

Priority and Deadline Assignment to Triggered Transactions in Distributed Real-time Active Databases*

Kam-yiu Lam, Gary C.K. Law and Victor C.S. Lee
Department of Computer Science
City University of Hong Kong
83 Tat Chee Avenue, Kowloon
HONG KONG
email: cskylam@cityu.edu.hk
fax: 852-2788-8614
tel: 852-2788-9807

Abstract

A Distributed Real-time Active Database System (DRTADBS) reacts to the critical events occurred in the external environment by triggering of transactions. In this paper, the priority and deadline assignment to triggered transactions under two coupling modes, the deferred and immediate, in a DRTADBS is discussed. Two new approaches, the data state dependent (DSD) and the transaction-data deadline (TDD) are proposed to assign criticality and deadlines to the triggered transactions, respectively. In the DSD approach, the criticality of a triggered transaction is defined according to the state of the temporal data object which is responsible for its triggering. The objective of the DSD approach is to increase the number of commit achieved by the triggered transactions especially the more critical ones. The performance of these two approaches under the two coupling modes has been investigated. The results show that the DSD approach is more effective under the immediate coupling mode than under the deferred coupling mode due to the late creation of the triggered transactions under the deferred coupling mode. The TDD approach can improve the system performance under both deferred and immediate coupling mode.

1 Introduction

In recent years, there is a strong move toward integrating Real-time Database Systems [Ozso95, Yu94] with Active Database Systems [Wido96a] with an attempt to provide a better solution to their potential applications [Bern96, DART96], e.g., the applications which have to react to the changes in the external environment in real-time manner. The new area is called *Real-time Active Database System (RTADBS)* [Kort90]. If the database is distributed in different sites, the system is called *Distributed Real-time Active Database System (DRTADBS)* [Andl96, Buch92]. The scheduling of

* An earlier version of the paper appears in the Proceedings of the Second International Workshop on Active, Real-Time, Temporal Database Systems, Como, Sept. 1997.

transactions in a DRTADBS is complicated by the distributed nature of the transactions, the additional overhead for the management of distributed data objects and the unpredictability of the system workload due to triggering of transactions. Some common applications of DRTADBS are international program stock trading systems and battlefield management systems.

In a DRTADBS, transactions that are dynamically created by other transactions in react to the changes (or the occurrences of critical events) in the external environment are called *triggered transactions*. For example in a program stock trading system, applications create transactions to access the database and check for trading opportunities. Once the conditions are satisfied, e.g., the price of a particular stock is lower than a certain value, an action will be triggered (in form of transaction) for buying or selling of the stocks.

The transactions, which have triggered other transactions, are called *triggering transactions*. The relationship between the triggering and the triggered transactions (e.g., the creation time of the triggered transaction and the commit dependency between them) is defined by the coupling mode. Different coupling modes have been proposed to cater for the requirements of different applications [Buch95, Wido96b]. Some examples are *deferred*, *immediate* and *detached* coupling modes. In the deferred and immediate coupling modes, the triggered transactions are defined to be dependent on the triggering transactions. The triggered transaction is a part of the triggering transaction (its sub-transaction). If a triggering transaction is aborted, its triggered transaction will be aborted as well. If the triggered transaction misses the deadline of the triggering transaction, it will also be aborted.

Comparing with the transactions, which have not triggered transaction (we call them non-triggering transactions), the triggered transactions are more critical. Failing to commit the triggered transactions is highly undesirable as they represent the reactions to the critical events occurred in the external environment. If the events are beneficial (e.g., in the program stock trading systems), it represents a serious loss in *opportunities*. If the events are harmful, it represents that some unattended *risk* is coming (e.g., in the missile tracking systems). The consequence can be catastrophic. Thus, how to maximize the number commit of the triggered transactions, especially the more important ones, for a given sequence of events occurred in the external environment is an important issue in the design of a DRTADBS.

Due to the high complexity of a DRTADBS and the unpredictability introduced from triggering of transactions, a number of factors can cause the abort of the triggered transactions. Although some recent work has been done on scheduling transactions which dynamically trigger other transactions,

most of them are concentrated on meeting the deadlines of the triggering transactions which lengths are suddenly increased as the result of the triggering [Chen96, Puri94, Siva96]. They have ignored the impact of the temporal data objects, especially the temporal data objects which are responsible for the triggering (we call them *triggering data objects*), on the timing constraints and criticality of the transactions. The timing constraints on the transactions due to temporal data objects has been discussed in [Rama96, Xion96a, Xion96b]. In [Xion96b], a forced wait method is suggested to reduce the number of transaction aborts due to the access of invalid temporal data objects. However, the impact of triggering data object on the criticality of the transactions has not been addressed.

In this paper, we discuss how to assign priorities and deadlines to triggered transactions with the objective to maximize the number of commit of triggered transactions especially the more important ones. New approaches, called *data state dependent* (DSD) and *transaction-data deadline* (TDD) are suggested. The remaining parts of the paper are organized as follows. Section 2 discusses the mechanisms to trigger transactions under the deferred and the immediate coupling modes. Section 3 discusses the effect of the scheduling of triggering transactions on the scheduling of the triggered transactions. Section 4 introduces our approaches to deadline and priority assignments. Section 5 is the performance study of our approaches under different workload and environment. The Conclusions of the paper is in Section 6.

2 Triggering of Transactions in DRTADBS

A DRTADBS has to monitor the status of the external environment. In the other words, the system has to keep an updated view of the external environment [Adel95]. So, some of the data objects in the database, called *temporal data objects*, are used to record the status of the associated objects in the external environment. Each temporal data object is associated with a temporal constraint to define its validity. The data object is *invalid* if the value recorded in the temporal data object may be significantly deviated from the actual status of the associated object in the external environment. The consistency between the value recorded in the temporal data objects and the actual status of the associated objects in the external environment is called *absolute consistency* [Kim94, Rama93].

Since events are occurring in the external environment, the status of the objects is also changing dynamically. Absolute consistency has to be maintained by timely installation of the update transactions which capture the up-to-date status of the objects in the external environment. Each update transaction is responsible to update one temporal data object. If an update transaction cannot be installed before the temporal data object becomes invalid, the absolute consistency will be violated.

When these data objects are accessed by transactions, the transactions have to be restarted or even aborted¹.

Transactions are created by applications to access the database. They may read some temporal data objects to check the conditions which are predicates on the state of the database. In addition to observe the absolute consistency, they also have to observe *relative consistency* which is defined on the relative *ages* of the temporal data objects accessed by a transaction [Kim94]. It is *relative inconsistent* if the temporal data objects accessed by a transactions are representing the views at a very different time span. In this case, the transaction has to be restarted or aborted.

In accessing a temporal data object, the conditions for triggering of transaction will be checked. If any one of the defined conditions is satisfied, a triggered transaction will be created according to the specified coupling mode. In the following, we will consider two coupling modes, the deferred and the immediate in which the dependency between the triggering and triggered transactions is greater as the triggered transactions are sub-transactions of the triggering transactions. The scheduling problem is more difficult as more constraints have to be satisfied before a triggered transaction is allowed to commit.

Under the deferred coupling mode, the creation and the execution of a triggered transaction is started only after the completion of the triggering transaction even though the conditions are satisfied in the course of execution of the triggering transaction as shown in Figure 1.

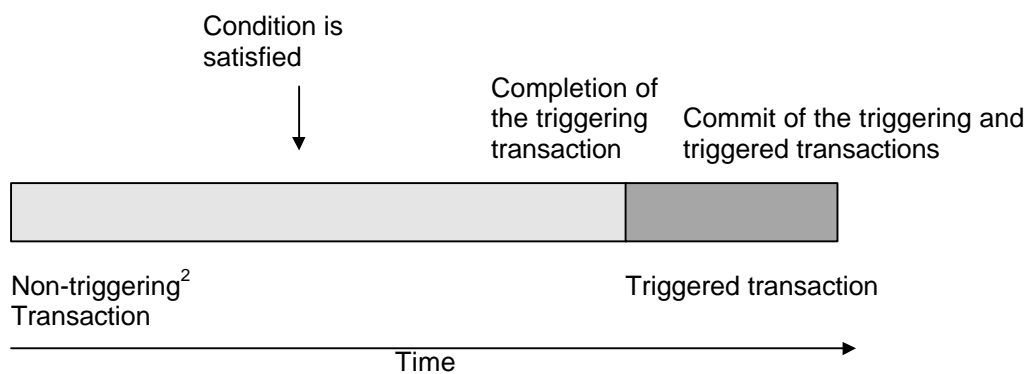


Figure 1 : Triggering of Transaction under the Deferred Coupling Mode

¹ When a transaction is restarted, it may miss its deadline and has to be aborted.

² When a transaction is created. It is a non-triggering transaction. When it triggers a transaction, it becomes a triggering transaction.

In a DRTADBS, the time required to process an operation is unpredictable. It is affected by the location of the required data object and the distribution of workload in the system. Thus, the waiting time of a triggered transaction may be very long under the deferred coupling mode.

Under the immediate coupling mode, once the conditions are satisfied, the triggered transaction will be created and executed “immediately” (Figure 2). At the same time, the execution of the triggering transaction will be suspended until the completion of the triggered transaction. Thus, the processing order of the remaining part of the triggering transaction and the triggered transaction is just the opposite of that under the deferred coupling mode.

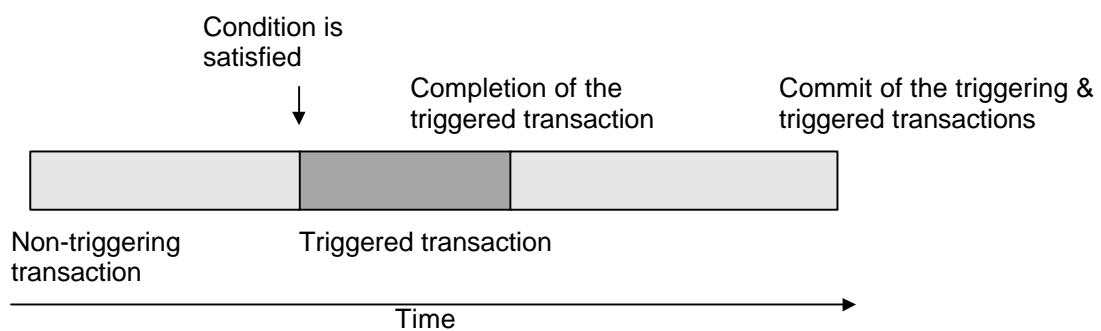


Figure 2 : Triggering of Transaction under the Immediate Coupling Mode

Since the triggered transaction is dependent on the triggering transaction, the triggered transaction cannot commit even though it is completed. It has to wait until the commit of the triggering transaction. Similar to the situation under the deferred coupling mode, the delay is unpredictable and is dependent on the scheduling of the triggering transaction.

3 Scheduling of the Triggering Transactions

The precedence and dependency relationships between the triggering and the triggered transactions as defined by the coupling modes (e.g., the deferred and the immediate coupling modes) make the scheduling of the triggered transactions highly affected by the scheduling of the triggering transactions. If the triggering transaction is completed earlier, it is more possible for the triggered transaction to be completed because much slack time is left. Otherwise, the triggered transaction may not be able to be completed due to short slack time. This in turn depends on the scheduling of the triggering transaction.

Triggering of transaction is an unpredictable event. None can forecast when the conditions will be satisfied. Therefore, none can predict which transaction will trigger a transaction and when it will

trigger the transaction until at the time of triggering. It is because triggering of transaction is resulted from the state of the database which is highly dynamic. Thus, the system cannot anticipate the additional resource requirements of a triggered transaction by scheduling the triggering transaction first (at that time, the system still recognizes it as a non-triggering transaction). If the system schedules the triggering transactions in the same way as the non-triggering transactions, the probability to commit the triggered transactions will be affected as the transaction length of a non-triggering transaction is usually shorter compared with the sum of the length of the triggering and triggered transactions. Consequently, the slack time available for the triggered transaction (and the triggering transaction under the immediate coupling mode) is also shorter. This is especially true when the triggering is occurred at the later stage of the transaction. Suppose there are two transactions T_1 and T_2 . They have the same criticality but the priority of T_1 is higher than T_2 due to a closer deadline. The system schedules T_1 before T_2 . When T_2 triggers a transaction, $T_{2,1}$, the slack for $T_{2,1}$ to complete may be very short as the system does not know in advance that T_2 will trigger a transaction and has not pre-allocated extra slack time for the execution of T_{tt} . As a result, T_{tt} is likely to be aborted due to deadline missing.

Triggered transactions increase the system workload and affect the schedulability of the system. If a large number of triggered transactions are created in a short period of time, the system will be suffered from a transient overloading. The most affected ones are the triggered transactions as their scheduling is dependent on the scheduling of the triggering transactions.

Although some sub-transaction priority assignment policies have been suggested to solve the scheduling problem due to triggering of transactions [Puri94, Siva96], these policies have two major problems. Firstly, deadlock is possible as they dynamically adjusted the priority of the transactions based on their remaining slack times. Resolving deadlock in a distributed environment is highly expensive and difficult. They should be avoided by adopting some deadlock free approaches. Secondly, they have assumed that the criticality of different transactions (different kinds of triggered and triggering transactions) are the same. This is not true in most cases (a detail explanation is given in Section 4.2.)

4 Scheduling of Triggered Transactions

Although to commit a triggered transaction is highly important to the usefulness of a DRTADBS, not much work has been addressed on how to assign deadlines and priorities to the triggered transactions. Mostly, they are considered as a part of the triggering transactions. However, due to the access of temporal data objects, the deadline and criticality of a triggered transaction may be different from that of its triggering transaction. In this Section, we introduce two approaches to assign

deadlines and priorities to the triggered transactions according to the current status of the temporal data objects.

4.1 Assigning Deadlines to Triggered Transactions

Due to the access of temporal data objects, the timing constraints on the triggered transactions may become tighter as they have to commit before any of their accessed temporal data objects becomes invalid. If a triggered transaction is dependent on the triggering transaction (as under the deferred and immediate coupling modes), its completion time (deadline) is constrained by :

- (1) the deadline of the triggering transaction; and
- (2) the temporal constraints of the accessed temporal data objects (to observe the absolute consistency).

The second constraints are called the *data deadlines*. Missing the deadline of the triggering transaction will abort the triggered transaction as it is defined to be dependent on the triggering transaction. Missing the data deadlines also causes the abort of the triggered transaction as the values in the temporal data objects are no longer valid. Thus, the deadline of a triggered transaction can be defined as:

$$\text{Deadline}_t = \min(\text{Deadline}_i, \text{Temporal}_D)$$

where, Deadline_i is the deadline of the triggering transaction and

Temporal_D is the earliest data deadline among the temporal data objects accessed by the transaction.

We call this approach for assigning the deadline to a triggered transaction as *transaction-data deadline (TDD)* approach. Whenever a transaction accesses a temporal data object, its deadline will compare with the data deadline of the temporal data object. If the data deadline is closer, its deadline will be replaced by the data deadline. With the TDD approach, any abort due to missing the data deadlines can be identified earlier so that the amount of resources wasted on those transactions which would be aborted can be minimized. This can also improve the schedulability of the whole system especially when the system workload is heavy.

4.2 Assigning Priority to Triggered Transactions

Not only the timing constraint of the triggered transaction is different from the triggering transaction, their criticality may not be the same. The creation of a triggered transaction implies that some specific attention has to be paid to the current system status because some interesting or important event has occurred, e.g., missile is coming. Therefore, the importance of the triggered transactions are usually much higher than the non-triggering transactions. Thus, higher criticality should be assigned to the triggered transactions.

Example 1: Consider a navigation system of robot in which detection transactions are created periodically to check whether there are any obstacle in front of the robot. Once it has found that there is an obstacle in its way, a reaction transaction will be triggered to stop the robot or to change its moving direction. The criticality of the detection transactions should be lower than the criticality of the reaction transactions as the existence of any reaction transactions implies that *a risk is coming*. On the other hand, missing the deadlines of the detection transactions is less serious as the detection transactions will find nothing in front of the robot in most of the cases. To reflect the higher importance, higher criticality, and thus higher priorities, should be assigned to the triggered transactions comparing with the non-triggering transactions.

No only the criticality of the triggered transactions may be higher than the non-triggering transactions, the criticality amongst the triggered transactions may be different. It is because the generation of triggered transactions is the result of satisfaction of some triggering conditions which importance may be different. The triggered transaction in connection with an important condition should be assigned to a higher criticality.

Example 2: In a program stock trading system, the condition to buy the stock X and Y is when their prices are lower than \$10. Assuming that the new price of stock X is \$9, thus a buy transaction, *B1*, is triggered. At the same time, the price of stock Y is \$2. Thus, another buy transaction *B2* is triggered to buy stock Y. To the system users, *B2* is more critical than *B1* as completing it can yield more profit. In this scenario, *B2* should be given a higher criticality.

This example shows that the criticality of a triggered transaction is affected by the current status of the triggering data object. Thus, in assigning criticality to a triggered transaction, both the criticality of the triggering transaction and the criticality resulted from the triggering data object have to be considered. We call this approach as the *data state dependent (DSD) approach* such as :

$$\text{Criticality}_{tt} = \alpha \times \text{Criticality}_t + \beta \times \text{DataState}_D$$

where α and β are the weighting factors. Criticality_{*t*} is the original criticality of the triggering transaction and DataState_{*D*} is a function which defines the criticality for the triggering data object. It is based on the current state of the triggering data object, *D*, and the condition for triggering. Referring to the above example on the program stock trading system, the DataState_{*D*} can be defined as the difference between the price of the stock set in the condition for triggering and the current price of the stock. A higher criticality will be assigned to the one which is triggered from a greater difference in stock prices.

The priority of a triggered transaction is a function of its deadline and criticality. In order to support the scheduling of triggered transactions with different criticality, the system is defined to have *multiple priority levels*. By the use of DataState_{*D*}, a triggered transaction can be assigned to different priority levels based on its criticality. It is expected that by promoting the triggered transactions which have higher criticality to a higher priority level, they will have a higher probability to commit even with a shorter slack time.

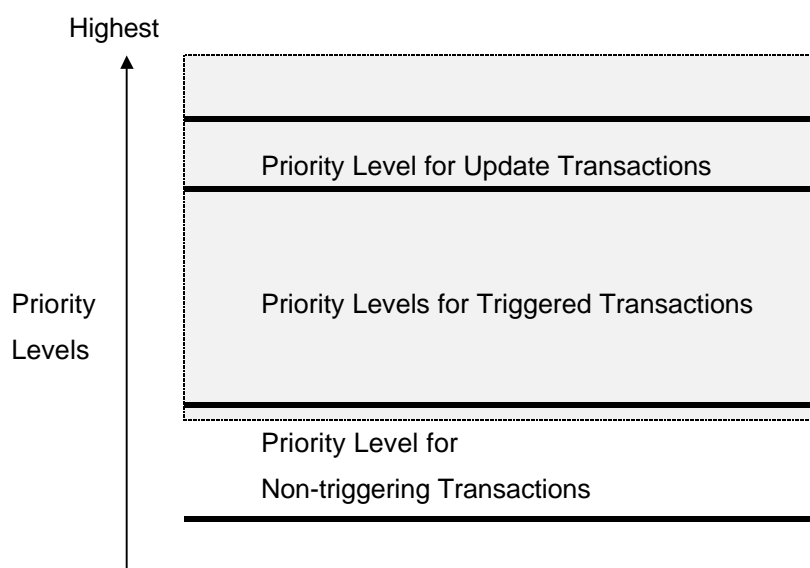


Figure 3 : Assigning Priorities to Triggered Transactions

How to raise up the criticality (and priority) of a triggered transaction is dependent on the semantic of the transaction and the condition for triggering. However, simply promoting the triggered transactions to a higher priority level may not be able to improve their performance effectively. It is affected by a number of factors such as the length of the triggered transactions, the data deadlines of the temporal data objects and the workload of the update transactions. The *avi* of the temporal data objects and the length of the triggered transactions determine the tightness of the timing constraint of a triggered transaction.

The workload of the update transactions affects the scheduling of the triggered transactions. In Figure 3, there exists multiple priority levels in the system. The non-triggering transactions are assigned to the lowest level. Since the update transactions are important to maintain the absolute consistency of the temporal data objects, they are assigned to the priorities higher than the non-triggering transactions. The probability to commit a triggered transaction is then dependent on where is the priority of a triggered transaction relative to the update transactions. If its priority is lower than the update transactions, they still have a high chance to miss their deadlines if the workload of update transactions is heavy.

4.3 Effect of Coupling Modes on Triggering

The effect of the TDD and the DSD approaches on the performance of the triggered transactions can be very different under different coupling modes. Under the deferred coupling mode, the creation of the triggered transactions are delayed until the completion of the triggering transactions. So, they are more affected by the scheduling of the triggering transactions. The effectiveness of promoting the triggered transactions to a higher priority level may be small if the remaining time for the execution of the triggered transactions is short. In case the remaining time is less than the required execution time of the triggered transaction, it will be useless to schedule the triggered transactions even though they are promoted to the highest priority level.

Under the immediate coupling mode, priority promotion can be much more effective especially if the triggering occurs at the early stage of a transaction. Since the triggering transactions are still in the middle of execution, there remains much slack time for the execution of the triggered transactions as compared with the deferred coupling mode. However, the commit of a triggered transaction requires the completion of its triggering transaction. It has to wait for the triggering transaction even though it has been completed. If the priority of the triggering transaction remains low, it is still possible for the triggered transaction to miss its deadline because of waiting for the completion of the triggering transaction.

Another more serious problem under the immediate coupling mode is that deadlock is possible if it is combined with some blocking based real-time concurrency control protocols such as High Priority Two Phase Locking (H2PL) [Abbo92]. Suppose there are two transactions, T_t and T_1 with priorities, P_t and P_1 , respectively. After T_t accesses a temporal data object such that the conditions to trigger a transaction T_{tt} (with priority P_{tt}) are satisfied. The priorities order of these three transactions is $P_{tt} > P_1 > P_t$. The triggered transaction T_{tt} may block T_1 if T_1 requests a lock after T_{tt} . After the

completion of T_{it} , T_t resumes its execution. Since P_t is lower than P_i . It is possible that T_t is blocked by T_i if it requests a lock holding by T_i . This is a deadlock even the original concurrency control protocol is deadlock free. In order to avoid the deadlock and to reduce the probability of missing deadlines of triggered transactions while they are waiting for the completion of their triggering transactions, the priorities of the resumed triggering transactions are set to be the priorities of the triggered transactions after the completion of triggered transactions.

5 Performance Evaluation

5.1 Simulation Model

A simulation model has been implemented to investigate the performance of the DSD and TDD approaches under different system environment and coupling modes. Figure 4 depicts the DRTADBS model consisting of a number of sites, fully-connected in a point-to-point network. Each communication link is modeled as a delay center, i.e., constant message delay³. Each site is a local database system consisting of two transaction generators, a scheduler, a CPU, a ready queue, a local database, a communication interface, and a block queue.

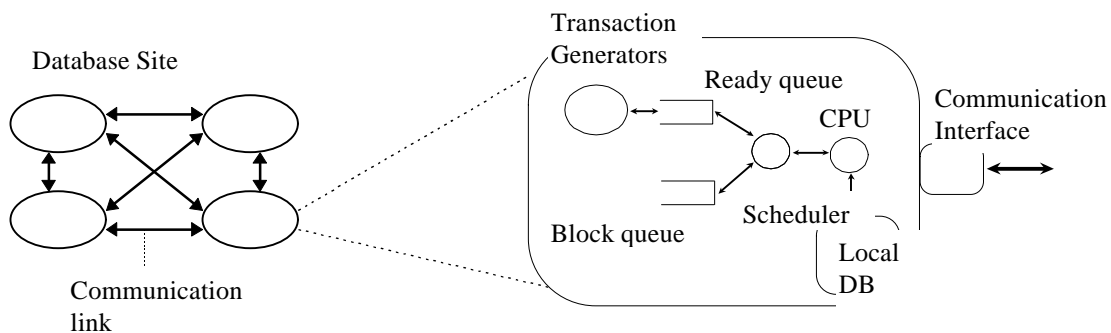


Figure 4 : The DRTADBS Model

The local database consists of two types of data objects: temporal and static data objects. *Static data objects* are for storing static information which does not change frequently with time such as the stock number of a particular stock in a stock market. Temporal data objects store dynamic information reflecting the current status of the associated objects in the external environment. For each temporal data object, an *absolute validity interval, avi* is defined for checking its validity. The temporal data objects are also constrained by *relative validity intervals (rvi)* for checking the relative consistency. Since main memory databases can better support real-time applications, it is assumed that the database is residing in the main memory.

One of the transaction generator generates update transactions periodically. They are single operation transactions. A time stamp of the current time is assigned to each update transaction to indicate under which snapshot the value is taken. When an update transaction is installed, the time-stamp will be recorded into the time-stamp of the temporal data object. A temporal data object is absolute consistent if the sum of its time-stamp and avi is greater than the current time. If the time-stamps of the data objects accessed by a transaction is greater than the rvi , then the transaction has violated the relative consistency. A transaction has to be restarted if it is absolute or relative inconsistent. Basically, the update transactions do not have deadlines. However, it is important to install the update as soon as possible in order to maintain the validity of the temporal data objects. Thus, they are assigned to the higher priority level.

Another transaction generator creates user transactions following a Poisson distribution. Each user transaction is defined with a deadline and a priority based on its deadline and criticality. It is assumed that all user transactions are of the same criticality upon their generation. Thus, a user transaction with a closer deadline will be assigned to a higher priority.

A user transaction is modeled as a sequence of database operations. The processing of the operations requires CPU computations and the access of data objects (some of them may be temporal data objects). To simplify the model, it is assumed that each operation will access one data object. If the required data object of an operation is resided in a remote site (different from the site where the transaction is generated), it will be transmitted to the remote site through the communication networks, and be processed there.

When a user transaction accesses a temporal data object, a transaction will be triggered if certain conditions are satisfied. The probability to trigger a transaction is defined by *Trigger_Probability*. The triggered transaction is also modeled as a sequence of operations. Dependent on the status of the temporal data objects, the criticality and deadlines of the triggering and the triggered transactions will be reassigned if the DSD and the TDD approaches are used. To simplify the analysis and without loss of generality, we define two levels of criticality for the triggered transactions (high and low). When a triggered transaction is created, the probability to have a high criticality is *High_Priority*. Otherwise, it has a low criticality. Thus, we can classify the committed transactions into three classes :

- (1) Class H : the triggered transactions with high criticality;

³ The impact of different communication model has been studied in [Lee96].

- (2) Class L : the triggered transactions with low criticality (although it is low, it is still higher than the non-triggering transactions); and
- (3) Class N : the user transactions which have not triggered any transaction

The transactions (the update, user, triggering and triggered transactions) queue in the ready queue for the CPU according to their priorities. The scheduling of the CPU is performed by the scheduler. It selects the transaction at the head of the ready queue (the one with the highest priority) to use the CPU when it is free.

In order to maintain the data consistency (internal consistency), a concurrency control protocol [Bern87] is adopted to manage the access of the data objects by transactions. It is assumed that the high priority two phase locking (H2PL) [Abbo92] is used in which each data object is associated with a lock. Before accessing a data object, a transaction has to seize the lock associated with the data object. If the lock is already seized by a higher priority transaction, the lock requesting transaction has to be blocked in the block queue until the lock holding transaction releases the locks. Otherwise, the lock holding transaction will be forced to release the lock and be restarted. Then, the lock requesting transaction can seize the lock and continue its execution.

When all operations of a user transaction has been completed, before it enters the commitment stage, the relative consistency of all its accessed temporal data objects will be checked. It will be allowed to enter the commitment stage only if all the relative consistent constraints have been satisfied. Otherwise, it has to be restarted. In the commitment stage, it performs a commit protocol to ensure the transaction atomicity. It is assumed that the two phase commit is used [Bern87]. Permanent updates of the data objects will be performed if the final decision is commit. If the decision is abort, all the effect to be applied to the database by the transaction will be cleared. Finally, the transaction releases all its seized locks upon its commit or at the time of abort.

It is assumed that all transactions, except the update transactions, are firm real-time [Abbo92, Yu94]. Before a transaction is granted to use the CPU, its deadline will be checked. If it has missed its deadline, it will be aborted immediately.

5.2 Model Parameters and Performance Measures

The simulator is built using OPNET [OPNE96], a proprietary graphical simulation package. The structure of the simulation model follows the system model described in the last Sub-section with

the following characteristics. The end-to-end deadline of a transaction (T) is generated according to the following formula :

$$Deadline = ar(T) + pex(T) \times (1 + SF) + T_{comm} \times N_{transit}$$

where SF : the slack factor which is a random variable uniformly chosen from a range;

$ar(T)$: the arrival time of transaction T ;

$N_{transit}$: the number of transit across the networks required in order to access the data objects in a transaction;

T_{comm} : the communication delay;

$pex(T)$: the predicted execution time of T . It is defined as :

$$pex(T) = (T_{lock} + T_{process} + T_{update}) \cdot N_{oper}$$

where N_{oper} : the number of operations in a transaction;

T_{lock} : the CPU time required to set a lock;

$T_{process}$: the CPU time required to process an operation; and

T_{update} : the CPU time to update a data object.

In most of the previous studies on real-time database systems, the missing rate is used as the primary performance measure [Abbo92, Ulus95]. They indicate the probability of missing deadline of the transactions. However, they are not suitable for DRTADBS in which it is not possible to determine the number of triggered transactions as their creations are system dependent. In a DRTADBS, the most important performance objective is how well the system can respond to the changes in the external environment. This can be measured in terms of the number of commit of triggered transactions for a given set of events occurred in the external environment. To the system users, it is more preferable to have a larger number of commit of triggered transaction in a given period of time even though the missing rate is higher. Thus, in our experiments, the primary performance measure is the *commit rate* (CR) which is the number of committed transactions per unit time.

In the simulation model, a small database (200 data objects per site) is used to create a high data contention environment. The small database also allows us to study the effect of hot-spots, in which a small part of the database is accessed frequently by most of the transactions. Table 1 summarizes the model parameters and their baseline values. In the baseline setting, the amount of workload for the update transactions is 15%. In the model, streams of update transactions are defined

to maintain the validity of the temporal data objects. Their periods are defined based on the *avi* of the temporal data objects. To simplify the model, it is assumed that all the temporal data objects have the same *avi* and *rvi*. The period of an update stream, for each temporal data object, is defined to be twice of the *avi* in the baseline setting.

| Parameter | Baseline Value |
|---|---|
| CPU Scheduling | highest-priority-first |
| Concurrency Control | H2PL |
| Database size / site | 200 data objects |
| # of database sites | 4 |
| T_{lock} | 2 msec |
| $T_{process}$ | 34 msec |
| T_{comm} | 100 msec |
| N_{oper} | 3 - 20 uniformly distributed |
| SF | randomly chosen from the range [2.5, 13.75] |
| T_{update} | 6 ms |
| I_{user} (arrival rate of user transactions) | 0.2 to 1.4 transactions per sec. |
| $Trigger_Probability$ | 0.6 |
| $High_Priority$ | 0.5 |
| avi | 4 sec |
| rvi | 4 sec |
| TL (length of the triggered transactions) | 20% of the length of the triggering transaction |
| P_{update} (period of each update stream) | 2 sec |
| $Temp_fac$ (Proportion of Temporal data objects) | 0.25 |

Table 1: Baseline Setting

5.3 Results Analysis and Discussions

Each simulation reports the results for a simulation length of 2400sec. About 10,000 user transactions are generated in the system with four sites when the arrival rate is 1 user trans/sec. In the experiments, the impact of different factors on the performance of the system with the application of the DSD and the TDD approaches have been studied. It is compared with the system using the deadline and

criticality of the triggering transaction for the scheduling of the triggered transaction. We call this approach as the *Single Priority Level (SPL)*.

Figure 5 depicts the commit rate (CR) of the three classes of transactions with and without the DSD under the deferred coupling mode when the arrival rate of the user transactions is increased from 0.2 to 2.0 transactions per second. Although the CR of all the three classes of transactions increase with the arrival rate, the rates of increase are different. The CRs of Class H and Class L become flat starting from medium workload whereas the CR of Class N transactions, which do not triggered any transaction, continue to increase steadily. It is because under a higher workload (due to increased arrival rate), the triggered transactions (Class H and Class L transactions) have higher probability to abort due to deadline missing as the remaining slack time for their execution decreases with an increase in system workload. The CRs of Class L and Class H are almost the same in the SPL case. Since the priorities of the triggered transactions in these two classes inherit the same priorities as their triggering transactions, they should attain a similar performance.

Surprisingly, as shown in Figure 5, the DSD approach helps little to improve the performance of the Class H and Class L transactions. Compared with the CRs in the SPL case, the differences are insignificant. Although the priorities of Class H transactions are promoted, their probability to commit is similar to the SPL case where there is no priority promotion. It is because, under the deferred coupling mode, the creation of the triggered transactions is delayed until the completion of the triggering transactions. At the start time of the triggered transaction, the remaining slack time may be very short. To promote the priorities of the triggered transactions may be too late to save them from missing their deadlines.

Under the immediate coupling mode, the improvement is much significant when the DSD approach is applied as shown in Figure 6. It is because the creation of the triggered transaction is earlier. To promote the priorities of the triggered transactions and the triggering transactions can make them to have higher probability to complete before their deadlines. However, the CRs of the Class L and Class N transactions are reduced as their priorities are lower than the priorities of the Class H transactions. They have to wait longer for the CPU and have higher chances to be restarted due to lock conflict with Class H transactions. The drop is more significant in the Class N transactions as their priorities are the lowest.

As can be seen in Figure 7, the performance of Class H and Class L transactions decreases as the length of the triggered transactions increases. If the length is increased, the triggered transactions

have a higher probability to miss their deadlines as the result of tightened time constraints. The amount of slack time is determined at the creation of their triggering transactions. At that moment, there is no prior knowledge of the length of the triggered transactions or even whether there is any triggering will occur. As a result, the longer is the length of the triggered transactions, it is more likely for them to miss their deadlines and the CRs decrease subsequently. Class N transactions are less affected by the increased length of the triggered transactions because they do not triggered any transaction. Consistent with the results in Figure 6 and Figure 7, the improvement with the application of DSD is only significant under the immediate coupling mode as shown in Figure 8. The performance of the Class H and Class L transactions is very similar under the deferred mode even with the application of the DSD approach due to the late start time of the triggered transactions.

Figures 9 and 10 show the CRs with different *avi* values. The performance of all classes improves as the *avi* increases. It is reasonable because a wider *avi* means that the validity of the temporal data objects can last for a longer time. In the other words, it is less likely for a transaction to access an invalid temporal data object or that the temporal data object accessed by a transaction becomes invalid at the time of commit. As a result, number of restarts and number of aborts due to temporal inconsistency are reduced. Consequently, the CR increases. Since the Class N transactions may or may not access temporal data objects and the data deadlines are not a big problem to them. Thus, the impact of this factor is less significant. The large improvement of the Class H transactions under the immediate coupling mode shows that the *avi* is a bound to their performance. To loosen the *avi* helps those triggered transactions which meet the deadlines to commit without violating the temporal consistency.

Figures 11 to 14 give the CRs with different update transaction rates. There are two sides to interpret this factor. If the update rate is low, given a fixed *avi* value, the temporal consistency of the data objects may not be able to be maintained. This will lead to an increase in the number of restarts and aborts. If the update rate is high, the temporal consistency can be maintained but it may be very expensive. Part of the resources in the system will be drawn by them and lead to the degradation of the performance of the other user transactions as the priority of the update transactions is set to be high. So, for some applications, it may be desirable to have a low update rate in spite of the possibility of temporal inconsistency.

Figures 11 and 12 give the performance when the priority of update transactions is set to be the highest. When the update rate is low (0.25 update per second), temporal inconsistency must exist even though all updates can be installed once they are generated. So, the CRs are low. When the update rate

is increased to 0.5 update per second, the temporal consistency can be maintained if all updates can be installed once they are generated. The CRs increased to larger values. However, further increase of the update rate helps little to improve the performance. It is reasonable because the highest priority of the update transactions allow them to install the updates into the database efficiently. So, when the update rate is further increased, deterrent effect due to the large volume of workload generated by the update transactions can be observed. The CRs of all classes drop slightly.

Figures 13 and 14 gives the performance when the priority of the update transactions is set to be higher than the priority of the non-triggering transactions but lower than the priority of the triggered transactions. In general, the performance and the analysis are very similar to Figures 11 and 12. But there are two points to note. When the update rate is 0.75 update per second, the CRs of Class L and Class H are increased comparing with the CRs when the update rate is 0.5 update per second. Since the priorities of the update transactions are lower than the priorities of the triggered transactions, the increase of update rate can now help to improve the degree of temporal consistency in the database. Compared with the case when the priority of the update transactions is the highest, the extent of increase is much significant. It is because the triggered transactions are now scheduled before the updates. They have a higher probability to meet their timing constraints. Consistent with the results in previous figures, the improvement with DSD is much more significant under the immediate coupling mode.

Figure 15 depicts the CRs of the three classes of transactions when both the DSD and the TDD approaches are applied under the deferred coupling mode. The CRs of all the three classes of transactions are improved due to early abort of missed deadline transactions with the use of the TDD approach. Similar to the previous results, the performance of Class H and Class L transactions is similar even though their priorities are different. On the contrary, further improvement can be obtained under the immediate coupling mode when both the DSD and the TDD approaches are applied as shown in Figure 16. All classes are benefit from early abort of missed deadline transactions. In this case, the performance of the Class L transactions is improved compared with the case when only the DSD is applied as shown in Figure 6.

6 Conclusions

Two fundamental requirements of a Distributed Real-time Active Database System is to process the time-constrained transactions so that they can be committed before their deadlines and to react to the occurrences of critical events in the external environment by the creation of triggered

transactions to be the event handlers. However, in most of the previous studies, the importance of the triggered transactions has been ignored. Mostly, they are considered as a part of the triggering transactions. The same deadline and criticality will be inherited from the triggering transactions for the scheduling of the triggered transactions. However, if we examine the implication of triggering in real life applications, triggered transactions represent very important actions to the critical events. Committing them is highly important and desirable.

Central to the importance of a triggered transaction is the status of the temporal data object which is responsible for the triggering. Thus, in this paper, we have suggested new approaches, the DSD and the TDD, to assign priorities and deadlines to the triggered transactions based on the status of the temporal data objects which are responsible for the triggering. Different status implies different importance and the triggered transactions created for a more important condition is given a higher criticality and thus a higher priority.

In the DSD approach, the triggered transactions are assigned with higher priorities. However, it may not be able to effectively increase their probability of commitment since it is highly affected by the coupling mode used for the triggering. Under the deferred coupling mode, the scheduling is affected by the scheduling of the triggering transaction to a great extent, even with the use of the DSD, the performance of the triggered transactions seems to be about the same. The DSD approach is much more effective under the immediate coupling mode as the triggered transactions are created earlier. Thus, one way to increase the performance of the triggered transactions under the deferred coupling mode is to apply the DSD approach at the time on the triggering transactions even though the generation of the triggered transaction is at their completion. In this way, the slack time for the triggered transaction can be much increased. The use of the TDD approach can improve the system performance under both coupling modes. However, the degree of improvement is affected by the availability of the temporal data objects, the relative priorities and workload of the update transactions and the length of the triggered transactions.

Reference

- [Abbo92] R. Abbott, & H. Garcia-Molina, "Scheduling Real-time Transactions: A Performance Evaluation", ACM Transactions on Database Systems, Volume 17, Number 3, pp. 513-560, 1992.
- [Adel95] B. Adelberg, H. Garcia-Molina and B. Kao, "Applying Update Stream in a Soft Real-time Database System", in Proceedings of the 1995 ACM SIGMOD Conference, California, pp. 245-256, 1995.

- [Andl96] S. F. Andler, J. Hansson, J.Eriksson, J. Mellin, M. Berndtsson, B. Efring, "DeeDS Towards a Distributed and Active Real-Time Database System", SIGMOD Record, volume. 25, number 1, pp.38-40, 1996.
- [Bern87] P.A. Bernstein, V. Hadzilacos & N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, Mass, 1987.
- [Bern96] M. Berndtsson, J. Hansson, "Workshop Report : The First International Workshop on Active and Real-Time Database Systems", SIGMOD Record, volume 25, number 1, pp. 64-66, March 1996.
- [Bran95] H. Branding, Alejandro P. Buchmann, "On Providing Soft and Hard Real-Time Capabilities in an Active DBMS", Proceedings of the First International Workshop on Active and Real-time Database Systems, Sweden, May 1995.
- [Buch92] A. Buchmann, H. Branding, T Kudrass and J. Zimmermann, "Reach : A real-time, active and heterogeneous mediator system", IEEE Quarterly Bulletin on Data Engineering, volume 14, number 1, pp. 44-47, December 1992.
- [Buch95] A. Buchmann, J. Zimmermann, J.A. Blakeley and D.L. Wells, "Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions", In Proceedings of International Conference on Data Engineering, 1995.
- [Chen96] Y.W. Chen & L. Gruenwald, "Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real-time Database Systems", Information Systems, Volume 21, Number 1, pp. 103-124, 1996.
- [DART96] International Workshop on Database: Active and Real-time, U.S.A., November 1996.
- [Kim94] Y.K. Kim & S.H. Son, "Predictability and Consistency in Real-time Database Systems", Advances in Real-time Systems, Edited by S.H. Son, Prentice Hall, 1994.
- [Kort90] Henry F. Korth, Nandit Soparkar, Abraham Silberschatz, "Trigger Real-Time Databases with Consistency Constraints", in Proceedings of the 16 VLDB Conference, Brisbane, pp.71 - 82, 1990.
- [Lee96] Victor C.S. Lee, Kam-yiu Lam, Sheung-lun Hung, "Impact of High Speed Network on Performance of Distributed Real-time Systems", Journal of System Architecture, volume 42, number 6-7, pp.531-546, 1996.
- [OPNE96] OPNET Modeling Manual, Release 2.5, MIL 3, Inc., Washington, DC, 1996.
- [Ozso95] G. Ozsoyoglu & R. Snodgrass, "Temporal and Real-time Database: A Survey", IEEE Transactions on Knowledge and Data Engineering, Volume 7, Number 4, pp. 513-532, 1995.
- [Prui94] B. Purimetla, R.M. Sivasankaran, J. Stankovic, K. Ramamritham and D. Towsley, "Priority Assignment in Real-time Active Databases", In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, 1994.
- [Rama93] K. Ramamritham, "Real-Time Databases", Distributed and Parallel Databases, volume 1, number 2, pp. 199-226, 1993.
- [Rama96] K. Ramamritham, Raju Sivasankaran, John A. Stankovic, Don T. Towsley, M. Xiong, "Integrating Temporal, Real-Time, and Active Databases", SIGMOD RECORD, Vol. 25, No. 1, p.8 - p.12, 1996.
- [Siva96] R. M. Sivasankaran, John A. Stankovic, Don Towsley, Bhaskar Purimetla, K. Ramamritham, "Priority Assignment in Real-Time Active Database", The VLDB Journal, volume 5, number 1, pp .19 - 34, 1996.

- [Ulus95] O. Ulusoy, "A Study of Two Transaction Processing Architectures for Distributed Real-time Database Systems", *Journal of Systems and Software*, volume 31, number 2, pp. 97-108, 1995.
- [Wido96a] J. Widom and S. Ceri, *Active Database Systems : Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann Publishers, In.c, San Francisco, 1996.
- [Wido96b] J. Widom, "The Starburst Active Database Rule System", *IEEE Transactions on Knowledge and Data Engineering*, volume 8, number 4, pp. 583-595, 1996.
- [Xion96a] M. Xiong, J. Stankovic, K. Ramamritham, D. Towsley & R.M. Sivasankaran, "Maintaining Temporal Consistency : Issues and Algorithms", In *Proceedings of First International Workshop on Real-time Databases: Issues and Applications*, California, 1996.
- [Xion96b] M. Xiong, R. Sivasankaran, J.A. Stankovic, K. Ramamritham and D. Towsley, "Scheduling transactions with Temporal Constraints: Exploiting Data Semantics", In *Proceedings of 1996 Real-Time Systems Symposium*, Washington, December, 1996.
- [Yu94] P. S. Yu, K. L. Wu, K.J. Lin and S.H. Son, "On Real-time Databases: Concurrency Control and Scheduling", *Proceedings of IEEE*, volume 82, number 1, pp. 140-157, 1994.

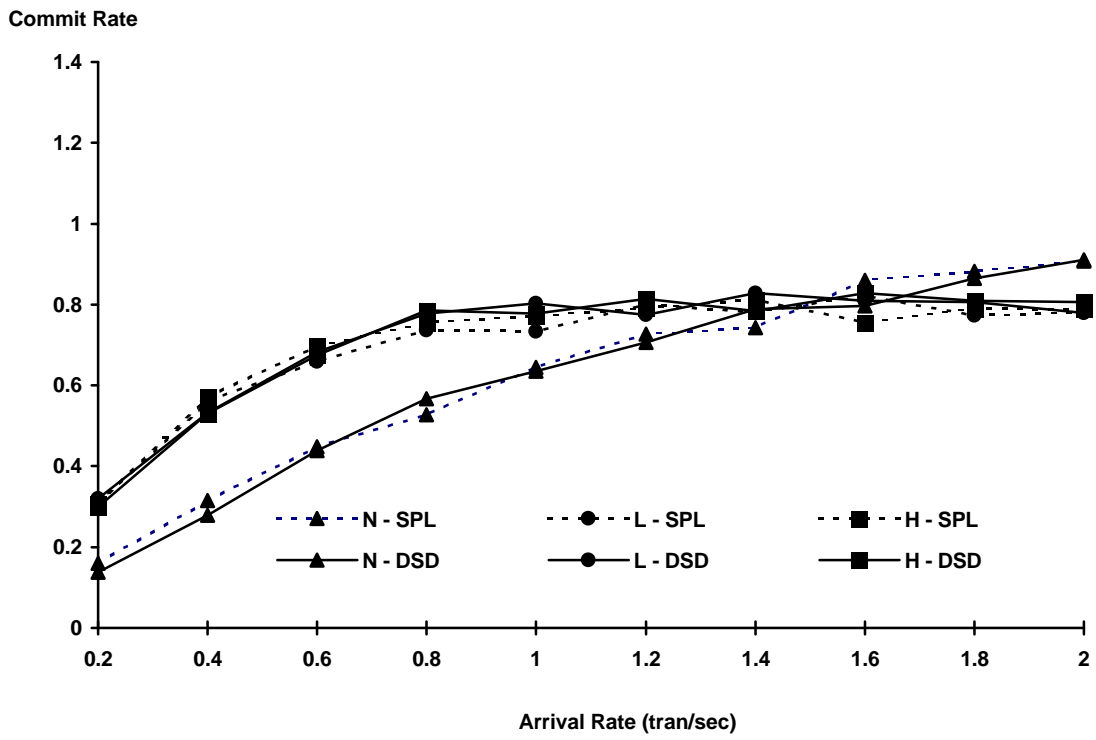


Figure 5: SPL vs DSD under Deferred Mode with Different Workload

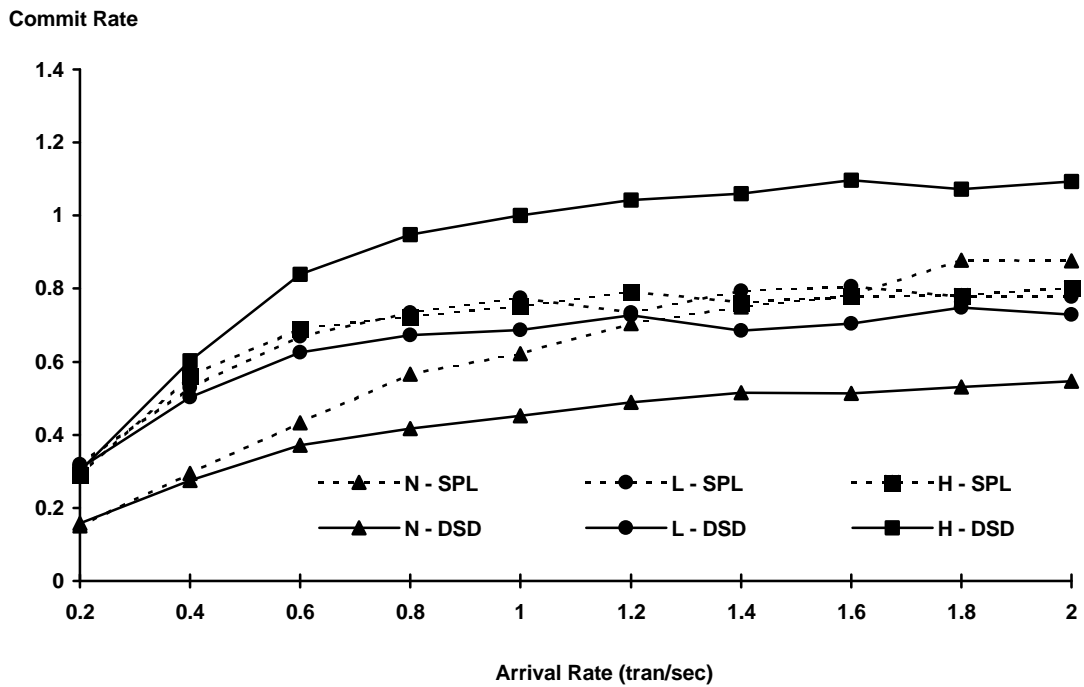


Figure 6: SPL vs DSD under Immediate Mode with Different Workload

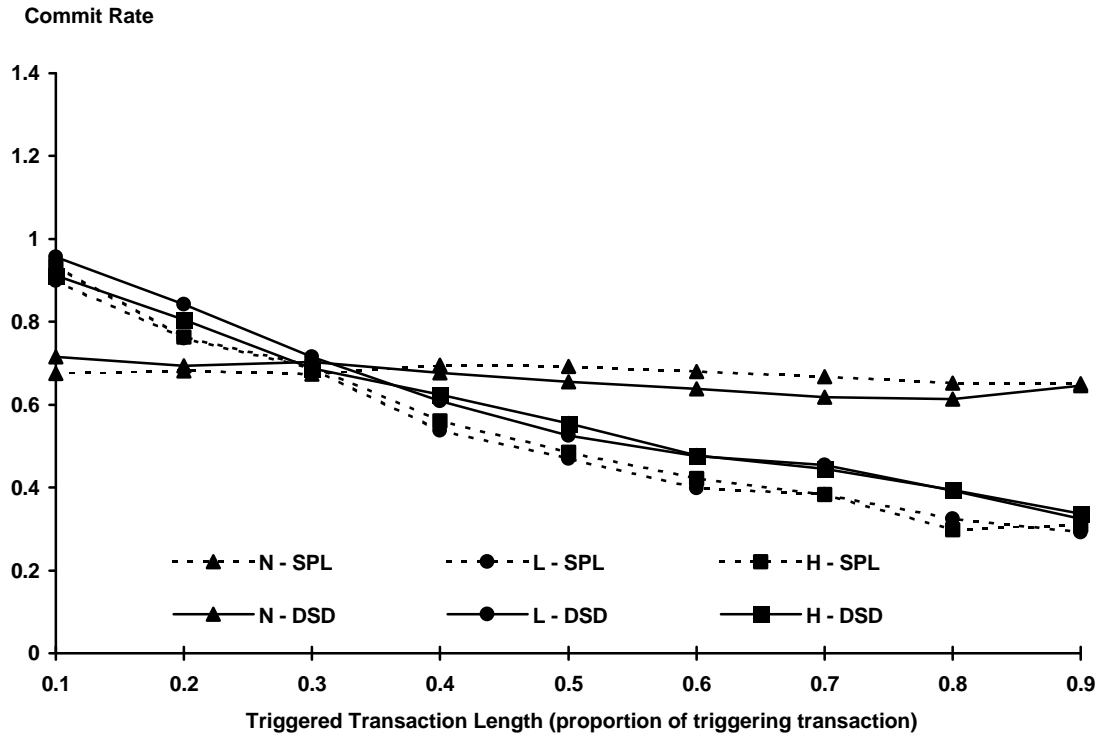


Figure 7: SPL vs DSD under Deferred Mode with Different Triggered Lengths

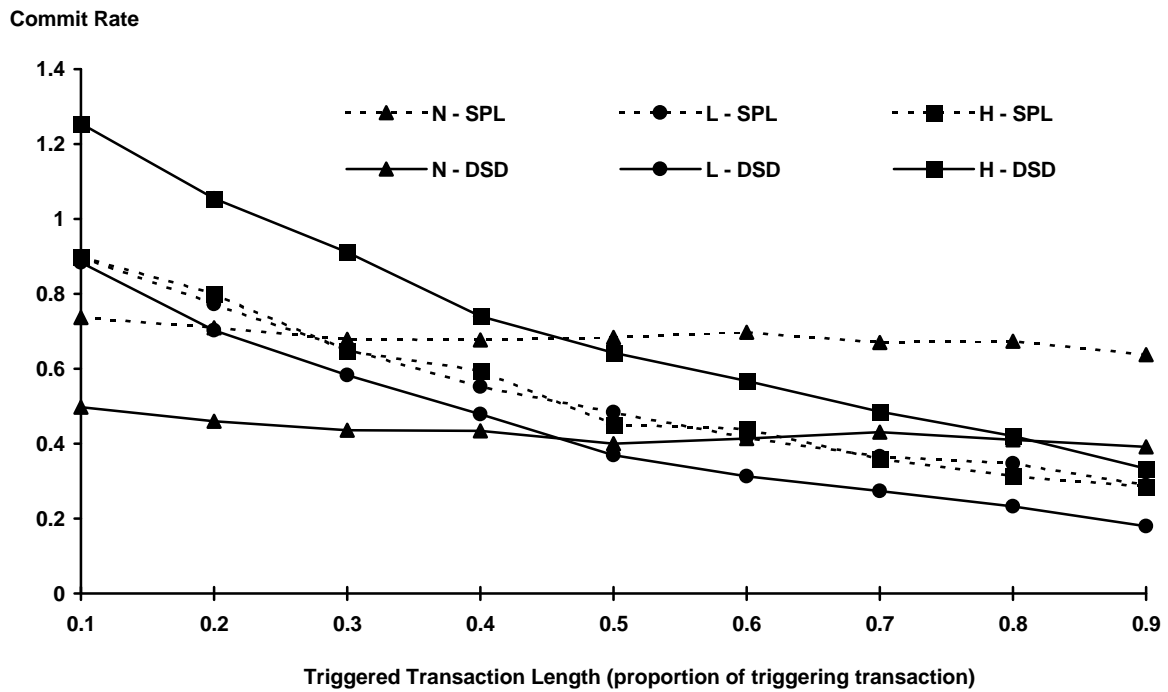


Figure 8: SPL vs DSD under Immediate Mode with Different Triggered Lengths

Commit Rate

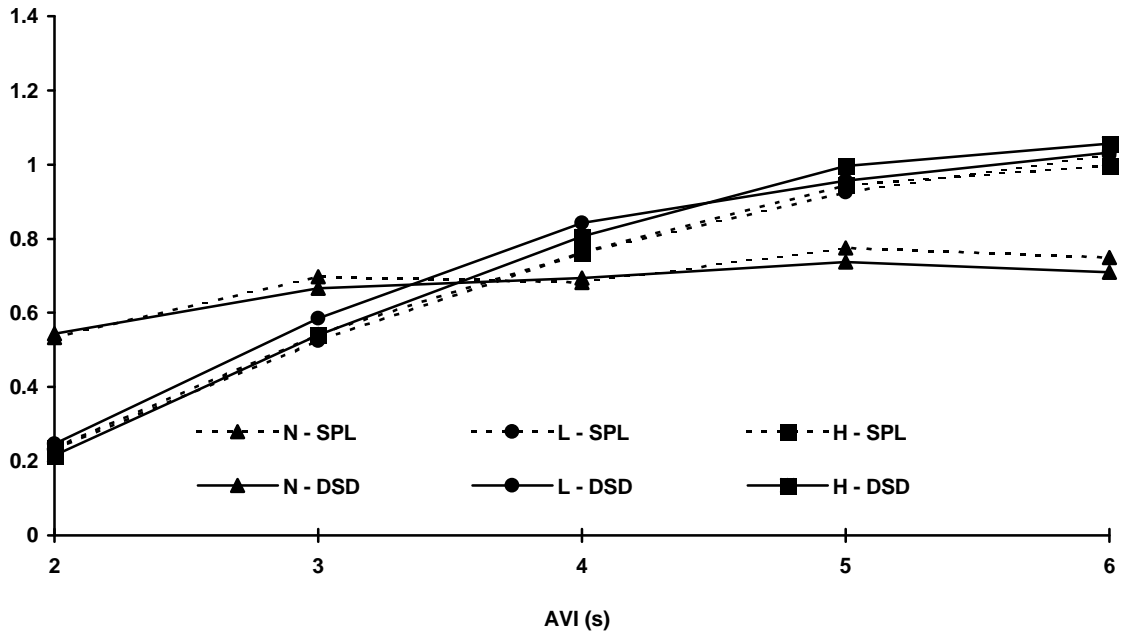


Figure 9: SPL vs DSD under Deferred Mode with Different AVIs

Commit Rate

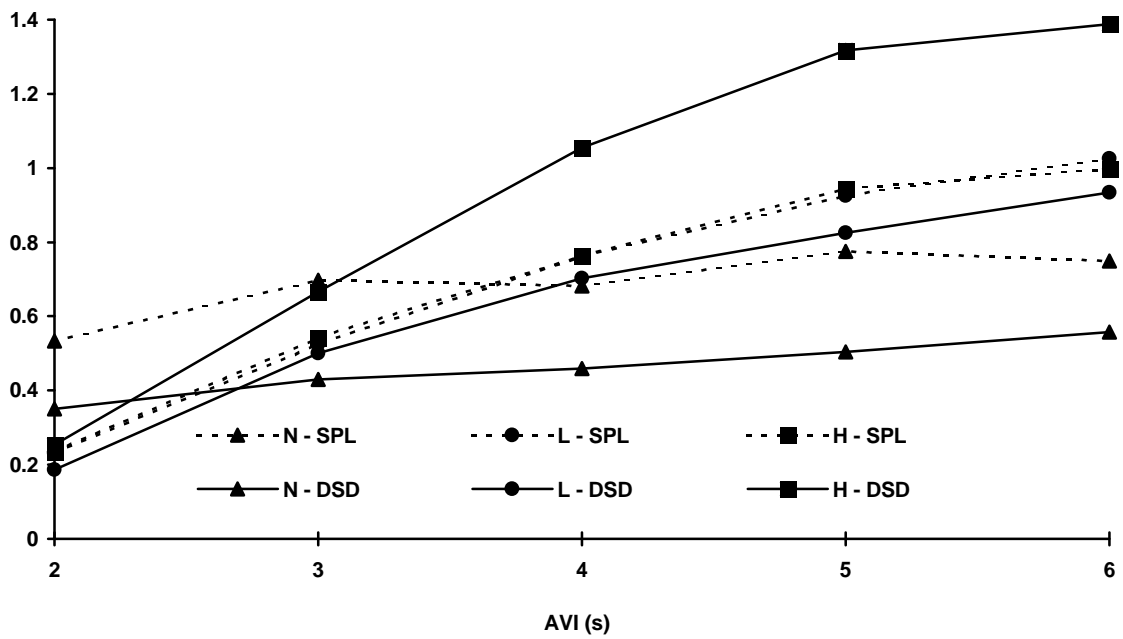


Figure 10: SPL vs DSD under Immediate Mode with Different AVIs

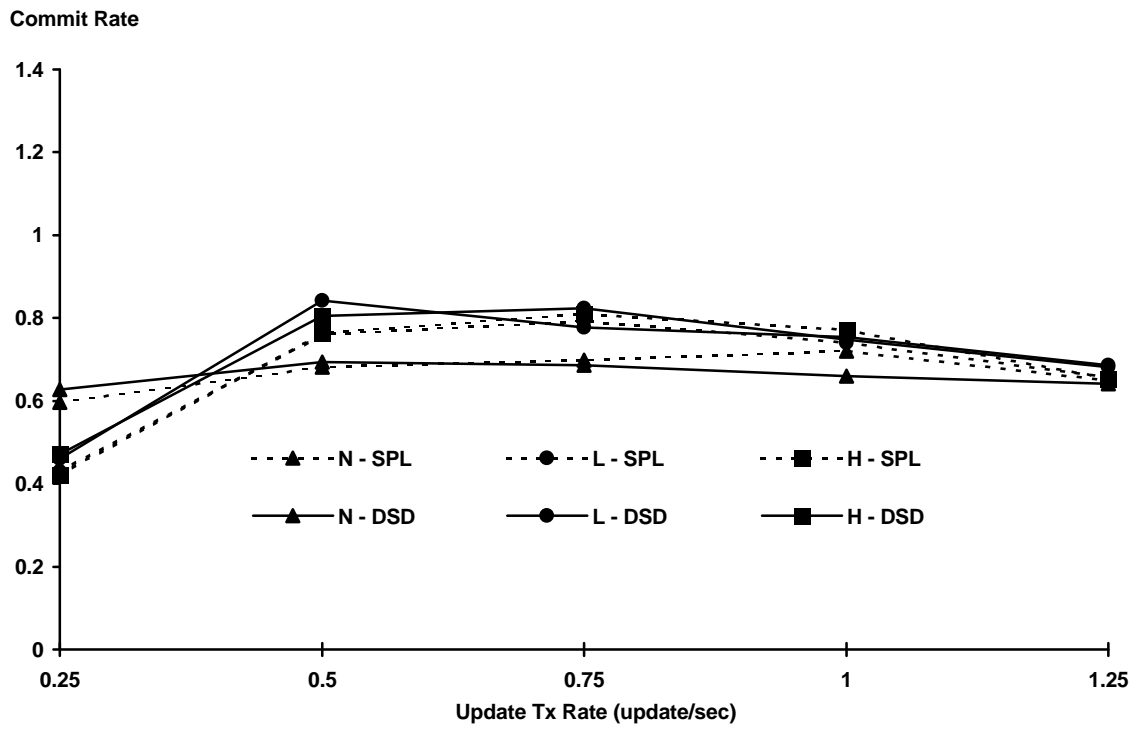


Figure 11: SPL vs DSD under Deferred Mode with Different Update Rates

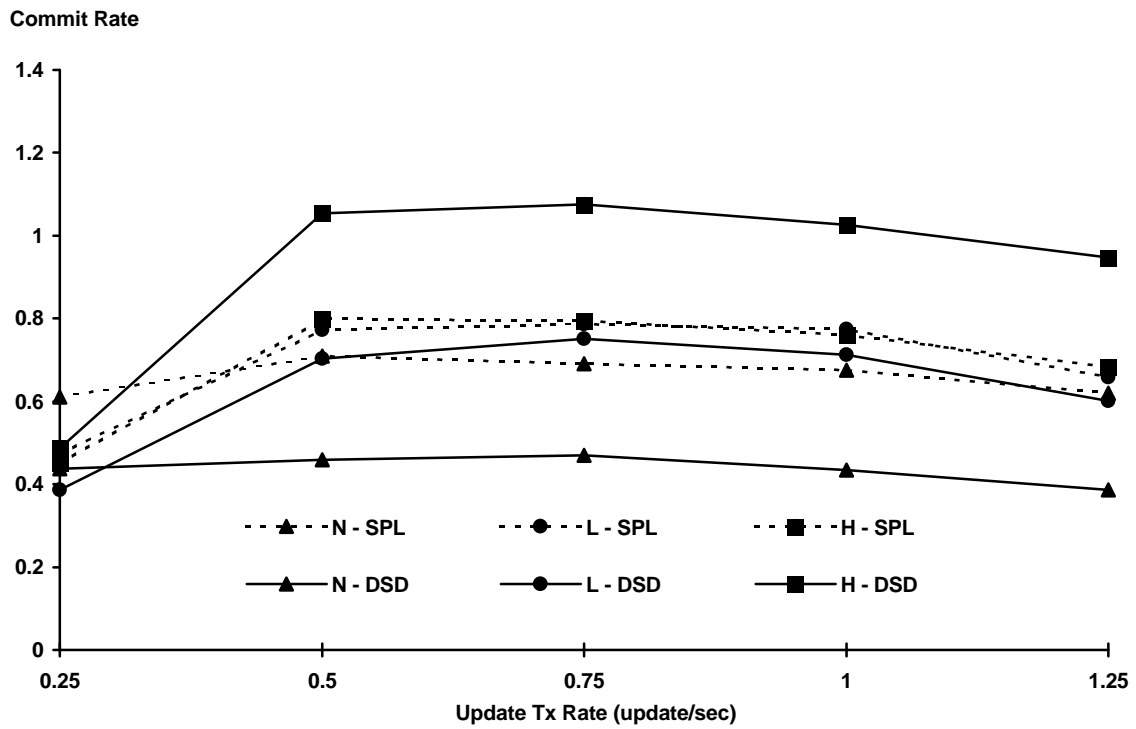


Figure 12: SPL vs DSD under Immediate Mode with Different Update Rates

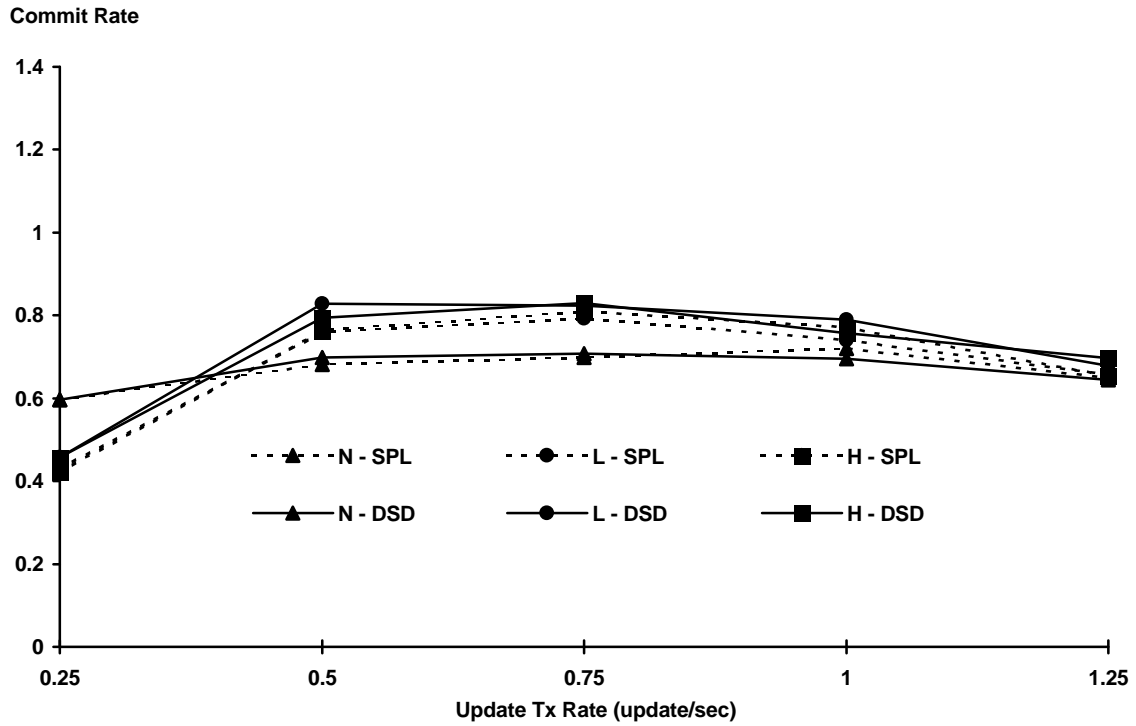


Figure 13: SPL vs DSD under Deferred Mode with Different Update Rates

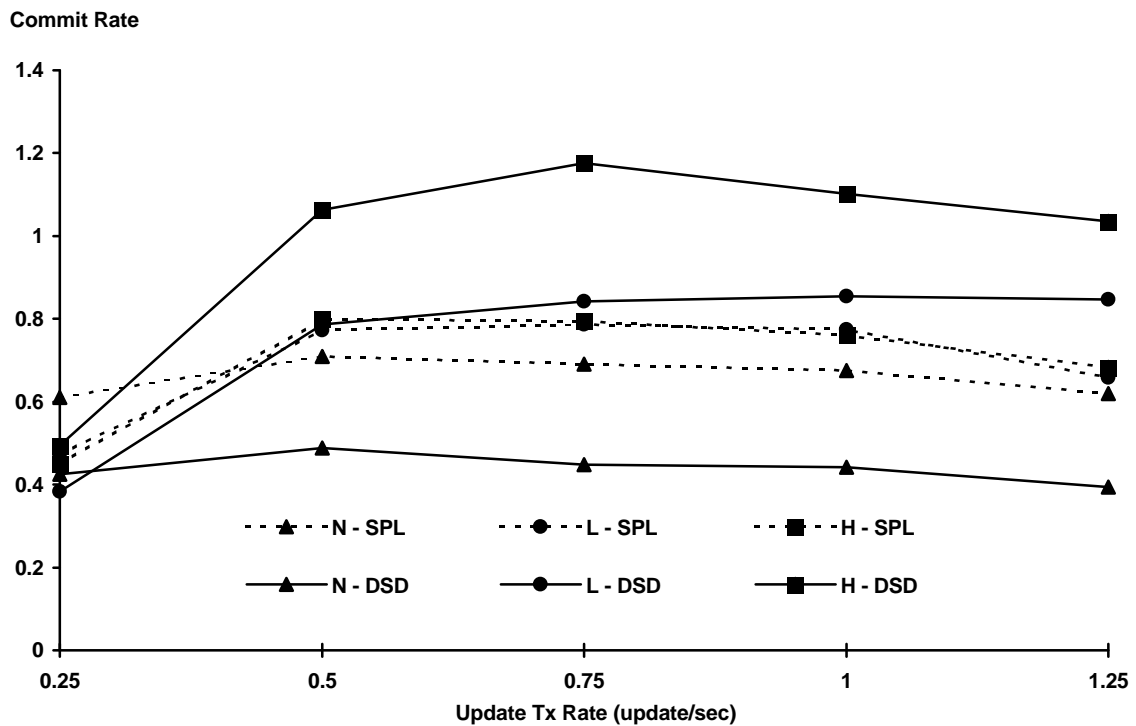


Figure 14: SPL vs DSD under Immediate Mode with Different Update Rates

Commit Rate

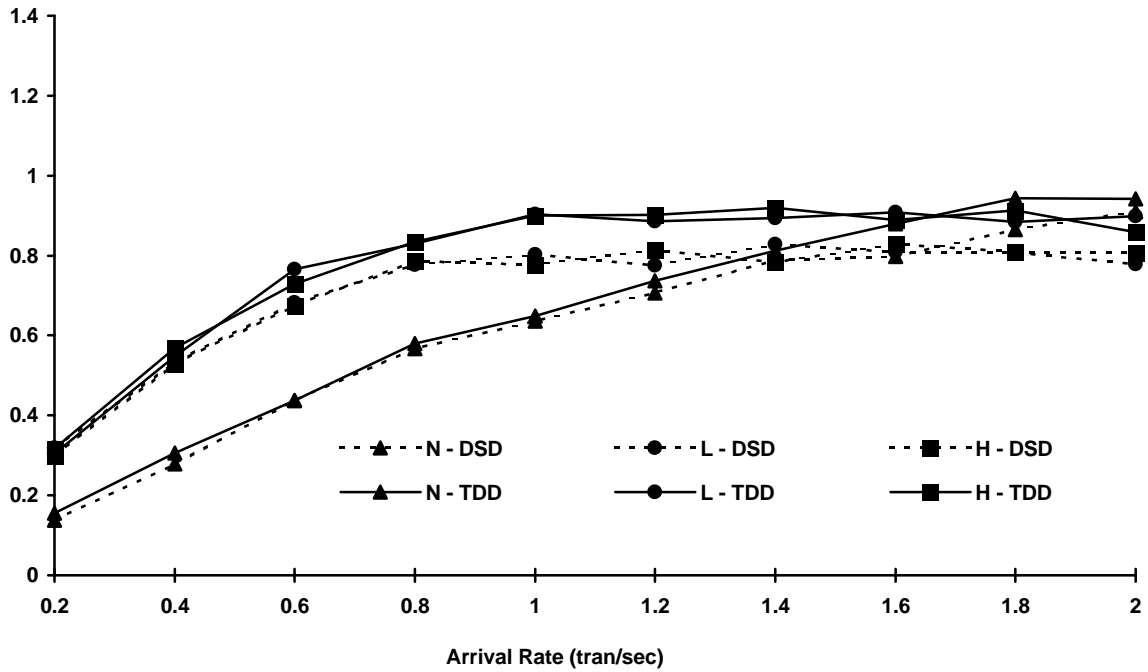


Figure 15: TDD vs DSD under Deferred Mode with Different Workload

Commit Rate

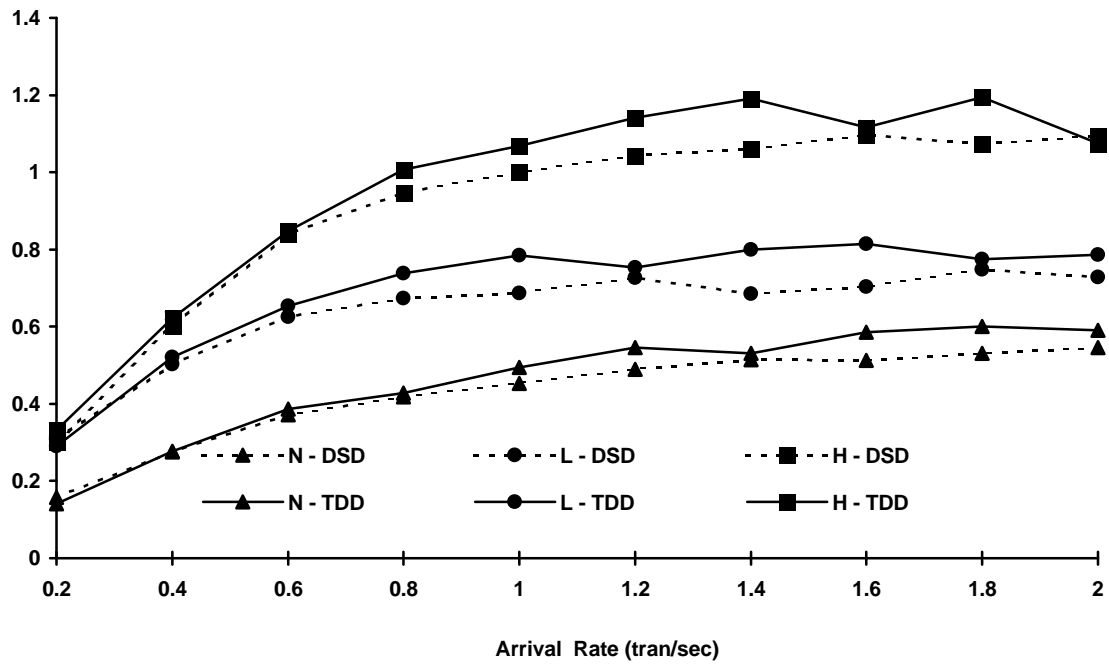


Figure 16: TDD vs DSD under Immediate Mode with Different Workload