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

Waqar Haque and Shah N. Mahmud
University of Northern British Columbia
Prince George, BC, Canada
{waqar.haque, mahmud}@unbc.ca

ABSTRACT

Transaction processing in a distributed real time database system is coordinated by a concurrency control protocol (CCP). In a real-time distributed database system, priority protocols work in conjunction with the CCP to prioritize transactions so that they do not miss their deadlines. The performance of priority protocols varies under load conditions. We have developed a dynamic protocol that monitors the system load and switches between priority assignment protocols at run time in order to maximize the performance of the overall system. The proposed protocol uses adaptive speculative locking as the concurrency control mechanism and outperforms the fixed priority protocols under all load conditions.

KEY WORDS
Real-time; Priority Assignment; Transaction Processing; Distributed Databases

1. Introduction

A database system provides a systematic and secure way to store information and answer queries in an organized manner. Access to a database system is controlled by transactions, which are combination of read and write operations. A transaction must follow the ACID properties of atomicity, consistency, isolation, and durability [1]. A real time database system (RTDBS) introduces another layer to ensure correctness in the form of temporal requirements by monitoring the number and effect of missed deadlines, and the average ‘lateness’ or ‘tardiness’ of late transactions [2]. A distributed database system (DDBS) contains one or more databases connected by a communication network. In such an environment, sharing of data and programs, coordination among transactions and sub-transactions, and load balancing becomes more challenging factors.

Concurrency control protocols (CCPs) coordinate concurrent access to data and are considered the core component of database systems. In an optimistic approach, operations are scheduled immediately; in a conservative or pessimistic approach, operations may be delayed [3]. Two-phase locking (2PL) is a popular concurrency control protocol in which locks can be held in either shared or exclusive mode. Exclusive locks can increase transaction’s execution time considerably due to blocking. In contrast, speculative locking (SL) protocol allows simultaneous access to data items and conflicts are resolved later [4]. The adaptive speculative locking (ASL) protocol extends the basic SL protocol by exploiting a variety of techniques: efficient memory management, hyper-threading, and transaction queue management [5] [6]. However, the ASL protocol uses a fixed priority assignment approach, where a given protocol is selected once and for all. This approach may not always produce optimum results in a real time system, especially where system conditions change frequently. Priority assignment protocols such as First Come First Serve (FCFS), Shortest Job First (SJF), Earliest Deadline First (EDF), and Minimum Slack First (MSF) perform differently under different load conditions. We have investigated the performance of the ASL protocol using these protocols under different system configurations. Results of the experiments indicated that a priority protocol that performs well under certain configuration, may perform poorly or moderately under other configurations. This allowed us to determine a set of load ranges in which different priority protocols perform superiorly. We propose an adaptive protocol, Adaptive Priority Assignment Protocol (APAP) which monitors the load condition at run time and then maps it to the load ranges determined above to decide which priority protocol should be used to assign the next priority while using ASL as the concurrency control protocol. With this approach, we observe significant improvement in the overall system performance.

2. Related Work

2.1 Concurrency Control Protocols

ASL inherits the concept of speculative locking of data from SL protocols. However, the concept of lending uncommitted data has also been used by other CCPs. Permits Reading of Modified Prepared-data for Timeliness (PROMPT) is a commit protocol based on firm deadlines [7]. It extends the concept of centralized 2PL high priority (2PL-HP) for distributed real time
environments by adding additional steps and interaction scenarios between the lenders and the borrowers. The studies presented in [7] demonstrate that PROMPT performed better than other protocols, especially in the low load condition. PROMPT’s performance, however, degraded under high load conditions due to excessive borrowing. Prompt-Early Prepare (PEP) is a one-phase (1PC) real-time commit protocol that integrates the early prepare (EP) protocol with the lending property of PROMPT [8]. PEP overlaps the prepare and commit phases into one phase and reduces the transaction execution time by removing the voting phase of the 2PC protocol. PEP is also optimized by a presumed commit mechanism, where the master sends the commit decision, but cohorts do not need to send ACKNOWLEDGEMENT messages to the master. PEP was shown to outperform PROMPT in all experiments except in the sequential high distributed transaction environment due to longer priority inversion periods [8]. Though PEP reduces message and logging overheads through the use of the 1PC protocol, it suffers from a higher number of transaction aborts, priority inversion and deadlocks. Adaptive Exclusive Primary (AEP) [3] is an adaptive concurrency control protocol which dynamically switches between an optimistic and a pessimistic CCP to improve data and resource contention issues [9]. However, the application of AEP in a real-time distributed database environment is unclear. Moreover, AEP cannot detect conflicts at a non-local database site. Speculative Locking (SL) protocols extend standard 2PL protocols to allow parallelism among conflicting transactions by allowing any transaction to borrow uncommitted data from the conflicting transactions [4]. The borrowing transaction then has access to two versions of data: one is the before image, which is the data before the conflicting transaction updates it, and the other is the after image, which is the updated data produced by the conflicting transaction. The borrowing transaction performs speculative operations on both images. If the conflicting transaction commits, the borrowing transaction retains the after image of the data, otherwise it keeps the before image. Therefore, transaction blocking time is low, making transaction processing faster. In SL, the number of parallel speculative transaction processing increases exponentially as the level of lending increases and this in turn causes a higher number of transaction aborts. To solve this issue, SL introduces some variants to restrict the number of speculative executions. All SL variants assume that memory is unlimited, which makes them inappropriate for many systems. Adaptive Speculative Locking (ASL) [5] [6] is based on the SL protocol and does not assume infinite system memory; instead, it uses a page-based virtual memory management mechanism. ASL also uses hyper-threading and transaction queue management to further enhance performance. ASL outperforms all versions of SL but its performance degrades in high load conditions.

2.2 Dynamic Switching Between Priority Protocols

Various adaptive approaches have been used to best capture the performance variations of different priority protocols under varying system conditions. In all techniques, a common practice is to switch between priority protocols of a system based on the load, to improve the overall performance of the system. Adaptive Earliest Deadline (AED) is an adaptive scheduling algorithm based on a RTDBS [10]. To improve the performance of EDF in an overloaded environment, AED utilizes an adaptive approach using a feedback control mechanism. AED divides the transactions into two groups: HIT and MISS. Transactions which have higher probability of meeting deadlines fall into the HIT group. On the other hand, transactions which are less likely to meet their deadline are categorized as the MISS group. Transactions in the HIT group follow EDF scheduling and transactions in the MISS group follow random priority (RP) mapping. AED performs like EDF during low load conditions and like RP during high load conditions [10], thus giving a better overall system performance. The performance of AED in a distributed environment has not been tested. Sectional Scheduling (SS) is an adaptive approach of task or transaction scheduling based on hard real-time systems [11]. SS measures the current load of the system and adequately adapts to changes in the system environment. An improved version of SS, robust sectional scheduling (RSS), is more effective in overload conditions as it tests tasks in order to reject one with the least value that is affecting the system load. SS is not implemented in a DRTDBS. Group-EDF (gEDF) is a scheduling algorithm designed for non-preemptive soft real-time systems. It uses both EDF and SJF to schedule a task or transaction in the system [12]. It creates groups of tasks based on deadlines which are close to each other. Groups are scheduled based on the EDF protocol; when the deadlines of all tasks in a group are earlier than the deadlines of all tasks in other groups, then the first group has higher priority. However, gEDF follows SJF to schedule individual tasks within a group. Though gEDF outperforms EDF, it does not guarantee fairness because, it has tendency to favour only small tasks. gEDF also does not consider distributed real-time systems where a transaction might have a number of sub-transactions, so grouping of tasks in different sites and coordinating between them can be difficult.

In summary, dynamically switching between priority protocols according to system load can optimize system performance, provided one can effectively quantize the system load. However, none of the existing solutions have been designed or tested for a distributed real-time database system.
3. The Simulator

The Distributed Real-Time Transaction Processing Simulator (DRTTPS) is a discrete event simulator that simulates a configurable distributed real-time database environment. It serves as a test bed for our performance study and fine tuning of pluggable concurrency control and other protocols. A detailed description of the components of DRTTPS can be found in [6] [13], but some key features are described in this section. In DRTTPS, events are executed sequentially from a queue using ticks as a unit to measure a discrete amount of time; the simulator is thus platform independent. The network configuration in a distributed system consists of one or more interconnected sites where each site has one or more interconnected nodes, and each node has one or more real-time databases. A network connection is characterized by the source, destination, bandwidth, latency, and external usage (percentage of the bandwidth occupied by external users). The node is the core component of DRTTPS which specifies the maximum number of active transactions and protocols for concurrency control, deadlock resolution, priority assignment and preemption. The node consists of several hardware and software components including resource managers, buffer, swap disk, and a workload generator. The data can be partitioned or replicated. The workload generator uses inter-arrival times to generate a statistically distributed load and also specifies each transaction’s characteristics such as the work size, update probability of pages accessed, and the slack time. In DRTTPS, the aborted transactions (preempted or deadlocked) are positioned in a priority queue for future execution.

The simulator provides a graphical user interface for setting up parameters, monitoring the simulations and viewing results. The SetupTool allows users to specify/create site structure, node structure, and network architecture via parameters. A particular setup, where all site components and parameters are encapsulated in a binary image file, can also be saved for later retrieval. The ReportTool provides a user interface to display statistics and visualization of results via graphs and charts. There is also provision to trace the simulation in order to identify any anomalies or other points of interest. The entire simulation can be stored as a single object for future reference.

DRTTPS provides a common pluggable framework for all components. The components are organized in a hierarchical structure which allows sharing of interface and inheritance of baseline methods and functionality. This also allows addition of new protocols without any major modifications to the code. The discrete event model is implemented by creating a global event queue which stores all the events and exists at the top of the hierarchical class structure. The events are executed one by one from the global event queue. DRTTPS uses Java reflection to create the event queue, which does not require message passing between classes. Java language was chosen for the development of the simulator for its versatility, efficiency, and platform portability.

4. The Proposed Protocol

A priority assignment protocol is an integral part of real-time transaction processing. The effect of this protocol on the overall performance is greatly impacted by data and resource contention resulting from factors such as arrival rate of transactions, disk size, cache size, total number of transactions, network topology, number of physical nodes in the distributed system, and the page update probability. The performance of a priority assignment protocol varies with different system environments; especially with different loads [11]. We propose an Adaptive Priority Assignment Protocol (APAP) which monitors the system load and dynamically selects the most suitable priority assignment protocol to optimize the performance of a distributed real-time database system. We calculated the load of the system using the following formula taken from [14]:

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} \quad \text{……….. (1)}$$

where $e_i$ and $p_i$ denote execution time and the available processing time of a transaction, $T_i$, respectively. The value of $e_i$ can be calculated by subtracting the start time of a transaction from the current time of the system. A combination of deadline and slack time was used to compute $p_i$. We ran the system with different load configurations until the number of completed transactions was maximized. After analyzing the loads, optimal ranges were determined for the tested priority protocols; the selected ranges are shown in Table 4.1.

<table>
<thead>
<tr>
<th>Priority Protocol</th>
<th>Load Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDF</td>
<td>0 to 1.25</td>
</tr>
<tr>
<td>FCFS</td>
<td>2.2 to 2.3</td>
</tr>
<tr>
<td>SJF</td>
<td>Rest of the time</td>
</tr>
</tbody>
</table>

The performance of MSF is relatively better under high load than in low load. However, in high load, SJF has a higher chance to complete a transaction than MSF because SJF minimizes the idle time of system during transaction execution. Therefore, we did not use MSF as a possible choice in APAP. As stated earlier, we use ASL as the underlying concurrency control protocol for our study.

In our model, APAP acts as a mediator between the system and the priority protocol engine (Figure 4.1). A sequence of events takes place whenever a new transaction arrives in the system. The APAP receives a load value from the system and then determines the most
suitable priority protocol. It then receives a list of transactions, obtains the value of each transaction from the priority protocol engine and passes those values to a comparator class. The transaction with the highest value is selected and returned to system for scheduling together with a priority queue for the remaining transactions. In order to minimize the switching operation overhead, APAP only tests load for switching upon arrival of new transaction and not when transactions are preempted and/or aborted and added to a priority queue.

Figure 4.1: Sequence of events to select transaction

5. Experiments and Results

DRTTPS was used to conduct a number of experiments in order to study our proposed protocol under varying system configurations representing various load scenarios. For each experiment, the performance of APAP is compared with the performance of static priority protocols: EDF, SJF, and FCFS. The key performance metric is percentage of transactions which complete on time (PTCT), that is, before their deadlines expire. In addition, we also observe the switching between various protocols and present the usage of specific protocol by the percentage of usage (POU) metric. We measure these metrics against inter-arrival time which is defined as the time interval between arrival of two consecutive transactions. Thus, a higher value of inter-arrival time implies a lower load.

5.1 Baseline Experiment

In a distributed real-time database system, a number of factors can impact the system load; these include system resources (cache size, number of disks and processors), inter-arrival time, slack time, execution pattern, and network topologies. For our baseline experiment we use the parameters as defined in Table 4.2. We used binary tree topology for network connections among sites, where a node might have at most two child nodes. We assume, bandwidth is unlimited and there is no network latency.

Table 4.2: Baseline Parameter Settings

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Network Topology</td>
<td>Binary Tree</td>
</tr>
<tr>
<td></td>
<td>Bandwidth</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Network Pipe Latency</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Number of nodes</td>
<td>7</td>
</tr>
<tr>
<td>Node</td>
<td>Active Transaction Count</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Disk Count per Node</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Max Pages held per Disk</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Disk Access Time</td>
<td>35 ticks</td>
</tr>
<tr>
<td></td>
<td>Cache Size</td>
<td>20 pages</td>
</tr>
<tr>
<td></td>
<td>Swap Disk Access Time</td>
<td>35 ticks</td>
</tr>
<tr>
<td></td>
<td>Transaction Process Time</td>
<td>15 ticks</td>
</tr>
<tr>
<td>Transaction</td>
<td>Pages per Transaction</td>
<td>4-12 pages</td>
</tr>
<tr>
<td>Generator</td>
<td>Slack Time</td>
<td>720-2160</td>
</tr>
<tr>
<td></td>
<td>Inter Arrival Time</td>
<td>5-55 ticks</td>
</tr>
<tr>
<td></td>
<td>Page Update Rate</td>
<td>100 percent</td>
</tr>
<tr>
<td></td>
<td>Transaction Count</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 4.2 illustrates the PTCT performance for the baseline experiment. For transaction inter-arrival time up to 30 ticks, SJF achieves significantly higher PTCT than EDF and FCFS, because in high load scenarios, EDF and FCFS protocols suffer from a high number of transaction aborts. MSF performs close to SJF, since a large number of transactions complete before the timing constraints expire. Under heavy load, EDF performs worse than the other protocols until an inter arrival time of 40 ticks. However, during low load, when the inter-arrival time is more than 40 ticks, the transactions have enough time to complete and EDF takes the lead, performing at 100% PTCT at 45 ticks. APAP performs better than all tested protocols by switching between the priority protocols when their performances are optimal. We do note here that APAP’s superior overall performance than the best protocol at a given load is attributed to the transition periods that effect system conditions.

Figure 4.2: PTCT for the Baseline Experiment
Figure 4.3 shows the percentage usage of priority protocols in APAP for the baseline experiment. When the inter-arrival time of the system is between 5 and 20 ticks, the system usually runs with SJF because of its superior performance during that time. SJF has the maximum POU up to the 20-tick inter arrival time. When the inter arrival time of the system increases, the load condition of the system decreases. Thus, APAP increases the usage of EDF, rather than SJF. The inter-arrival period between 20-30 ticks shows a transition period during which each of the two protocols is used approximately 50% of the time. The usage then becomes more distinct in favour of EDF due to decreased system load. Throughout the experiment, usage of FCFS is minimal ranging between 2.4-3.8%.

This experiment was repeated for transaction count of 200, while keeping all other parameters fixed. This resulted in increased data conflicts and more speculative executions leading to a large number of transaction rejections. Therefore, the performance of all priority protocols, including APAP, degraded. The overall trend, however, remains the same and APAP has the highest PTCT as before. For sake of space, these results are not presented here.

5.2 Varying Baseline Parameters

In this section, we present performance results by varying baseline parameters that affect the system load.

5.2.1 Changing page update rate

Page update rate is an important factor in a transaction processing system that indicates the percentage of operations which are write operations during a transaction’s lifetime. The write operations use exclusive locks on the data, thus blocking access to the locked data for a certain time. Consequently, a high or low page update rate affects the system load. We studied the performance of APAP for varying update rates and present two representative cases here, one with page update rate of 0 (read-only transactions) and the other with the update rate of 0.5 (half of the pages read by a transaction also require a write back).

As expected, the overall performance of the system is much superior when update rate is 0. Figure 4.4 shows EDF demonstrating highest PTCT at inter-arrival times of 5 and 10 ticks, because there is no data conflict. At an inter-arrival time of 20 ticks, EDF shows a sudden decline of performance because of resource contention. SJF exhibits higher PTCT than in the baseline experiment, but does not perform well in comparison with other protocols. APAP demonstrates slightly lower PTCT than EDF at inter-arrival time of 5 ticks, but then outperforms all priority protocols and achieves PTCT of 100% before EDF. Figure 4.5 shows that APAP runs mostly with SJF until an inter-arrival time of 15 ticks. During that period, the usage of SJF varied from 85.2% to 76.1% whereas the usage of EDF increased from 13.8% to 20.3%. The usage of FCFS remains low (between 1-5%). As the inter-arrival time increases, APAP uses EDF exclusively.

At a page update rate of 50 percent, EDF performs relatively poorly during the low inter-arrival times (Figure 4.6). As the inter-arrival time increases (low system load), EDF outperforms SJF, MSF, and FCFS. APAP exploits all the priority protocols and consistently performs better than the other protocols except for a brief period between the inter-arrival time range of 20-25 ticks when the performance of APAP is 3-4% lower than EDF.

The POU chart (Figure 4.7) indicates that during the low inter-arrival times, APAP runs SJF frequently until an inter-arrival time of 25 ticks. The POU of SJF is 83.5% at the 5-tick inter-arrival time. After that, the usage of SJF gradually decreases while usage of EDF increases. APAP uses EDF exclusively beyond inter-arrival time of 40 ticks.
5.2.2 Effect of cache size

ASL must find space available in cache or swap disk before it requests a page from the database. If there is not enough space, then the transaction needs to wait. Therefore, cache size affects the system load. Figure 4.8 shows that performance of all protocols improve as the cache size is increased to 50 pages. Compared with the baseline study, the PTCT of SJF increases linearly and that of MSF, improves by 4.5% on average. FCFS shows 20.8% jump at the 30-tick inter-arrival time. At low loads, EDF outperforms all other protocols. Both EDF and APAP show a 100% PTCT beyond an arrival rate of 40 ticks. The POU results are similar to those obtained for the 50 percent update rate experiment except that the crossover point is observed at 35 ticks.

5.2.3 Effect of Slack time

Slack time indirectly affects the system load because it defines the deadline of a transaction. A higher slack time implies that transactions have more time to complete and this allows fewer preemptions and resource contention. For this experiment, we increased the range of slack time to 720-3600 ticks. Consequently, the performance of all priority protocols increased (Figure 4.9) compared to the baseline experiment. EDF outperforms SJF at the 20-tick inter arrival time which is earlier than the baseline experiment. FCFS shows 11.4% and 14.5% higher PTCT than EDF during the inter-arrival times of 5 and 10 ticks. APAP outperforms all protocols, but we did observe an unexplained anomaly at the inter-arrival rate of 25 ticks. Because of the improved performance, the POUs of SJF and EDF in APAP at the inter-arrival time of 5 ticks are now closer, at 30% and 70%, respectively (Figure 4.10). The crossover point also moves earlier and the usage change is more gradual.

5.2.4 Effect of system disk space

The number of disks affects the data availability for a transaction during execution. By increasing the number of disks, a transaction has a high probability of getting the required data in the local disk, which reduces its blocking and execution times. Therefore, the number of disks affects the system load. Figure 4.11 shows that performance of all protocols improves significantly when the number of disks is doubled. However, during very high load (inter-arrival time between 5 and 10 ticks), SJF outperforms EDF, FCFS, and MSF. Beyond that point, EDF quickly starts exhibiting superior performance. FCFS and MSF always perform close to each other and
surpass SJF after 10 ticks. APAP outperforms all priority protocols under all load conditions except at an inter-arrival time of 20 ticks when EDF exhibits a slightly (4%) better value for PTCT. The POU of EDF and SJF at low inter-arrival times are at 37.67% and 56.5%, respectively (Figure 4.12). APAP gradually switches to using EDF more than SJF much sooner than in previous experiments. After 30 ticks, APAP only runs EDF. FCFS is used 5.9% of the time at high load, and gradually drops to under 1% as the inter-arrival rate increases.

6. Conclusion

A real-time database system uses priority protocols when a CCP coordinates transaction processing to order transactions. EDF is an optimal priority protocol for ordering transactions in a real-time database system. However, EDF’s performance degrades in high load conditions whereas other priority protocols (such as SJF) perform better under these conditions. We have designed a protocol that optimizes the overall performance of a distributed real-time database by dynamically changing from one priority protocol to another according to the system load conditions. This has been done by first finding the most optimal protocol for individual load scenarios and determining appropriate load ranges. Our proposed protocol (APAP) uses these load range values to dynamically switch between protocols thus providing the best results under all conditions. The ASL protocol is used as the underlying CCP for all of our experiments. We have presented the performance results obtained by varying cache size, and page update rates. We also observed the percentage of usage of the priority protocols (POU) in APAP. To summarize the results presented in this paper, we observe that

- APAP outperformed all priority protocols especially in high load conditions. However, in some low load conditions EDF completed 2% to 6% more transactions than APAP. We consider this to be an insignificant cost of achieving overall optimization of APAP.
- A change in the page update rates and number of disks greatly improved the performance of all priority protocols.
- An increase in the cache size beyond a certain value did not have a significant effect on the performance of any priority protocol.
- APAP uses mostly SJF when the system load is high. For an inter-arrival time of 5 ticks, POU of SJF varied from 90% to 50%.
- Under low load situations, the POU of EDF increased, and always reached 100%.
We also conducted experiments by increasing number of nodes and using different network topologies. Though some of those results are interesting, they are beyond the scope of this paper.

Acknowledgement

This research was funded by a Discovery Grant from NSERC Canada.

References


