assigned the highest priority. Therefore, deadline information is not considered during priority assignment. In other words, a new transaction with a close deadline will get a lower priority than an old transaction which may not have a close deadline. This is not desirable in a real time system [18].

2. Shortest Job First (SJF): In SJF, transactions with the smallest run time are executed next [19]. This protocol is suitable when a system has prior knowledge about the run time of the transactions. This process produces the best result when the load is high because it minimizes the average waiting time for a given set of transactions.

3. Earliest Deadline First (EDF): In EDF, a transaction with an early deadline gets higher priority. A drawback of this protocol is that it allocates higher priority to a transaction which is close to its deadline, but might miss it, over a transaction that still has a chance to meet its deadline [20].

4. Minimum Slack First (MSF): In MSF, a transaction with a shorter slack time gets higher priority. Slack time is the maximum amount of time that a transaction can be

idle, but still complete before its deadline [1]. Therefore, MSF depends on both the execution time and the deadline of a transaction.

1.6 Preemption

In DRTDBS, transactions should be preempted to avoid blockage of high priority transactions [21]. If a lower priority transaction has a lock on a data item and a higher priority transaction issues a request for that lock, then the higher priority transaction has to wait until the lower priority transaction completes. This situation is called priority inversion [22]. Due to priority inversion, high priority transactions might miss their deadlines. There are two popular methods to solve the priority inversion problem: priority inheritance and priority ceiling. In priority inheritance, if a lower priority transaction T_L holds a data lock and a higher priority transaction T_H also requests that data lock, then T_L temporarily inherits the priority of T_H until it completes its critical section [22]. The critical section is the time when a transaction accesses shared data and is not allowed to be preempted. After the critical section, T_L releases the lock and returns to its initial priority. The priority inheritance methods reduce the blocking time of T_H from the entire execution time of T_L to the execution time of its critical section. However, this process might suffer from deadlocks, and the block duration can be significant if there is a chain of blocking [22].

In priority ceiling, a transaction T_i can preempt a blocking transaction T_j if T_i has higher priority than other preempted transactions. Otherwise, the transaction T_i is suspended, and the transaction T_j inherits T_i's priority. Priority ceiling not only minimizes the blocking time but also prevents deadlocks because a transaction with an exclusive lock (discussed in the next section) will never be blocked by a lower priority transaction [22]. However, in priority ceiling, a low priority transaction is unnecessarily blocked by a high priority

transaction even when an application is idle while reading data from or writing data into the database [23].

1.7 Locking Protocols

A locking protocol guarantees serialization of transactions within the system by utilizing locks on data items [24]. If a transaction or a sub-transaction wants access to a shared data item, then it needs to request a lock on that data item. There are two types of locks [25]: shared and exclusive. A shared lock is required when a transaction only needs to read a data item. An exclusive lock is needed when a transaction needs to modify a data item. When a scheduler gets a request for a data item from a transaction, it checks the state of the lock. If the data item is not currently locked or has a shared lock, then the scheduler permits the transaction to hold the lock. Otherwise, if the data item is exclusively locked, the transaction needs to wait until the current lock has been released. This ensures that only one transaction gets accesses to a data item at a given time. However, this blocking behaviour of a locking protocol greatly degrades the performance of a DRTDBS because of time constraints [11]. In a DRTDBS, during a read operation, a single copy of the data item is locked (shared) by the scheduler. During a write operation, all copies of the data item are locked (exclusive) by the scheduler until the data modification completes [9].

One of the most common locking protocols is the two-phase locking protocol (2PL) [11], which includes a growing phase and a shrinking phase (Figure 1.3). In the growing phase, a transaction only acquires locks on the required data. In the shrinking phase, a transaction frees all the acquired locks. During growing phase, no lock is released, and

during shrinking phase no new lock is acquired. Every transaction has to go through these phases in order to guarantee the consistency of data [6].

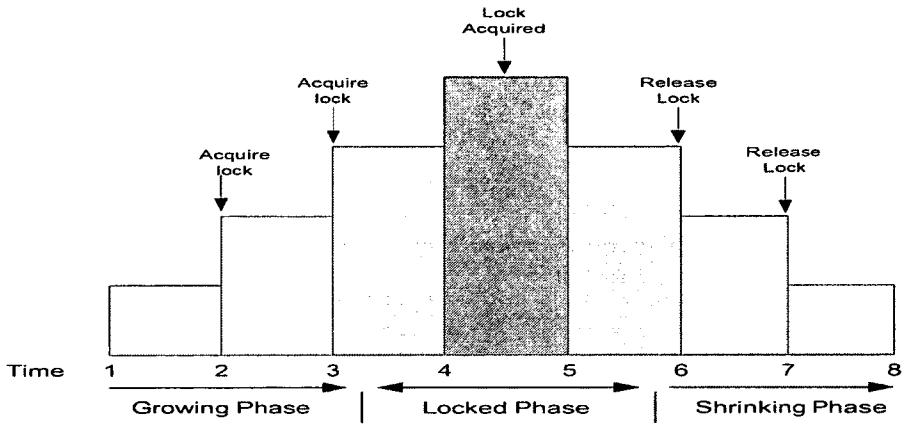

Figure 1.3: Two-Phase Locking - Growing and Shrinking Phase [9]

The 2PL can be static or dynamic [1]. Dynamic two-phase locking (D2PL) and static two-phase locking (S2PL) work similarly, but have different lock settings. D2PL sets locks on the data item required for a transaction and keeps the data locked until the transaction completes. S2PL sets locks on the data item beforehand, using prior knowledge of the transactions that will access the data item. In a DRTDBS, especially for hard real time transactions, this prior knowledge is easily accessible [1]. Distributed S2PL decreases the number of messages transmitted between sites in comparison to D2PL, because all lock requests of a transaction are transmitted as one message. This also reduces the time delays for setting remote locks. Another advantage of S2PL is that a blocked transaction cannot hold locks, meaning deadlocks do not occur. Therefore, D2PL, with its shorter average lock holding time, is preferable over S2PL for conventional non real time database systems [1].

1.8 Commit Protocols

Commit protocols ensure that modification of data by transactions will be permanent after transactions successfully complete [9]. One important feature of transactions is atomicity, which can be secured by commit protocols. To ensure atomicity of a transaction, commit protocols prevent locks on data from being released until the modification of data becomes permanent [26].

In the distributed environment, atomicity is violated if some transactions commit at some sites and abort at other sites [1]. Therefore, all the participating sites need to agree on committing or aborting. Moreover, to maintain atomicity, once a cohort is ready to commit "it has to retain all its data locks until it receives the global decision from the master", which might cause priority inversion [1].

The two-phase commit protocol (2PC) is the most commonly used distributed commit protocol. The fundamental workflow of the 2PC protocols explained by Gupta et al. [27] is described below: The 2PC protocol has at least two-phases: the prepare phase and the commit phase (Figure 1.4). A commit protocol starts execution, when the master receives a WORKDONE message from all the cohorts. In the prepare phase, the master sends PREPARE messages to all cohorts in parallel. After getting the PREPARE messages, the cohorts vote for committing or aborting the execution. If a cohort finds a suitable environment for committing, it sends a YES vote to the master and writes a prepare log record to their local storage. This is called the "prepared state" for the cohorts. However, the cohort cannot commit until they get the final decision from the master. On the other hand, if a cohort cannot complete the execution, it sends a NO vote to the master. The cohort writes

an abort log to their local storage and aborts immediately as a NO vote is considered a veto [27]. This is the end of the prepare phase.

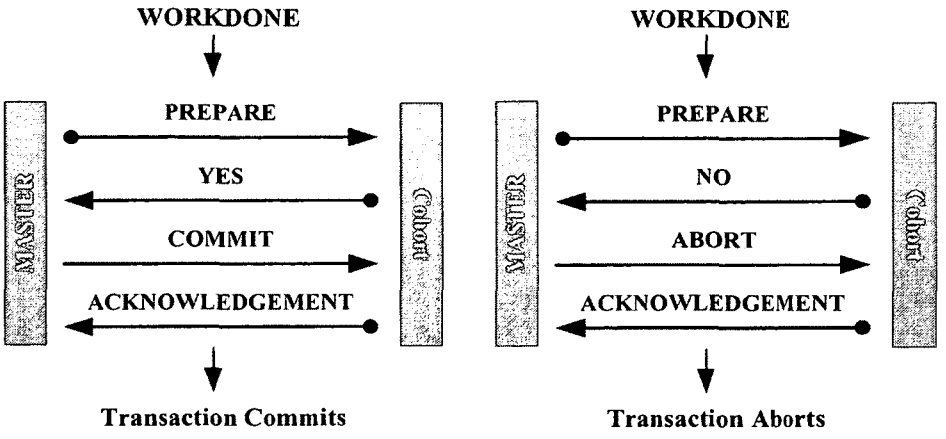

Figure 1.4: Commit Protocol - Commit and Abort Paths

The commit phase starts when the master receives the votes from all the cohorts. If there is not a single NO vote, then it writes a commit log record and sends the global decision, which is a COMMIT message to all the cohorts. This is called "committing state" for the master. When the global decision reaches the cohorts they write a commit log and enter the "committing state". The cohorts commit by sending an ACKNOWLEDGEMENT message to the master. On the other hand, if a single NO vote is received, the master writes an abort log and sends the global decision as an ABORT message to all the cohorts. This is called "aborting state". After receiving the global decision of abortion, all the cohorts write an abort log, and abort the transaction by sending an ACKNOWLEDGEMENT message to the master. Upon receipt of this message from all cohorts, the master writes an end log record and discards the transaction.

There are several variants of 2PC [1]: presumed abort/commit protocols, one-phase commit protocols, and three-phase commit protocols. The presumed abort/commit protocols

reduce the message and logging overheads by making an explicit commit presumption about the committing or aborting of transactions. When a cohort recovers from the failure state, it communicates with the master for available information about the transaction. If the master does not have information available about the transaction, then the cohort can assume that it has aborted. On the other hand, if the cohort gets the commit decision from the master, it commits. The cohort then does not need to send an acknowledgment for the ABORT or COMMIT message and also does not need to write an abort/commit record to the log. The master also does not write the abort/commit record and the end record [27] [28].

One-phase commit protocols (1PC) combine the commit and prepare phases into one phase by removing the cohort voting phase to commit or abort. The cohorts enter into "prepared state" at the time of sending the WORKDONE message. Thus 1PC eliminates one entire phase, which reduces commit processing overhead and delay [29] [30]. However, due to the long prepared state, 1PC suffers from priority inversion, because data locks cannot be preempted in the prepared state.

In 2PC and 1PC, even if a single site fails, all participating cohorts "remain blocked until the failed site recovers" [27]. Three-phase commit protocols remedies this problem by using an extra phase, which is called "precommit phase". This phase occurs between the two phases of 2PC, and makes a preliminary decision about committing or aborting transactions. The preliminary decision then helps all the participating sites, to reach a global decision even though the master fails. However, three-phase commit protocols increase the communication overhead by adding an extra message exchange between the cohorts and the master. Moreover, it forces the cohorts and the master to write a record to the logs in the "precommit phase" [27] [31].

1.9 Concurrency Control Protocols (CCPs)

CCPs make sure that multiple users can access data concurrently in a database management system. A CCP must protect data updates of one user from access and updates of another user until the first update becomes permanent [32]. CCPs maintain the serialization of the transaction operations, and guarantee that the transactions will maintain atomicity. Therefore, the main goal of a CCP is to maximize the concurrency and maintain consistency of the databases. On the other hand, CCPs in a DRTDBS ensure that transactions are meeting their deadlines, in addition to maintaining consistency constraints of the databases [33]. In a DRTDBS, maximizing the concurrency is not enough. The transactions need to be prioritized to maximize the schedulability, which helps transactions maintain their timing constraints. It is also important that transactions are preemptible to reduce the blocking time of transactions. Thus CCPs in a DRTDBS minimize the duration of blocking time by utilizing efficient priority assignment and preemption protocols. As stated earlier, CCPs are classified into two types: optimistic or aggressive and pessimistic or conservative [6].

1. Optimistic protocols do not block transactions; rather they optimistically schedule them instantaneously. This immediate scheduling can violate the serialization order of operations if the scheduler receives an operation later which should have been scheduled earlier than an executed operation. In this situation, optimistic protocols abort the transactions to maintain the serialization. The optimistic process is a faster process, but it might result in a higher number of transaction rejections.

2. Pessimistic protocols block an operation of a transaction immediately if there is any data conflict and continue blocking until the possibility of data conflicts disappears. Delaying the operations by blocking decreases the possibility of data conflict and abortion during transactions, but excessive delays can cause transactions to exceed deadlines. Pessimistic protocols are suitable for transactions that rarely conflict.

1.10 Contribution

Few studies have been done on concurrency control protocols in a DRTDBS as it is difficult to manage distributed data and deadlocks, and coordinate transactions and their sub-transactions performing at different sites. A priority protocol plays an important role in a real time system as it determines whether a transaction will be completed on time or not [16]. EDF is an optimal priority protocol, because if EDF cannot schedule a transaction, then it is not possible for other priority protocols to schedule that transaction [34]. However, the concept of assigning higher priority to transactions with the earliest deadlines is not suitable in high load conditions, because transactions might miss their deadlines due to lack of time [16].

An important CCP in DRTDBSs is ASL which controls a transaction's access to data based on a fixed priority protocol. We investigated the performance of ASL protocol using several common priority protocols under different system configurations. Results of the experiments indicated that a priority protocol that performs well under certain configuration, may perform poorly or moderately under other configurations. This allowed us to determine a set of load ranges in which different priority protocols perform superiorly. To maximize the performance under all system conditions, we concluded that an adaptive approach to

selecting priority protocols is needed. Researchers have been trying to achieve adaptive approaches to utilize the performance variations of different priority protocols under varying system conditions [16] [35] [36]. In all techniques, a common practice is to switch between priority protocols of a system based on the load, to improve the overall performance of the system. However, no research has been done so far to improve the performance of ASL protocol by dynamically switching between priority assignment protocols.

We use the load ranges determined as explained earlier to create an adaptive protocol, called Adaptive Priority Assignment Protocol (APAP). APAP uses the load condition at run time to decide which priority protocol should be used next while keeping ASL as the underlying CCP. Using this approach we observe significant improvement in the overall system performance. The protocol and results are presented in Chapter 4.

Chapter 2

Related Work

We use Adaptive Speculative Locking (ASL) as the underlying concurrency control protocol (CCP) in this thesis; therefore we needed to understand ASL and its techniques. We evaluated ASL and reviewed current research about the speculative locking approach, which is the underlying structure of ASL. We also studied transaction scheduling techniques which consider changing the system environment during run time.

This chapter is divided into two sections. In the first section, we review the ASL protocol with other locking and commit protocols, from which ASL inherited properties such as lending uncommitted data and adaptive approach. We also describe inheritance techniques among those protocols. In the second section, we discuss some scheduling techniques that adaptively switch between priority protocols depending on the system environment and that are most relevant to our research.

2.1 Evaluation of the ASL protocol

ASL inherits the concept of speculation locking of data from SL protocols [7]. However, the concept of lending uncommitted data has also been used in the past in other CCPs, such as PROMPT, PEP, SL, etc. ASL also borrowed the concept of monitoring the system performance and making adaptive decisions to change system behaviour from AEP [8]. In this section we discuss those CCPs briefly.

2.1.1 Permits Reading of Modified Prepared-Data for Timeliness

Permits Reading Of Modified Prepared-data for Timeliness (PROMPT) is a commit protocol based on firm-deadline designed for the DRTDBS [37]. It also extends the concept of centralized 2PL high priority (2PL-HP) for distributed real time environments. In the 2PL-HP protocol, if a higher priority transaction is holding a lock on a data item, then all requests for that lock will be blocked until the lock is released. On the other hand, if the requesting transaction has higher priority, then the lock holding transaction is aborted immediately to release the lock. PROMPT extends this concept further by adding three more steps: 1) in the prepared state, read locks are released by the cohorts just after the cohorts receive the PREPARE message from the master. However, update locks are still held by the cohorts until the global decision about committing or abortion is available; 2) it is not possible to abort a cohort if it is in the prepared state; and 3) transactions can lend uncommitted data optimistically when the lending transaction is only in the commit phase. When the borrowing transactions have access to the uncommitted data, there are three scenarios that describe the interaction between the lenders and the borrowers:

1. The global decision of committing or aborting for the lender is available, but the borrower's local execution is still incomplete, In this case, the lender commits, or

aborts depending on the global decision, and the borrower follows the lender to commit or abort.

2. The borrower completes its local execution, but the global decision for the lender is still not available. In this case, the borrower has to wait and is not allowed to take any initiative related to committing until the global decision for the lender is available or the borrower misses its deadline. This situation is called "put on the shelf". If the lender receives the global decision to commit, then the lender commits and the borrower initiates commit related processing that is called "taken off the shelf". If the lender aborts then borrower's data becomes useless and the borrower aborts.

3. If the borrower aborts during data processing, then the lending is cancelled and the borrower's updates are rolled back.

PROMPT has three additional features to make the data lending process faster and avoid wasting system resources: active abort, silent kill, and healthy lending.

1. In active abort, if a participant cohort is about to abort locally, it sends this information immediately to the master, rather than waiting for the commit phase. Active abort provides a transaction more time to complete and also facilitates proper usage of both logical and physical system resources.

2. In silent kill, if a transaction is rejected before the commit phase of the master, then the rejection is recognizable by the cohorts without communication with the master. Therefore, the master does not need to invoke the abort protocol, because abortion happens silently. The silent kill process saves system resources by eliminating message passing between the master and the cohort.

3. In healthy lending, if a transaction is about to miss the deadline, then the transaction is disallowed to lend data to avoid abortion of the borrower transaction.

PROMPT permits borrowing locked data, but it does not have a cascading abort problem for two reasons. First, the lending transaction is always expected to commit, because it is in the prepared state, so local data conflicts cannot abort the lender. Secondly, the sibling cohorts are going to commit, because all prior data conflicts are handled. Moreover, PROMPT has a controlled lending policy which does not permit the borrower to be a lender simultaneously, so PROMPT affects only the immediate borrower [37].

PROMPT's performance was studied by Haritsa et al. [37] against 2PC, presumed commit, presumed abort, and 3PC for sequential and parallel transactions during both high level of data and resource contention, only high level of data contention, slow and high network speed, and high and low degree of data distribution. In all experiments, PROMPT performed better than the other protocols, especially in the low load condition. PROMPT also showed a higher borrowing rate (the average number of data items borrowed per transaction) during low to medium load. PROMPT's success ratio (the fraction of times that a borrowing was successful) was 1 during low load, but decreased when the system load increased. Therefore, PROMPT performs poorly in high load condition. However, the success of the borrower depends on the success of the lender, because the borrower has to abort if the lender aborts. On the other hand, if the borrower completes before the lender, the borrower cannot commit until the lender completes, resulting in increase of transaction execution time.

2.1.2 Prompt-Early Prepare

Prompt-Early Prepare (PEP) is a one-phase (1PC) real time commit protocol based on a RTDBS. PEP integrates the early prepare (EP) protocol with the lending property of the PROMPT protocol [29]. The standard (2PC) protocol has higher master/cohort communication overhead. To reduce the communication overhead, PEP overlaps the prepare and commit phases into one phase using EP protocol, which is a 1PC protocol. EP reduces the transaction execution time by removing the voting phase of the 2PC protocol. In PEP, the prepared state of a cohort starts at the time of sending the WORKDONE message to the master [38]. PEP is also optimized by a presumed commit mechanism, where the master sends the commit decision, but cohorts do not need to send ACKNOWLEDGEMENT messages to the master. The master also does not write an end log record, rather it writes a membership log record to identify all the cohorts involved in the execution.

However, being a 1PC protocol, EP suffers from priority inversion because of the long duration of the prepared state. The situation deteriorates if the participating transactions are sequential, where cohorts execute one after another rather than parallel. PEP deals with this problem by incorporating the concept of lending prepared data. PEP also incorporates the active abort policy of PROMPT to reduce the response time.

Haritsa and Ramamritham [29] compared the performance of PEP with PROMPT, EP, and CENT (a centralized system) in both parallel and sequential transaction environments. In both environments there were four experiments: 1) data and resource contention, 2) pure data contention, 3) fast network interface, and 4) highly distributed transaction. During the first two experiments, both EP and PEP outperformed PROMPT and PEP outperformed EP, because of the message passing overhead of PROMPT. In the case of

fast network interface, PROMPT performed better, but was still outperformed by PEP. PEP was only outperformed by PROMPT in the sequential high distributed transaction environment when the priority inversion period of PEP is much longer than PROMPT.

PEP exploits the concept of optimistically lending uncommitted data from PROMPT. Moreover, PEP reduces message and logging overheads through the use of 1PC protocol. However, there are a few issues with PEP. PEP suffers from a high number of transaction aborts because it goes into the prepared state when data processing is still unfinished. The extension of the prepared state duration increases priority inversion. Also, deadlocks can occur, because a lender transaction, which has already lent data items, can still access new data items.

2.1.3 Adaptive Exclusive Primary

Adaptive exclusive primary (AEP) is an adaptive concurrency control protocol from which ASL borrowed the concept of dynamically changing behaviour during run time [9]. AEP is designed for distributed database systems and dynamically switches between an optimistic and a pessimistic CCP to improve data and resource contention issues [39]. The optimistic and pessimistic CCPs are the exclusive writer with locking option (EWL) protocol and the priority site locking (PSL) protocol, respectively.

EWL has a controlling site called exclusive writer and primary site (EW/PS). In EWL, a transaction updates data in the database optimistically without considering any data conflict. After the transaction completes, it sends a request for the update to the EW/PS. The request would be approved and the data update would be permanent if there is no data conflict; otherwise, the transaction needs to wait in a queue.

PSL controls the access to each file in the database system using a primary site (PS), which also supports transaction execution as an ordinary site. An update transaction sends a lock request for a file in the database to PS. PS checks the file status and approves the lock request if the file is available; otherwise, PS disapproves the lock request, which forces the transaction to wait in a queue.

AEP dynamically switches between the optimistic and pessimistic protocols if potential data conflicts are found. A transaction begins with executing the EWL protocol. When data conflict occurs, AEP switches to the PSL protocol. AEP also uses PS to control access to a distributed file system like PSL and EWL. AEP maintains a registry to keep track of the local active transactions which are incomplete. The registry helps a new transaction in determining any potential data conflicts. According to this information, AEP performs PSL if there is a potential conflict; otherwise, it performs EWL, from then on.

Tai et al. [39] state that AEP has three assumptions: 1) "with EWL and AEP, an access conflict is detected using a method based on sequence numbers"; 2) "all transactions are read/write transactions"; and 3) "there is only one file in the database system in question and the file is replicated at and shared among all the sites". The operational procedures of PSL, EWL, and AEP are similar and can be divided into two steps. First, the primary site gives permission for a transaction to update a file. Second, the transaction checks if there are any conflicts, and then it decides whether or not to update the file.

The performance of AEP is discussed by Tai et al. [39] in comparison to PSL and EWL by varying the transaction inter-arrival times in two different transaction execution rates. In high transaction execution rates, EWL always outperformed PSL, but in low

transaction execution rates, EWL showed little improvement over PSL. However, AEP always outperformed PSL and EWL.

The adaptive concept of AEP inspired ASL; however, AEP is not properly implemented in a real time distributed database environment. Transaction deadlines and time constraints were discussed, but the implementation of those variables is not obvious. Moreover, AEP cannot detect any conflict at a non-local database site.

2.1.4 Speculative Locking

Speculative locking (SL) protocols extend standard 2PL protocols to allow parallelism among conflicting transactions [40] [7]. The 2PL protocol does not permit an uncommitted data item to be shared. SL allows any transaction to borrow uncommitted data from the conflicting transactions. The borrowing transaction can have access to two versions of data: a before image, which is the data before the conflicting transaction updates it, and an after image, which is the updated data produced by the conflicting transaction. The borrowing transaction then performs speculative operations on both the before and after images of the data. If the conflicting transaction commits, the borrowing transaction retains the after image of the data, otherwise it keeps the before image of the data. Therefore, transaction blocking time is low, making transaction processing faster.

Transaction processing in a database system is shown in (Figure 2.1 [7]). The notation S_i indicates the start of execution, E_i is the completion of execution, C_i is the completion of commit processing and A_i is the abortion of a transaction, T_i. Figure 2.2 illustrates the transaction processing for 2PL, where two transactions, T_1 and T_2, need access to the pages X, Y and X, Z, respectively. The transaction T_2 needs to wait until transaction T_1 commits and releases the lock on page X.

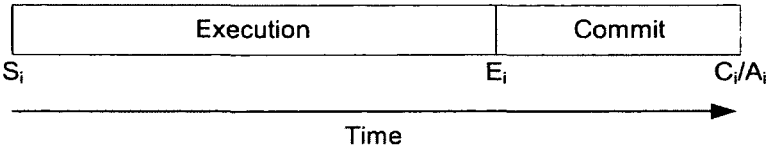

Figure 2.1: A Transaction Processing [7]

Speculative locking processing is shown in Figure 2.3 taken from [7]. Transaction T_1 has locks on pages X and Y. T_1 releases the locks just after it completes processing and creates before and after images of the data X (X_1) and Y (Y_1). T_2 requests for locks on X and gains locks on both X and X_1. T_2 starts speculative executions T_{21} and T_{22} right away and creates after images of both X and X_1, which are X_2 and X_3. If T_1 commits then X_3 will remain, otherwise X_2 will remain.

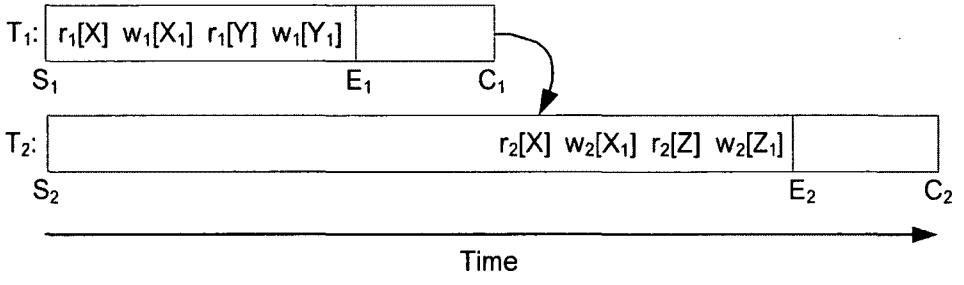

Figure 2.2: 2PL Processing [7]

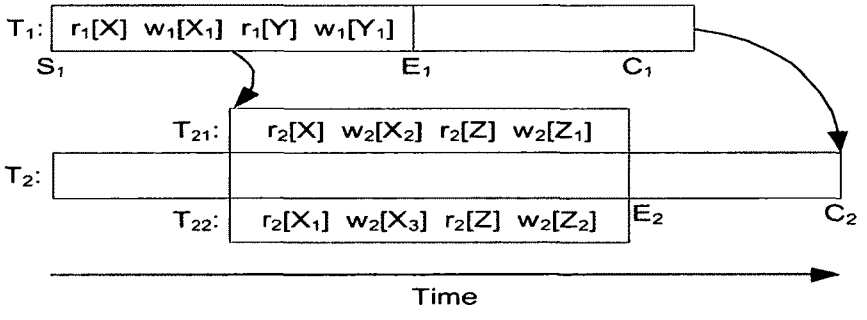

Figure 2.3: SL Processing [7]

Lock Requested By T_i	Lock Held by T_j	
	R	W
R	yes	no
W	no	no

(a)

Lock Requested By T_i	Lock Held by T_j		
	R	EW	SPW
R	yes	no	sp_yes
EW	sp-yes	no	sp_yes

(b)

Table 2.1: Lock Compatibility Matrix (a) 2PL and (b) SL [7]

Lock compatibility matrixes for both 2PL and SL are shown in Table 2.1 [7]. A typical 2PL protocol has two types of locks: read (R) and write (W). A transaction requests a read lock to read a data item and a write lock to update a data item. On the other hand, to perform speculative operations, SL has two forms of write locks: execution-write (EW) and speculative-write (SPW). An update transaction requests EW locks on the data. When the operation creates an after image, SL converts the EW lock into SPW lock, and the data becomes available to other transactions.

In SL, the number of parallel transaction processing increases exponentially as the level of lending increases. For example, for n number of transactions, there can be 2^n number of speculative executions. Therefore, if n transactions conflict, there are 2^n number of possibilities for termination of the transactions. This version is known as the naïve version and is indicated by SL(n). Due to the high number of speculative executions, SL suffers from a higher number of transaction abortions. To solve this issue, SL introduces some variants to restrict the number of speculative executions. These variants are described below:

- **SL(0):** This variant is very optimistic. It assumes that a transaction will abort and terminate if any prior transactions abort.
- **SL(1):** In this version, if more than one prior transaction aborts then current transaction will abort.
- **SL(2):** In this version, if more than two prior transactions abort then current transaction will abort.

SL is a faster process than 2PL, because transactions can start executing sooner. SL opens a new door in concurrency control protocol research. However, a large number of speculative executions can occur with SL, causing data contention that degrades the performance of the system. All the SL variants assume that memory is unlimited, which makes them inappropriate for many systems.

2.1.5 Adaptive Speculative Locking

Adaptive Speculative Locking (ASL) is based on the SL protocol and follows the adaptive nature described in the AEP protocol [9] [8]. ASL uses the same underlying architecture of SL for transaction processing. However, SL assumes infinite system memory, which is unrealistic. To remedy this, ASL maximizes the size of the local buffer and uses a page-based virtual memory mechanism. This mechanism is a memory management algorithm that controls allocation and de-allocation of the memory space. Since SL suffers from high data contention when the number of speculative executions explodes, three variants were introduced, SL(0), SL(1), and SL(2). These variants restrict the number of speculative executions depending on the number of previous transaction aborts. ASL does not depend on the number of previously aborted transactions and introduced the following

techniques: hyper-threading, memory management including virtual memory, and transaction queue management [9] [8] for improving performance.

Hyper-threading (HT) is a Simultaneous Multi-Threading (SMT) technology. HT utilizes instruction level and thread level parallelism to achieve performance gains. Therefore, a single physical processor can run concurrent executions of multiple separate instructions. In HT, one physical processor has two architectural states. One architectural state represents a logical processor and can execute an instruction stream. Therefore, one physical processor can act as two logical processors and can process two concurrent processes or threads simultaneously. HT shows a 65% performance increase over previous generation processors. However, HT is application and hardware dependent. ASL exploits HT by creating a thread for each speculative execution or concurrent process [41].

ASL protocol has a very effective memory management system. It uses two types of memory concepts: system cache and virtual memory [9] [8]. The cache is a volatile and easily accessible storage area, managed by a cache manager, which is used for storing short term data. On the other hand, virtual memory is a technique that implements an operating system paging concept to improve limited memory issues, where data are moved to the physical disk when the cache is full, as if part of the cache. A new transaction requires enough memory space to be reserved either in the cache or in the swap disk before requesting data from the database. When the transaction has enough space reserved, it reads data from the disk into the cache, which are then locked by the transaction manager; otherwise, it waits until enough space is available. The transaction then processes the data and releases the locks. In memory management systems, a transaction cannot block other transactions until it locks all required pages in the memory ASL considers all versions of a page as a unique page

when they are in the memory or moving back and forth between the memory and the disk. [9] [8].

ASL protocol shows its adaptive nature by using the transaction queue management (TQM). Page swapping is an effective way to solve limited memory issues, but it creates high communication overhead. TQM balances between cache utilization and page swapping. To improve the cache utilization, TQM holds or releases transactions from the queue depending on the available space in the system cache. TQM has two parameters: hold level (HL) and enter level (EL). HL and EL help to determine the available system cache (ASC) which helps the transaction manager to compute the amount of total system cache utilization (TSCU). If TSCU is greater than HL, then the transaction manager does not deliver any transactions or subsequent transactions to the scheduler, and keeps checking the ASC value. The transaction manager releases transactions from queue when the ASC value becomes less than EL. By using the correct configuration of HL and EL values, TQM helps to minimize data contention and maximize the cache use [9] [8].

The performance of ASL was tested against all variants of SL by varying cache sizes, number of transactions, inter-arrival times, disk sizes, percentage of read/write operations, and HL and EL values. ASL outperformed SL in all experiments. However, ASL's performance degraded in high load condition [9] [8]. In those experiments, ASL used a static priority protocol approach which is not effective under all load conditions. Therefore, performance of ASL can be improved by switching between priority protocols at run time. In this thesis, we propose an adaptive priority protocol approach for ASL.

2.1.6 Synchronous Speculative Locking Protocol for Read-Only Transactions

Synchronous speculative locking protocol for read-only transactions (SSLR) is based on the SL protocol [42]. Transactions can be divided into two types: read-only transactions (ROT), which only read data, and update transactions (UT) which modify data. In SSLR, a ROT can access data items, which are held by a UT and perform speculative executions. On the other hand, a UT- is not allowed to access data items that are held by a UT and has to wait until the data items are released.

The lock compatibility matrix for SSLR is shown in Table 2.2 [42]. For write locks in SSLR, if one transaction holds a SPW lock then unlike SL no other transactions can get the EW lock. There are two types of read locks in SSLR: a read lock for UTs (RU) and a read lock for ROTs (RR).

Lock Requested By T_i	Lock Held by T_j			
	RR	RU	EW	SPW
RR	yes	yes	no	sp_yes
RU	yes	yes	no	no
EW	no	no	no	no

Table 2.2: Lock compatibility matrix for SSLR [42]

In SSLR, committing of a ROT does not depend on preceding conflicting transaction like it does in SL. If the preceding transaction is still uncommitted then the ROT commits with the before images of the data items. However, if the preceding transaction commits before the ROT commits, then the ROT commits with the after images of the data items. For example, T_2 is a ROT and T_3 is a UT (Figure 2.4). T_1 is a running UT which produces an

after image x_1 of data x_0. T_1 accesses both x_1 and x_0 and performs two transactions T_{21} and T_{22}. As T_2 completes before T_1, T_{21} will remain. Whereas, as T_3 is a UT, it will wait until T_1 completes [42].

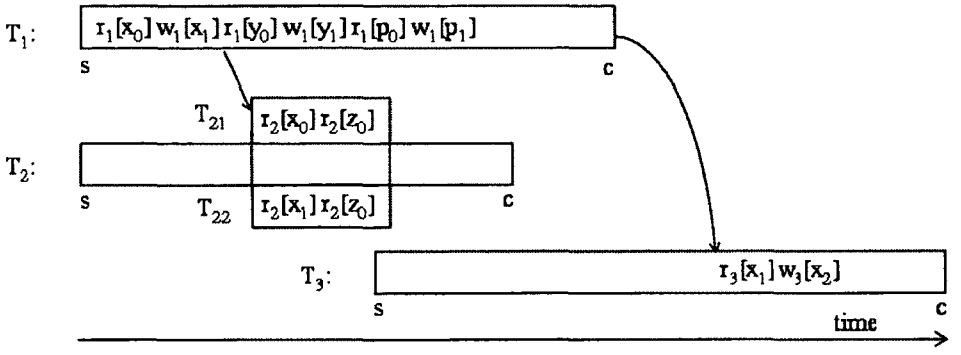

Figure 2.4: SSLR Processing [42]

SSLR not only outperforms other read-only transaction based protocols, but also improves issues with the correctness of transactions (serialization) and the data currency which represents how recently a requested data item was changed. However, when a ROT completes execution before the preceding UT, and commits before the UT commits, the order of transaction executions changes, which might make the system unstable when the system load changes frequently.

The SSLR has two variants. The first is Asynchronous Speculative Locking Protocol for ROTs (ASLR) [43] where a ROT can execute asynchronously, reducing the waiting time of the speculative transactions. Rather than waiting for the conflicting UT to produce after images, the ROT is allowed to access available data item versions to carry out speculative executions. The transaction can start other speculative executions independently based on the available after images. For example, in Figure 2.5, T_2 is a ROT which is conflicting with a UT T_1. The speculative transaction T_{21} of T_2 can access x_0 and start speculative executions.

When transaction T_1 produces x_1, the speculative transaction T_{22} is started. T_2 can commit when any one of the speculative transactions complete.

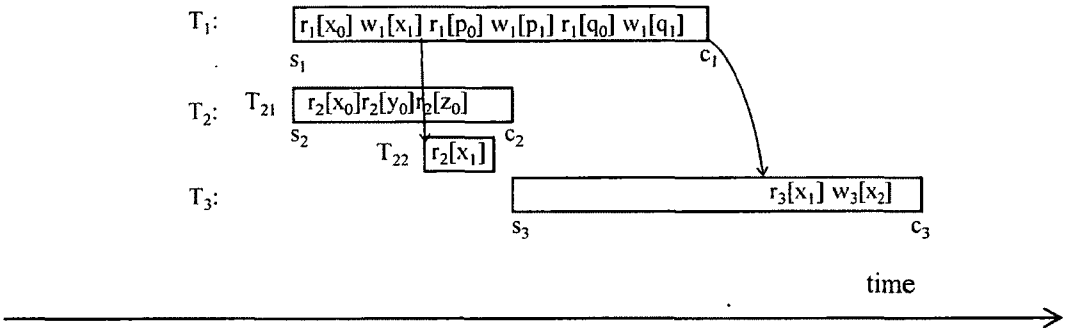

Figure 2.5: ASLR Processing [43]

The second variant is Synchronous Speculative Locking Protocol for ROTs exploiting Semantics (SSLR-S) [44] [45]. Parallelism can be improved by using a property of ROTs, called "compensatability" that reduces waiting time significantly. In "compensatability", when a ROT is in conflict with a UT, a list is created and recorded with identification numbers of the UT and the data item modified by UT. In the commit process, the ROT reads the update value of the data item from the transaction log by using identification numbers. The SSLR-S classifies ROTs into two types: compensatable ROTs (CROTs) and non-compensatable ROTs (NCROTs). A CROT is processed without blocking. When a CROT conflicts with an UT, it shows "compensatability". However, NCROT follows synchronous speculation like SSLR. For example, T_1 is a CROT and T_2 is an UT. Here T_1 conflicts with T_2 on data item x_0, but it performs a parallel execution with T_2 without any blocking. When T_1 commits, it reads the update value of x_0 from the transaction log and performs a compensation operation.

2.2 Switching Between Priority Protocols

This section describes some scheduling algorithms most relevant to our thesis, which switch between priority protocols during run time. EDF is the most used priority protocol in real time transaction processing. However, EDF does not perform well during high load conditions. The following methods use EDF under low load conditions, but switch to a different priority protocol when the system load becomes high.

2.2.1 Adaptive Earliest Deadline

Adaptive Earliest Deadline (AED) is an adaptive scheduling algorithm based on a RTDBS [20]. Under low or moderate resources and data contention, EDF results in the fewest missed deadlines, but when the load increases gradually, performance of EDF degrades abruptly. To improve the performance of EDF in an overloaded environment, AED incorporates an adaptive approach using a feedback control mechanism. AED divides the transactions into two groups named HIT and MISS. Transactions which have higher probability of meeting deadlines fall into the HIT group. On the other hand, transactions which are less likely to meet their deadline are categorized as the MISS group. It is always expected that the transactions that can meet their deadlines should be in the HIT group. Transactions in the HIT group follow EDF scheduling and transactions in the MISS group follow random priority (RP) mapping.

To uniquely identify transactions, a randomly generated key is assigned to a new transaction, which is then used to order a list of transactions. The position in the list is very important as it is used to determine the relevant group for the transaction. To determine a group, the position of the transaction within the list is compared with a dynamic control variable, called HITcapacity. The transaction goes to the HIT group if the value of position is

less than or equal to the HITcapacity value, otherwise, it goes to the MISS group. A feedback process is used to set the value of the HITcapacity. Initially, a scheduler called the priority mapper initializes the value of the HITcapacity. Two parameters, HITbatch and ALLbatch, are used for computing two ratios called hit ratios, HitRatio(HIT) and HitRatio(ALL), that make sure only the transactions which can complete are in the HIT group. HitRatio(HIT) represents the fraction of transactions in the HIT group that meet their deadlines. HitRatio(ALL) represents the same measurement in terms of all transactions in the system. In an ideal case, the HIT group has a HitRatio(ALL) of 1.0 and the MISS group has a HitRatio(ALL) of 0.0. The hit ratios are continuously checked and fed back to the priority mapper. The hit ratios values help the priority mapper to re-evaluate the HITcapacity value, which is an iterative process [20].

Haritsa et al. [20] compared the performance of AED with EDF, random priority (RP), no priority (NP), and latest deadline (LD). AED performed like EDF during low load conditions when EDF outperformed all other priority protocols, and performed like RP during high load conditions when RP outperformed all other priority protocols. Therefore, EDF or RP performs well in a particular load condition while AED performs well under all load conditions. They also show that the HitRatio(ALL) for HIT group varied from 1.00 to 0.98 when the system moved from low to high load, whereas, the HitRatio(ALL) for MISS group varied from 0.0 to 0.1. However, depending on a random number key for determining the transaction groups might not always be accurate. Moreover, calculating HITcapacity is a complicated process, which requires prior knowledge of the transaction characteristics. AED does not target distributed systems, so its performance in a distributed system is not tested.

2.2.2 Sectional Scheduling

Sectional scheduling (SS) is an adaptive approach for transaction scheduling based on hard real-time systems, which changes behaviour according to the system environment [16]. The system environment does not remain constant. It changes as the system evolves, which makes the real time transactions vulnerable to fail their executions before deadlines. EDF is the most stable scheduling algorithm in real time systems, but it performs unpredictably when system characteristics change, especially when the system load changes. SS measures the current load of the system and adequately adapts to changes in the system environment.

According to SS, the system load can be partitioned into three cases: normal load, overload, and serious overload. The load is indicated by ρ. In normal load or low load ($\rho <= 1$), as EDF shows 100% processor utilization, SS uses EDF for transaction scheduling. In overload conditions ($1 < \rho <= 3$), SS uses Deadline/Value First protocol which prioritizes transactions according to their values. A value represents the importance of a transaction in relation to other transactions [34]. SS has three principles. The first principle is that a transaction has higher priority if it has an earlier deadline or a larger importance value than another transaction. The second principle is if two transactions have the same deadline, the transaction with the larger importance value will have the higher priority, or if two transactions have the same importance value, then the transaction with the earlier arrival time will have the higher priority. Finally, the third principle is if two transactions have the same importance value and deadline, then the transaction with the earlier arrival time will have the higher priority. In serious over load conditions ($\rho > 3$), SS follows Highest Value Density First (HVDF) where a transaction has the highest priority if it has the highest value density. HVDF is based on the Highest Value First (HVF) algorithm where a transaction has the

highest priority if it has the highest importance value. The assumption is that a transaction has higher priority if it has a higher value density and a lower slack time [16].

SS has an improved version, which is called robust sectional scheduling (RSS), which is more predictable than SS in overload condition. In this condition, RSS rejects a transaction with the least value that is affecting the system load. RSS has a recovery mechanism in which the rejected transactions are stored in a queue, and when the system is idle, RSS recovers those transactions [16].

Ding and Guo [16] compared SS with EDF, HVF, and HVDF by varying the load of the system. SS outperformed all other priority protocols under all load conditions. In comparison with SS, RSS had a lower number of missed jobs under high load conditions. However, SS is not implemented in a DRTDBS where transaction operation type (read or write) might affect the system load. It is also unclear how changes in system load were implemented.

2.2.3 Maximum Miss First

Maximum Miss First (MMF) is a non-preemptive scheduling algorithm for soft real time systems [35]. MMF schedules a transaction by calculating the miss ratio of the transaction. Miss ratio is the number of missed jobs at a certain time divided by the number of released jobs during that time. The miss ratio of a transaction τ_i is defined as [35]:

$$MR_i(t) = \frac{N_i^{miss}(t)}{N_i^{job}(t)}$$

where $N_i^{miss}(t)$ is the number of missed jobs of transaction τ_i at time t and $N_i^{job}(t)$ is the number of released jobs of transaction τ_i at time t. The values of $N_i^{miss}(t)$ and $N_i^{job}(t)$ are zero for a new transaction τ_i, but they increase as τ_i releases jobs.

MMF assigns priority to the transaction which has the highest miss ratio. If two transactions have the same miss ratio, then MMF uses the EDF priority protocol for scheduling transactions. When the system load is low, the miss ratio is zero; therefore, MMF acts like EDF, which is preferable under low load conditions. When the load increases gradually, the miss ratio becomes the key factor for scheduling [35].

Asiaban et al. [35] compared MMF with EDF, gEDF and other similar scheduling algorithms. MMF showed the same miss ratio for different transactions with different periods while the other algorithms did not. Therefore, MMF works well for all transactions. MMF produced a lower number of consecutive missed jobs in comparison with other algorithms. MMF also showed better jitter (the maximum time variation between the finishing times of any two consecutive jobs that completed successfully). MMF demonstrated the same system utilization as EDF and gEDF.

MMF does not depend on the execution time of the transactions which makes it more stable. However, if the number of consecutive missed jobs is high, then MMF can block the transactions for a long time because of the high number of missed attempts. This can cause transactions to miss their deadlines, which degrades the performance of the system.

2.2.4 Group-EDF

Group-EDF (gEDF) is a scheduling algorithm designed for non-preemptive soft real time systems. It uses both Earliest Deadline First (EDF) and Shortest Job First (SJF) to schedule a transaction in the system [36]. It creates groups of transactions based on their deadlines, where deadlines of the transactions are close to each other. A group in the gEDF algorithm is created based on a group range parameter, Gr, represented by a percentage value. If the deadline value of a transaction falls under the Gr percent of the deadline value of

the current transaction, the transaction is considered to be in the same group as the current transaction. Groups are scheduled based on the EDF protocol; when the deadlines of all transactions in a group are earlier than the deadlines of all transactions in other groups, then the first group has higher priority. However, gEDF follows SJF to schedule individual transactions within a group.

Li et al. [36] studied the performance of gEDF by varying load tolerance (what extent a transaction can miss the deadline), the deadline value of the transactions, the execution time of the transactions, etc. gEDF performs like EDF during low load, but performs better than EDF during high load. The gEDF protocol also outperforms best-effort algorithms which switch priority protocols according to the system load.

According to Li et al. [36], though gEDF outperforms EDF, it does not guarantee fairness because it has tendency to favour only small transactions. However, if we concentrate in increasing the number of completed transactions before the deadlines, then this is a good strategy. Nevertheless, gEDF does not consider distributed real time systems where a transaction might have a number of sub-transactions, so grouping of transactions at different sites and coordinating between them can be difficult.

2.3 Summary

In a RTDBS, a transaction not only needs to maintain the serialization, but also needs to complete before the deadline. Traditional CCPs guarantee serialization in transaction execution, but cannot guarantee meeting deadlines. Again, a RTDBS in a distributed environment (DRTDBS) needs to consider data availability, system resources availability, and communication overhead between transactions and sub-transactions. Because of these reasons, extending traditional CCPs by using an optimistically lending data technique or

adaptively changing the behaviour of the system according the system state became very popular in designing a CCP. The ASL protocol adapted both features and proved itself as an effective CCP in a DRTDBS. However, the performance of ASL is affected by the system attributes (i.e. network latency, network topology, priority protocols). Therefore, the performance of a system can be improved by finding an optimal solution for these attributes. Dynamically switching between priority protocols according to the system load is an optimal solution to improve the system performance. From various solutions presented in this chapter, we found that if we can quantize the system load, then it is easy to switch between priority protocols considering the value of the load. However, none of the solutions targeted a DRTDBS. Therefore, finding a solution for switching between priority protocols in a DRTDBS is the focus, and main contribution, of this thesis.

Chapter 3

The Simulator

To analyze the performance of the ASL protocol in a DRTDBS, a distributed real-time database model implemented in a distributed real time transaction processing simulator (DRTTPS) is used. DRTTPS was developed in the parallel and distributed computing research lab at UNBC. The key features of DRTTPS are provided in this chapter. A detailed description of the simulator can be found in [46] [9].

3.1 Distributed Real-Time Transaction Processing Simulator

A model is not a real system, but describes the real system with all necessary information in a simpler manner. A simulation of a model describes the workflow of the model [47]. The discrete event simulation model is a type of simulation model, where a representable system only changes its states at discrete points of time [48]. The continuous event simulation model is another type of simulation model, which changes states of the system continuously over time. The discrete event simulation model is preferable over the continuous event simulation model because of its simplicity [47]. A simulation model

requires development of a software application to implement the workflow of a real system. The software application consists of entities that represent physical elements of the real world. The entities interact with each other to perform actions, which represents the behaviour of the real system [9]. The software application is called the simulator.

Distributed Real-Time Transaction Processing Simulator (DRTTPS) is a discrete event simulator that simulates a DRTDBS [46] [9]. DRTTPS is flexible to incorporate new concurrency control protocols (CCPs) and can be a test bed for analyzing their performance. DRTTPS also allows other components to be added such as a new priority protocol or new network architecture. In DRTTPS, events are executed sequentially. An event can be an action that a component of a simulator performs during execution. To maintain the sequence of events, they are inserted into a list in the order they need to be executed based on the event's execution time. The events are then executed by removing them one by one from the beginning of the list to the end by incrementing time. A tick is the unit to measure a discrete amount of time in the simulator.

The network configuration in a distributed system is shown in Figure 3.1 [49]. It consists of one or more sites where each site has one or more nodes, and each node has one or more real-time databases. It maintains a local area network between all nodes in a site and a wide area network between all sites. The system provides virtual routers to each site to maintain the connection between nodes inside the site or with other sites. It also provides routing tables to each node, where each routing table contains routes to all conceivable destination nodes. The destination nodes may be inside the container site or in other sites. The network connection is very reliable. A failure message is always re-sent to confirm the arrival of the message at the destination and the system also updates the routing table during

any connection failure or recovery to have an efficient routing path. A network connection has the following properties:

1. Source: a network connection originating node.
2. Destination: a network connection ending node.
3. Bandwidth: maximum capacity of a network connection.
4. Latency: the time taken by a message to travel from source to destination.
5. External usage: the percentage of the bandwidth occupied by external users, if any. The bandwidth excluding the external usage is considered as the effective bandwidth of the system.

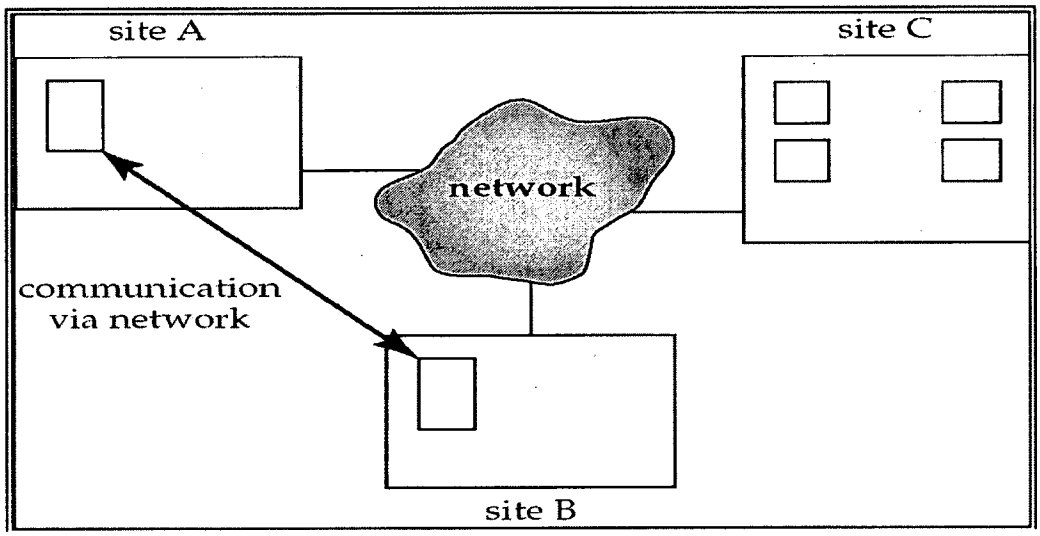

Figure 3.1: Network configuration [49]

3.1.1 Node Architecture

A node is the core component of DRTTPS and has the following characteristics:

1. Concurrency Control Protocol: It indicates which CCP is followed by the transactions in the node.

2. Preemption Protocol: A preemption protocol controls how the transactions are preempted in the system to avoid priority inversion. The options are described below:

    - High Priority: A transaction which is holding a lock on a data item can be preempted only if a transaction which requests a lock on the same data item has higher priority than the lock holding transaction.
    - Priority Inheritance: The lower priority lock holding transactions inherits the priority of the higher priority waiting transaction to complete rather than being aborted.
    - No preemption: No preemptions will be attempted.

3. Deadlock Resolution Protocol: This protocol defines which transaction will be aborted if a deadlock occurs. The options are shown below:

    - First Deadlock Resolution: A list is generated with the transactions which are involved in a deadlock. First deadlock resolution aborts the transaction at the top of the list to resolve the deadlock.
    - Priority Deadlock Resolution: Priority deadlock resolution requires a list of the transactions sorted according to their priority. It aborts the lowest priority transaction from the list to resolve the deadlock.

4. Priority Protocol: A priority protocol defines the order of execution of transactions, for example EDF, SJF, MSF, and FCFS, as described in Chapter 1.

APAP selects one of the existing priority protocols and switches between them depending on the load conditions.

5. Max active transactions: It sets the maximum limit on the number of transactions that can concurrently run in a node. Excess transactions need to wait in a queue.
6. Timeout: It represents the maximum time limit a transaction can be idle during execution before it is aborted. This time limit helps to resolve distributed deadlocks by predicting that the transaction is trapped in a deadlock if it exceeds the time limit.

A node consists of several hardware and software components: **processor manager**, **disk manager**, **buffer**, **swap disk**, and optionally **workload generator** [46]. The **processor manager** contains and arranges processors in the node. A **processor** can use a hyper-threading technique which allows processing more than one page at a time [46]. A processor has two attributes:

1. Process Time: It indicates the processing time of a page measured by ticks.
2. Hyper-threading: It represents if the hyper-threading technique is enabled or disabled in the node. System performance can be analyzed with or without hyper-threading.

The **disk manager** arranges disks in a node which store pages. Disks are non-volatile storage where data can be partitioned or replicated. A disk has two attributes:

1. Access time: It represents the number of ticks the system takes when it reads or writes a page from or to the disk.

2. Page Range: It represents the number of pages in that disk. Data partition or replication can be controlled by page ranges. However, CCPs in DRTTPS are not designed to handle replicated data.

A **buffer** is a volatile storage where transactions perform read/write operations on the pages. A transaction needs to have all requested pages to be loaded to the buffer from the disk if the buffer has enough space. The buffer has a parameter which represents the maximum number of pages a buffer can support during read and write operations. If the buffer does not have enough space, pages must be swapped out to a swap disk. The **swap disk** component indicates the physical disk and it contains a parameter, called access time, which represents the number of ticks the system requires to swap a page .

A **workload generator** is an optional component of a node, which is used for generating transactions in the node. It has the following parameters to control the number and nature of the transactions:

1. Size: It represents how many transactions are generated.
2. Arrival: It represents the inter-arrival time of transactions in the system generated by the workload generator. At a low inter-arrival time, many transactions enter the system within a short time period which causes more transactions to run concurrently, resulting in high system load.
3. Slack time: It represents the deadline of a transaction, as well as the extra time after the deadline. The range for the baseline experiment has been selected to represent a deadline which includes a slack of 2-6 times the execution time.
4. Work size: It represents the number of pages a transaction might access in its lifetime.

5. Pages: It defines the total number of pages in the system.
6. Update: It represents a percentage of update operations in transaction's execution.

3.2 Graphical User Interface

SetupTool is the core of DRTTPS that allows users to set up and run simulations by creating site structure, node structure, and network architecture (Figure 3.2). The SetupTool contains a number of site components and corresponding parameters, where parameters control the characteristics of all the site components. A simulation is run using combinations of specified parameter values. The SetupTool also has a component called variation, which defines the number of simulations that can be run at the same time. The SetupTool has two graphical panels. The left panel shows the site components and the right panel shows parameter settings of that particular site component. The SetupTool allows users to save simulations, where all site components and parameters are encapsulated in a binary image file. The binary image file is used later to load the simulation. After running a simulation, another component of DRTTPS, called ReportTool (Figure 3.3), provides a user interface to create and save charts showing the results.

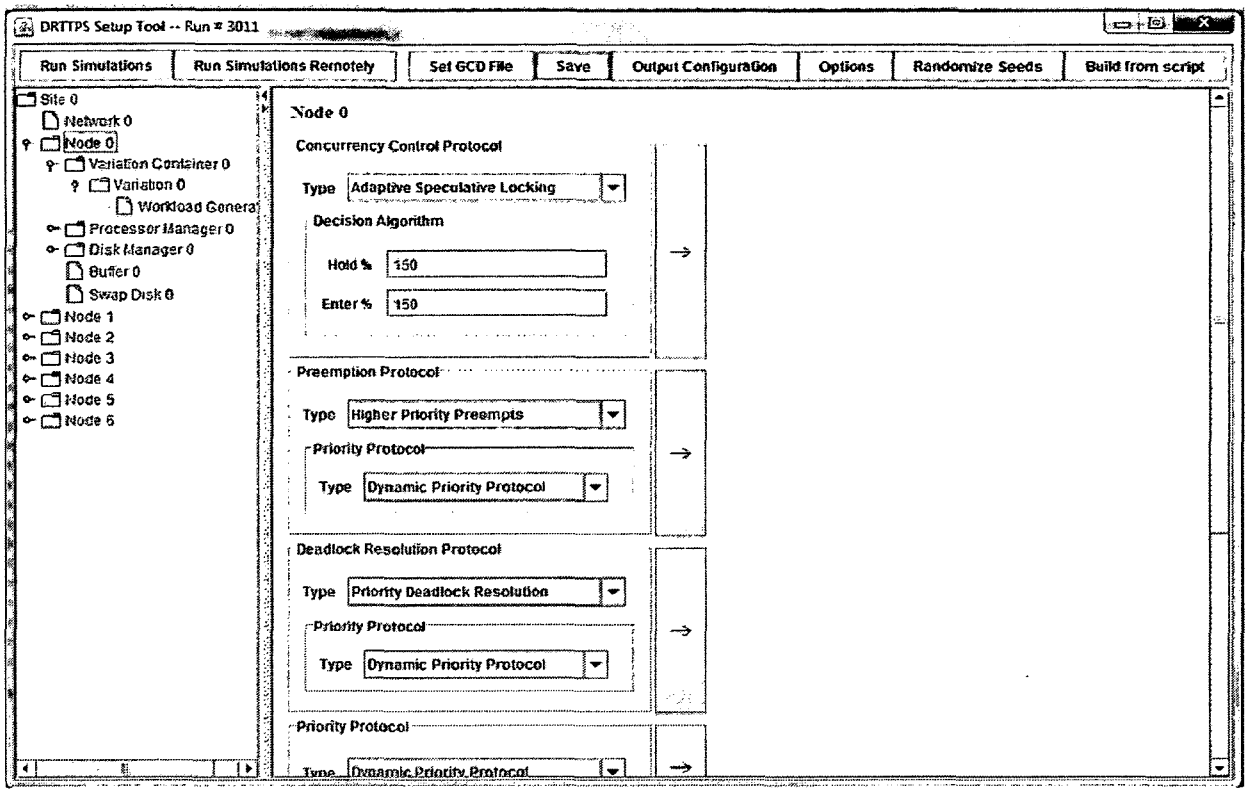

Figure 3.2: Simulator SetupTool

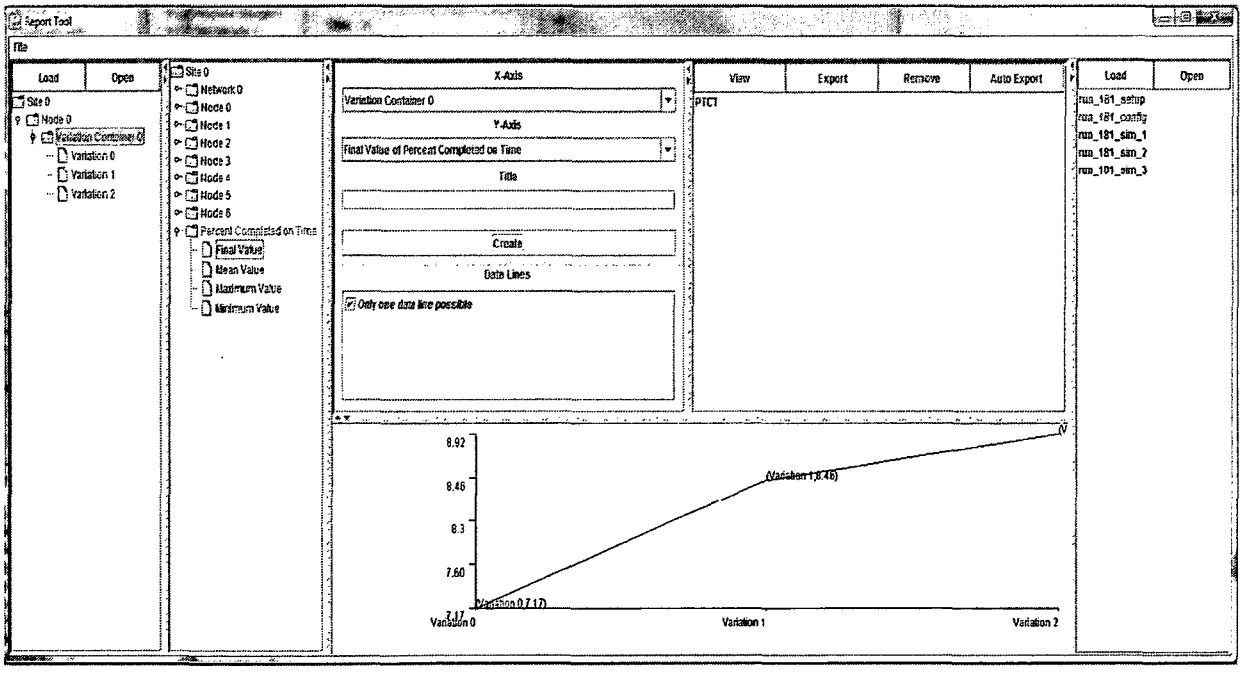

Figure 3.3: Simulator ReportTool

3.3 Software Design:

DRTTPS provides a common pluggable framework for all components. The components are organized in a hierarchical structure (Figure 3.4 [46]) where all functioning components are children of the node module. All children of same base class have the same interface for interacting with other modules. There is a base-line CCP which is at the top of the CCP inheritance tree, and contains basic functionality. CCP Level II inherits from the base-line protocol and CCP level III inherits from the CCP level II. All protocols at the same level with CCP Level II or CCP Level III will have the same physical structure (discussed in detail in the following subsection).

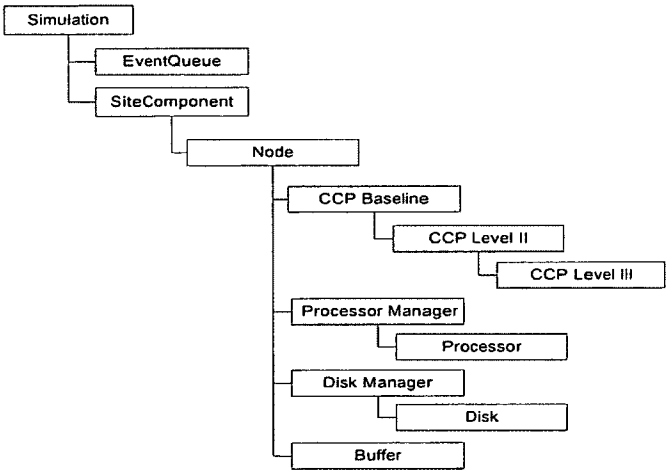

Figure 3.4: Simulation class structure [46]

The simulator architecture is flexible to incorporate new protocols without major modification of the code. An example could be priority protocols. If a new priority protocol follows the common architecture given by DRTTPS, then the protocol would be easily recognized by DRTTPS. The discrete event model is implemented by creating a global event queue which stores all events and exists at top of the hierarchical class structure. DRTTPS uses Java reflection [50] to create an event queue, which does not require message passing

between classes. Java language was chosen for the development of the simulator for its versatility, efficiency, and platform portability. For a new event task, a new event object is created and added to the event queue. The events are executed one by one from the global event queue.

The performance analysis of the protocols is executed by a separate class which keeps track of any output values in the system. The values are then displayed in graphs in real-time as the simulation progresses (Figure 3.5). Finally, the ReportTool displays the statistics associated with the graph. DRTTPS can save or load a simulation for future use by encapsulating user interface, statistics, parameter setting, graphs etc. in a single object, by utilizing the serialization technique of Java language [50].

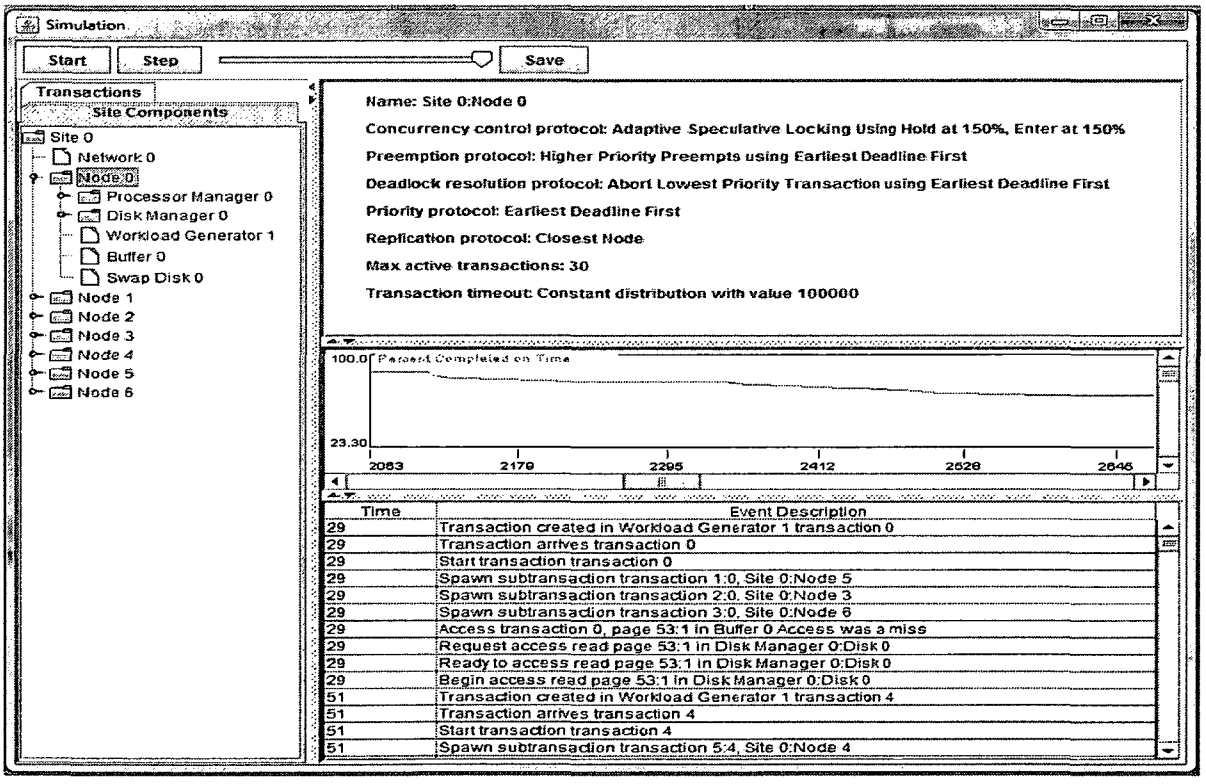

Figure 3.5: A running simulator

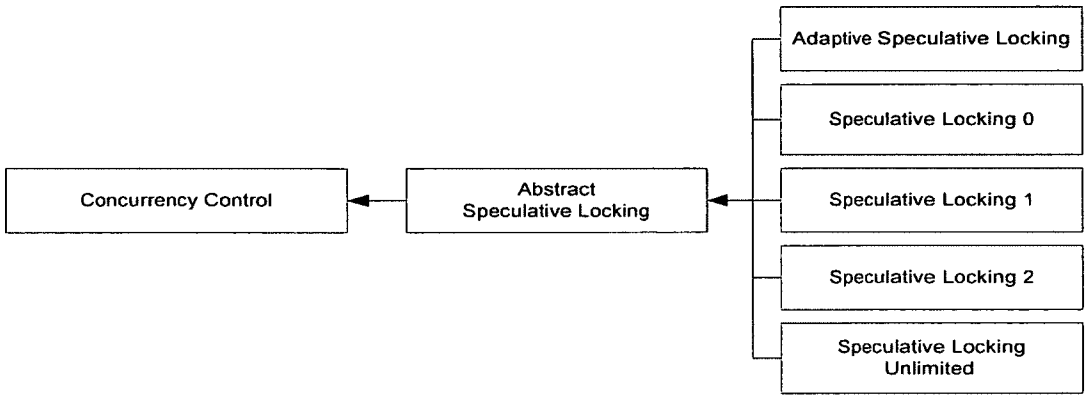

Figure 3.6: Speculative locking protocol dependencies [46]

3.3.1 Concurrency Control Protocols

Figure 3.6 shows speculative locking protocol dependencies in DRTTPS. The implementation of a CCP follows a hierarchical order. Concurrency Control is at the top of all CCPs' dependency hierarchies which provides default methods that are common to descendant CCPs. Abstract Speculative Locking inherits Concurrency Control for all default methods of CCP and adds other methods with specific speculative locking functionality. Abstract Speculative Locking provides a generic structure for all speculative locking protocols. The structure includes common speculative functionalities as well as some distinct functionality (such as restricting number of speculative executions). Abstract Speculative Locking also provides a version tree structure to keep track of all versions of a page. Speculative locking protocols are implemented by inheriting all methods from Concurrency Control and Abstract Speculative Locking. In addition it adds some extra features, which are not available in other speculative protocols; for example, ASL adds hyper-threading, memory management including virtual memory, and transaction queue management techniques.

For a new transaction, the executing CCP checks the required pages on the version tree to lock the versions of the pages. Not all versions of a page will be locked. It depends on the executing CCP's page restriction criteria. The CCP sets speculative read or write locks on the pages and creates a group of all the locks to keep track of all locks of a transaction.

In DRTTPS, a transaction is aborted if it is pre-empted by other transactions or if it becomes deadlocked. The aborted transaction releases all locks that were obtained or requested. All versions of the pages on the speculative tree created by the transaction are removed and protected for future use. The transaction is then positioned in a queue for future execution.

In the next chapter, we present our proposed method protocol and demonstrate its superior performance as compared with other priority protocols.

Chapter 4

The Proposed Protocol, Experiments and Results

This chapter describes our proposed protocol, Adaptive Priority Assignment Protocol (APAP) in detail. This is followed by the presentation of experiments and results.

4.1 Adaptive Priority Assignment Protocol

Adaptive Priority Assignment Protocol (APAP) is an adaptive protocol for assigning priority to transactions in order to improve the performance of a DRTDBS under varying system loads. A priority assignment protocol is an integral part of transaction processing. The effect of this protocol on the overall performance of the system is greatly impacted by data contention (number of users' requests to a database system at any time) and resource

contention (conflict of access to shared resources). The factors that cause data and resource contention include inter-arrival times of transactions, disk size, cache size, number of transactions, network topology, number of physical nodes, and the page update rate.

The performance of a priority assignment protocol varies with different system environments; especially with different loads [16]. The system load "fluctuates drastically from day to day, hour to hour, minute to minute, even second to second" [51]. The load in a database system can be defined as the demand of the database system, when a transaction performs queries and analysis through the DBMS. Moreover, any batch jobs or system commands can also create a demand [51]. The load can be quantified by the utilization of the system, denoted by the following formula taken from [36] [52]:

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} \quad \text{----------} \quad 3.1$$

where e_i and p_i denote elapsed execution time and the total assigned processing time of transaction T_i, respectively. EDF achieves 100% processor utilization during low load [16]. Conversely, during high load, EDF exhibits a significantly poor performance, thus prompting selection of a better priority protocol [53]. The load calculation takes place whenever a new transaction arrives in the system. We ran the system with different load configurations until the number of completed transactions were maximized, and then recorded the load. After analyzing the performance with different loads, a range was determined for a particular priority protocol (Table 4.1).

Priority Protocol	Load Range
EDF	0 to 1.25
FCFS	2.2 to 2.3
SJF	Rest of the time

Table 4.1: Load ranges for priority protocols

The performance of MSF protocol is comparatively better under high load than in low load. However, in high load, SJF has a higher chance to complete a transaction than MSF because SJF minimizes the idle time of system during transaction execution. Moreover, in our experiments, during high load SJF performed better than MSF. Therefore, we did not use MSF as one of the options for APAP. FCFS performed the best over a small load range from 2.2 to 2.3, beyond which the performance of SJF was superior.

APAP uses the load range to decide which priority protocol to execute. When the system selects a new task it sends a load value to APAP. After getting the load value, APAP matches the load value with the load ranges recorded in it. If the load falls within the range of the currently executing priority protocol then no change happens. If the load falls within a different range then a switch operation is performed to change the executing priority assignment protocol to the newly determined priority protocol.

4.1.1 Implementation

We use DRTTPS as a test bed for APAP and use ASL as the underlying concurrency control protocol. As previously discussed, DRTTPS provides a common architecture to select the next transaction from a list of transactions through a priority protocol engine.

When a new transaction arrives in the system, it is placed in a list where it waits until scheduled. The transaction selection process proceeds as follows: The priority protocol

engine receives a request from the system to select a transaction from the waiting list. The priority protocol engine calculates and sends all the transactions' values in the list to the backend comparator, which selects the highest priority transaction (Figure 4.1). The value is the deadline when the priority protocol is EDF or the execution time when the priority protocol is SJF. In the case of EDF, the highest priority transaction will have the earliest deadline, whereas in the case of SJF, it will have the lowest execution time. The transaction with the highest priority is then returned to the system for scheduling. While the transactions are executing, the preempted and/or aborted transactions are added to a priority queue in the backend comparator for future consideration.

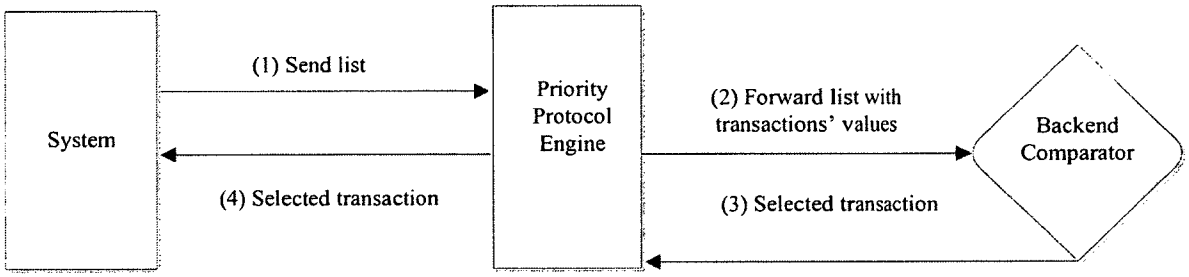

Figure 4.1: Sequence of transaction selection process without APAP

When APAP is introduced in DRTTPS architecture, it acts as a mediator between the system and the priority protocol engine (Figure 4.2). The selection of a transaction from the list of transactions waiting to be scheduled proceeds as follows: The APAP engine receives a load value from the system and determines the most suitable priority protocol based on that value. This modification ensures the dynamic selection of the priority protocol. The APAP engine receives a list of transactions from the system instead of the priority protocol engine. The APAP engine communicates with the priority protocol engine to get the values of the transactions in the list and sends the list with the transactions' values to the backend

comparator so that transaction selection can be completed and returned to the system. The communication between the APAP engine and the priority protocol engine is transparent to the system, as it only communicates directly with the APAP engine.

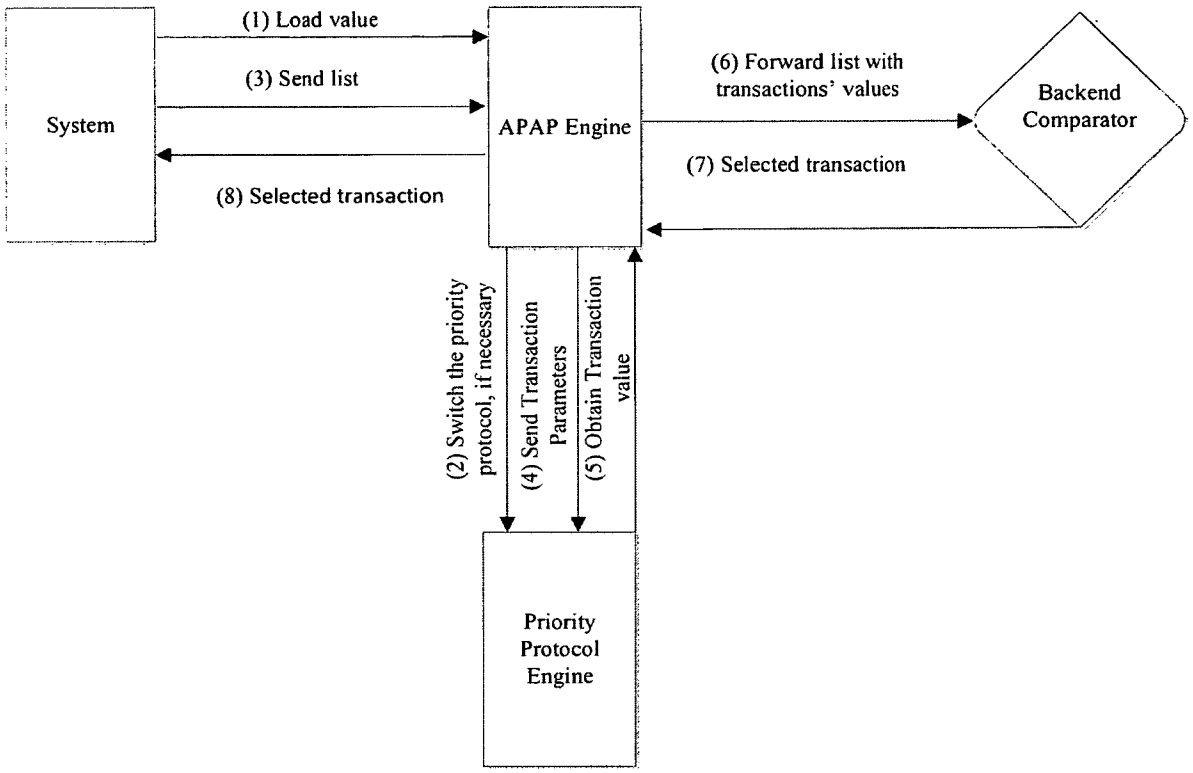

Figure 4.2: Sequence of transaction selection process with APAP

In order to minimize the switching operation overhead, APAP only tests load for switching upon arrival of a new transaction and not when transactions are pre-empted and/or aborted and added to a priority queue. However, it continues acting as a mediator to add transactions to the priority queue. The inclusion of APAP in the DRTTPS does not cause any overhead to the original workflow of transaction selection, as APAP simply selects transactions in the same manner (i.e. using the values for the transactions returned by the priority protocol engine).

4.2 Experiments and Results

In this section, we describe the experiments conducted in this study. For each experiment, the performance of APAP was compared with other priority protocols including: EDF, SJF, FCFS, and MSF (discussed in chapter 1). The key performance metric is the percentage of transactions completed on time (PTCT), that is, before the deadlines. In addition, we observed the switches between the protocols in APAP and present it as the percentage of usage (POU) of priority protocols given a particular system configuration.

4.2.1 Baseline Experiment

In a DRTDBS, there are many parameters that can change the system load, such as system resources (cache size, number of disks), transaction inter-arrival time, slack time, network topologies etc. In our baseline experiment, we followed the parameter settings given in Table 4.2. Later we varied these parameters to change the overall system load. For all experiments, PTCT and POU are measured at a transaction inter-arrival time range of 5 to 55 ticks, because there was no observable result below or above this range. We used binary tree topology for network connections among sites, where a node could have at most two child nodes. We assume bandwidth is unlimited and there is no network latency.

In Figure 4.3, when the transaction inter-arrival time is between 5 and 30 ticks, SJF demonstrates significantly higher PTCT than EDF and FCFS, because in high load scenarios, EDF and FCFS protocols suffer from a high number of transaction aborts. Under heavy load, EDF performs worse than the other protocols until an inter-arrival time of 40 ticks. Beyond this point, the transactions have enough time to complete and EDF climbs to 100% completion rate. APAP performs better than all tested protocols by switching between the priority protocols.

Parameter Type	Parameter	Value
Network	Network Topology	Binary Tree
	Bandwidth	1000 bit/tick
	Network Pipe Latency	0
Node	Active Transaction Count	30
	Disk Count per Node	2
	Maximum Pages Held per Disk	100
	Disk Access Time	35 ticks
	Cache Size	20
	Swap Disk Access Time	35 ticks
	Transaction Process Time	15 ticks
Transaction Generator	Pages per Transaction	4-12 pages
	Slack Time	720-2160
	Inter-arrival time	5-55 ticks
	Page Update Rate	100 percent
	Transaction Count	100

Table 4.2: Parameter Settings

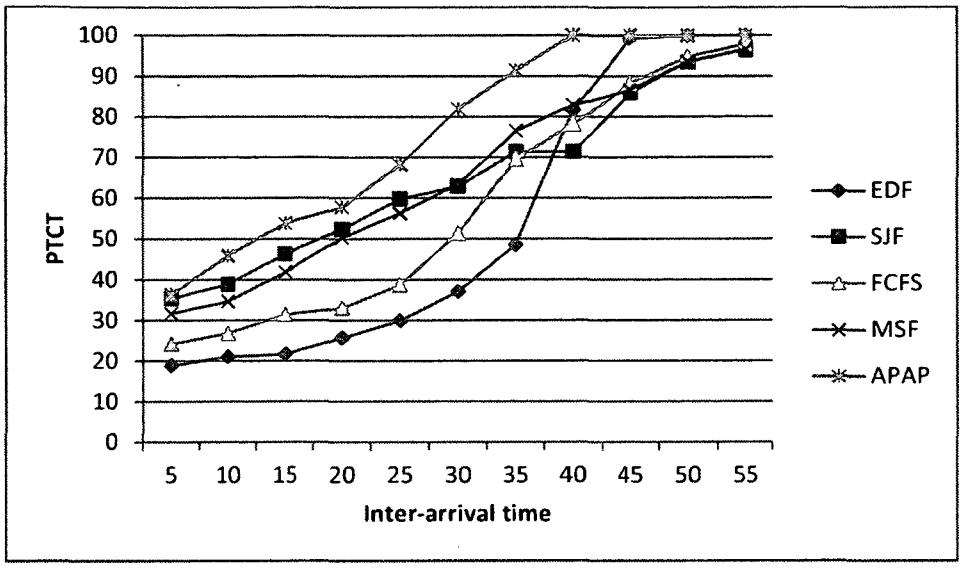

Figure 4.3: PTCT for the Baseline Experiment

Figure 4.4 shows the POUs of priority protocols in APAP for the baseline experiment. When the inter-arrival time of the system is between 5 and 20 ticks, the system usually runs with SJF because of its superior performance during that time. When the inter-arrival time of the system increases, the load condition of the system decreases. Thus, APAP increases the usage of EDF, rather than SJF. The inter-arrival time between 20-30 ticks shows a transition period during which each of the two protocols is used approximately 50% of the time. The usage then becomes more distinct in favour of EDF due to decreased system load. APAP follows FCFS during the inter-arrival times of 5 to 10 ticks and15 to 30 ticks, with POUs up to 2.4% and 3.8%, respectively.

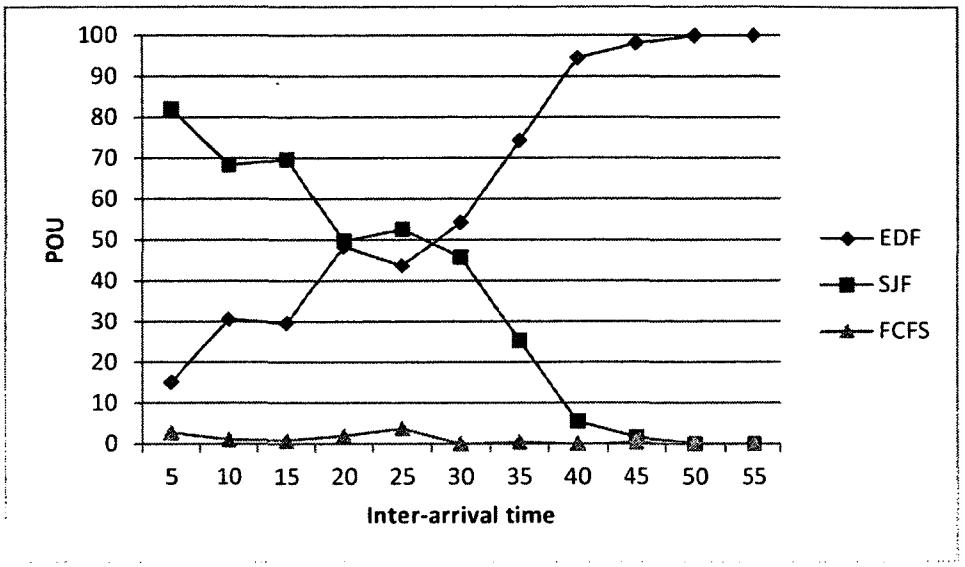

Figure 4.4: POU for the Baseline Experiment

4.2.2 Performance of APAP varying the transaction load

More transactions means more data conflicts and more speculative executions, which increase the system load. We used the parameter settings in Table 4.2, except we increased the number of transactions to 200. As a large number of transactions are executed, data contention and resource contention also increase, resulting in a large number of transaction rejections. Therefore, the performance of all priority protocols, including APAP, degrade. Figure 4.5 shows the PTCT for all protocols for this experiment. APAP continues to exhibit superior performance overall.

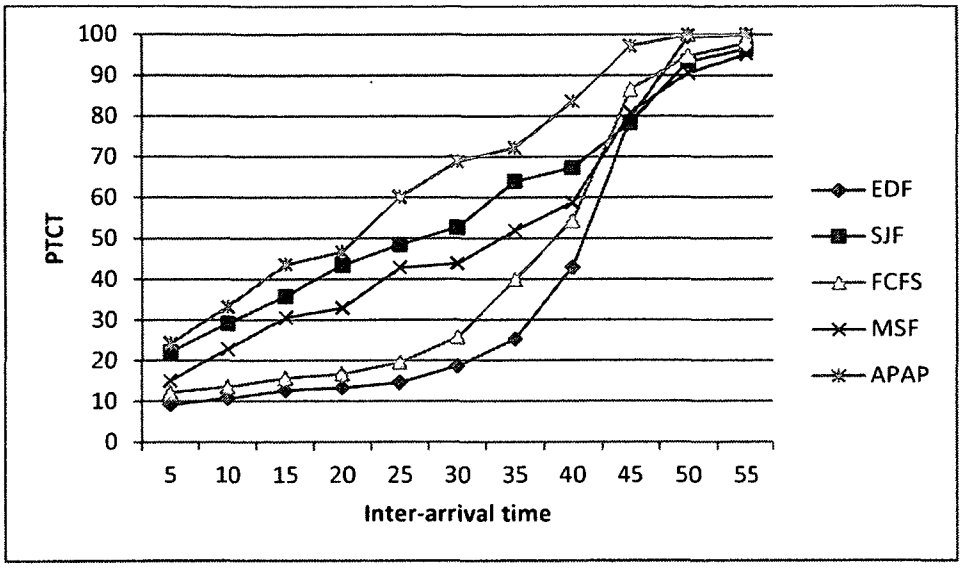

Figure 4.5: PTCT for 200 transactions

The POUs of all protocols with 200 transactions are shown in Figure 4.6. It is clear that APAP runs SJF for more time than the baseline experiment. The FCFS is used during the inter-arrival time ranges from 5 to 15 ticks (maximum 4% POU) and 20 to 30 ticks (maximum 2.2% POU).

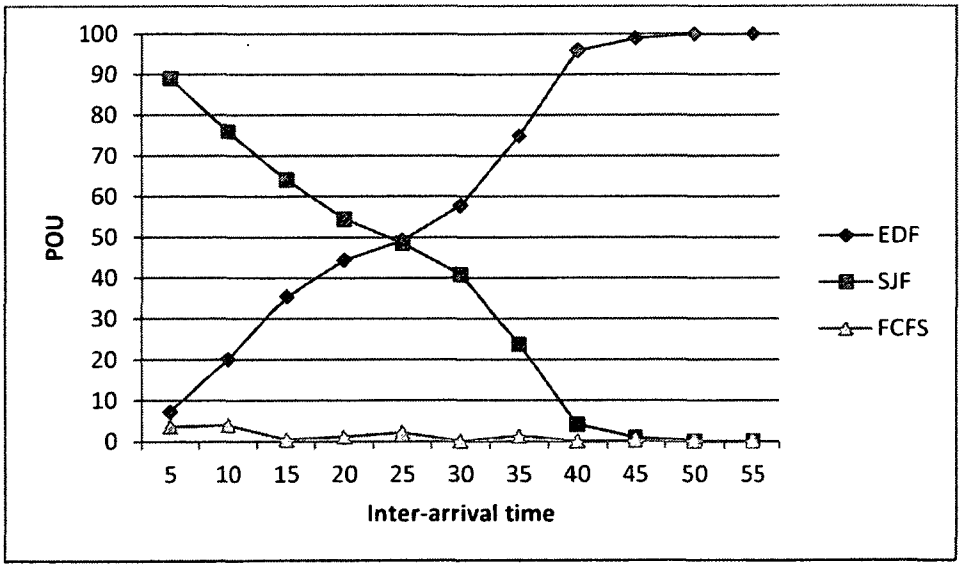

Figure 4.6: POU for 200 transactions

4.2.3 Performance of APAP with reduced page update rate

Page update rate is an important factor in transaction processing and indicates the percentage of write operations in a transaction execution. The write operations use exclusive locks on the data, thus blocking other transactions from accessing that data for a certain time. This blocking time increases the transaction execution time. Consequently, a high or low page update rate affects the system load. We show results from two experiments in this section. In the first experiment, we changed the page update rate to 0 which represents a read-only scenario.

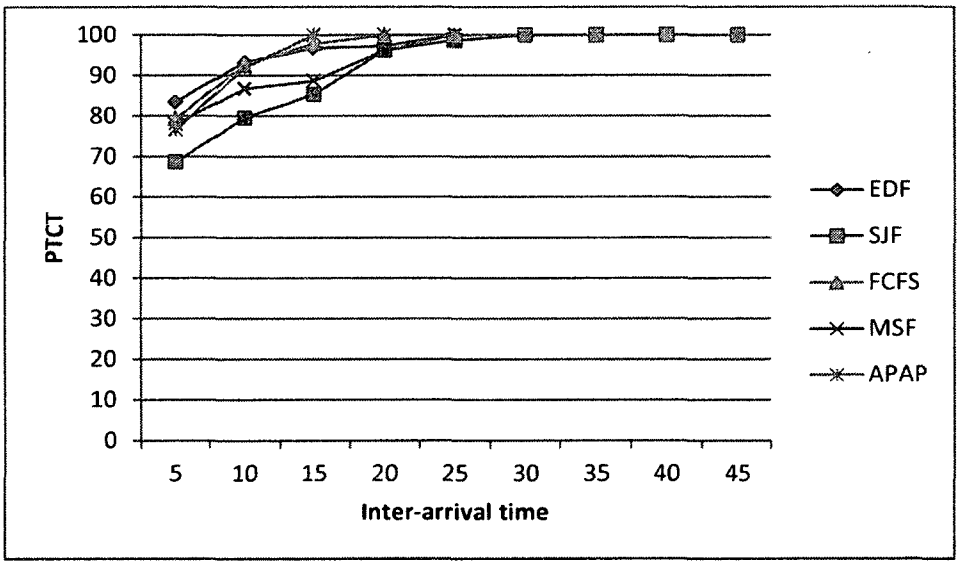

Figure 4.7: PTCT for the zero page update rate

From Figure 4.7, EDF demonstrates maximum PTCT at inter-arrival times of 5 and 10 ticks, because there is no data conflict and thus no blockage of transactions. SJF exhibits higher PTCT than in the baseline experiment, but in comparison to other protocols, this is a poor performance. APAP demonstrates 6.9% less PTCT than EDF at inter-arrival time of 5 ticks. However, when the system runs with an inter-arrival time of more than 10 ticks, APAP

outperforms all priority protocols and achieves PTCT of 100% before EDF. Figure 4.8 shows that APAP runs mostly with SJF until an inter-arrival time of 15 ticks. During that period, the usage of SFJ varied from 85.2% to 76.1%, the usage of EDF increases from 13.8% to 20.3%, and the usage of FCFS varies from 1% to 5%. When the inter-arrival time is more than 25 ticks, APAP only uses EDF because it shows 100% efficiency during that period.

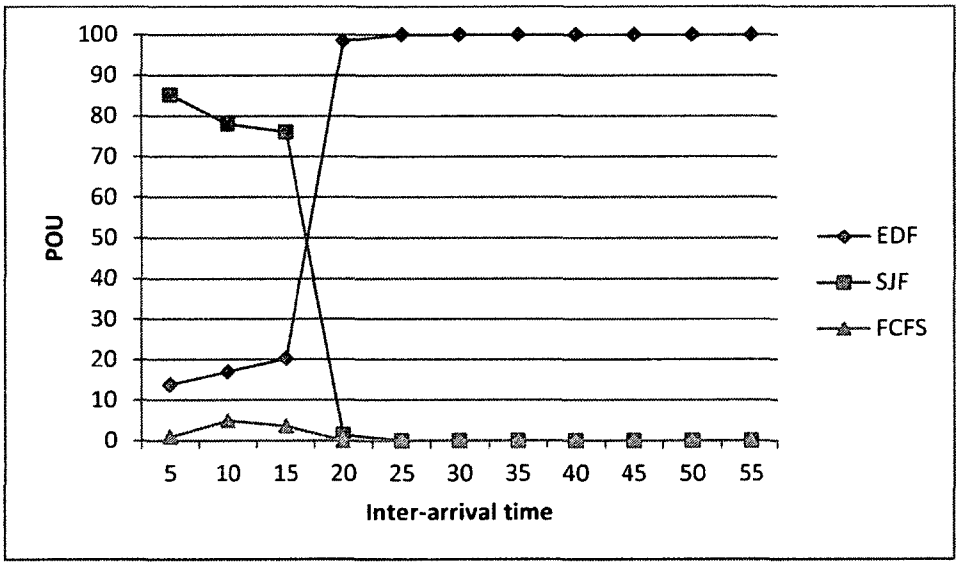

Figure 4.8: POU for the zero page update rate

Next, we changed the page update rate 50 percent which implies that half of the pages accessed are also modified (Figure 4.9). The system load is higher in this case than that of the zero page update rate in the previous experiment. Due to this higher load, EDF performs more poorly during the low inter-arrival times. However, during the high inter-arrival times, EDF outperforms SJF, MSF, and FCFS. APAP exploits all the priority protocols and consistently performs better than the other protocols except when the inter-arrival time is 20 and 25 ticks. At these inter-arrival times, EDF exhibits a PTCT of 3.1% and 4.5% greater

than APAP while transitioning from high to low system load. Beyond inter-arrival time of 30 ticks, APAP performs the same as EDF at 100% PTCT.

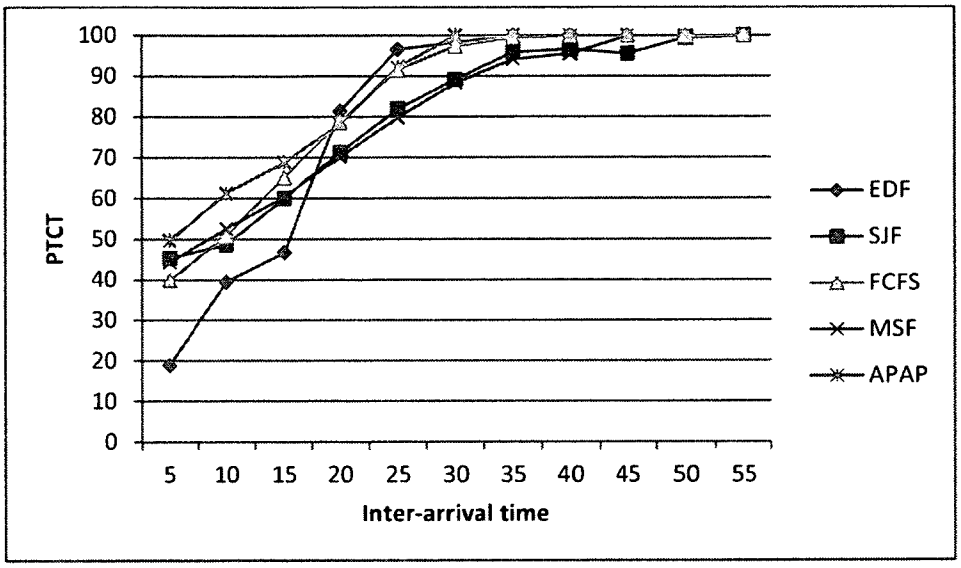

Figure 4.9: PTCT for the 50 percent page update rate

The POU of APAP for the page update rate of 50 percent is shown in Figure 4.10. During the low inter-arrival times and high system load, APAP runs SJF frequently until an inter-arrival time of 25 ticks. The POU of SJF and EDF is 83.52% and 14.8%, respectively, at the 5-tick inter-arrival time. After that, the usage of SJF gradually decreases and that of EDF gradually increases with a crossover point at 25 ticks. The POU of FCFS remains low varying from 0 to 5.35%. APAP only uses EDF beyond the inter-arrival time of 40 ticks.

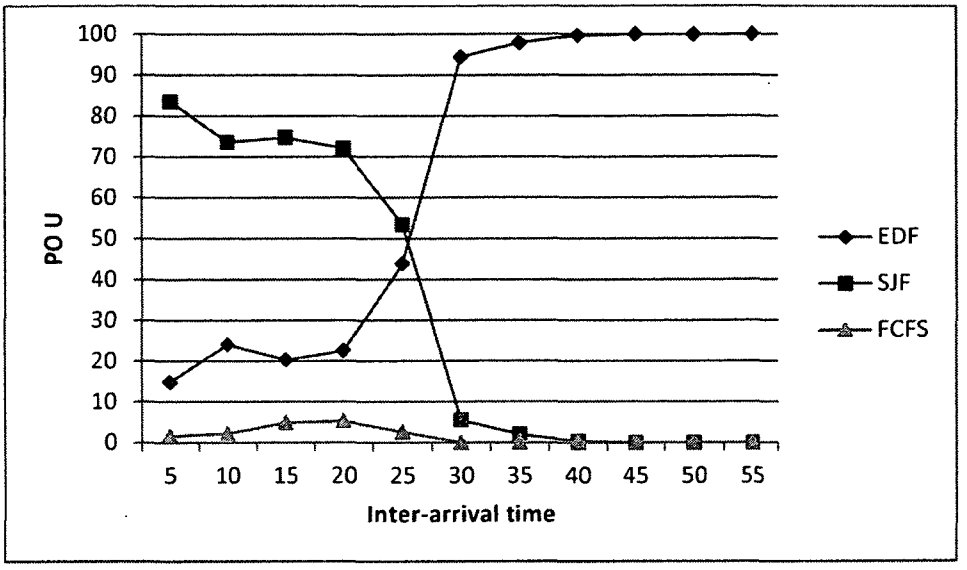

Figure 4.10: POU for the 50 percent page update rate

4.2.4 Performance of APAP with larger cache size

In this experiment, we study the effect of cache size on APAP and other protocols. Cache size is important to the performance of the ASL protocol because it must find space available in the cache or swap disk before it requests a page from the database. If there is not enough space, then the transaction needs to wait. Therefore, cache size affects the system load. When the cache size was increased from 20 to 50 pages (Figure 4.11), the PTCT of all the priority protocols improve because of the larger memory. The PTCT of SJF increases linearly, unlike the baseline experiment which has a drop in PTCT at the 40-tick inter-arrival time. The PTCT for MSF improves 4.45% on average. FCFS has a maximum 20.8% jump at the 30-tick inter-arrival time from the baseline experiment. EDF displays a large increase of PTCT at 35 ticks and outperforms all other protocols. APAP is exceeded by EDF slightly (3.4%) at 35 ticks, but attains a PTCT of 100% at 40 ticks with EDF.

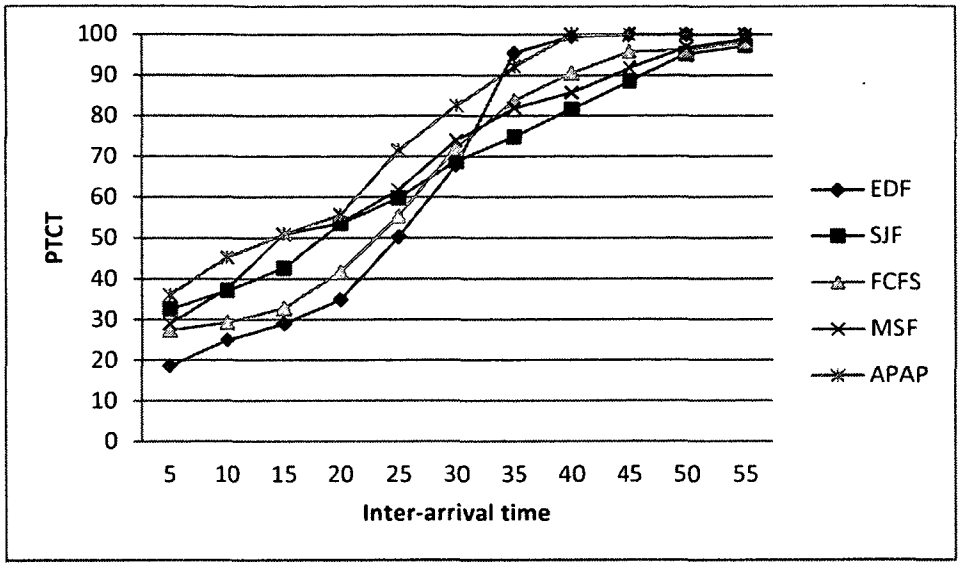

Figure 4.11: PTCT for 50 pages cache size

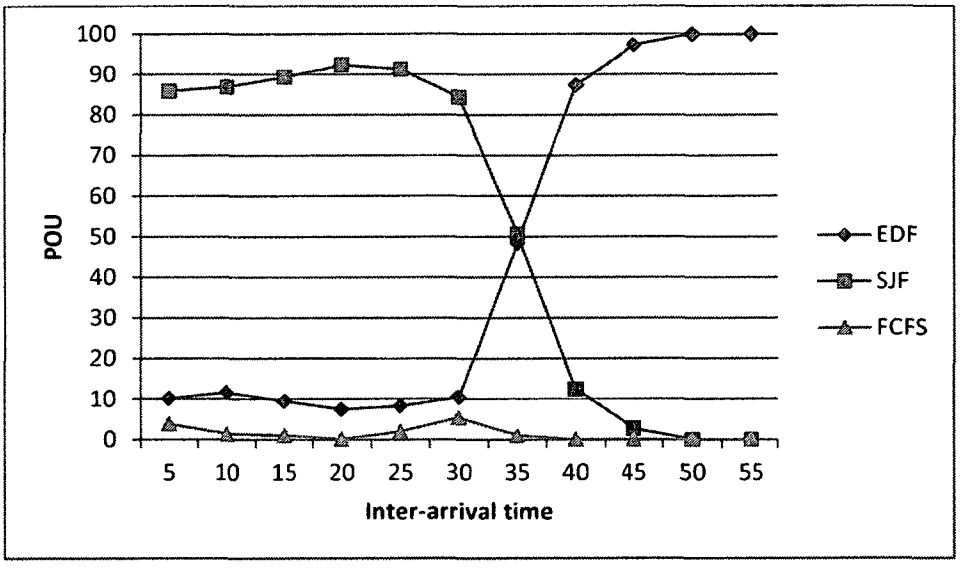

Figure 4.12: POU for 50 pages cache size

Figure 4.12 indicates that the POU of SJF varies from 84.3% to 92.4% until an inter-arrival time of 30 ticks. The POU of EDF varies from 5.6% to 11.7% until 30 ticks. FCFS has maximum 5.34% POU at 30 ticks. When the PTCT of EDF increases sharply at 35 ticks during the transition from high to low system load, the POU of EDF also increases to 48% at

35 ticks. Afterwards, the POU of EDF gradually reaches 100% at an inter-arrival time of 50 ticks.

When we increased the cache size to 100 pages, the performance for all protocols was observed to be almost the same as the performance with a memory size of 50 pages.

4.2.5 Performance of APAP with increased slack time

An increase in slack time relaxes the deadlines and allows enough time for a transaction to complete. When the slack time of the system was increased to 720-3600 ticks, the PTCT of all priority protocols improved (Figure 4.13), since the system now had enough time to execute all transactions. All priority protocols have a PTCT between 30% and 50% when the inter-arrival time is at 5 ticks. As the inter-arrival time increases, the performance of all protocols also improves. EDF outperforms SJF at the 20-tick inter-arrival time, earlier than the baseline experiment. FCFS shows 11.4% and 14.5% more PTCT than EDF during the inter-arrival times of 5 and 10 ticks. However, after 15 ticks FCFS is outperformed by EDF. MSF performed the same as in the baseline experiment, indicating that it is not affected by the increased slack time.

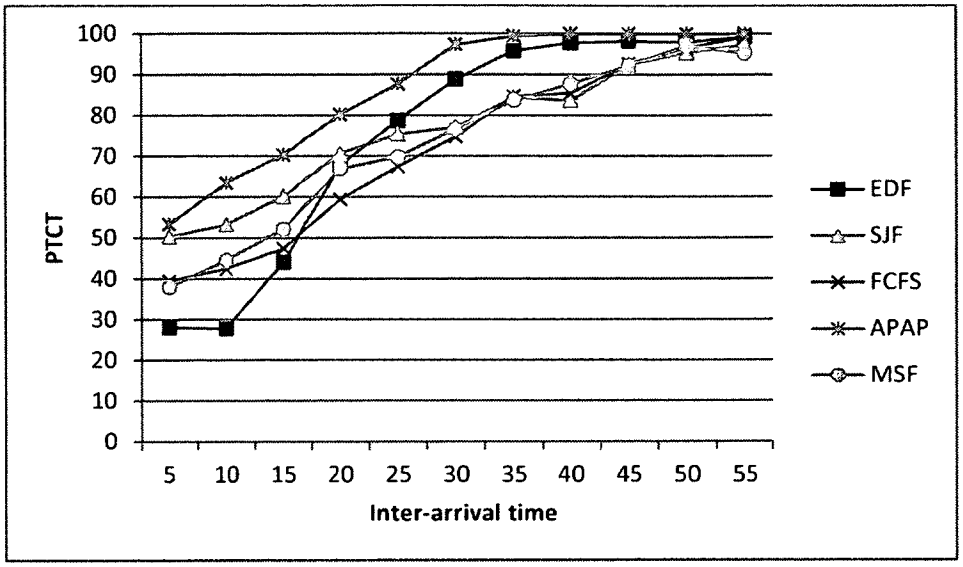

Figure 4.13: PTCT for the 720-3600 ticks slack time

One noticeable finding is the performance of the protocols increase rapidly until the 20-tick inter-arrival time, after which it increases slowly. APAP outperforms all protocols under all load conditions.

Figure 4.14 shows the POUs of the priority protocols in APAP for slack time of 720-3600 ticks. The POUs of SJF and EDF in APAP at the inter-arrival time of 5 ticks are now closer, at 30% and 70%. The POUs then change quickly with the POU of EDF becoming greater than that of SJF after an inter-arrival time of 15 ticks. As the PTCT of EDF slowly increases to 100%, the POU of EDF also slowly goes up to 100%. In APAP the POU of FCFS is up to 4.3% between the inter-arrival times of 5 and 20 ticks. FCFS is also used in the inter-arrival times between 20 and 30 ticks, but only with a POU of 1%.

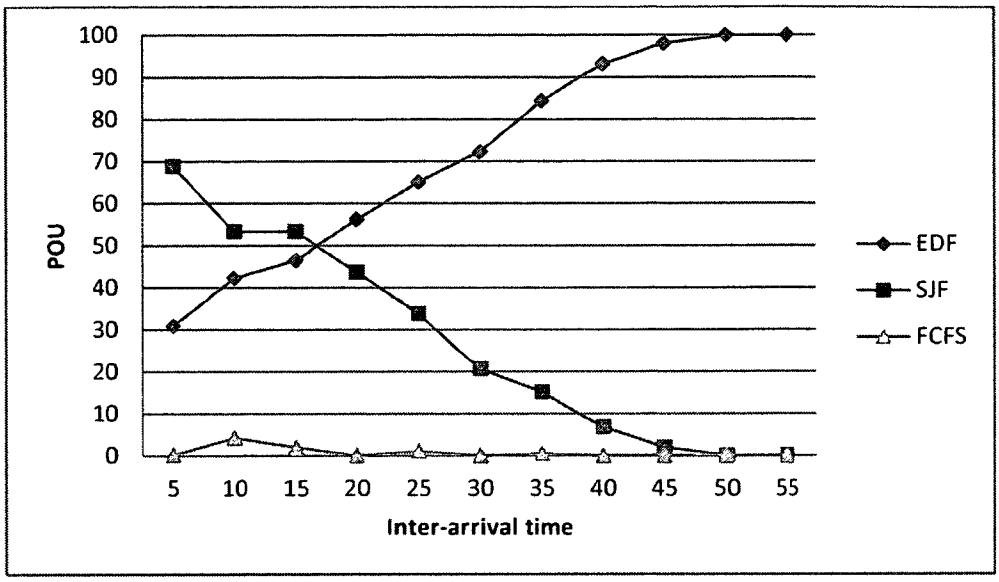

Figure 4.14: POU for the 720-3600 ticks slack time

4.2.6 Effects of system disk space on the performance of APAP

In this experiment, we study the effect of system disk space on the performance of APAP and other protocols. The number of disks affects the data availability for a transaction during execution. If we increase the number of disks, a transaction has a high probability of getting required data in the local disk, which reduces the blocking and execution times of the transaction. Therefore, the number of disks affects the system load. We increased the number of disks from 2 to 4 for this experiment. Because of the increase in resources, there is a large change in the PTCTs of all protocols over the baseline experiment (Figure 4.15). However, during low load when the inter-arrival time is between 5 and 10 ticks, SJF outperforms EDF, FCFS, and MSF. After 10 ticks, the performance curve of EDF shows a steep rise confirming its superior performance during low load. FCFS and MSF always perform close to each other and surpass SJF after 10 ticks. APAP outperforms all priority protocols under most load conditions, except an inter-arrival time of 20 ticks where EDF performs slightly (4%) better.

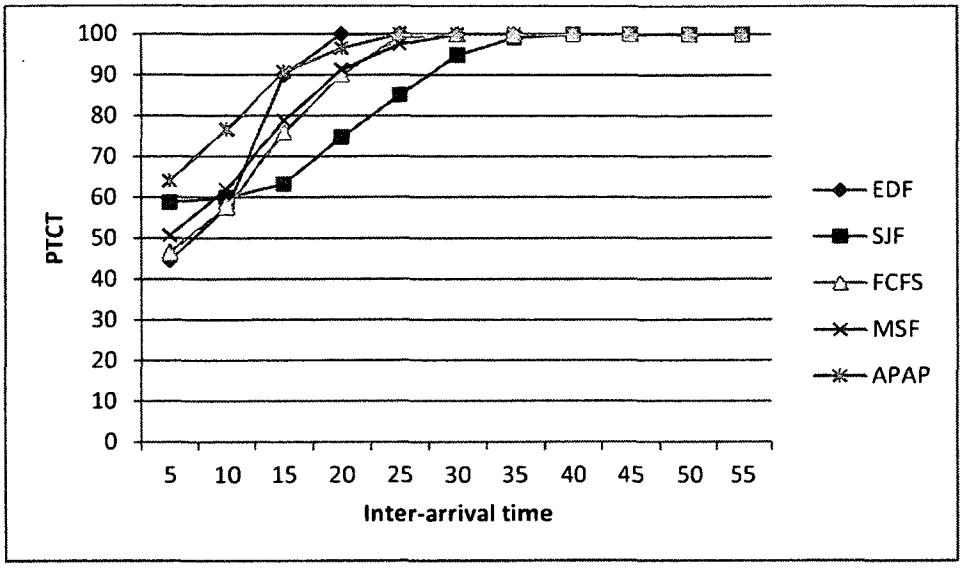

Figure 4.15: PTCT for 4 disks

The POUs of EDF and SJF at lower inter-arrival times (5 to 10 ticks) are at 37.6% and 56.5%, respectively (Figure 4.16). As the system is a suitable environment for EDF when the inter-arrival time is more than 10 ticks, APAP gradually switches to using EDF more than SJF. After 30 ticks, APAP only runs EDF. APAP also uses FCFS 5.85% at the inter-arrival time of 5 ticks, which gradually levels to around 1% between 15 and 20 ticks before going to zero at 25 ticks.

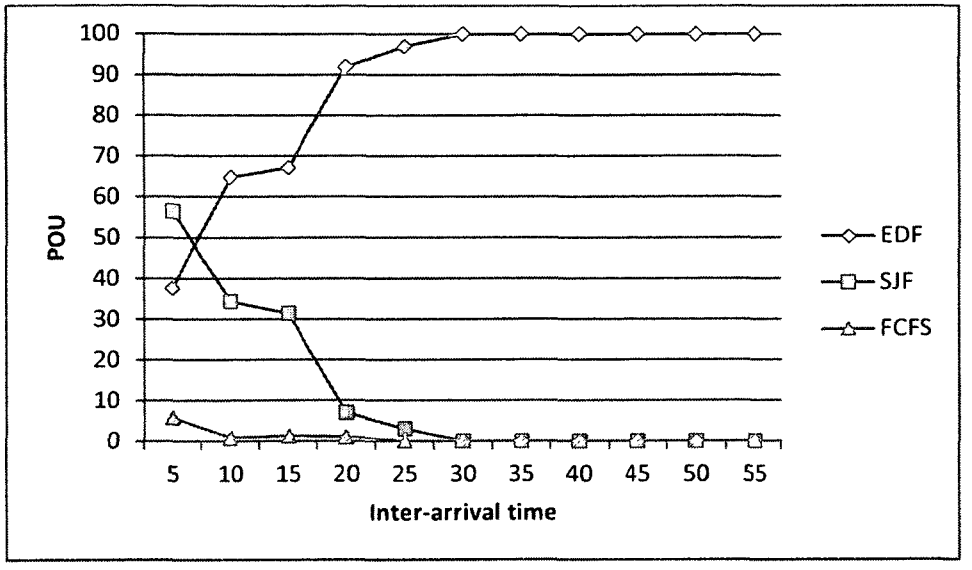

Figure 4.16: POU for 4 disks

4.2.7 Effect of system distribution on the performance of APAP

An increase in the number of nodes in a DRTDBS increases data availability and load distribution. Therefore, it reduces the overall system load. In the first experiment, we changed the number of nodes from 7 to 11 (Figure 4.17). Due to the high data locality, the PTCT for EDF is now almost same as SJF even at an inter-arrival time of 5 ticks. However, the performance of EDF does not increase at a high rate when the inter-arrival time is more than 5 ticks. FCFS shows 7.7% more PTCT than EDF at an inter-arrival time 20 ticks. MSF exhibits the lowest PTCT until the 15-tick inter-arrival time after which it performs close to SJF. In fact, MSF performs the same as it did in the baseline experiment, because slack time does not improve by increasing the number of nodes. APAP continues to outperform all other priority protocols.

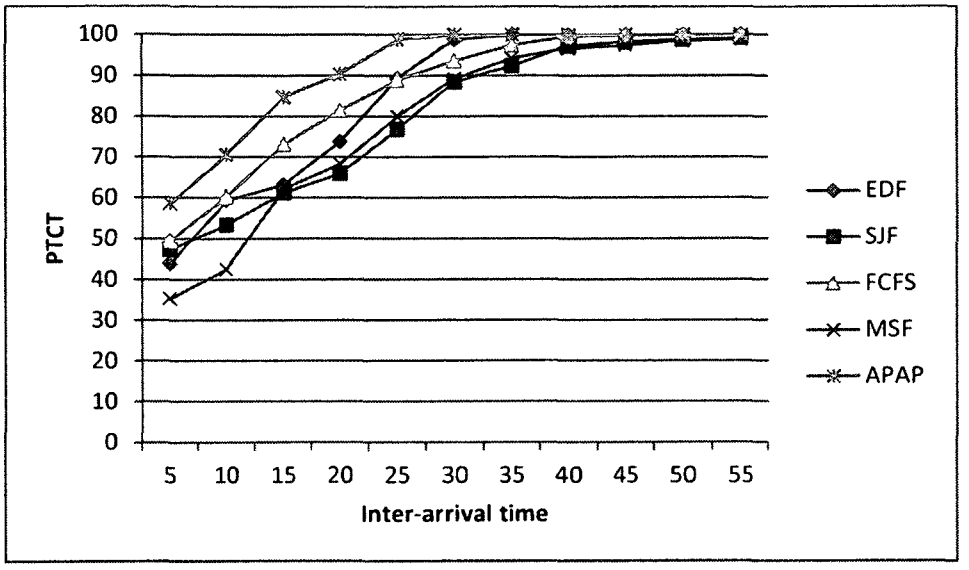

Figure 4.17: PTCT for 11 nodes

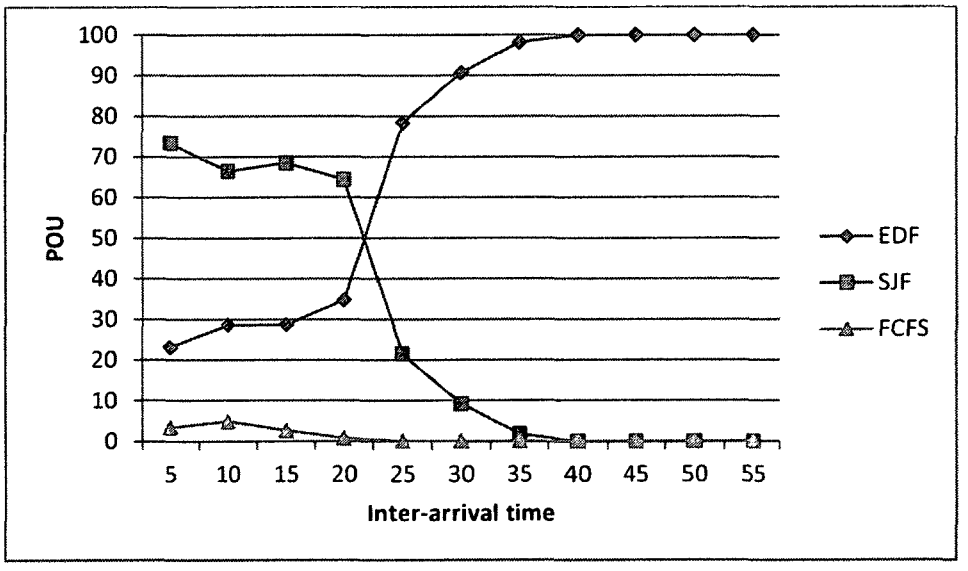

Figure 4.18: POU for 11 nodes

Figure 4.18 indicates that the POU of SJF decreases from 73.4% to 66.5% during the inter-arrival times between 5 and 10 ticks, whereas the POU of EDF increases from 23.2% to 28.6% during the same period. After that, the POUs of SJF and EDF remain more or less flat, until the inter-arrival time of 20 ticks when APAP clearly shows preference for EDF over

other priority protocols. The POU of FCFS increases to 4.9% at 10 ticks, then decreases gradually until it becomes zero at 35 ticks.

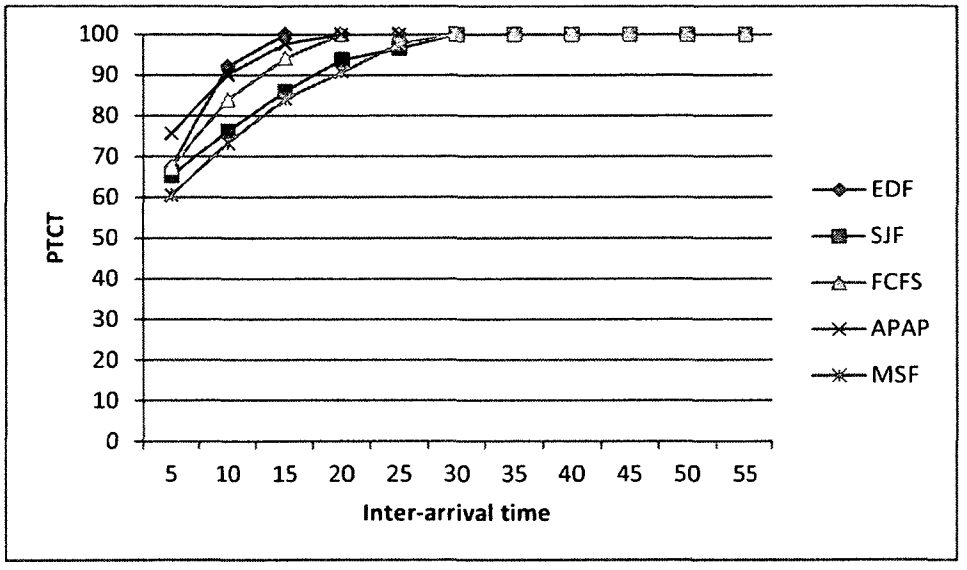

Figure 4.19: PTCT for 15 nodes

Figure 4.19 shows performance results when the number of nodes was further increased to 15. Since there is high data locality, EDF surpasses SJF, MSF, and FCFS consistently, and achieves a PTCT of 100% at the inter-arrival time of 15 ticks. APAP shows 8.23% more PTCT than EDF at 5 ticks, but 2% and 2.3% less PTCT than EDF at 10 and 15 ticks, respectively. Hereafter, APAP attains 100% of PTCT.

Because of the superior performance of EDF, APAP prefers EDF over SJF from the beginning which proves the adaptive nature of APAP (Figure 4.20).

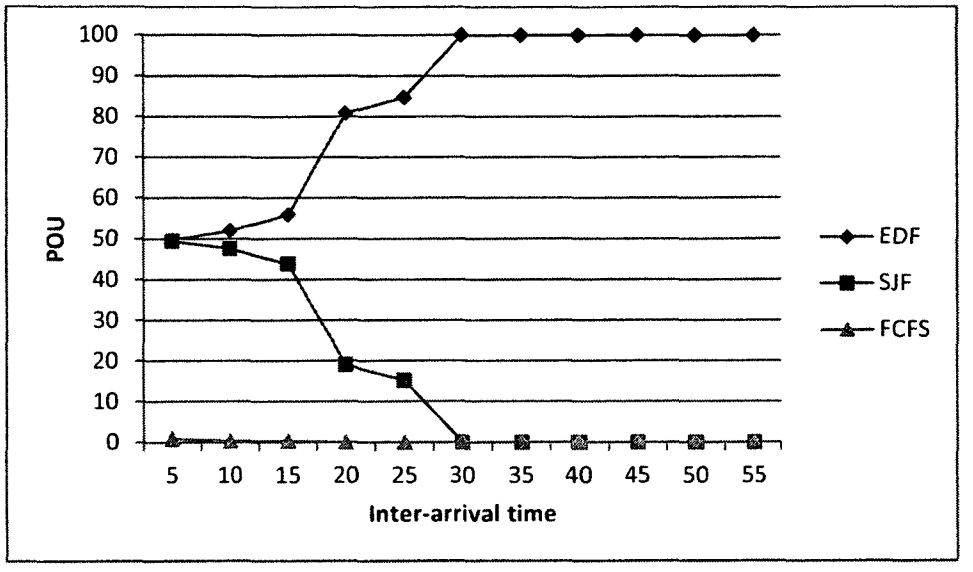

Figure 4.20: POU for 15 nodes

4.2.8 Effect of increasing the number of processors

In this experiment, we study the effect of more than one processor in a node on APAP. We increased the number of processors from 1 to 2, and no significant improvement in performance was observed for any of the protocols including APAP (Figure 4.21). It should be noted that the number of processors is not the real bottleneck in our experiments. As we discussed, the performance of the protocols is greatly affected by the page update rate, cache size, slack time etc.; increasing the number of processors does not affect the performance of priority protocols. The PTCT of APAP increases to a maximum of 5.7% at the 20-tick inter-arrival time over the single processor experiment. The maximum increase for EDF is 5.9% at the 35-tick inter-arrival time and 4.6% for SJF at 40 ticks.

The POUs in Figure 4.22 for two processors are visibly different than the single processor, because slight performance changes of the protocols increase the usage of EDF at the inter-arrival times from 5 to 10 ticks and decrease the POU at the inter-arrival time of 20 ticks.

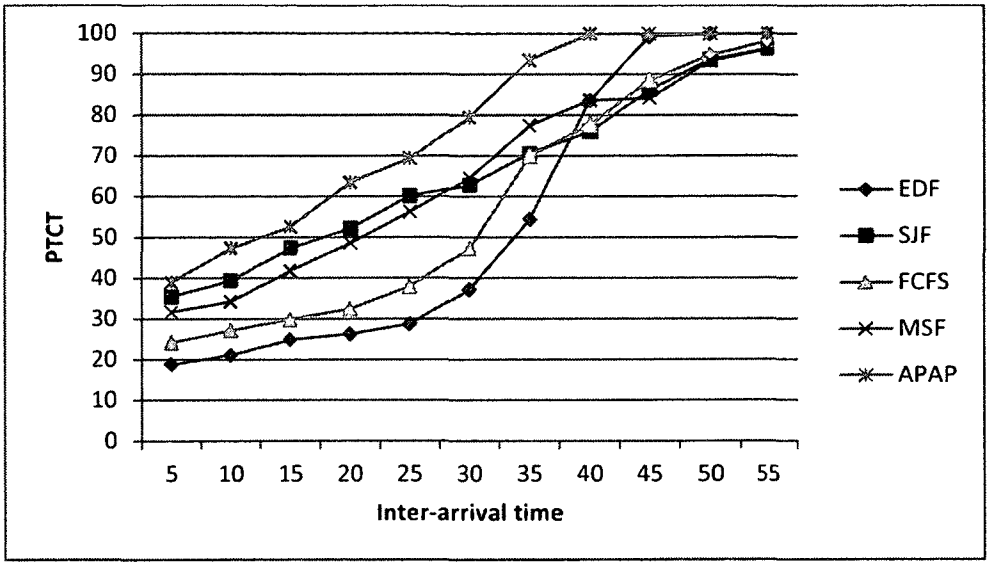

Figure 4.21: PTCT for 2 processors

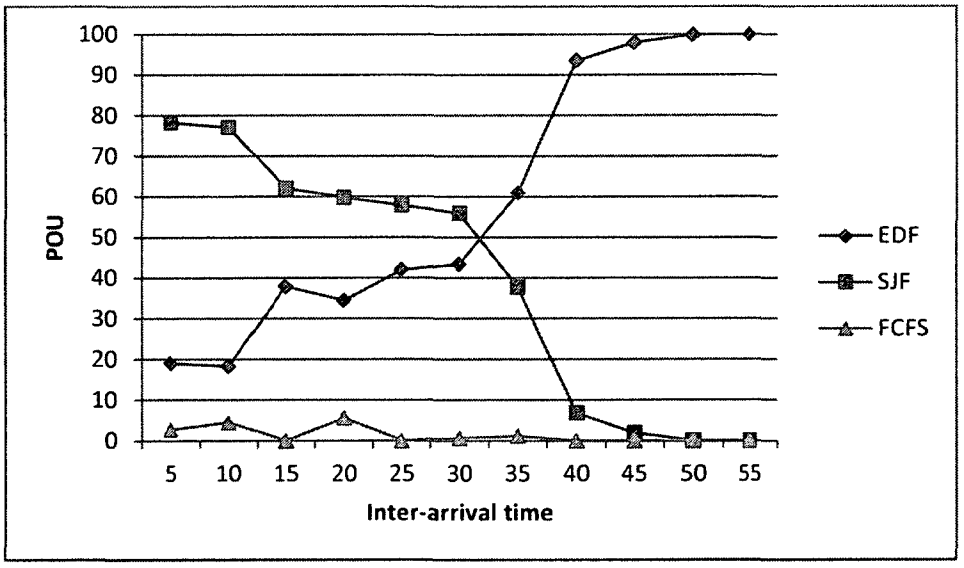

Figure 4.22: POU for 2 processors

4.2.9 Effect of network topologies on the performance of APAP

Network topologies affect the performance of ASL protocol [54]. In order to study this with APAP, that is, using dynamic switching between priority protocols, we changed the

network topology from binary tree to 2D-Torus using 16 nodes. Due to the high data locality, all priority protocols perform better than the baseline experiment with 7 and 15 nodes. Here SJF and FCFS perform close to each other and better than EDF when the inter-arrival time is between 5 and 10 ticks (Figure 4.23). However, EDF outperforms SJF, MSF, and FCFS priority protocols when the inter-arrival time is more than 10 ticks and the system load is low. APAP demonstrates the maximum PTCT consistently.

As shown on Figure 4.24, between the inter-arrival times of 5 and 10 ticks, APAP performs with SJF around 75%-70% and with EDF around 25%-20%. After the inter-arrival time of 10 ticks, the POU of SJF decreases and the POU of EDF increases rapidly until it is used 100% by APAP at the inter-arrival time of 25 ticks. The POU of FCFS increases to 7.9% at the inter-arrival time of 10 ticks and goes back to zero at 15 ticks. We get similar results for 2D-Mesh, Hypercube-4, and ring topologies.

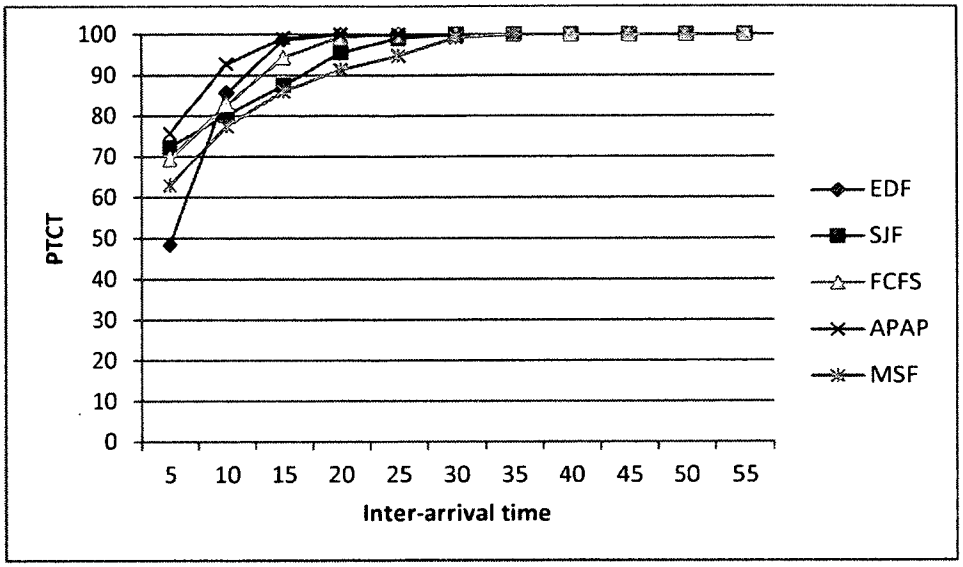

Figure 4.23: PTCT for the 2D-torus network topology

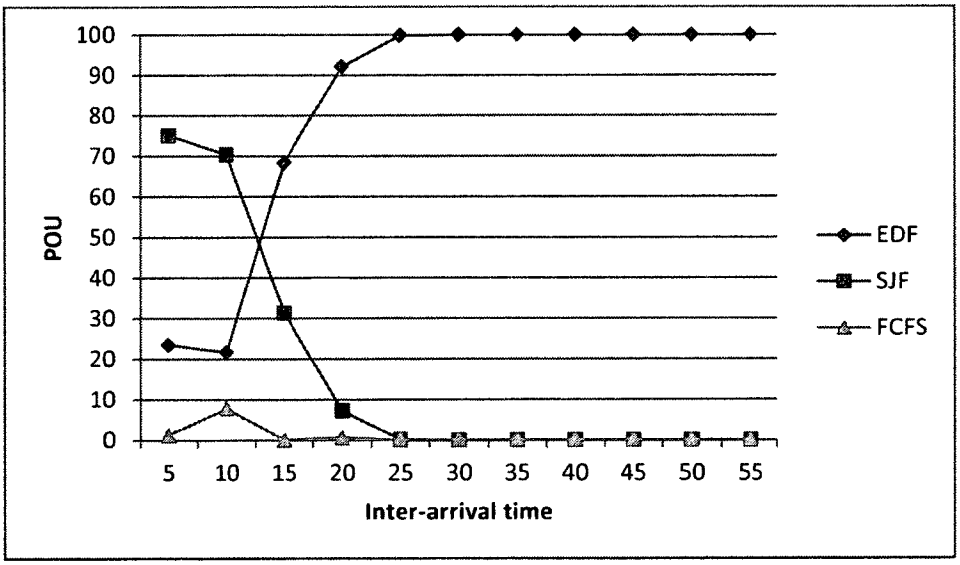

Figure 4.24: POU for the 2D-torus network topology

4.3 Summary

- APAP outperformed all priority protocols especially in high load conditions. However, in some low load conditions EDF completed 2% to 6% more transactions than APAP.

- A change in the page update rates from 100% to 0%, disk size 2 to 4, and the number of nodes from 7 to 15 greatly improved the performance of all priority protocols.
- An increase in the number of processors from one to two in each node and the cache size from 50 to 100 did not have a significant effect on the performance of any priority protocol.
- The network topologies we tested had similar effect on all priority protocols.
- When the inter-arrival time was increased, the POU of EDF increased, and always reached 100%.

Chapter 5

Conclusion

CCPs ensure consistency of a database when multiple transactions request the same data in a database. In a distributed environment, CCPs also need to coordinate between transactions and their sub-transactions, which execute at different sites. The ASL protocol is a CCP for a DRTDBS which follows the underlying structure of Speculative Locking (SL) protocols as well as provides additional features (discussed in Chapter 2). ASL outperformed SLs, but its performance degrades when the system is in a high load condition [9].

A database system uses priority protocols when a CCP coordinates transaction processing to order transactions. EDF is an optimal priority protocol for ordering transactions in a database system. However, EDF's performance also degrades in high load conditions. On the other hand, some other priority protocols (such as SJF) perform better than EDF in such conditions. A common trend to optimize the performance of priority protocols is dynamically changing from one to another according to the system load conditions. However, the existing solutions are not amenable to a DRTDBS.

Our proposed method, Adaptive Priority Assignment Protocol (APAP), improved the performance of a CCP to a large extent in a DRTDBS under all load conditions. APAP switches between the priority protocols according to the system load using a load range table

which contains load values where a given priority protocol is expected to perform better. This is done at run time. The ASL protocol is used as the underlying CCP for all of our experiments. We observed ASL's improved performance by varying the number of transactions, cache sizes, number of processors, page update rates, number of disks, and network topologies. APAP outperformed all priority protocols in most conditions. In some low load conditions, when the inter-arrival time varied, EDF exhibited 2% to 6% more completed transactions than APAP.

We can summarize the observations in the following way:

1. APAP yields an overall superior performance when system load is high.
2. Longer slack time and more resources (more disks nodes, or cache size) decrease the system load, improving performance of all priority protocols. However, increasing cache size beyond a certain value does not further improve the performance of priority protocols.
3. In some low load conditions, EDF performs slightly better than APAP. However, any improvement attempt in these load conditions degrades the overall performance of APAP. Therefore, the fact that EDF outperformed APAP is considered as a limitation of APAP and can be considered negligible.

As APAP switches between priority protocols according to the system load, we also observed the percentage of usage (POU) of the priority protocols in APAP. The POU charts indicate that:

1. APAP uses mostly SJF when the system load is high.
2. When the system load decreases, APAP increases the use of EDF.

5.1 Future Work

We found that changing priority protocols at run time improves the performance of a CCP in real-time distributed database systems. However, in this research, we assumed that there exists a single workload generator. In a practical application, transactions can be generated at more than one node. Therefore, we can further our research by testing the performance of APAP under multiple workload generators. Another area of improvement would be the use of replicated databases. They improve the performance of a transaction processing system by increasing data locality where a data item has one or more copies at different nodes. However, the test bed of this research, DRTTPS, is not designed to handle replicated databases. We can modify DRTTPS for this scenario and test the performance of APAP. Finally, a real-time distributed system is also susceptible to failure. In our experiments, the DRTTPS was not built as a fault tolerant system which continues to function even when some components fail. Therefore, we can test how APAP performs in such an environment.

Bibliography

[1] U. Shanker, M. Misra and A. K. Sarje, "Distributed real time database system: background and literature review," *Distributed and Parallel Databases,* vol. 23, pp. 127-149, January 2008.

[2] P. A. Bernstein and . E. Newcomer, Principles of transaction processing (Second Edition), San Francisco, California: Morgan Kaufmann Publishers, Inc., 2009.

[3] B. Kao and H. Garcia-Molina, "An Overview of Real-Time Database Systems," in *proceedings of NATO Advanced Study Institute on Real-Time Computing*, St. Maarten, Netherlands., 1992, pp. 463-486.

[4] A. Buchmann, "Real-Time Databases," *Encyclopedia of database technologies and applicatios* , pp. 524-529, 2005.

[5] C. F. Yeung and S. L. Hung, "A new deadlock detection algorithms for distributed real-time database system.," in *14th Sysmposium on Reliable Distributed System.*, Bad Neuenahr, 1995, pp. 146-153.

[6] P. A. Bernstein, V. Hadzilacos and N. Goodman, Concurrency Control and Recovery in Database Systems, M. A. Harrison, Ed., Manlo Park, California: Addison-Wesley Publishing Company, 1987, pp. 265-304.

[7] P. K. Reddy and Kitsuregawa, Masaru, "Speculative Locking Protocols to Improve Performance for Distributed Database Systems," *IEEE Trans. on Knowl. and Data Eng.,* vol. 16, pp. 154-169, February 2004.

[8] W. Haque and P. R. Stokes, "Adaptive speculative locking protocol for distributed real-time database system," in *Procedings of the 19th IASTED Interntional Conference on Parallel and Distributed Computing and Systems*, Cambridge, Massachusetts, 2007, pp. 382-390.

[9] P. R. Stokes, Design and Simulation of an Adaptive Concurrency Control Protocol for Distributed Real-Time Database System, Prince George, Canada: Master's Thesis,University of Northern British Columbia, 2007.

[10] O. Ulusoy, "Distributed Concurrency Control," in *Real-time database systems: architecture and techniques*, Norwel, Massachusetts 02061, USA, Kluwer Academic Publishers, 2001, pp. 205-215.

[11] D. Agrawal, A. El Abbadi and R. Jeffers, "Using delayed commitment in locking protocols for real-time databases," in *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, San Diego, California, USA, 1992, pp. 104-113.

[12] A. E. Abbadi and S. Toueg, "Availability in partitioned replicated databases," in *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, Cambridge, Massachusetts, United States, 1986, pp. 240-251.

[13] K. Ramamritham, "Real-Time Databases," *International Journal of Distributed and Parallel Databases,* vol. 1, pp. 199-226, 1996.

[14] R. Obermacrk, "Disctributed Deadlock Detection Agorithm," *ACM Transactions on Database System,* vol. 7, pp. 187-208, 1982.

[15] M. Singhal, "Deadlock Detection in Distributed Systems," *Computer*, vol. 22, no. 11, pp. 37-48, 1989.

[16] W. Ding and R. Guo, "Design and Evaluation of Sectional Real-Time Scheduling Algorithms Based on System Load," in *The 9th International Conference for Young Computer Scientists*, Hunan, 2008, pp. 14-18.

[17] D. Levine, C. Gill and D. Schmidt, "Dynamic Scheduling Strategies for Avionics Mission Computing," in *Digital Avionics Systems Conference*, Bellevue, WA , USA, 1998, pp. 1-8.

[18] L. Gruenwald, M. Montealegre and C. N. Lau, "Performance Comparison of Scheduling Techniques to Manage Transactions for Real-Time Mobile Databases in Ad Hoc Networks," in *http://paginas.usco.edu.co/proyeccion/Documentos/entornos/entornos20/ArticuloMatildeMontealegre.pdf.*

[19] [Online]. Available: http://www.personal.kent.edu/~rmuhamma/OpSystems/os.html.

[20] J. R. Haritsa, M. Livny and M. J. Carey, "Earliest Deadline Scheduling for Real-Time Database Systems.," in *in Proc. IEEE Real-Time Systems Symposium*, San Antonio, TX , USA, 1991, pp. 232-243.

[21] K. W. Lam and S. L. Hung, "A pre-emptive transaction scheduling protocol for controlling priority inversion," in *Third International Workshop on Real-Time Computing Systems and Applications*, Seoul, 1996, pp. 144 - 151.

[22] L. sha, R. Ragunathan and L. John, "Priority Inheritance Protocols: An approach to Real-Time Synchronization," *IEEE transaction on Computers,* vol. 39, pp. 1175-1185, September 1990.

[23] Y. S. Philip , . K.-l. Wu , K.-j. Lin and S. H. Sang , "On real-time databases: Concurrency control and scheduling," *Proceedings of the IEEE,* pp. 140-157, 1994.

[24] Z. Kedem and A. Silberschatz, "Controlling concurrency using locking protocols," in *20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979, pp. 274 - 285.

[25] Z. M. Kedem and A. Silberschatz, "Locking Protocols: From Exclusive to Shared Locks," *Journal of the ACM (JACM),* vol. 30, no. 4, pp. 787-804, Oct 1983.

[26] D. Skeen, "Nonblocking commit protocols," in *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, Ann Arbor, Michigan, 1981, pp. 133-142.

[27] R. Gupta, H. Jayant and K. Ramamritham, "Revisiting Commit Processing in Distributed Database Systems," *Proceedings of the 1997 ACM SIGMOD international conference on Management of data,* vol. 26, no. 2, pp. 486-497, 1997.

[28] Y. Al-Houmaily, P. K. Chrysanthis and S. P. Levitan, "An argument in favor of the presumed commitprotocol.," in *Proceedings of the IEEE International Conference on Data Engineering*, Birmingham, 1997, pp. 255--265.

[29] J. R. Haritsa and K. Ramamritham, "Adding PEP to real-time distributed commit processing," in *Procedings of the 21st IEEE Real-time Systems Symposiums*, Orlando, USA, 2000, pp. 37-46.

[30] M. Abdallah, R. Guerraoui and P. Pucheral, "One-Phase Commit :Does It Make Sense ?," in *Procedings of the International Conference on Parallel and Distributed Systems*, Tainan,

Taiwan, 1998, pp. 14-16.

[31] M. Atif, "Analysis and Verification of Two-Phase Commit & Three-Phase Commit Protocols," in *International Conference on Emerging Technologies*, 2009, pp. 326 -331.

[32] P. A. Bernstein and N. Goodman , "Concurrency Control in Distributed Database Systems," *ACM Comput. Surv.,* vol. 13, no. 2, pp. 185-221, 1981.

[33] L. Sha, R. Rajkumar and J. P. Lehoczky, "Concurrency Control for Distributed Real-Time Databases," *SIGMOD RECORD,* vol. 17, pp. 82-98, March 1988.

[34] G. Buttazzo, M. Spuri and F. Sensini, "Value vs. Deadline Scheduling in Overload Conditions," in *Procedings of the 16th IEEE Real-Time Systems Symposium*, Pisa, Italy, 1995, pp. 90-99.

[35] S. Asiaban, M. E. Moghaddam and M. Abbaspur, "A Real-Time Scheduling Algorithm for Soft Periodic Tasks," *JDCTA: International Journal of Digital Content Technology and its Applications,* vol. 3, pp. 100-111, 2009.

[36] W. Li, K. Kavi and R. Akl, "A non-preemptive scheduling algorithm for soft real-time systems," *Computers and Electrical Engineering,* vol. 33, no. 1, pp. 12-29, Januray 2007.

[37] J. R. Haritsa, K. Ramamritham and G. Ramesh, "The PROMPT real-time commit protocol," *IEEE Transactions on Parallel and Distributed Systems,* vol. 11, no. 2, pp. 160-181, 2000.

[38] M. Abdallah and P. Pucheral, "A Non-Blocking Single-Phase Commit Protocol for Rigorous Participants," in *In Proceedings of the National Conference Bases de Donnes Avances*, Grenoble, France, 1997.

[39] A. T. Tai and J. F. Meyer, "Performability Management in Distributed Database Systems: An Adaptive Concurrency Control Protocol," in *Proceedings of the 4th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, San Jose, CA , USA, 1996, pp. 212-216.

[40] P. K. Reddy and M. Kitsuregawa, "Improve performance in distributed database systems using speculative transaction processing," in *Second IASTED European Parallel and Distributed Systems Conference*, Vienna, Austria, 1998, p. 275– 285.

[41] T. Marr and et al., "Hyper-threading technology architecture and microarchitecture," *Intel Technology Journal,* vol. 6, pp. 4-15, 2002.

[42] T. Ragunathan and P. K. Reddy, "Performance enhancement of Read-only transactions using speculative locking protocol," in *Sixth Annual Inter Research Institute Student Seminar in Computer Science (IRISS 2007)*, Hyderabad, India, 2007.

[43] T. Ragunathan and P. K. Reddy, "Improving the performance of read-only transactions through asynchronous speculation," in *HPCS 2008*, San Diego, CA, USA, 2008, p. 467–474.

[44] T. Ragunathan and . P. Krishna Reddy, "Exploiting Semantics and Speculation for Improving the Performance of Read-only Transactions," in *International Conference on Management of Data COMAD 2008*, Mumbai, India, December 17–19, 2008, pp. 162-173.

[45] T. Ragunathan, P. Krisna Reddy and M. Goyal, "Semantics-Based Asynchronous Speculative Locking for Improving the Performance of Read-only Transactions," in *Proceedings of the 2010 Spring Simulation Multiconference*, Orlando, Florida, 2010, pp. 238:1--238:4.

[46] W. Haque and P. R. Stokes, "Simulation of a complex distributed real-time database system," *Proceedings of the 2007 spring simulation multiconference,* vol. 2, pp. 359-366, 2007.

[47] A. Maria, "Introduction to Modeling and Simulation," in *Winter Simulation Conference*, Atlanta, Georgia, United States, 1997, pp. 7-13.

[48] J. Banks and S. J. Carson, "INRODUCTION TO DISCRETE_EVENT SIMULATION," *Proceeding of the 1986 Winter Simulation Conference proceedings,* pp. 17-23, 1986.

[49] A. Silberschatz, H. F. Korth and S. Sudarshan., Database Systems Concepts, New York, NY, USA: McGraw-Hill, Inc., 2006.

[50] Sun MicroSystem, "Java Language Specification," *Palo Alto California : Sun Microsystem,* vol. 1.4.2, 2003.

[51] C. S. Mullins, "Defining Database Performance," October 2010. [Online]. Available: http://www.dbta.com/Articles/Columns/DBA-Corner/Defining-Database-Performance-70236.aspx.

[52] G. C. Buttazzo, J. A. Stankovic , S. Superiore and S. Anna, "RED: Robust Earliest Deadline Scheduling," *Proc. of 3rd International Workshop on Responsive Computing Systems,* pp. 100-111, 1993.

[53] C. L. Liu and J. W. Layland , "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM (JACM),* vol. 20, no. 1, pp. 46-61, Januray 1973.

[54] W. Haque, Q. Pai and S. N. Mahmud, "Effect of Network Topology on the performance of Adaptive Speculative Locking Protocol," in *Parallel and Distributed Computing and Systems*, Dallas, USA, 2011.