
Hong-Ren Chen

SIMULATION 2008; 84; 263
DOI: 10.1177/0037549708096298

The online version of this article can be found at: http://sim.sagepub.com/cgi/content/abstract/84/6/263

Hong-Ren Chen
Department of Digital Content and Technology
National Taichung University
No. 140, Min-Shen Rd., Taichung 403
Taiwan
hrchen@mail.nutn.edu.tw

The use of service-oriented technologies has been used increasingly to speed up the application process and respond to changing business needs for many e-commerce systems. However, current e-commerce transaction services are poor at processing data access conflicts in a timely manner. The different timely data access for e-commerce applications should be considered with concurrency control mechanisms. In this paper, we study the concurrency control problem among interleaved web service transactions under differentiated real-time demands. We propose a new method called two-phase locking with fairness principles (2PL-FP), which resolves the concurrent data access for both real-time and non-real-time support operations. The design issue aims to meet the deadline requirements of real-time transactions and minimize the response time of non-real-time transactions in the web service and service-oriented computing environment. Simulation results show that our method can achieve a significant performance improvement compared to current approaches.

Keywords: Information service, electronic commerce, database system

1. Introduction

The web service and service-oriented computing are currently hot topics for many e-commerce applications as they dramatically speed up the application process and become more agile in responding to changing business needs. Web service-oriented technologies are important for building distributed applications, which are typically constructed from a set of services that are independently designed. The widespread adoption of web services offers advantages including interoperability, stability and implementation reuse. The structure of a typical web services application is shown in Figure 1 [1]. It consists of resources, application logic and a message-processing layer that deals with message exchanges. When a service-oriented message arrives at web services, the message processing changes it into something more tangible for applications to deal with. Such web service transactions include the execution of short-running transactions within an organization and long-running transactions across organizations [2].

Each activity in web service transactions may request access data executing by sub-transactions in real-time or non-real-time situations. For practical purposes, the activity is considered to consist of a nested web service transaction. An Internet purchasing example as illustrated in Figure 2 involves the steps: (1) selecting the product; (2) providing personal data that allows the credit card to be authorized; (3) checking the number of products in the inventory management services and (4) confirming a customer’s order and total payment [3, 4]. The activities of credit authorization and customer accounting must be executed timely, and the inventory management services are commonly encapsulated with several activities to execute. In the familiar example of stock market analysis and program trading, information on stock prices is gathered through multiple sources and piped through a series of filters for refinement. The information is then used by an expert system that spots trading opportunities. Some serious activities in web-based stock trading systems perhaps involve real-time supports. The purpose of activities...
for system management belongs to traditional web-based transactions [5].

In fact, the need to manage different styles of resource access conflicts and ensure the consistency of data in web services is based on concurrency control mechanisms. The services broker receives many requests for its information and it needs to be able to decide who wants what and whether they are granted access.

Locking mechanisms are the standard method of concurrency control for most web-based database applications. The idea behind locking is intuitively simple and effective as stated in many previous publications [6]. Each data object has a lock associated with it. Before a web service transaction can access a data object, the concurrency control must first examine the lock associated with that data object. If the lock is free, the lock is granted to the web service transaction and the data object can be accessed by the web service transaction. The two-phase locking with high priority (2PL-HP) used in most commercially available database systems gives good performance in processing real-time transactions on the Internet [4, 5, 7]. This approach ensures that a higher priority transaction is never blocked by a lower priority transaction, and the conditional restart avoids the starvation problem through the high priority technique. The problem of priority inversion is resolved by using priority inheritance [4].

In spite of many research efforts concerning the concurrency control problem, the findings often assume that the system may consist of only one single type of data that is required in the real-time access. We know that real-time or non-real-time activities may exist simultaneously in the nested web service transactions described in Figure 2. The impacts of different non-real-time features on these concurrency control mechanisms have been conspicuously neglected in previous studies in this topic.

To correct this omission, a new concurrency control mechanism must incorporate the access needs of real-time and non-real-time features. The design must minimize the number of missed real-time transactions and maximize the throughput of non-real-time transactions. A new concurrency control mechanism, 2PL-FP, is proposed to satisfy the concurrent data access of nested web service transactions supporting different degrees of real-time constraints. The essential concepts of 2PL-FP are: (1) giving priority data access to real-time web service transactions; (2) utilizing slack time to allow non-real-time web service transactions to access data; and (3) using conditional restart and priority inheritance to avoid the problem of starvation and priority inversion [4]. Simulations demonstrate that 2PL-FP delivers good performance when web service transactions require data access in different real-time supports in electronic commerce systems.

The remainder of this paper is organized as follows. Section 2 introduces the framework of web services-oriented electronic commerce systems. Section 3 describes the proposed concurrency control mechanism. Section 4 provides the simulation model and performance results. Finally, a conclusion is drawn in Section 5.

2. Web Service-oriented Electronic Commerce Systems

In web service-oriented electronic commerce systems, a communication network interconnects a number of sites as shown in Figure 3. They provide many services to support business-to-consumer applications, business-to-business applications and intra-company applications. Each site is typically composed of three types of servers: a web server, an electronic commerce server and a database server [7–9]. Web clients browse information and request accessing data supported by web server.

Electronic commerce server contains a transaction generator, pre-analysis manager, transaction manager, com-
Figure 2. Business process of internet purchasing [3]

Figure 3. High-level architecture of web service-oriented electronic commerce systems [4, 8, 10, 11]
Concurrency control mechanisms are designed to maintain database consistency despite concurrent execution of transactions [7]. The two-phase locking protocol has been a popular mechanism to solve the problem of concurrent accesses to shared data objects in most web-based database applications [6]. Some notable concurrency control protocols based on two-phase locking mechanisms are introduced and notation used is listed briefly.

3. Proposed Concurrency Control Mechanism

The real-time web services-oriented application is generally defined as a distributed database system where some transactions have deadlines on their completion times. Missing the deadlines can seriously affect the usefulness of completing the transactions. The goals are to satisfy transaction deadlines and to maintain database consistency. Concurrency control protocols strive to schedule data and to resolve data conflicts in such a way that transaction deadlines are taken into account [7]. The web services stack as shown in Figure 4. Here we focus on the problem of data access conflicts among interleaved web service transactions in real-time and non-real-time support operations resolved by concurrency control mechanisms [2].

In the literature on distributed database applications, we discovered that concurrency control mechanisms could be divided into three types. The first type is the locked-based approach where the data being accessed is locked until the transaction has been executed completely. This approach, however, requires a means of detecting and resolving deadlocks. Another approach is to prioritize data access based on the transaction timestamp. In other words, each transaction is assigned a timestamp once it enters the system. When there is an access conflict, this method compares the timestamp of each transaction to verify the serializability of data access. As concurrency control based on timestamps does not require the locking of data, this avoids the problem of deadlocks. However, locked-based concurrency control requires checking the data current lock status while timestamp concurrency control requires checking a transaction’s timestamp [15].

Too many checks affect the efficient execution of transactions. This led to the proposal of optimistic concurrency control where no checks are performed during the execution of the transaction. Only when the transaction is actually written to data is a check on access conflict made. This approach has three phases: read, validation and write. A transaction is initially permitted to read and write data in temporary storage. This is then validated to determine if it generates an access conflict with other transactions. If there are no access conflicts, the system moves into the write phase and puts the data on disk. If there is an access conflict, the transaction is restarted [16].

Concurrency control mechanisms are designed to maintain database consistency despite concurrent execution of transactions [7]. The two-phase locking protocol has been a popular mechanism to solve the problem of concurrent accesses to shared data objects in most web-based database applications [6]. Some notable concurrency control protocols based on two-phase locking mechanism are introduced and notation used is listed briefly.
TRANSACTION MANAGEMENT ISSUES IN WEB SERVICE-ORIENTED ELECTRONIC COMMERCE SYSTEMS

Figure 5. Pseudo-code of 2PL-HP [7, 9].

```
Algorithm 2PL-HP
Input: transaction Ti and Tj for accessing data object X, ∀Ti and Tj ∈ transaction set
Output: Ti or Tj accesses data object X
Begin
    If for all Tj holding a lock on data object X and P(Tj) > P(Ti) Then
        Restart each lock holder on data object X
    Else Ti blocked until release lock on data object X
End-if
End-begin
```

below. Ti denotes a (sub-) transaction in the system. S(Ti) represents the slack time of transaction Ti. The remaining execution time of transaction Ti is denoted Ert(Ti).

3.1 Conventional Two-Phase Locking (2PL)

In 2PL, the execution of a transaction consists of two phases: the grow phase and the shrink phase. In the grow phase, locks are acquired but may not be released. In the shrink phase, locks are released but new locks may not be acquired [7]. For transactions to be executed timely, priority inversion occurs when a transaction with high priority is blocked by a transaction with lower priority.

3.2 Two-Phase Locking with High Priority (2PL-HP)

2PL-HP proposes restarting the lower priority lock holder and letting the higher priority lock requester get the lock. This resolves the problem of priority inversion for transactions with real-time constraints. Figure 5 states the rule of 2PL-HP: if the priority of a lock-requesting transaction is higher than the lock-holding transaction, the lock-holding transaction is restarted. Otherwise, the lock-requesting transaction is blocked [9]. Although it eliminates priority inversion, 2PL-HP causes severe access conflicts of system resources among transactions with different real-time support operations, and does not use the nested structure of web services applications.

3.3 Two-Phase Locking with Fairness Principle (2PL-FP)

In 2PL-HP, the basic principle in resolving data conflicts between two web services transactions is to restart the lower-priority transaction [7]. In web services applications supporting real-time and non-real-time constraints, real-time transactions are always assigned higher priority than non-real-time transactions. Thus, when there is an access conflict between a real-time transaction and a non-real-time transaction in web services applications, the non-real-time transaction is likely to be restarted leading to a poor performance of non-real-time transactions.

To improve the total system performance, 2PL-FP is proposed using the fairness principle based on the high priority approach as shown in Figure 6.

2PL-FP assigns the unique lowest priority value to all non-real-time web service transactions as those transactions have no deadlines. When web service (sub-) transactions have access conflicts among different interleaved web service transaction families, the high priority technique is used: it blocks or aborts the lower priority of web service (sub-) transactions. The conditional restart procedure using adjustable slack time is incorporated in 2PL-FP to avoid the starvation problem of the high priority approach. This allows web service (sub-) transactions with a lower priority to access data first instead of aborting, if the slack time is long enough. Otherwise, lower priority web services (sub-) transactions must be aborted. In a nested structure of web services-oriented applications, (sub-) transactions execute on behalf of their root transaction. Thus, sub-transactions of the same web services family should not abort each other when access conflict occurs. When access conflict does occur within the members of a web service transaction family, the priority inheritance method is used to avoid the problem of priority inversion [4].

The adjustable value of $\beta$ is designed in response to the load of the practice system. $\beta$ satisfies deadlines for real-time web service transactions and maximizes the throughput of non-real-time web service transactions. Table 1 shows the nested structure of web service transactions. The different conflict resolution strategies adopted in 2PL-FP are listed within and across a web service transaction family. The symbol A represents blocking the lock-requester; B represents the technique of inheritance priority; and C represents using adjustable slack time to evaluate the opportunity of non-aborted (sub-) transactions.

4. Simulation Model and Performance Evaluation

The architecture of the simulation model as shown in Figure 7 is defined as an open queuing model that has a network of $n$ sites with a single external source and destination for transactions [17, 18]. Tsi indicates a transaction leaving the source for the queue in site i and Tid presents a
Algorithm 2PL-FP
Input: (sub)transaction $T_i$ and $T_j$ for accessing data object $X$, $\forall T_i$ and $T_j \in$ transaction set
Output: $T_i$ or $T_j$ accesses data object $X$
Begin
  Case 1: for all $T_j$ holding a lock on data object $X$ are same family members of $T_i$
  $T_i$ blocked until release lock on data object $X$
  If $P(T_j)>P(T_i)$ for transactions with real-time constraints Then
  $T_i$ inherits $P(T_i)$
  End-if
End-if
Case 2: for not all $T_j$ holding a lock on data object $X$ are same family members of $T_i$
  If $P(T_j)>P(T_i)$ with real-time constraints for non-family member Then
    If $\beta \cdot S(T_j) \geq \sum_{i=1}^{n} E_n(T_i)$ with real-time constraints then
      If $\beta \cdot S(T_j) \geq \sum_{i=1}^{n} E_n(T_i)$ with non-real-time constraints then
        $T_i$ blocked until release lock on data object $X$
        $T_j$ inherits $P(T_i)$
      Else
        $T_i$ blocked until release lock on data object $X$
        $T_j$ with real-time constraints inherits $P(T_j)$
        abort $T_j$ with non-real-time constraints
      End-if
    Else
    abort all $T_j$ except family members
  End-if
Else
  available family member $T_j$ is processed by case 1
End-if
End-if

Figure 6. Pseudo-code of 2PL-FP

<table>
<thead>
<tr>
<th>Lock-holder</th>
<th>Lock-requestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same web service transaction family</td>
<td>Different web service transaction family</td>
</tr>
<tr>
<td>Real-time</td>
<td>Non-real-time</td>
</tr>
<tr>
<td>A, B</td>
<td>A</td>
</tr>
<tr>
<td>Non-real-time</td>
<td></td>
</tr>
</tbody>
</table>

transaction leaving the queue in site $i$ for the destination. A transaction leaving the queue in site $i$ for the queue in site $j$ is $T_j$. All accesses of local transactions existing in the central subsystem keep circulating from one queue to the next and re-enter the system immediately. When a remote transaction issues the remote request, the central subsystem has external arrivals and departures. The simulation model at each site consists of four CPUs and two disks. This structure is similar to that which has been previously described in the literature [4, 7, 19–22].

4.1 Workload Model and Performance Metric

Most of the workload parameters have similar values to those used in previous studies [4, 16, 19–21, 23]. Table 2 lists the workload model parameters and their baseline values. The parameters page_cpu and page_io determine the CPU and disk time needed to access a data page, respectively. The parameter used to model the load of the system is arrival_rate, which specifies the mean rate of transaction arrivals and has a Poisson distribution. Sub_trans signifies the number of sub-transactions varying randomly in a (sub-) transaction tree. Tran_size represents the number of leaf sub-transactions in a nested transaction, which is the mean of a uniform distribution varying range between $0.5 \times$ tran_size and $1.5 \times$ tran_size. The parameter leaf_size determines the number of leaf sub-transactions in a nested transaction tree varying uniformly from $0.25 \times$ leaf_size and $1.75 \times$ leaf_size. The parameter level_size represents the depth of a nested transaction tree varying uniformly from $0.25 \times$ level_size to $2.5 \times$ level_size.
1.75 × level_size. The parameter of fault ratio is responsible for simulating the proportion of transaction faults.

The performance metric of MissRatio [4, 5, 7] was used:

\[ \text{MissRatio} = \frac{\text{number of transactions missing the deadline}}{\text{total number of submitted transactions}} \times 100\% . \]

The smaller the value of MissRatio, the better is the performance achieved. We compared the performance of the concurrency control protocols of the 2PL, 2PL-HP and 2PL-FP under various conditions, and also investigated the variety of response times for non-real-time transactions. For the transaction scheduling policy, we used the approach of earliest deadline first, adopted widely for most types of real-time database research [5, 19].

Figure 7. Architecture of the simulation model [4]
Table 2. Workload parameters and baseline values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_sites</td>
<td>number of sites in the system</td>
<td>4</td>
</tr>
<tr>
<td>num_proc</td>
<td>number of processors in the site</td>
<td>4</td>
</tr>
<tr>
<td>page_cpu</td>
<td>CPU time for accessing a data page</td>
<td>0.03 ms</td>
</tr>
<tr>
<td>page_io</td>
<td>disk time for accessing a data page</td>
<td>4.8 ms</td>
</tr>
<tr>
<td>arrival_rate</td>
<td>the rate of real-time transaction arrivals</td>
<td>40 trans/sec</td>
</tr>
<tr>
<td>restart_delay</td>
<td>delay time to restart a transaction</td>
<td>5 ms</td>
</tr>
<tr>
<td>remote_trans</td>
<td>the ratio of remote transactions in the system</td>
<td>0.3</td>
</tr>
<tr>
<td>min_slack</td>
<td>minimal slack factor</td>
<td>2</td>
</tr>
<tr>
<td>max_slack</td>
<td>maximal slack factor</td>
<td>8</td>
</tr>
<tr>
<td>sub_trans</td>
<td>number of sub-transactions in a (sub-) transaction</td>
<td>4</td>
</tr>
<tr>
<td>tran_size</td>
<td>number of leaf sub-transactions in a nested transaction tree</td>
<td>8</td>
</tr>
<tr>
<td>leaf_size</td>
<td>number of operations per real-time leaf sub-transaction</td>
<td>4</td>
</tr>
<tr>
<td>level_size</td>
<td>the depth of a nested transaction tree</td>
<td>4</td>
</tr>
<tr>
<td>remote_op</td>
<td>the ratio of remote operations for a remote transaction</td>
<td>0.5</td>
</tr>
<tr>
<td>db_size</td>
<td>number of pages in database</td>
<td>1600</td>
</tr>
<tr>
<td>write_prob</td>
<td>write probability for accessing a data page</td>
<td>0.5</td>
</tr>
<tr>
<td>transfer_rate</td>
<td>transfer rate of the network</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>commit_time</td>
<td>commit time for completing a decision phase</td>
<td>40 ms</td>
</tr>
<tr>
<td>comm_delay</td>
<td>communication delay between any two sites</td>
<td>dynamic</td>
</tr>
<tr>
<td>fault_ratio</td>
<td>the proportion of transaction faults</td>
<td>0.1</td>
</tr>
</tbody>
</table>

4.2 Basic Model

The settings for the basic parameters listed in Table 2 are based on previous studies [4, 9, 19, 20]. In this experiment, we varied the arrival rate from 20 real-time web service transactions/second (real-time trans/sec) to 120 real-time trans/sec in increasing steps of 20 in order to model different system loads. As shown in Figure 8a, the performance order based on the MissRatio metrics is 2PL-FP > 2PL-HP > 2PL.

From the figure, we see that the system misses more deadlines as the workload increases. This is consistent with our intuition: a heavier workload induces a longer queuing time, a higher probability of data conflicts and transaction blocking, and thus fewer transactions can meet their deadlines. 2PL performs the worst because the requesting transaction always blocks and waits for the data object to become free. This is the standard method for most database management systems which do not execute real-time transactions. 2PL causes some real-time demands to be delayed. The 2PL-FP algorithm gives different real-time demands access to data without delay, and avoids useless restarts. 2PL-FP uses slack time to execute the transactions with higher priorities completely instead of restarting them. 2PL-FP uses the techniques of conditional restart and priority inheritance so that access conflict will not cause the sub-transactions of the same web service transaction family to abort each other.

The 2PL-FP delivers better performance than 2PL-HP, which directly restarts lower priority transactions when they have data access conflicts with high-priority transactions. Since the restarted transaction would have much less slack time than its first incarnation, chances are that the restarted transaction will also miss its deadline. For the performance of non-real-time transactions as illustrated in Figure 8b, we found that 2PL-FP using the factor of adjustable slack time can maintain a faster response time than 2PL and 2PL-HP with a normal real-time transaction workload.

To study how data contention affects the performance, we plot the transaction rollback frequencies in Figure 8c. We see that 2PL-FP causes fewer restarts than 2PL-HP. This is because 2PL-FP keeps adjustable slack time to execute the lower priority transactions, and fewer transactions must be restarted. However, the restart problem only occurs in 2PL when a previously aborted transaction is restarted in the deadlock processing.

4.3 Real-time/Non-real-time Transaction Ratio

This experiment examines the impact of increasing the real-time/non-real-time web service transactions (Rt/nrt Ws-trans) from 100/0 to 0/100 in increasing steps of 20 for non-real-time web service transactions and decreasing steps of 20 for real-time web service transactions, with other parameters fixed as the basic model.
Figure 8. (a) MissRatio for basic model; (b) response time for basic model; and (c) rollback frequency for basic model
We observed performance changes in various concurrency control mechanisms for increasing the number of non-real-time web service transactions involved. The result for MissRatio can be seen in Figure 9 and the performance order is 2PL-FP > 2PL-HP > 2PL at various ratios. When the number of non-real-time web service transactions is increased, the access conflict at the same data object occurs frequently between real-time and non-real-time support operations. 2PL-FP performs best because the conditional restart procedure is running by utilizing the adjustable slack time. Hence, web service transactions with a lower priority can access data first instead of aborting. The response time will be reduced effectively for non-real-time web service transactions.

4.4 Read/Write Ratio

This experiment examines the impact of varying the read/write operations per nested web service transactions from 100/0 to 0/100 in increasing steps of 20 for write operations and decreasing steps of 20 for read operations, with other parameters fixed as the basic model. The result for the MissRatio is shown in Figure 10 and the performance order is 2PL-FP > 2PL-HP > 2PL, regardless of the ratios.

As the number of write operations increase, the access conflicts among interleaved nested web service transactions play the dominant role in the MissRatio. 2PL-FP considers the influence of adjustable slack time and conditional restart procedure. When the read/write ratio of a nested web service transaction is 100% (i.e. an extreme case of read-only), there are no access conflicts among nested web service transaction, and the system performance is mainly determined by transaction scheduling policies. Hence, the concurrency control mechanism has no influence.

5. Conclusion

It can be difficult to keep up with the rapid changes in technology. The advent of web services-oriented architectures intensifies competition, because these technologies are fundamentally changing the way we build our systems and how internal and external systems will interact. Various concurrency control mechanisms have been studied for different types of web service transactions. However, the techniques proposed for the concurrency control of real-time transactions may not be suitable to web services e-commerce applications due to the existence of non-real-time requests; their conflict resolution mechanisms may significantly affect the performance of non-real-time web service transactions.

In this paper, a new method called 2PL-FP based on high priority algorithms is proposed to provide concur-
rent data access for both real-time and non-real-time requests. The essential works of 2PL-FP realizes the fairness principle and includes the actions: (1) give prompt data access to real-time requests, (2) utilize slack time for non-real-time data access, and (3) use the techniques of conditional restart and priority inheritance to avoid the problems of starvation and priority inversion. Simulation results demonstrate the performance order from the best to the worst based on the metric of MissRatio is $2PL-FP > 2PL-HP > 2PL$ as workload increases. 2PL performs poorly because the requesting transaction always blocks and waits for the data object to become free. Therefore, 2PL increases the chances of real-time web services transactions missing its deadlines. 2PL-HP causes more restarts than 2PL-FP because 2PL-HP does not consider the possibility of utilizing slack time to execute restarted web service transactions. For the performance of non-real-time web service transactions, 2PL-FP can maintain quick response times at normal real-time web service transaction workload.

For an application programmer, we summarize the following brief guideline. To expect an 80% probability of completing an electronic commerce transaction on time in a web services-oriented computing environment, the programmer should code an electronic commerce transaction with the following limitations. When the system is under high load, the transaction size should be reduced to 6 operations or less, the transaction ratio should be less than 55% while the read/write ratio should be over 50%. When the system is under low load, the transaction size can be adjusted to 8–10 operations, the transaction ratio to between 75% and 65%, and the read/write ratio to below 50%. For a system administrator, the appropriate setting for cache size is 650 pages and the communication delay about twice the unit time of a page io. In such settings, 78–84% of the real-time transactions can meet their deadlines on time, and the response time of the non-real-time transactions is decreased to 1000 ms (or less) under the simulation results.

6. References


Hong-Ren Chen received M.Sc. and Ph.D. degrees in computer science in 1998 and 2002 at National Tsing Hua University in Hsinchu, Taiwan. He is currently an assistant professor with the Department of Digital Content and Technology at National Taichung University, Taiwan. His research interests include database management systems, real-time services processing and web services computing. He is a member of the IEEE Computer Society.