A Hybrid Concurrency Control Strategy for Mobile Database Systems

ABSTRACT

The inherent limitations of mobile database systems make concurrency control an important problem. Several pessimistic and optimistic strategies for achieving concurrency control in mobile database systems are presented in literature. The pessimistic strategies achieve concurrency however they face the problem of blocking, whereas the optimistic strategies may not always keep the data consistent. The conflict resolution strategies may result in starvation of some of the mobile requests. This paper presents a hybrid concurrency control strategy by using the good properties of both pessimistic and optimistic approaches. The results specify that the proposed strategy performs better as compared to both pessimistic and optimistic strategies and reduces the starvation of transaction requests of various mobile devices.

Keywords: Concurrency control, Mobile Database, Pessimistic strategy, Optimistic strategy, Transaction

1. INTRODUCTION

Concurrency control is one of the important building blocks of the transaction management. The goal of the transaction management is to ensure the correctness of data irrespective of the anomalies that arise due to concurrent request for the shared resources. The inherent features of mobile computing viz. disconnections and mobility makes the implementation of two phase locking protocol impractical. Several attempts to compromise ACID (Atomicity, Consistency, Isolation, Durability) properties are proposed in the literature. In general the pessimistic and optimistic approaches were proposed that are suitable to mobile database systems.

The pessimistic approaches use the traditional locking mechanism in mobile environments. To manage the disconnections and mobility, timeout strategies were implemented to increase the success rate of the transaction. However it results in increase in waiting time and rollback operations. The optimistic strategies avoid blocking giving more importance to the transactions which can be perform faster by mobile devices in offline mode. But it results in starvation of few requests which emerged from the wireless devices with less computing power and weak connectivity. The proposed hybrid concurrency control strategy combines the good features of both pessimistic and optimistic approaches. The simulation results show better throughput, less waiting time and a gradual decrease in number of starved transaction requests.

The access to shared data items by multiple mobile devices may be lead to concurrency access anomalies. The consistency of data can be maintained in presence of concurrent access using Pessimistic and Optimistic concurrency control strategies.

Several pessimistic and optimistic concurrency control strategies are proposed in literature. The state of art of these strategies is presented, their characteristics are compared and the current issues and challenges are highlighted [1].

The pessimistic strategies use locking protocols to overcome the problems of concurrent access. However these approaches are not being suitable for mobile environments. A transaction which is initiated at the
mobile host locks the required data items at the fixed host and may be disconnected. If another mobile device
needs the same fragment of data for its transaction execution, it has to wait till it is unlocked by the former.
The second mobile device in the queue has to wait for invariant period of time leading to starvation. Since
disconnections are normal characteristics in mobile environments, concurrency control has to be achieved in
presence of mobility and disconnections.

Andre et al [2] proposed four new isolation levels to ensure data consistency for read only transactions is
proposed. These levels are different from traditional isolation levels such as serializability. A Push mechanism
is used to disseminate information to large number of mobile hosts and point to point technology to satisfy on
demand requests. However this approach uses hybrid mechanism only for data delivery. Carmine et al [3]
proposed a modified two-phase locking protocol to handle frequent disconnections. In this strategy the
transaction execution takes place entirely on fixed host. However due to conflicting transactions, less
bandwidth and disconnections the abort rate increases. In [4] different concurrency control strategies are
used for different sets of relations. Based on their applicability, some relations use pessimistic strategies, few
use optimistic strategies and the remaining use no check concurrency control strategies. However this
approach cannot guarantee the hybrid nature of concurrency control on single relation. In [5] an attempt is
made to minimize the inconsistency problems during disconnections. Prior to disconnection a choice is made
between a token and optimistic method. However the local client practically implements either of the
concurrency control mechanisms. Further in token method, the high probability upon disconnection is given to
the site with high probability of occurrence of transaction. S.Cho et al [6] proposed a hybrid concurrency
control strategy. However in this strategy, for ready only operations optimistic strategy is adopted and for write
operations the conflict resolution is realized by the fixed host resulting in increase in aborted transactions. It
also results in increase in uplink bandwidth. In [7] a hybrid concurrency control is proposed which is free from
deadlocks. In this approach the transactions can be re-started utmost once otherwise they are aborted.
However in the scenarios where there is a weak connectivity, this protocol may result in increased aborted
transactions and also increase in uplink bandwidth. In [8] two protocols which realize hybrid strategy is
proposed. It uses dynamic adjustment of serialization order to resolve conflicts. However in presence of
disconnections and mobility, this protocol may not perform better as it has three stages of conflict resolution.
To overcome these anomalies, the good features of the pessimistic and optimistic strategies can be adopted
to implement effective concurrency control strategy as a hybrid approach. In [10], a comparison of several
pessimistic and optimistic strategies is presented.

The remaining part of this paper is organized as follows: Section-2 describes the mobile database
architecture and the execution modes of operation, section-3 describes the proposed strategy, section-4
specifies the results and comparisons and section-5 concludes the paper.

2. MOBILE DATABASE SYSTEM

A Mobile Database System (MDS) consists of two basic entities viz., a Mobile Host (MH) and a Fixed Host
(FH) respectively. The Mobile Hosts (MH) like cellular phones, PDAS, Laptops etc, is not always connected to
the fixed hosts. They may be intentionally disconnected for saving the power or may be disconnected
accidentally while moving from one cell to another. In mobile database systems, the disconnections are
realized as a normal scenario and are not treated as an anomaly. The desktops, servers etc., are fixed hosts
which are connected to fixed network. The fixed host has the capability of running large databases and at the
same time guarantees the efficient processing of the requests initiated by mobile hosts. The following figure
describes the conceptual architectural of a mobile database system.
The two basic elements of the MDS are Mobile Host and Fixed Host. The Mobile hosts are connected to the nearest base station by a wireless connectivity and the elements in the fixed network are connected in a wired mode.

The emergence of smart devices has lead to increase in the amount of storage space and thereby realizing the processing capability on mobile hosts. These characteristics made the offline transaction processing a reality.

The offline execution enables to execute the transactions in a soft disconnection state. Hence when the mobile host moves from one cell to another, the transaction may still continue its execution even when there is little or no connectivity. In the offline transaction processing, the fragment needed to execute the transaction is first copied onto the mobile host, then the mobile host may not be strongly connected to fixed host. The transaction is executed locally on the mobile hosts. Once the transaction successfully executes, the results are integrated with the fixed host.

### 2.1. Transaction Execution Modes

Transactions in mobile database systems are initiated at mobile hosts and are executed on mobile host or fixed host or the execution of the transaction is distributed among mobile and fixed host respectively. The nature of the execution depends on the type of transaction i.e. flat or distributed [1]. In all execution modes, the transactions are initiated at the mobile host. The execution of transaction is characterised by following models:

- **Execution on the Fixed Host**
  In this execution mode, the transactions are completely executed on fixed host.

- **Execution on a MH**
  In this mode, transactions are executed on mobile host. This model can be realized for smart phones. However reconciliation of the final state of transaction is needed with the fixed host.

- **Distributed Execution on a MH and the Wired Network**
  In this mode, execution of transaction is distributed among mobile host and fixed host respectively. A sub-transaction is executed at mobile host and another one at fixed host. This helps in minimizing the communication between the fixed host and mobile hosts respectively.

- **Distributed Execution among several MHs**
  In this mode, the transaction execution is distributed among several mobile hosts. A mobile host acts as a server for other mobile hosts so that the execution is distributed between them. The selection of a mobile host for execution of a transaction is location based. The participating mobile hosts are smart phones. This scenario realizes Mobile Adhoc Database Systems.

- **Distributed Execution among MHs and FHs**
  This mode provides a fully distributed environment where a transaction execution is distributed among several mobile hosts and fixed hosts respectively.
2.2. Mobile Deployment Model

Mobile Middleware is software that acts as an interface between the Mobile host and fixed host respectively. The Mobile Middleware ensures that the information is delivered to the proper device irrespective of the constraints.

The architectural elements of a mobile middleware are depicted in Figure 2. It is three-tier architecture and realizes cross platform characteristics. The mobile hosts works on different platforms and operating systems and the applications need to be accessed from any of the devices. Further it supports the heterogeneous databases concept because the transaction management is realized at mobile middleware.

![Image of Mobile Middleware and Fixed Host](image)

Figure 2. High-level architecture of Mobile Middleware

Mobile middleware communicates with the fixed host on behalf of the mobile hosts. It supports location transparency thereby reducing the overhead of hard coding the applications on mobile devices. The data synchronization and the conflict resolution strategies are realized at mobile middleware.

3. HYBRID CONCURRENcy CONTROL

The concurrency control can be achieved using pessimistic or optimistic strategies which guarantee isolation property of transaction management. The timeout based strategies proposed in literature to reduce the starvation issues still suffer from blocking issues and also affects the throughput of the system. To overcome the blocking problem the optimistic strategies were supposed to be suitable for mobile environments. However in the conflict resolution stage the device which returns the results first is reconciled with the fixed host at the earliest. In this scenario, the mobile host which requested for shared items first has to wait for invariant time. In the proposed strategy, the good features of both pessimism and optimism are adopted. This is possible by setting the priority to transactions. If the transaction is not executed, its priority is increased. Once the priority reaches the max_priority level it enters into the pessimistic mode, locks the transaction once the current transaction is executed and again moves to the optimistic mode. As long as max_priority is not reached the transactions are executed in optimistic mode. The value of max_priority depends on nature of transaction, which is maintained and recorded at mobile middleware.

3.1 Priority of Transactions

Initially the priority of all transactions requesting for same shared data items is set to zero. The max_priority of each type of transaction is maintained at mobile middleware. The priority can be increased during conflict resolution stage of the algorithm. This information of each type of transaction is maintained in the Transaction_Manager relation at mobile middleware. The structure of Transaction_Manager relation is as follows:
Table 1. Transaction_Manager relation

<table>
<thead>
<tr>
<th>Trans_name</th>
<th>Relation</th>
<th>Data_Items</th>
<th>Max_Priority</th>
<th>Timer</th>
</tr>
</thead>
</table>

Trans_name is the name of transaction or transaction id requested by a mobile host. To execute this transaction, the mobile host reads the data specified in “Data_Items” of “Relation”. Max_Priority specifies the maximum priority of the transaction which when reached, the data items are locked. To avoid indefinite locking of resources, a “Timer” value is used. This is the time period within which the mobile host is expected to return the results to fixed host before moving to the optimistic mode.

3.2 Concurrency Control Strategy

The Hybrid Concurrency control algorithm starts by setting the priorities of all transactions to zero. Whenever there is a request for shared data item, conflict is tolerated and the transaction execution starts locally on mobile hosts. The following figure describes the basic steps of the algorithm.

![Figure 3. Hybrid Concurrency Control](image)

When the transaction request is initiated by mobile host, the mobile middleware checks for the validity of transaction in the Transaction_Manager relation, then reads the data items needed for executing the transaction and copies the same to the requesting mobile host cache and the transaction starts executing locally. If some other mobile host request for same transaction, the conflicts is tolerated and the execution starts locally. The information pertaining to the transactions currently under execution is maintained by mobile middleware in Current_Trans relation.

Table 2. Current_Trans relation

<table>
<thead>
<tr>
<th>Mobile Host</th>
<th>Trans_name</th>
<th>Current_Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mobile Host specifies the wireless device which requested for transaction identified by trans_name and Current_Priority specifies the priority of the transaction which is initially set to zero.

The Conflict resolution strategy specified in figure 4 is realized as follows:

When a transaction is successfully executed locally, the results of transactions are returned to the mobile middleware. The mobile middleware updates the results in the actual database at the fixed host. The mobile middleware then checks the Current_Trans relation to check whether some other wireless device is using the shared data items for which the result are returned. If there exists other mobile host which is using the same shared data items, the current priority of that mobile host is increased.

If the current_priority has exceeded the max_priority of the transaction, then the transaction starts executing in pessimistic mode by locking the data items for particular period of time specified by “timer” attribute in Transaction_Manager relation. Once the transaction completes its execution successfully, it returns to the optimistic mode of operation. The lower priority transaction may be suspended for a period of time. However this doesn’t reflect the uplink bandwidth because the new values of shared data items are sent to the mobile host after completing the current transaction.

If the current_priority is well below the max_priority the new values of the data items are multi-casted to the mobile hosts requesting for shared data items and the transaction starts locally and the execution proceeds in an optimistic mode of operation.

![Conflict Resolution Diagram](image_url)

Figure 4. Conflict Resolution

The pessimistic mode of execution of the transaction specified in figure 5 is realized as follows:

When the max_priority is reached for a particular transaction, the data items specified in Transaction_Manager relation are locked and the data items are copied onto the mobile host and the execution starts locally.

The results of transaction executed locally are to be reconciled with the fixed host before the expiry of timer. If the transaction successfully completes its execution before the expiry of timer, the results are updated onto the fixed host and the data items are unlocked and the execution proceeds in optimistic mode. If the request of same transaction by another mobile host is pending then the new values of data items is sent to the mobile hosts and transaction execution proceeds as usual.
If the transaction is not executed successfully within the specified time period, the transaction may be aborted. However in order to guarantee the execution of the transaction, the dynamic timer adjustment strategy [9] may be adopted. In this approach, the timer value is increased by a step factor and continues the execution. However if the transaction is not successfully executed within the threshold limit, it might be aborted.

![Diagram](image)

**Figure 5. Pessimistic mode of Execution**

4. RESULTS

The proposed Hybrid concurrency control strategy is simulated for M-Banking application. The domain specific relations are maintained at the fixed host. The transaction management relations are maintained at mobile middleware.

The relations that are maintained at the mobile middleware are Transaction_Manager and Current_Trans relation. The Transaction_Manager relation describes the list of banking transactions, the data items needed to execute the particular transaction, name of relation from where the data items are to be read, the max_priority level of each transaction and the timer value to be used for pessimistic strategy. The max_priority depends on the criticality of the transaction. Current_Trans relation describes the list of transactions which are active and are in waiting state.
Table 3. Transaction_Manager relation for M-Banking

<table>
<thead>
<tr>
<th>Trans_Name</th>
<th>Relation</th>
<th>Data Item(s)</th>
<th>Max_Priority</th>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Account</td>
<td>Amount</td>
<td>3</td>
<td>3 sec</td>
</tr>
<tr>
<td>T2</td>
<td>Account</td>
<td>Amount</td>
<td>4</td>
<td>4 sec</td>
</tr>
<tr>
<td>T3</td>
<td>Account</td>
<td>Accountno, Amount</td>
<td>4</td>
<td>5 sec</td>
</tr>
</tbody>
</table>

Table 4 lists the transaction requests by mobile hosts in order of arrival. Two mobile hosts may execute the same transaction Tᵢ if their account numbers are different. A row locking mechanism is simulated to implement the M-Banking scenario.

<table>
<thead>
<tr>
<th>SiteId</th>
<th>Transaction request</th>
<th>Accountno</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>T1</td>
<td>101</td>
</tr>
<tr>
<td>M2</td>
<td>T2</td>
<td>102</td>
</tr>
<tr>
<td>M3</td>
<td>T1</td>
<td>103</td>
</tr>
<tr>
<td>M4</td>
<td>T1</td>
<td>101</td>
</tr>
<tr>
<td>M5</td>
<td>T1</td>
<td>101</td>
</tr>
</tbody>
</table>

Transactions initiated by mobile host M1, M2 and M3 can be executed in parallel. Though M1 and M3 have requested for same transaction request but their account numbers are different. However M4 has to wait till the completion of transaction by M1. Similarly M5 has to wait for the completion of transaction M1 & M4. The simulation of proposed strategy is depicted in the Table 5 (current_trans relation). If the request for same shared resource arrives, the priority of the transaction request will be increased by one. Initially all priorities are assumed to be zero.

Table 5. Current_Trans (M-Banking)

<table>
<thead>
<tr>
<th>Mobile Host</th>
<th>Trans_Name</th>
<th>Current_Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>T1</td>
<td>2</td>
</tr>
<tr>
<td>M2</td>
<td>T2</td>
<td>0</td>
</tr>
<tr>
<td>M3</td>
<td>T1</td>
<td>0</td>
</tr>
<tr>
<td>M4</td>
<td>T1</td>
<td>1</td>
</tr>
<tr>
<td>M5</td>
<td>T1</td>
<td>0</td>
</tr>
</tbody>
</table>

The transactions starts locally executing at all mobile hosts by tolerating conflicts to occur. M1, M4 & M5 are using the same shared data items. If M5 completes the execution first, it updates the values of data items on the fixed host and multicast new shared values to M1 and M4 respectively. But before that the priority of is increased. The M1 priority is set to 3 and M4’s priority is set to 2. Since M1 has reached the Max_Priority, it is locked and the transaction at M4 is suspended for some period of time. Once the transaction at M1 successfully executes the new values are sent to M4 for transaction to restart locally in optimistic mode.

5. CONCLUSION

The proposed Hybrid concurrency control approach combines the good characteristics of both pessimistic and optimistic concurrency control mechanisms. It reduces starvation of transactions and also maintains the
data consistency. The proposed strategy works in offline mode of transaction processing, thereby mobility and disconnections are handled without aborting the transactions. This increases the throughput of the system. There is a considerable decrease in re-submitting the transaction requests from mobile hosts. Hence the uplink bandwidth is considerably reduced.

REFERENCES


