Performance Evaluation of Speculation-Based Protocol for read-only transactions

by

T. Ragunathan, P Krishna Reddy

in

ACM Compute 2010
(ACM Compute 2010)

Report No: IIIT/TR/2010/45

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
January 2010
Performance Evaluation of Speculation-Based Protocol for Read-only Transactions

T. Ragunathan  
International Institute of Information Technology  
Hyderabad, India.  
ragunathan@research.iiit.ac.in

P. Krishna Reddy  
International Institute of Information Technology  
Hyderabad, India.  
pkreddy@iiit.ac.in

ABSTRACT
In the literature, speculation-based protocols have been proposed to improve the performance of read-only transactions (ROTs) over the existing two-phase locking (2PL) and snapshot isolation (SI)-based protocols. In this paper, we have compared the performance of asynchronous speculation-based protocol with 2PL and SI-based protocols through analytical and simulation methods. The results show that asynchronous speculation-based protocol improves the performance over 2PL and SI-based protocols.

Categories and Subject Descriptors
H.2.4 [Database management]: System—c ocurrency

General Terms
Performance

Keywords
Concurrency control, speculation, performance evaluation, transaction management

1. INTRODUCTION
A read-only transaction (ROT) performs read operations on the database, whereas an update transaction (UT) performs both read and write operations on the database. The performance of two-phase locking (2PL) protocol deteriorates with data contention. Snapshot isolation (SI)-based protocols suffer from correctness and data currency problems. The main research issue is to investigate a protocol which processes ROTs correctly with high data currency and performance.

Speculation-based protocols are proposed in the literature [3] and [4], for improving the performance of ROTs over 2PL and SI-based protocols, without compromising correctness and data currency.

In this paper, we have compared the performance of asynchronous speculation-based protocol with 2PL and SI-based protocols through analytical and simulation methods. The results show that asynchronous speculation-based protocol improves the performance over 2PL and SI-based protocols.

2. 2PL, SI-BASED AND SPECULATION-BASED PROTOCOLS
Some of the notations we employed are as follows. Transactions are represented with T1, T2, ... Data objects are denoted with 'x', 'y'. For the data object 'x', 'xi' (i = 0 to n) represents ith version of 'x'. The notation ri[xi] indicates that read operation is executed on 'xi' by the transaction Ti, and wi[xi] denotes that the transaction Ti performs a write operation on a particular version of 'x' and produces 'xi'. The notations 'si', 'ci', and 'ai' denote the start, commit and abort of Ti respectively. Tij indicates jth speculative execution of Ti.

2.1 2PL protocol
The 2PL protocol processes ROTs by following two-phase rules. The processing of ROTs is delayed if they conflict with UTs. As a result the performance decreases with data contention.

In Figure 1, both T1 and T3 are UTs and T2 is an ROT. It can be observed that T2 has to wait for a lock on the object 'x' until T1 commits. Similarly, T3 has to wait for a lock on 'y'. In this way, in 2PL, performance of ROTs suffers as data contention increases.

![Figure 1: Depiction of transaction processing with 2PL](image)

2.2 SI-based protocols
In SI-based protocols, an ROT (T_i) reads data from the snapshot of the data available when T_i has started or generated the first read operation. The modifications performed by other concurrent UTs, which have started their execution after T_i are unavailable to T_i. We consider one of the SI-based protocol “first committer wins rule” (FCWR) [1]. In FCWR, a transaction (T_i) commits if and only if no concurrent transaction (T_j) has already committed writes of data objects that T_i intends to write.
The processing of ROTs using FCWR is depicted in Figure 2. In this figure, both T1 and T3 are UTs, and T2 is a ROT. It can be observed that T2 reads the currently available values \( y_0 \) and \( z_0 \) and proceeds with the execution. Simultaneously, T3 also reads \( x_0 \) and produces \( x_2 \). Note that, FCWR allows only one of the conflicting UTs to commit. So, T3 has to be aborted as T1 commits. However, as per FCWR, T2 commits with the old values and it has not accessed the updates produced by T1 even though T1 commits before its completion and therefore receives low data currency.

\[
\begin{array}{c}
T_1 \quad \begin{array}{c}
| x | w | y | \ldots | z | \ldots | x |
\end{array} \\
T_2 \quad \begin{array}{c}
| x | j | u | x | \ldots | x | \ldots | x |
\end{array} \\
T_3 \quad \begin{array}{c}
| x | l | w | x | \ldots | x | \ldots | x |
\end{array}
\end{array}
\]

Figure 2: Depiction of transaction processing with FCWR

2.3 Speculation-based Protocols

In speculation-based protocols [3] [4], ROTs are processed using speculation and UTs are processed with 2PL. In speculation-based protocols, an ROT carries out speculative executions by accessing before- and after-images of conflicting active transactions. After completion, that ROT retains one of the executions based on the termination status of preceding UTs. Note that, speculation-based protocols do not violate correctness and compromise data currency. In this paper, we have considered asynchronous speculative locking protocols for ROTs (ASLR) for performance evaluation.

The processing of ROTs under ASLR is shown in Figure 3. Here T2 accesses the before-image \( x_0 \) and other available values of data objects \( y_0 \) and \( z_0 \) and starts speculative execution T21. Once the after-image \( x_1 \) becomes available, another speculative execution T22 is started. Note that T21 and T22 are executed in a parallel manner. Whenever the processing is completed for any one of the speculative execution, the ROT can be committed provided it contains the effect of committed transactions at that instant. We can observe that, T21 does not depend on T1 and it is committed once it completes the execution without waiting for T1. So, T22 is aborted. Note that being UT, T3 waits for T1 for the release of the lock on \( x \) as per 2PL rule.

3. PERFORMANCE EVALUATION THROUGH ANALYTICAL MODELING

The transaction response time depends upon the occurrence of data conflict in accessing the database and also on the queuing and processing delay in accessing hardware resources. We discuss these aspects in the following subsections after covering transaction model.

3.1 Transaction model

The transaction model consists of \( n_L + 2 \) states, where \( n_L \) is the number of the data objects accessed by the transaction. The transaction has an initial setup phase (including program fetch), state 0. Following the initial set up, a transaction progresses to states 1, 2, . . . , \( n_L \) in that order. This is the execution phase. At the start of each state, i, the transaction begins to access a new data object and moves to state \( i + 1 \) when the next data object access begins. At the end of the state \( n_L \), if successful, the transaction enters into the commit phase at state \( n_L + 1 \).

3.2 Hardware resource model

We consider that there are K tightly coupled processors available and a database is spread over 2K disks. Assuming poisson arrivals of transactions, we model the processors and disks as M/M/m (m refers to multiple CPU and I/O servers) with FCFS discipline. The CPU and I/O waiting time can be calculated based on the discussion in [2].

Let \( T_{cpuServer} \) be the CPU time required to access a database object and to perform the computations on that data object, \( N_{cpuServers} \) be the number of CPUs present in the system, cpuUtilization be the utilization of CPU and \( P_{tasks \geq N_{cpuServers}} \) be the probability that there are as many more requests than we have CPUs. Then the average waiting time of CPU is given by

\[
T_{cpuWaitingTimeQueue} = T_{cpuServer} \times \frac{P_{tasks \geq N_{cpuServers}}}{N_{cpuServers} \times (1 - cpuUtilization)} \tag{1}
\]

Let \( T_{ioServer} \) be the time required to access a database object from the disk, \( N_{ioServers} \) be the number of disks present in the system, ioUtilization be the IO utilization and \( P_{tasks \geq N_{io Servers}} \) be the probability that there are as many more requests than we have disks. Then the average waiting time of the transaction requests waiting in the queue maintained for disks is given by

\[
T_{ioWaitingTimeQueue} = T_{ioServer} \times \frac{P_{tasks \geq N_{ioServers}}}{N_{ioServers} \times (1 - ioUtilization)} \tag{2}
\]

3.3 Analytical model for 2PL, ASLR and FCWR protocols

1. Calculating response time for 2PL protocol

Let \( P_{UT} \) and \( P_{ROT} \) be the percentage of UTs and ROTs respectively, \( R_{UT} \) and \( R_{ROT} \) be the average response time for UTs and ROTs respectively. Then average response time can be calculated as given below.

\[
R = P_{UT} R_{UT} + P_{ROT} R_{ROT} \tag{3}
\]

Let \( R_{INPL} \) be the time taken for loading and initiating the transaction, \( L_U \) and \( L_R \) be the number of data objects accessed by the UTs, and ROTs respectively. \( T_{Commit} \) be the commit time of a UT. Then average response time by considering UTs and ROTs is computed.

\[
R_{UT} = R_{INPL} + L_U \bar{a} + L_U P_{RW} R_W + T_{Commit} \tag{4}
\]
\[ R_{\text{ROT}} = R_{\text{INPL}} + N_{\text{LR}} a + N_{\text{LR}} P_{\text{RW}} \] \hfill (5)

Next, commit time required for UTs is computed.
\[ T_{\text{Commit}} = c N_{\text{LU}} \] \hfill (6)

In the above equation, \( c \) is the time taken to commit single data object. Let \( a \) be the mean time spent for reading a data object and performing computations. We compute \( a \) as given below.
\[ a = \frac{\text{Time}_{\text{cpuServer}} + \text{cpuWaitingTime}_{\text{queue}}}{\text{Time}_{\text{ioServer}} + \text{ioWaitingTime}_{\text{queue}}} \] \hfill (7)

In the above equation, \( \text{Time}_{\text{cpuServer}} \) and \( \text{Time}_{\text{ioServer}} \) are the average CPU time and I/O time required for serving a request given by a transaction for a data object. Let \( N_{\text{LRU}} \) be the average number of data objects accessed by a transaction and \( P_W \) be the probability of lock contention. Next, \( \text{cpuUtilization} \) and \( \text{ioUtilization} \) is computed.
\[ \text{cpuUtilization} = \frac{\text{arrivalRate(\text{Time}_{\text{cpuServer}} + N_{\text{LRU}} \text{Time}_{\text{cpuServer}})}}{P_W \text{arrivalRate(\text{Time}_{\text{cpuServer}} + N_{\text{LRU}} \text{Time}_{\text{cpuServer}})}} \]
\[ \text{ioUtilization} = \frac{\text{arrivalRate(\text{Time}_{\text{ioServer}} + N_{\text{LRU}} \text{Time}_{\text{ioServer}})}}{P_W \text{arrivalrate(\text{Time}_{\text{ioServer}} + N_{\text{LRU}} \text{Time}_{\text{ioServer}})}} \]

Next, \( N_{\text{LRU}} \) is computed.
\[ N_{\text{LRU}} = N_{\text{LR}} P_{\text{ROT}} + N_{\text{LU}} P_{\text{UT}} \] \hfill (8)

The \( \text{cpuWaitingTime} \) and \( \text{ioWaitingTime} \) are calculated as per the Eqs. (1) and (2). The calculation of sum of mean lock holding time is modified by considering both the ROTs and UTs. The time a UT conflicts waits for a conflicting ROT is calculated as given below.
\[ G_1 = \sum_{i=1}^{N_{\text{LU}}} (ia + (i-1)b) \] \hfill (9)

Next, the time a UT waits for a conflicting UT is calculated.
\[ G_2 = \sum_{i=1}^{N_{\text{LU}}} (ia + (i-1)b) + N_{\text{LU}} T_{\text{Commit}} \] \hfill (10)

Next, the time an ROT waits for conflicting UT is computed.
\[ G_3 = \sum_{i=1}^{N_{\text{LU}}} (ia + (i-1)b) + N_{\text{LU}} T_{\text{Commit}} \] \hfill (11)

As per our assumption, in ROT processing environment, at a time \( P_{\text{ROT}} \) and \( P_{\text{UT}} \) percentage of ROTs and UTs are present in the system. So, the mean lock holding time \( G \) is calculated as given below.
\[ G = P_{\text{ROT}} G_3 + P_{\text{UT}} (P_{\text{ROT}} G_1 + P_{\text{UT}} G_2) \] \hfill (12)

Next, the probability of lock contention is calculated.
\[ P_W = \frac{\text{arrivalRate} G}{L} \] \hfill (13)

Next, we compute mean lock waiting time. The time a UT waits for conflicting ROT or UT, is computed as given below.
\[ R_{W1} = \frac{((a + b)(N_{\text{LU}} - 1)) + N_{\text{LU}} T_{\text{Commit}}}{3} \] \hfill (14)

\[ R_W = \frac{((a + b)(N_{\text{LU}})) + N_{\text{LU}} T_{\text{Commit}}}{3} \] \hfill (16)

The time an ROT waits for a conflicting UT is computed as given below.
\[ R_{W3} = ((a + b)(N_{\text{LU}})) + N_{\text{LU}} T_{\text{Commit}}/3 \]

Next, the mean lock waiting time by considering both the ROTs and UTs is computed.
\[ R_W = P_{\text{ROT}} R_{W3} + P_{\text{UT}} (P_{\text{ROT}} R_{W1} + P_{\text{UT}} R_{W2}) \] \hfill (17)

\[ b = P_W R_W \] \hfill (18)

We compute \( a, P_W \) and \( R_W \) by using iterative procedure. Initially, we assume a small fractional value for \( P_W \) to compute \( \text{cpuWaitingTime}_{\text{queue}} \) and \( \text{ioWaitingTime}_{\text{queue}} \). Then, we can calculate \( a \). Next, we calculate \( G, P_W \) and \( R_W \) by initializing \( b \) to 0. After this, we calculate the next value of \( b \) and \( a \). Then, calculation of \( G, P_W \) and \( R_W \) are performed by considering new \( a \) and \( b \). This process continues and typically converges in a few iterations. Once this process is completed, the values of \( a, P_W \) and \( R_W \) are known.

2. Calculating response time for ASLR protocol

In ASLR, only UTs will wait for locks. The time a UT conflicting with ROT waits, is calculated as given below.
\[ G_1 = \sum_{i=1}^{N_{\text{LU}}} (ia) \] \hfill (19)

Next, the time a UT conflicting with another UT is computed.
\[ G_2 = \sum_{i=1}^{N_{\text{LU}}} (ia + (i-1)b) + N_{\text{LU}} T_{\text{Commit}} \] \hfill (20)

Next, the mean lock holding time \( G \) is calculated.
\[ G = P_{\text{UT}} (P_{\text{ROT}} G_1 + P_{\text{UT}} G_2) \] \hfill (21)

Then, the probability of lock contention \( P_W \) is calculated.
\[ P_W = \frac{\text{arrivalRate} G}{L} \] \hfill (22)

The time a UT waits for another conflicting UT, is calculated as given below.
\[ R_{W1} = (((a + b)(N_{\text{LU}} - 1)) + N_{\text{LU}} T_{\text{Commit}})/3 \] \hfill (23)

Next, the mean lock waiting time by considering both UTs and ROTs is calculated.
\[ R_{W2} = (a(N_{\text{LU}}))/3 \] \hfill (24)

Next, we calculate the mean lock waiting time by considering both the ROTs and UTs as given below.
\[ R_W = P_{\text{UT}} (P_{\text{UT}} R_{W1} + P_{\text{ROT}} R_{W2}) \] \hfill (25)

The iterative procedure discussed for 2PL is also followed here for computing \( a, P_W \) and \( R_W \). Next, average response time is calculated by considering both UTs and ROTs.
\[ R_{\text{UT}} = R_{\text{INPL}} + N_{\text{LU}} a + N_{\text{LU}} P_{\text{RW}} + T_{\text{Commit}} \] \hfill (26)

\[ R_{\text{ROT}} = R_{\text{INPL}} + N_{\text{LR}} a \] \hfill (27)

Then average response time is computed by using Eq. 3.
3. Calculating response time for FCWR protocol

Let nuts be the total number of UTs and we calculate cpuUtilization, ioUtilization, probability of aborts (PA) and mean data object holding time (TH) as given below.

\[
\text{cpuUtilization} = \frac{\text{arrivalRate}(\text{Time}_{\text{cpuServer}} + N_{LU} \text{Time}_{\text{cpuServer}} \text{nuts})}{\text{N}_{\text{cpuServers}}} + P_A(N_{LU} \text{Time}_{\text{cpuServer}} \text{nuts}) (28)
\]

\[
\text{ioUtilization} = \frac{\text{arrivalRate}(\text{Time}_{\text{ioServer}} + N_{LU} \text{Time}_{\text{ioServer}}) + P_A(N_{LU} \text{Time}_{\text{ioServer}} \text{nuts})}{\text{N}_{\text{ioServers}}} (29)
\]

\[P_A = \text{nuts}(N_{LU}^2(TH + T_{\text{COMMIT}})/L) \ldots (30)\]

\[TH = ((N_{LU} + 1)\alpha)/2 \ldots (31)\]

Let TBackoff be the average time required to start rerunning a transaction after its abort. Then, the average response time by considering the UTs, is computed as given below.

\[R_{UT} = R_{INPL} + N_{LU}a + T_{\text{COMMIT}} + P_A(T_{\text{Backoff}} + N_{LU}a) \ldots (32)\]

We consider TBackoff is equivalent to TCommit [5]. Next, average response time by considering the ROTs is computed.

\[R_{ROT} = R_{INPL} + N_{LRA} \ldots (33)\]

Let RE be the time required for executing a transaction which is computed as given below.

\[R_E = N_{LRA} \ldots (34)\]

Let R′E be the time required for rerunning a transaction which is same as RE. Then, the average response time can be calculated as given below.

\[R = R_{INPL} + R_E + P_A(T_{\text{Backoff}} + R_E) + T_{\text{Commit}} \ldots (35)\]

4. PERFORMANCE RESULTS

We have developed a discrete event simulator based on open queuing model for performance evaluation. In the performance graphs, we have used the symbols ‘-S’ and ‘-A’ to indicate ‘simulation’ and ‘analytical’ results of the protocols, respectively. Figure 4 shows the average response time performance. We have found similar trends in the results of both analytical methods and simulation experiments. The performance graphs, we have used the symbols ‘-S’ and ‘-A’ to indicate “simulation” and “analytical” results of the protocols.

5. CONCLUSIONS

In this paper, we have compared the performance of asynchronous speculation-based protocol with 2PL and SI-based protocols. The results of simulation and analytical studies indicate that the the asynchronous speculation-based protocol performs better than both 2PL and SI-based protocols in processing ROTs.