Impact of Mobility on Concurrent Transactions
Mixture
Ahmad Alqerem
Zarka University, Jordan
ahmad_qerm@zu.edu.jo

Abstract
This paper presents a simulation analysis of the impact of mobility on concurrent transaction processing over a mixture of mobile and fixed transactions. Our results show that the traditional concurrency protocol yield low throughput in mixture mobile and fixed environments due to the fact that traditional protocol cannot discriminate implications of restart of both types of transactions. As a result, the number of restarts can be unbounded and the ongoing mobile transactions that share the same wireless resources will be affected. We investigate adaptive adjustment the serialization order of transactions such that a conflict resolution should be give more rooms for mobile transaction. We show that by effectively capitalizing this information, the number of unnecessary restarts can be highly reduced. Based on the simulation analysis, we show that this protocol provide a significant performance gain in mixed transaction environments.

Keywords: concurrency control, mixed transaction, mobile computing, performance evaluation.

1. Introduction
Concurrency control protocols proposed in the previous works [2], [3], [4] and [5] mainly focused on homogeneous environments. That is, there is only one type of transactions present in the system, either fixed transactions or mobile transactions. However, it is not uncommon to have mobile applications that consist of both types of transactions. Due to the high expensive resources used by mobile transactions, it will be a serious problem for mobile transactions to suffer from restarts caused by other fixed or mobile transactions. As a result, the conflict resolution should be in favour of mobile transaction. In MDBS, transaction restart has two negative implications. First, since the number of restarts can be unbounded, resources and time spent on the restarted transactions will be wasted. Second, reserving extra resources and time to process restarted mobile transactions may seriously affect other ongoing mobile transactions that share the same wireless resources. In view of these two factors, it is very important to reduce the number of transaction restarts which negatively affect amount of resources wasted in this aspect, save the expensive resources of the wireless environment and improves the overall system performance. In this paper, to reduce the number of restarts, a new optimistic based concurrency control protocol, called OCC-Mix, is proposed. By deferred adjustment the serialization order of transactions, the OCC-Mix protocol can successfully resolve many conflicts between the validating transaction and the executing transactions. Consequently, the number of unnecessary restarts can be highly reduced. The characteristics of these protocols have been examined in detail by simulation. The results show that the performance of these protocols is consistently better than the pure OCC protocol over a wide range of system settings. In particular, these protocols provide a more significant performance gain in mixed transaction environments. The remainder of this paper is organized as follows: Section 2 describe the mixed system model. Section 3 discusses the differences between fixed timestamp and timestamp interval. The OCC-Mix approach is described in Section 4. In Section 5, we discuss the performance results and we conclude the study in Section 6.

2. Mixed System model
This section is to define a mixed system model Figure 1; it is assumed that the system consists of stationary and mobile part. Stationary units are classified as either fixed hosts, base stations BS or mobile support station MSS. Fixed host are database server connected to the existing wired network, this server is accessible by two types of users, users whom used the wired reliable network and the second type of users are not capable to connect directly to these information servers. A MSS is equipped with a wireless interface and work as a coordinator and communication interface between the mobile unit and the database.
server through BSC. Transactions involved in this system consist of a sequence of read and writes operations, and ends with a commit or an abort operation. This transaction considers an atomic process. That is, translate a database from a consistent state into another consistent state. There are two types of transactions in this environment: fixed or wired (FT) transactions and mobile transactions. A fixed transaction is submitted directly to a database on the same host while the mobile transaction submitted by mobile device. Like in [3], an (MT) is a mobile transaction which issued by a mobile host. The participation of an MH introduces dimensions inherent to mobility such as: movement, disconnections and variations on the quality of communication.

Figure 1: Mixed System Environment

3. Time interval versus fixed timestamp
In a fixed timestamp method timestamps are chosen for transactions when they begin. Whenever a transaction makes a request that would create a conflict between itself and another transaction, the timestamps of the two transactions are compared. If the order of the timestamps is the same as the serialization order required by the conflict, the request is allowed; otherwise the requesting transaction is aborted and restarted with a new timestamp. Thus, the transaction serialization order is essentially fixed in advance, which has the potential to cause many unnecessary aborts. Using the Time Intervals method [7] each transaction has two timestamps. These timestamps can be thought of as the upper and lower bounds of an interval of timestamp time in which the transaction must appear in the serialization order. Time intervals are partially ordered, with the relations ‘<’ and ‘>’ applying only to intervals that are disjoint (note that non-disjoint intervals can always be truncated in such a way as to impose either ordering on them). Every transaction’s initial interval spans the entire allowable timestamp range, representing the fact that there is no restriction on its place in the serialization order until it encounters conflicts with other transactions. When a conflict is encountered, the time intervals of the transactions involved are compared. If the intervals are disjoint, then their relative ordering has already been established; in this situation the algorithm is exactly the same as for the tied timestamp case. On the other hand, if the intervals overlap, then they can certainly be truncated so as to effect the desired ordering, after which the request can be granted. In the limiting case, an interval may be shrunk down to a single point, which is then no different in its interpretation than a fixed timestamp.

4. Approach details
Our OCC-Mix is an optimistic approach based on the dynamic adjustment of serialization order. Like OCC-TI [7], OCC-Mix uses the notion of timestamp intervals to record and represent serialization orders induced by concurrency dynamics. Timestamps are associated with both transactions and data items. Each data item has a read and a write timestamps, where the read and the write timestamps are the timestamps of last committed transactions that have read or written to the data item, respectively. For each
transaction, OCC-Mix associates with each active transaction a timestamp interval expressed as a [lower bound (lb), upper bound (ub)] pair. The timestamp interval denotes the validity interval of a transaction. The timestamp intervals are also used to denote serialization order between transactions. For example, if \( T_i \) (with timestamp interval \([lb_i, ub_i]\)) is serialized before \( T_j \) (with timestamp interval \([lb_j, ub_j]\)), denoted \( T_i \rightarrow T_j \), then the following relation must hold: \( ub_i < lb_j \).

### 4.1 Adjustment of timestamp interval

Each transaction at the start of execution is assigned a timestamp interval of \([0, \infty]\), i.e., the entire timestamp space. As the transaction proceeds through its lifetime in the system, its timestamp interval is adjusted to reflect serialization dependencies as they are induced. Serialization dependencies may need to be reflected while transaction is accessing data items in its read phase or by being in the conflict set of a different validating transaction.

#### 4.1.1 Adjustment at the read phase

In this case, the timestamp interval is adjusted with regard to the read and write timestamps of the data item read or updated. In the process of adjusting, the timestamp interval may ‘shut out’, i.e., become empty. In that case, the transaction cannot be successfully serialized and needs to be restarted. Note that this is one of the major differences between conventional protocols and protocols based on dynamic adjustment of serialization order. In conventional OCC algorithms, restarts can only occur at validation times. In our case, however, transactions can restart at other times if a timestamp interval shut out is detected. The exact mechanics for these adjustments are shown in the procedures given in Figure 2. In the remainder sections, we use the notation \( TI(T_i) \) to denote the timestamp interval of transaction \( T_i \) and RTS \((D_i)\) and WTS \((D_i)\) to denote the read and write stamps of data item \( D_i \), respectively. As a transaction successfully validates, a final timestamp is assigned to it.

#### 4.1.2 Adjustment at the validation phase

In the case of being in the conflict set of a different validating transaction, the timestamp interval of the active transaction is modified to dynamically adjust the serialization order. The adjustment of the serialization order for both mobile and fixed transactions implemented with timestamp intervals creates a partial order between transactions based on conflicts and transaction type. Suppose we have a validating transaction \( T_v \) and an active transaction \( T_a \). Let \( TS(T_v) \) be the final timestamp of the validating transaction \( T_v \) and \( TI(T_v) \) the timestamp interval of the active transaction \( T_v \). Let \( TI(T_a) \) be the timestamp interval of the validating transaction and \( TI(T_a) \) be the transaction type where \( (T_a) \in \{\text{mobile, fixed}\} \). We assume here that there is no blind write. So, there are two possible types of conflicts which are resolved using adjustment of serialization order between \( T_v \) and \( T_a \): (1) read-write conflict which occur when the RS \((T_v) \cap WS(T_a) \neq \emptyset \) which can be resolved by forward adjustment (2) write-read conflict which occur when the RS \((T_v) \cap WS(T_a) \neq \emptyset \) and resolved by backward adjustment.

![Figure 2: Adjustment of TI \((T_a)\) at the read phase](image1)

![Figure 3: Iterate Readset/Writeset of Validating Transaction](image2)
The adjustment of timestamp intervals (TI) in Figure 3 iterates through the read set (RS) and write set (WS) of the validating transaction (Tv). First we check that the validating transaction has read from committed transactions. This is done by checking data item’s read timestamp (RTS) and write timestamp (WST). These values are fetched when the read/write operation to the current data item is made. Then the algorithm iterates the set of active conflicting transactions. When access has been made to the same objects both in the validating transaction and in the active transaction, the temporal time interval of the active transaction is adjusted. Non-serializable execution is detected when the timestamp interval of an active transaction becomes empty. If the timestamp interval is empty the transaction is restarted.

4.1.2.1 Forward adjustment
A read-write conflict between Tv and Ta can be resolved by adjusting the timestamp interval of the active transaction forward (i.e. Tv → Ta). If the validating transaction is a mobile transaction and the active transaction is a fixed host transaction, forward adjustment is correct. If the validating transaction is a fixed host transaction and has a conflict with a mobile transaction, we should favour the mobile transaction. This is achieved by reducing the timestamp interval of the validating fixed host transaction and selecting a new final timestamp earlier in the timestamp interval see section 4. Normally, the current time or the maximum value from the timestamp interval is selected but now a different value is selected based on a predefined σ-value. As the σ-value increases, the opportunity for the active mobile transaction to commit is increased on account of the fixed host transaction. When the σ-value = 2 this means that the validating fixed host transaction reduces by half of its interval to the advantage of active mobile transactions. Forward adjustment can be described by procedure in Figure 6.

4.1.2.2 Backward adjustment
A read-write conflict between Tv and Ta can be resolved by adjusting the timestamp interval of the active transaction backward, (i.e. Ta → Tv). If the validating is a mobile transaction and the active conflicting is a fixed transaction, backward adjustment is correct. If the validating is a fixed transaction and the active conflicting transaction is mobile, then backward adjustment is done if the active transaction is not aborted in backward adjustment. Otherwise, the validating transaction is restarted. This is wasted execution, but it is required to ensure the execution of the mobile transaction. In backward adjustment, we cannot move the validating transaction to the future to obtain more space for the mobile transaction. We can only check if the timestamp interval of the mobile transaction would become empty. In forward ordering we can move the final timestamp backward if there is space in the timestamp interval of the validating transaction. Again we check if the timestamp interval of the mobile transaction would shut out. We have chosen to abort the validating transaction when the timestamp interval of the mobile transaction shuts out. Thus, this protocol favours the mobile transaction that uses scarce and expensive resources. Backward adjustment can be described by procedure in Figure 5.

---

**Forward Adjustment (Tv, Ta)**

Begin

If Tt_type = Fixed Then
   If Tm_type = Mobile Then
      \[ TS(Tv) = \max(TS(Tv)) + \left\lceil \frac{TS(Tv) - \max(TS(Tv))}{2} \right\rceil \]
   Else
      If TI(Tv) > \max(TI(Tv)) Then
         Restart(Tv)
      Else
         TI(Tv) = TI(Tv) ∩ [TS(Tv), ∞)
      End
   Else
      TI(Tv) = TI(Tv) ∩ [TS(Tv), ∞)
   End

Else

If Tt_type = Mobile Then
   TI(Tv) = TI(Tv) ∩ [TS(Tv), ∞)
Else
   TI(Tv) = ∅ Then Restart(Tv)
End

---

**Backward Adjustment (Tv, Ta)**

Begin

If Tt_type = Fixed Then
   If Tm_type = Mobile Then
      If TS(Tv) - 1 < Min(TI(Tv)) Then
         Restart(Tv)
      Else
         TI(Tv) = TI(Tv) ∩ [0, TS(Tv) - 1]
      End
   Else
      TI(Tv) = TI(Tv) ∩ [0, TS(Tv) - 1]
   End

Else

If Tm_type = Mobile Then
   TI(Tv) = TI(Tv) ∩ [0, TS(Tv) - 1]
Else
   TI(Tv) = ∅ Then Restart(Tv)
End

---

Figure 4: Forward adjustment
Figure 5: Backward Adjustment
4.2 Final timestamp selection

In the OCC4Mix protocol we should select the final (commit) timestamp $T_v$ in such a way that room is left for backward adjustment. In our validation algorithm Figure 6 we set $T_v$ as the validation time if it belongs to the time interval of $T_v$ or the maximum value from the time interval otherwise. To justify our choice consider the following example:

**Example 4:** Let $RTS(x)$ and $WTS(x)$ are initialized as 100. Consider transactions $T_1$, $T_2$ and history $H$, where $T_1.type = T_2.type = mobile$:

$T_1: R_1(x) W_1(x) v_1 c_1$
$T_2: R_2(x) v_2 c_2$

$H = R_1(x) R_2(x) W_1(x) v_1$.

Consider the two cases (a) and (b):

**Case a:** $T_v = \max (TI(T_v))$

In a similar way, transactions $T_1$ and $T_2$ are forward adjusted to $[100, \infty)$. Transaction $T_1$ starts the validation at time 1000, and the final (commit) timestamp is selected to be $T_v = validation_time = 1000$. Because we have one write-read conflict between the validating transaction $T_1$ and the active transaction $T_2$, the timestamp interval of the active transaction must be adjusted: $TI(T_2) = [100, \infty) \cap [0,999] = [100,999]$. Thus the timestamp interval is not empty, and we have avoided unnecessary restart. Both transactions commit successfully. History $H$ is acyclic, that is, serializable. Therefore the selection $T_v = \max (TI(T_v))$ avoids the unnecessary restart problem.

**Case b:** $T_v = \min (TI(T_v))$

Transaction $T_1$ executes $R_1(x)$ which causes the timestamp interval of the transaction to be forward adjusted to $[100, \infty)$. Then $T_2$ executes a read operation on the same object, which causes the timestamp interval of the transaction to be forward adjusted similarly, e.g. to $[100, \infty)$. $T_1$ then executes $W_1(x)$ which causes the timestamp interval of the transaction to be forward adjusted to $[100, \infty)$. $T_1$ starts the validation, and the final (commit) timestamp is selected to be $T_v = \min ([100, \infty)) = 100$. Because we have one write-read conflict between the validating transaction $T_1$ and the active transaction $T_2$, the timestamp interval of the active transaction must be adjusted: Thus $TI(T_2) = [100, \infty) \cap [0,99] = \emptyset$. The timestamp interval is shut out, and must be restarted. However this restart is unnecessary, because history $H$ is acyclic, that is, serializable. Taking the minimum as the commit timestamp ($T_v$) was not a good choice here.

We have also used a deferred dynamic adjustment of serialization order. In the deferred dynamic adjustment of serialization order all adjustments of timestamp interval are done to temporal variables (i.e. the timestamp intervals of all conflicting active transactions are adjusted after the validating transaction is guaranteed to commit). If a validating transaction is aborted no adjustments are done. Adjustment of the conflicting transaction would be unnecessary since no conflict is present in the history after abortion of the validating transaction. Unnecessary adjustments may later cause unnecessary restarts. Finally, in Figure 7 current read timestamps and write timestamps of accessed data items are updated and changes to the database are committed.

---

**Figure 6: Select commit timestamp**

```
Select final timestamp ($T_v$)
Begin
  If (validation time) $\in TI(T_v)$ Then
    TS ($T_v$) = validation time;
  Else
    TS ($T_v$) = $\max (TI(T_v))$.
End
```

**Figure 7: Update Data Item Timestamps**

```
Update Data Item Timestamps ($RS(T_v), WS(T_v))$
Begin
  For all $D_i \in (RS(T_v) \cup WS(T_v))$
    Begin
      If ($D_i \in RS(T_v)$) Then
        RTS ($D_i$) = $\max (RTS(D_i)), TS(T_v))$;
      If ($D_i \in WS(T_v)$) Then
        WTS ($D_i$) = $\max (WTS(D_i)), TS(T_v))$;
    End
  End
End
```

Figure 6: Select commit timestamp

Figure 7: Update Data Item Timestamps
A backward and forward with deferred adjustment algorithm previously described, creates order between conflicting transaction timestamp intervals. A final (commit) timestamp is selected from the remaining timestamp interval of the validating transaction. Therefore the final timestamps of the transactions create partial order between transactions.

5. Experiments and Results

We have performed three sets of experiments. In the first set, mobile transactions percentage is 20%. For the second and the third sets, the utilizations of the mobile transactions are approximately 50 percent and 80 percent, respectively. We can therefore observe whether the domination of any one kind of transaction will have any impact on the performance of the implemented approaches. All these experiments show that the OCC-Mix approach outperforms both the pure OCC protocol and the 2PL protocol for a wide range of parameter settings.

5.1 Impact of Mobility

Figure 9 gives the PCR as a function of the mean mobility value when 20% of transactions present in the system are mobile transaction. Since the number of base station crossed over by a mobile transaction increase when the mobility increase, the delay between mobile transaction operations increase And the average numbers of active transactions increase, the blocking of 2PL protocol badly affect the PCR. For the OCC protocol, there is no blocking overhead. But the PCR reduction problem still exist because of mobile transaction restarts, the improvement made by OCC-Mix approach over other protocols is clear especially when the percentage of mobile and fixed transactions are equal as we can see in Figure 10. This is because of OCC-Mix have no blocking overhead compared to the 2PL and have less number of restarts than OCC. Figure 11 shows the PCR under all protocols when the mobile transaction are dominated, since the increase of PCR under OCC and OCC-Mix are mostly caused by the restart frequency, the difference between these protocols is reduced as we see in Figure 11 when the percentage of mobile transaction is 80%. So the best improvement which can be gained by the OCC-Mix when the percentages of mobile and fixed transactions are equals in the system.

Figure 9: Power consumption rate (MT = 20%)

Figure 10: Power consumption rate (MT = 50%)

Figure 11: Power consumption rate (MT = 80%)

Figure 12: Fixed rollback Frequency
5.2 Impact of $\sigma$-value

Sigma value can highly affect the nature of interactions between fixed and mobile transactions. A larger $\sigma$-value means a little opportunity for fixed transactions to commit in their time intervals. Figure 12 shows the fixed rollback frequency FRF for different values of $\sigma$. This measure determines the number of fixed transactions aborted because they are in conflict with other mobile transactions. As we can see in Figure 13, the number of fixed transactions roll backed by other mobile transactions increase as the $\sigma$-value increase. A choice of $\sigma$-value =2 leads to the smallest value of FRF. Smaller $\sigma$-value leads to more restarts in mobile transactions as we can see in the Figure 13. For mobile rollback frequency MRF which mean that the adjustments of an active mobile transaction that is in conflict with a validating fixed transaction whose timestamp interval bigger will short the timestamp intervals of other mobile and fixed transactions which may be in conflict in the future. As we shown in Figure 12 and 13, intermediate value of $\sigma$ leads to the best reduction of both fixed and mobile rollback frequency.

6. Conclusion

In this paper, a concurrency control approach called OCC-Mix has been proposed to alleviate the problem in mixed transactions environment. The idea is to avoid wasting of scars and expensive resources of mobile environment by (1) avoid unnecessary transaction restarts by dynamically adjusting the serialization order of the conflicting transactions with respect to the validating transaction and (2) make more room for mobile transaction which gives it a better chance to commit. Under the OCC-Mix approach, These done by exploiting the semantics between read and write operations in transactions in addition to the transaction type such that the serializability can be preserved without restarting the conflicting transactions of the validating transaction. Only those transactions with serious conflicts with the validating transaction have to be restarted. In case of non serious conflicts, we only need to adjust the serialization order of those conflicting transactions with respect to the validating transaction. As a result, unnecessary restarts can be removed. In addition to allowing those non serious conflicting fixed and mobile transactions to have opportunity to complete their executions, resources can be saved from being utilized by restarted transactions such that other ongoing transactions will not be affected. A series of simulation experiments has been done to investigate the performance of the OCC-Mix approach. It is found that the proposed protocols outperform the traditional optimistic concurrency protocol for a wide range of workload parameters. The first order improvement can be observed in decreasing the number of mobile transaction restarts that leads to a significant saving of resources. The second order improvement can be observed in the reduction of power consumption rate that is crucial to the mobile computing environments.
References


