Semantics-Based Asynchronous Speculative Locking Protocol for Improving the Performance of Read-only Transactions

by

T. Ragunathan, P Krishna Reddy, Mohit Goyal

in

2010 Spring Simulation Multi Conference
(SpringSim 2010)

Report No: IIIT/TR/2010/42

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
April 2010
ABSTRACT

Speculative locking (SL) protocols have been proposed in the literature for improving the performance of read-only transactions (ROTs) without correctness and data currency issues. In these protocols, ROTs carry out speculative executions and update transactions (UTs) follow two-phase locking (2PL). In these protocols, UTs are blocked if they conflict with ROTs. To reduce blocking of UTs, semantics-based protocol has been proposed in the literature by exploiting the “compensatability” property of ROTs. In that protocol compensatable ROTs are processed without blocking and non-compensatable ROTs are processed by using synchronous speculation method. In this paper, we have proposed a semantics-based speculative locking protocol for ROTs in which non-compensatable ROTs are processed with asynchronous speculation method. The simulation results show that the proposed approach improves the performance of ROTs over other protocols.

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

General Terms
Performance

Keywords
Concurrency control, speculation, performance evaluation, transaction management, simulation

1. INTRODUCTION

A read-only transaction (ROT) performs only read operations on the database. In the emerging e-commerce scenario, improving the performance of ROTs without correctness and data currency issues is an important research problem. The two-phase locking (2PL) protocol processes ROTs by following two-phase rules. The processing of ROTs is delayed if they conflict with update transactions (UTs). As a result the performance decreases with data contention. Snapshot isolation (SI)-based protocols suffer from correctness and data currency problems.

In the literature, speculative locking (SL) protocols [4] have been proposed to improve the transaction processing performance in distributed database systems. 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. We have proposed speculation-based protocols [2], [3] and [1], for improving the performance of ROTs over 2PL and SI-based protocols, without compromising correctness and data currency.

A semantics-based synchronous speculative locking protocol for ROTs (SSSLR) to improve the performance ROTs is proposed in [6]. The SSSLR protocol exploits the notion of “compensatability” and synchronous speculation for improving the performance of ROTs. In this paper, we have processed the ROTs using asynchronous speculation method by exploiting “compensatability” property of ROTs. The simulation results indicate that the proposed semantics-based asynchronous approach improves the performance of ROTs over the 2PL, snapshot isolation (SI)-based, speculation-based and semantics-based synchronous speculative locking protocols.

The rest of the paper is organized as follows. In Section 2, we discuss the speculation and semantics based protocols for processing ROTs. In Section 3, we discuss the details of the proposed protocol. In Section 4, we present the performance results. The last section contains conclusions and future work.

2. SPECULATION- AND SEMANTICS-BASED PROTOCOLS FOR ROTS

In speculation-based protocols [2] [3], ROTs carry out speculative executions and UTs are processed with 2PL. In semantics-based speculative locking protocols, the ROTs are classified as “compensatable ROTs (CROTs)” and “non-compensatable ROTs (NCROTs)” based on the notion called “compensatability”, which is defined in [6]. These protocols process ROTs without correctness and data currency issues.
2.1 Synchronous speculative locking protocol for ROTs

The processing of ROTs under synchronous speculative locking protocol for ROTs (SSLR) is illustrated in Figure 1. Here, T₁ and T₃ are UTs which are processed with 2PL and T₂ is an ROT which is processed with SSLR. T₁ obtains EW-lock on data object ‘x’. It reads ‘x₀’ and produces ‘x₁’ and converts the EW-lock on ‘x’ to SPW-lock. T₃, being a UT, waits till T₁ releases the lock on ‘x’. The ROT T₂ is processed as follows. Note that even though both T₁ and T₂ have arrived at the same instant, T₂ waits till T₁ produces after-image ‘x₁’. T₂ carries out two executions T₂₁ and T₂₂ by accessing ‘x₁’ and ‘x₀’ respectively. Note that, T₂₁ and T₂₂ are carried out synchronously. After T₂’s completion, T₂₁ is retained even though T₁ is not yet committed. We can observe that, T₂ is committed without waiting for the termination of T₁.

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

2.2 Asynchronous speculative locking protocol for ROTs

The processing of ROTs under asynchronous speculative locking protocol for ROTs (ASLR) is shown in Figure 2. Here T₂ accesses the before-image ‘x₀’ and other available values of data objects ‘y₀’ and ‘z₀’ and starts speculative execution T₂₁. Once the after-image ‘x₁’ 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, T₂₂ does not depend on T₁ and it is committed once it completes the execution without waiting for T₁. So, T₂₁ is aborted. Note that being UT, T₃ waits for T₁ for the release of the lock on ‘x’ as per 2PL rule.

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

2.3 Semantics-based synchronous speculative locking protocol for ROTs

Figure 3 depicts the processing under SSSLR. Here T₂ is a NCROT, T₃ is a CROT, T₁ and T₄ are UTs. Note that, T₂ waits for T₁ to produce after-image. Once the after-image is available, being a NCROT, T₂ proceeds with speculative executions. In the figure, T₃ conflicts with T₁ based on the data objects ‘y’ and ‘p’. Also T₃ conflicts with T₄ based on the data object ‘z’. Being a CROT, T₃ proceeds its execution without speculation and blocking. During commitment, T₃ 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 T₁ and T₄ are committed before T₃. In SSSLR protocol, the UTs conflicting with CROTs alone, are executed without blocking. By following this procedure, T₄ is executed without blocking even though it is conflicting with T₃ based on the data object ‘z’. Note that, this type of processing does not violate the serializability criteria.

![Figure 3: Depiction of transaction processing with SSSLR](image)

3. PROPOSED PROTOCOL

3.1 Overview of the protocol

In the proposed semantics-based asynchronous speculative locking protocol for ROTs (SASLR), the ROTs are classified into CROTs and NCROTs based on the notion called “compensatability”. In the proposed protocol, the NCROTs are executed with asynchronous speculation and CROTs are executed without speculation. However, CROTs have to perform compensating computations during their commitment by reading the concurrent updates of committed UTs. In the proposed protocol, UTs are processed with 2PL. However, UTs conflicting with CROTs alone, can be processed without blocking.

3.2 Types of locks

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 SASLR is shown in Figure 4. 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. The entry “asp_yes” (asynchronous speculation yes) specified for SPW-NR conflict 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. The entry “asp_no” (asynchronous speculation no) specified for EW-NR conflict indicates that the requesting NCROT continue its current speculative executions with the before-image. Once the after-image becomes available, further speculative executions can be started dynamically.

<table>
<thead>
<tr>
<th>Lock requested by $T_j$</th>
<th>Lock held by $T_i$</th>
<th>CR</th>
<th>NR</th>
<th>RU</th>
<th>EW</th>
<th>SPW</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>sm_yes</td>
<td>sm_no</td>
<td></td>
</tr>
<tr>
<td>NR</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>asp_no</td>
<td>asp_no</td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>EW</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Lock compatibility matrix for SASLR

Figure 5: Depiction of transaction processing with SASLR

### 3.3 Transaction processing in SASLR

Figure 5 depicts the processing under SASLR. Here $T_2$ is a NCROT, $T_3$ is a CROT, $T_1$ and $T_4$ are UTs. Note that, $T_2$ starts its execution by reading before-image of the data object ‘x’. Once the after-image of ‘x’ becomes available $T_2$ starts another speculative execution based on asynchronous speculation policy. In the figure, $T_3$ conflicts with $T_1$ based on the data object ‘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 ‘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 SASLR 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. PERFORMANCE RESULTS

We have developed a discrete event simulator based on a closed-queueing model [5]. The description of parameters with values is shown in Table 1. In the following experiments, we have reported the results by simulating environments in which 30% and 50% UTs are kept. Note that, the performance of 2PL, FCWR, SSLR and ASLR protocols is not affected because of change in the % of CROTs. This is because, these protocols consider both CROTs and NCROTs as simple ROTs.

![Figure 5: Depiction of transaction processing with SASLR](image)

![Figure 6: % of CROTs vs Throughput (30% UTs)](image)

Figure 6 shows how throughputs performance for 2PL, FCWR, SSLR, SSSLR and SASLR vary with % of CROTs. We can observe that, SSSLR protocol performs marginally better than SSLR in 30% UTs environment. Also, the figure shows that the performance of SASLR is better than ASLR protocol. In SSSLR and SASLR, UTs conflicting with CROTs are not blocked. However, UTs conflicting with UTs are blocked. In SASLR, CROTs and NCROTs are not blocked. So, the performance of SASLR is better than SSSSLR, SSLR and ASLR protocols. Similar trend can be observed in Figure 7.

![Figure 7: % of CROTs vs Throughput (50% UTs)](image)

Figure 8 shows how UT throughput performance of 2PL, FCWR, SSLR, ASLR, SSSLR and SASLR protocols vary with % of CROTs. It can be observed that, both SSSSR and SASLR protocol perform better than the remaining protocols in 30% UTs environment. Both SSSSR and SASLR protocols allow the UTs conflicting with CROTs to continue
Table 1: Simulation Parameters, Meaning and Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbSize</td>
<td>Number of objects in the database</td>
<td>1000</td>
</tr>
<tr>
<td>cpuTime</td>
<td>Time to carry out CPU request</td>
<td>5ms</td>
</tr>
<tr>
<td>ioTime</td>
<td>Time to carry out I/O request</td>
<td>10ms</td>
</tr>
<tr>
<td>rotMaxTranSize</td>
<td>Size of largest ROT transaction</td>
<td>20 objects</td>
</tr>
<tr>
<td>rotMinTranSize</td>
<td>Size of smallest ROT transaction</td>
<td>15 objects</td>
</tr>
<tr>
<td>utMaxTranSize</td>
<td>Size of largest UT transaction</td>
<td>15 objects</td>
</tr>
<tr>
<td>utMinTranSize</td>
<td>Size of smallest UT transaction</td>
<td>5 objects</td>
</tr>
<tr>
<td>noResUnits</td>
<td>Number of RUs (1 CPU, 2 I/O)</td>
<td>8</td>
</tr>
<tr>
<td>MPL</td>
<td>Multiprogramming Level</td>
<td>20</td>
</tr>
<tr>
<td>% of UTs</td>
<td>Percentage of UTs currently active</td>
<td>30% and 50%</td>
</tr>
<tr>
<td>% of CROTs</td>
<td>Percentage of CROTs currently active</td>
<td>Simulation variable</td>
</tr>
<tr>
<td>logOverhead</td>
<td>Time to search transaction log</td>
<td>5ms</td>
</tr>
</tbody>
</table>

Figure 8: % of CROTs vs UT throughput (30% UTs)

their executions without blocking. So, the UT throughput performance for SSSLR and SASLR protocols is higher than the remaining protocols. From the Figure 9, we can observe that UT throughput performance of SSSLR and SASLR is better than that of 30% UTs environment. As more CROTs are added into the system, more UTs complete their executions. So, the UT throughput performance for SSSLR and SASLR protocols has been further improved in 50% UTs environment.

Overall, the proposed SASLR protocol performs better than 2PL, FCWR, SSLR, SSSLR and ASLR protocols by considering centralized database environment.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed an improved protocol for ROTs by exploiting the notion of asynchronous speculation and "compensatability" property of ROTs. In this protocol, the waiting time of ROTs and UTs can be reduced due to processing of ROTs with asynchronous speculation. The simulation results show that the proposed approach improves the performance over 2PL, FCWR, SSLR, ASLR and SSSLR protocols. As a part of future work, we are planning to extend the proposed protocol to distributed database environment.

6. REFERENCES