Speculation-Based Protocols for Improving the Performance of Read-only Transactions

by

T. Ragunathan, P Krishna Reddy

in

COMAD 2009 PhD Workshop


Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2009
Speculation-Based Protocols for Improving the Performance of Read-only Transactions

T. Ragunathan and P. Krishna Reddy
International Institute of Information Technology
Gachibowli, Hyderabad, India
ragunathan@research.iiit.ac.in, pkreddy@iiit.ac.in

Abstract
This paper presents an ongoing Ph.D. thesis work which aims at improving the performance of read-only transactions (ROTs) in database systems using the notion of speculation. In the literature, speculative locking approach has been proposed to improve the transaction performance in online transaction processing environments. In this thesis, we have proposed two speculation-based protocols to improve the performance of ROTs, by making appropriate modifications to the speculative locking protocol. Further, we have come up with semantics-based protocols for improving the performance of ROTs with less number of speculative executions. The proposed protocols process ROTs without any data currency and correctness issues. We have also evaluated the performance of speculative locking protocols with manageable extra processing resources (0.2 times).

1 Research problem and its importance
We have taken “Read-only transaction (ROT) processing” as the research problem. A read-only transaction does not modify any data. The main issues in processing ROTs are correctness (serializability), data currency, and performance. In the emerging e-commerce scenario, information systems should meet intensive information requirements from a large number of users. The information systems frequently process ROTs or queries. In such systems, the ROTs should be processed efficiently without any correctness issues. Also, in many web-based information systems like online stock exchange systems, data currency provided to ROTs is very important. So, effective processing of ROTs has become an important research problem in the current e-commerce and web databases context.

2 Related Work
Four isolation levels are specified in ANSI/ISO SQL-92 standard [1] for processing transactions. The popular two-phase locking (2PL) protocol [6] [9] processes the transaction correctly with serializability as correctness criteria, but its performance deteriorates with data contention. Snapshot isolation (SI)-based protocols [10] are proposed for improving the performance of ROTs. But, these protocols compromise on correctness and data currency issues [3][21]. In the literature, speculative locking (SL) protocols [11] have been proposed to improve the transaction processing performance in distributed database systems without compromising correctness and data currency. By carrying out multiple executions for a transaction, SL increases parallelism by trading extra processing resources and without violating serializability criteria. However, in SL, the speculative executions explode with the data contention. There are efforts to improve the performance by processing ROTs with multi-version based approach [20] and by proposing separate protocols for ROTs and update transactions [7] [20].

Use of commutative property of operations is discussed in [8], [5] [22] [23], [7], [2], [4], [12], [19] and [13]. In [14], analytical models for locking and optimistic protocols are discussed.

3 Research approach
The widely used 2PL protocol [6] [9] processes the transactions with serializability as correctness criteria. 2PL performs poorly, as both ROTs and update transactions (UTs) are made to wait whenever conflicts occur. Even though 2PL provides high data currency to transactions, it is not a good choice for processing ROTs.

Snapshot isolation (SI)-based methods [10] are used in popular DBMSs like Oracle and Microsoft SQL
server to process ROTs. These DBMSs use variations of SI like “First committer wins rule (FCWR)” or “First updater wins rule (FUWR)” for processing ROTs. Note that, ROTs processed under FCWR or FUWR protocols violate serializability criteria [3] [21]. Also, under these protocols, ROTs are provided with low data currency.

We have chosen to use speculation-based protocols, as they have got the scope for improving the performance of ROTs without correctness and data currency problems. As a part of the thesis work, we have proposed two speculation-based protocols for processing ROTs [16], [18]. We have investigated further regarding ROT processing environment and come up with semantics-based protocols. The semantics-based protocols can process the ROTs with less number of speculative executions by executing certain class of ROTs and UTs without blocking.

Next, we discuss the basic idea of speculative- and semantics-based protocols with transaction-processing examples. Also, we present some of the performance results of our simulation and analytical studies on these protocols.

3.1 Speculative locking protocol for ROTs

We have extended the basic speculative locking protocol and come up with two protocols namely (i) Synchronous speculative locking protocol for ROTs (SSLR) (ii) Asynchronous speculative locking protocol for ROTs (ASLR). Next, we discuss these protocols one by one.

i. Synchronous speculative locking protocol for ROTs

We have proposed SSLR protocol [16], by adding two aspects to the basic SL protocol.

(a) In SSLR only ROTs are processed with speculation. The UTs are processed with 2PL. We assume that a UT releases the locks (converts EW-lock into SPW lock) whenever it produces after-images. Whenever a ROT conflicts with a UT, it carries out speculative executions by accessing both before- and after-images of the preceding UTs.

(b) The other aspect is regarding commitment of ROTs. In the SL [11], a waiting transaction carries out speculative executions and waits for the commitment of preceding transactions. Whereas, in SSLR whenever ROT completes execution, it commits by retaining appropriate execution. In SSLR an ROT does not wait for the termination of conflicting active transactions. However, it can be noted that, a UT waits for the termination of preceding UTs and ROTs.

The lock compatibility matrix of SSLR is shown in Figure 1. Similar to the case of basic SL protocol, W-lock is divided into EW-lock and SPW-lock. UTs request EW-lock for writing the data object. The EW-lock is converted into the SPW-lock after the work on

\[
\begin{array}{c|c|c|c|c|c}
\text{Lock requested} & \text{Lock held by } T_i & \text{Lock requested} & \text{Lock held by } T_i \\
\hline
\text{RR} & \text{yes} & \text{RR} & \text{yes} & \text{EW} & \text{no} & \text{SPW} & \text{yes} \\
\text{RU} & \text{no} & \text{no} & \text{no} & \text{no} & \text{no} & \text{no} & \text{no} \\
\end{array}
\]

Figure 1: Lock compatibility matrix for SSLR

Figure 2: Depiction of transaction processing with SSLR

the data object is completed. Separate read-locks are employed for UTs and ROTs. A UT requests RU-lock (read lock for UT) for reading a data object and an ROT requests RR-lock (read lock for ROT) for reading a data object. The entry "sp yes" indicates that the requesting transaction carries out speculative executions and forms commit dependency with the lock holding transaction. In [16], SSLR is discussed in detail.

It can be noted that commit dependency in SSLR is different from SL. Let \( T_i \) be an ROT and \( T_j \) be a UT. Suppose \( T_i \) forms commit dependency with \( T_j \). In SL, \( T_i \) commits only after the termination of \( T_j \). Whereas in SSLR, whenever \( T_i \) completes, it can commit by retaining one of the speculative executions without waiting for \( T_j \) to terminate.

Figure 2 depicts processing under SSLR. Here, \( T_2 \) is an ROT and \( T_1 \) and \( T_3 \) are UTs. Whenever \( T_1 \) produces after-image 'x1', \( T_2 \) accesses both 'x0' and 'x1' and carries out two executions \( T_21 \) and \( T_22 \), respectively. After \( T_2 \)'s completion, \( T_21 \) is retained even though \( T_1 \) is not yet committed. Note that, being a UT, \( T_3 \) waits for \( T_1 \) for the release of the lock on 'x' as per 2PL rule.

ii. Asynchronous speculative locking protocol for ROTs

In SSLR, an ROT carries out speculative executions in synchronous manner. Whereas in ASLR [18], the speculative executions of an ROT can be carried out in an independent manner. The ROT is allowed to access the available data object versions and carry out speculative executions. Whenever preceding transaction produces after-image, further speculative executions can be started in a dynamic manner. The asynchronous method of processing ROTs reduces waiting and improves the performance.

Figure 4 depicts the processing under ASLR. Here \( T_2 \) accesses the before-image 'x0' and other available
values of data objects 'y_0' and 'z_0' and starts speculative execution T_{21}. Once the after-image 'x_1' becomes available, another speculative execution T_{22} is started. Note that T_{21} and T_{22} are executed in a parallel manner. Whenever the processing is completed for any one of the speculative execution, the ROT can be committed provided if it contains the effect of committed transactions at that instant. Note that being UT, T_4 waits for T_1 for the release of the lock on 'x' as per 2PL rule. The lock compatibility matrix of ASLR is shown

<table>
<thead>
<tr>
<th>Lock requested by T_i</th>
<th>Lock held by T_j</th>
<th>RUC</th>
<th>RUW</th>
<th>EWR</th>
<th>SPW</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUC</td>
<td>yes</td>
<td>yes</td>
<td>asp-yes</td>
<td>asp-yes</td>
<td></td>
</tr>
<tr>
<td>RUW</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>EWR</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Lock compatibility matrix for ASLR

Figure 4: Depiction of Transaction processing with ASLR

in Figure 3. Locks used here are similar to SSLR protocol. The entry “asp-yes”(asynchronous speculative yes) indicates that the ROT carries out the possible speculative executions by accessing available versions of data object and forms a commit dependency with the preceding UTs.

3.2 Semantics-based speculative locking protocol for ROTs

We have proposed semantics-based protocols which can exploit the semantics of the applications and speculation to process the ROTs in [17]. We have identified that an ROT which performs compensatable computations can be executed without speculation and without waiting for the conflicting UTs. Also, we have observed that the UTs conflicting with such ROTs can be executed without blocking. Based on these observations, we have come up with a notion called “compensatability” for classifying the ROTs as compensatable ROTs (CROTs) and non-compensatable ROTs (NCROTs) (please refer to [17] for details). We have proposed two protocols namely (i) Semantics-based SSLR (SSSSLR) (ii) Semantics-based ASLR (SASLR). Due to space considerations, we discuss only the details of SSSLR protocol here.

**Semantics-based SSLR** In the proposed semantics-based SSLR (SSSSLR) protocol, CROTs are executed without blocking and NCROTs are processed using synchronous speculation policy as per SSLR protocol. Note that, CROTs do not perform speculative executions, but they have to perform compensating computations during commit time. 2PL is chosen to process the UTs. However, the UTs conflicting with CROTs alone are processed without blocking. The CROTs request compensating read locks (CR-locks) for reading. The NCROTs request non-compensating read locks (NR-locks) for reading. The UTs request read update locks (RU-locks) for reading and exclusive write locks (EW-locks) for writing. The lock compatibility matrix of SSSLR is shown in Figure 5. The entry “yes” indicates that the corresponding locks are compatible and “no” indicates that the corresponding locks are incompatible. The entry “sm-yes” (semantic yes) indicates that the requesting CROT is allowed to continue the execution. Note that, the UTs conflicting with CROTs are allowed to continue without blocking, which is different from the 2PL procedure. The entry “sp-yes” (speculation yes) indicates that the requesting NCROT carries out speculative executions with the after-image produced by the preceding UT and forms a commit dependency with that UT.

Figure 6 shows the processing of transactions in SSSLR protocol. Here T_2 is a NCROT, T_3 is a CROT, T_1 and T_4 are UTs. Note that, T_2 waits for T_1 to produce after-image. Once the after-image is available, being a NCROT, T_2 proceeds with speculative executions. In the figure, T_3 conflicts with T_1 based on the data objects 'y' and 'p'. Also T_3 conflicts with T_4 based on the data object 'z'. Being a CROT, T_3 proceeds its execution without speculation and blocking. During commitment, T_3 performs compensating computations by reading the modified values of 'y', 'p' and 'z' from the transaction log. These values are available in the log as the UTs T_1 and T_4 are committed before T_3. In SSSLR protocol, the UTs conflicting with CROTs alone, are executed without blocking. By following this procedure, T_4 is executed without blocking even though it is conflicting with T_3 based on the data object 'z'. Note that, this type of processing does not violate the serializability criteria.

4 Current status of research

The following works have been completed in this thesis work. (i) SSLR, ASLR, SSSLR, SASLR protocols have been developed. (ii) Simulation study based on open and closed queue models have completed. (iii)
Correctness proof for SSLR and ASLR protocols are developed. Towards completion of thesis work, performance study through analytical modeling has to be completed.

5 Results achieved

The performance results of 2PL, FCWR, SI-2PL, SSLR, ASLR, SSSLR and SASLR protocols are discussed here. Note that, in SI-2PL protocol, ROTs are processed using SI and UTs are processed using 2PL. For simulation and analytical studies, we have considered an environment where 70% ROTs and 30% UTs are present.

(i) Results under unlimited resources

The results obtained under unlimited resources are discussed here. Figure 7 shows how throughput performance for 2PL, FCWR, SL, SSLR, ASLR and SI-2PL vary with MPL. It can be noted that the performance of ASLR is significantly higher than 2PL and FCWR. 2PL performs poorly as the waiting time of the transactions is more in 2PL. In FCWR, the performance deteriorates due to its "first committer wins rule" as more number of UTs get aborted as data contention increases. Note that ASLR performance is better than both SL and ASLR due to the reduced waiting as a result of asynchronous speculation. Note that in both SL and SSLR, ROTs wait for the lock conversion from EW-lock to SPW-lock. We observe that the performance of SL and SSLR is close. Also, it can be observed that the performance of ASLR is more than SI-2PL.

(ii) Results under limited resources

Throughput
Average response time (in milliseconds) 2PL-A
FCWR-A
FCWR-S
ASLR-A
ASLR-S
Throughput
Average response time
Time
Total Memory Units (in multiples of MPL)
#UTs = 30%, #RUs= 8
MPL
2PL
SSLR
ASLR
SL

Figure 8: MPL versus Throughput in limited resources environment

The results obtained under unlimited resources are discussed here. Figure 8 shows the performance of 2PL, SL, SSLR and ASLR protocols by simulating limited resources environment. The resources are allocated in terms of memory units (MUS). We assumed that each memory unit carries out one speculative execution. If sufficient number of MUS is not available to carry out speculative executions, the transaction is put to wait. From Figure 8, it can be observed that the performance of both ASLR (also SSLR) reaches maximum value and saturates at MUS values equal to 1.2*MPL. Note that the performance of SL does not reach the performance of ASLR even after doubling the MUS values equal to 2*MPL. Note that the performance of 2PL is not affected with the number of additional MUS. Overall, the simulation experiments show that the performance of ASLR is better than 2PL, FCWR, SL and SSLR protocols. ASLR requires a fraction of additional resources equal to 0.2*MPL to achieve high performance.

(iii) Performance comparison with analytical and simulation results

Here, we discuss response time performance comparison of 2PL, FCWR and ASLR protocols, based on the results obtained through analytical and simulation methods.

Figure 9: Transaction arrival rate versus Average response time

Figure 9 shows the average response time performance of transactions by considering both the UTs and ROTs with the increase in transaction arrival rate. We have found similar trends in the results of both analytical methods and simulation experiments. As expected, the performance of 2PL is poor due to high
data contention. This is exhibited by both analytical and simulation results. We can observe that as transaction arrival rate increases beyond 20, the response time of 2PL shoots up, due to high data contention. In FCWR, the conflicting UTs are aborted and re-submitted. As the transaction arrival rate increases, more UTs arrive into the system and so the probability of aborts increases for UTs. As more number of UTs are aborted, the performance of FCWR comes down. It can be observed in figure 9 that ASLR protocol performs better than both 2PL and FCWR protocols. Even in a high data contention situation, the average response time for ASLR protocol is very less in comparison with 2PL and FCWR protocols. This is due to the fact that ASLR protocol processes ROTs without waiting by following asynchronous speculation policy."

(iv) Speculative executions under semantics-based protocols

Here, we compare the average number of speculative executions required for SSLR, ASLR, SSSLR and SASLR protocols based on the results obtained through simulation experiments. Figure 10 shows average number of speculative executions per transaction required for SSLR, ASLR, SSSLR and SASLR by varying “% of CROTs” under 30% UTs environment. We can observe that for SSSLR and SASLR protocols, the average number of speculative executions per transactions has been less. This is because, CROTs are processed without speculation in SSSLR and SASLR protocols.

6 Expected contribution

One of the main contributions of this Ph.D. thesis is the development of two speculation-based protocols namely (i) Synchronous Speculative Locking Protocol for ROTs (SSLR)(ii) Asynchronous Speculative Locking Protocol for ROTs (ASLR) [16], [18] and [15]. The second contribution is the correctness proof for SL-based protocols [15]. The third contribution is development of semantics-based protocols for processing ROTs [17]. As a part of thesis work, we are also investigating the performance evaluation of SL-based protocols through analytical modeling.

As a part of future work, we like to investigate speculation-based protocols for processing ROTs in distributed database systems. Also, we are planning to develop analytical model for evaluating the performance of semantics-based protocols.

References