OCC-Mix Approach for Concurrent Mixture of Fixed and Mobile Transactions

Ahmad Al-Qerem[1], Ala Hamarsheh[2], Shadi Nashwan[3]
[1]Department of CIS, Zarqa University
Zarqa, Jordan
ahmad_qerm@zu.edu.jo

[2]Faculty of Engineering and Information Technology, Arab American University
Jenin, Palestine
ala.hamarsheh@aauj.edu , alaauj@gmail.com

[3]Computer Science and Information Department
Aljouf University, KSA
shadi_nashwan@ju.edu.sa

Journal of Digital Information Management

ABSTRACT: *This paper addresses a new optimistic-based concurrency control protocol, called OCC-Mix. The proposed protocol could be used to reduce the scars wasting and expensive resources of mobile environment. This can be achieved by avoiding the unnecessary mobile transaction restarts. Consequently, a dynamic adjustment of serialization order in the conflicting transactions with respect to the validating transaction will be used. As a result, much resources of mobile environment are saved and a many fixed host transactions finished their execution without being affected by other mobile transactions. The performance of the proposed protocol was measured along with other related CC protocols; the simulation results show that the performance of these protocols is consistently much better than the traditional 2PL and OCC protocols in mixed transaction environments.*

Subject Categories and Descriptors
C.2.1 [Network Architecture and Design]; Wireless communication: **H.2.4 [Systems]; Concurrency: C.1.3 Architecture Styles];** Cellular architecture

General Terms: 26 May 2017, Revised 2 July 2017, Accepted 8 July 2017

Keywords: Concurrency Control, Mixed Transaction, Mobile Computing, Performance Evaluation

Received: 19 May 2017, Revised 28 June 2017, Accepted 4 July 2017

1. Introduction

Obviously, there are two aspects of the most concurrency control algorithms. These aspects are classified as when and how they deal with conflicts. The first aspect focuses on when these algorithms detect the conflict. However, the second aspect focuses on how these algorithms will overcome these conflicts. The two approaches 'Locking' and 'Optimistic' represent are implemented in concurrency control algorithms to represent these two aspects. For example, Locking can find conflicts and then it resolves them using blocking techniques. On the other hand, Optimistic approach finds conflicts only at the transaction's commit time and resolves them using restarts. In environments where these transactions are mixed, resolving data conflicts among these approaches is a big challenge to concurrency control protocols. Studying the performance of these concurrency control algorithms is one of the most important theme of these algorithms especially when they used in mixed database systems.

The effect of blocking and restarting in conflict resolution methods should be studied carefully in situations where the system resources are limited. Consequently, studies show that blocking-based conflict resolution policies conserve system resources. Whereas restart-based policies waste system resources. Previous studies related to the measurement of performance on traditional database

systems show that the performance of locking-based techniques that resolve data conflict using blocking approach is better than the performance restart-based approach techniques particularly when the system resources are limited. Besides, recent studies show that in case of the low resource utilization and hence, some of wasted resources could be ignored, as well as there are many available transactions are waiting for execution, then a restart-based technique that supports concurrent execution will be a better choice. This paper investigates the effect of the blocking and restarting approaches in environments where mixed transactions are used. The authors believe that the detecting time of conflicts as well as resolutions has a major impact on overall performance.

Optimistic algorithms can use two types of validation, backward and forward validations. When the optimistic algorithm uses backward validation, the delayed conflict resolution results in wasting more resources than locking protocol as a transaction can end up with restart after being paid for most of its execution. In contrast, when using forward validation, this problem is eliminated, since any transaction that reaches the validation phase is guaranteed to commit, and early transactions involved in any nonserializable execution will restart in their read phase. Besides the delayed conflict resolution of an optimistic approach helps in making better decisions in conflict resolution as more information about conflicting transactions is available at this later stage. The policy of immediate conflict resolution of locking schemes may lead to useless restarts and cause blocking in mixed systems due to the shortage of information on conflicting transactions at the time of conflict resolution.

The remainder of this paper is organized as follows: Section 2 briefly discusses some related work on concurrency control (*CC*) schemes. Section 3 describes the mixed database model. Section 4 illustrates the differences between fixed and interval timestamps. Section 5 discusses the two *OCC* validation techniques. The *OCC*-Mix approach is presented in Section 6. Section 7 discusses the performance results and finally section 8 concludes the paper.

2. Related Work

Many variations of *CC* schemes have been introduced to improve concurrency and system performance (e.g., see [1], [2], [3]), in conventional database environments. Lately, there have been several studies dealing with *CC* in mobile databases considering the transaction scheduling aspect (e.g., see [4][5][6] These *CC* schemes are mainly extended or adapted from existing *CC* schemes to the mobile environments, such as two phase locking and optimistic *CC* schemes, and multiversion schemes. Some of the *CC* schemes are more naturally adaptable to mobile requirements, while others are less so. For example, in optimistic concurrency control (*OCC*) schemes, the *CC* decision tends to be more or less independent of the *CPU* scheduling [7][8]. However, in the two-phase locking (2PL) approach, it's not practical for a mobile transaction to hold the lock on a certain data item then disconnect, as this will cause a massive performance degrading for both mobile and other fixed transactions concurrently running on the system and sharing the same database access [9]. Several techniques have been developed from conventional *CC* schemes to cope with the wireless requirements, especially for transaction processing in broadcasting environments, such as Update First Ordering (UFO) [10], Multi-Version Broadcast [11], Serialization Graph [12], Broadcast Concurrency Control with Time Stamp Interval (BCC-TI) [13], F-Matrix [14], and Certification Report [15]. The analysis and drawbacks of these methods can be found in [16]. As a summary, some of these methods only support client read-only transactions [17], and some of them could have substantial processing overhead. Protocol inefficiency is also introduced in some of these solutions because of the support of strict global serializability [18] as their correctness criterion. It is necessary to design a new broadcast concurrency control protocol to overcome these drawbacks. Very importantly, to improve transaction processing efficiency, we need to overcome these problems caused by global serializability, which is very difficult to achieve in distributed broadcast environment. In other type of mobile computing environment such as distributed and multi database system, most of the works were concentrated in proposing a mobile transaction models and execution frameworks for these models over such architectures and using the same traditional concurrency technique at the database server. for a mixture transaction processing a little work [24] has been done in this field and this will be consider one of our objective in this paper to propose a suitable concurrency protocol that take into account the wireless constraints of the mobile computing environment and at the same time keep some place for the fixed transaction which use a tolerable resources. In addition to the wireless requirements, other major differences also exist between conventional and mobile database systems.

3. Mixed System Model

This section describes the mixed system model. As shown in Figure 1, the database system consists of two major components, transaction manager (*TM*) and data manager (*DM*). *TM* records the execution status of the transactions of both fixed and mobile in a certain transaction table. The *TM* has two modules: scheduler and data manger. The scheduler module is responsible for concurrency control. For example, when the scheduler gets an operation, it decides whether this operation should be processed, blocked, or rejected. If the operation is rejected, then the corresponding transaction will be restarted. The scheduler records any possible data conflicts in the access-status. For example, if two phase locking (2PL) is used for concurrency control, the scheduler records in a lock table the locking status of data items that are used by all running transactions. This table is also be used for the optimistic methods. Generally, all requests to data items that are generated by a certain

operation, will be handled by the *DM*. All types of transactions that are involved in the mixed system model consist of a set of read and writes operations, and ends with either a commit or an abort operation. This type of transactions is considered as an atomic process. This means, it translates a database from a consistent state into other consistent state. There are two types of transactions in the mixed system model environment, fixed (*FT*) or wired transactions and mobile transactions. A fixed transaction is a transaction that is submitted directly to a database on the same host while. Whereas, a mobile transaction is a transaction that is submitted to a database by mobile device. Similar to [19], an (*MT*) is a mobile transaction which is issued by a mobile host. The contribution of mobile hosts requires handling the mobility features such as movement, disconnections and variations on the quality of communication. *TM* should capable to deal with mobility features of mobile hosts. The scope of this paper focuses only on a client-server system architecture, where clients are either *MH* or *FH*. These clients access a shared databases by calling certain transactions.

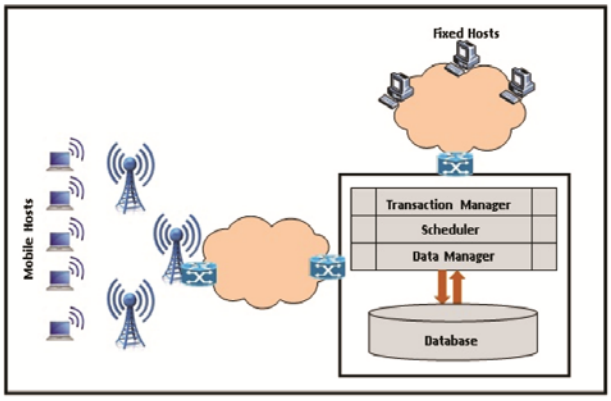

Figure 1. Mixed transactions environment

4. Time Interval Versus Fixed Timestamp

This section differentiates between time interval and fixed timestamp-based transactions. In a fixed timestamp-based approach [13], timestamps are assigned to transactions at the beginning of execution of these transactions. For example, when a transaction initiates a request, that request will create a conflict between the requesting transaction and other transaction, then timestamps of both transactions are compared. Thereafter, if the order both timestamps is similar to the serialization order required by the conflict, then the request is allowed. Otherwise, the requesting transaction will be aborted and restarted with a new timestamp. As a result, fixing serialization order of transactions in advance is essential as it is potentially may cause many unnecessary transaction aborts.

In the time intervals approach [20], each transaction is assigned two timestamps. The assigned timestamps can be understood as an upper and lower bounds of an interval of timestamp time. This interval represents the serialization order of the transaction. It is worth noting that these 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). The interval of every transaction should span the entire acceptable timestamp range. This implements the rule that the timestamp range has no restriction where to place it in serialization order on condition that there are no conflict with other transactions has occurred.

5. Forward Versus Backward Validation

Basically, the transaction in optimistic concurrency control are eligible to run until reaching the commit point. There are three phases in running a transaction, reading, validating and writing phase. In the reading phase, all write operations occur in local copies of the records. If these writes can be made within the validation phase, and the resulting changes on the transaction are not going to violate the serializability with respect to all other committed transactions, in this case the local copies are made global. In the writing phase only, the local copies will be available to all other transactions.

Generally, there are two properties of *OCC* protocols that differentiate it from other *CC* protocols. First, synchronizing transactions is accomplished by restarts, hence there will be no blocking. Second, taking the decision to restart the transaction or not is made after it has finished the execution. The key point among phases is the validation phase where the density of the transaction is defined. Validating the transactions comes in different forms, but every validation phase depends on the following principle to ensure the transaction serializability: "If a transaction Ti is serialized before transaction *Tj*, the write of *Ti* should not affect the read phase of *Tj*".

Traditionally, most validation processes can be carried out in any of the following schemes [1]. (1) Forward validation: states that validating a transaction is made against currently executing transactions. This process depends on the assumption that the validating transaction comes after every concurrent running transaction that is still in the reading phase of the serialization order. Therefore, detecting data conflicts can be achieved by comparing the write set of the validating transaction and the read set of active transactions. For example, if an active transaction, *Ti*, is reading an object that concurrently is being written by a validating transaction, the value of the object that is used by *Ti* is not consistent. These data conflicts is resolved by restarting either the validating transaction or the conflicting transactions in the reading phase. Optimistic algorithms that depends on this validation process are studied in [19], [21]. (2) Backward validation: means that validating transaction is made against committed transactions. Detecting data conflicts can be achieved by comparing the write set of committed transactions with read set of the validating transaction.

6. Approach Details

OCC-Mix is an optimistic protocol which depends on the dynamic adjustment of serialization order. Similar to *OCC-TI* [22], *OCC*-Mix uses the concept of timestamp intervals to record and represent serialization orders caused by concurrency dynamics. Timestamps are used in both transactions and data items. With respect to data items, the read and the write timestamps are the timestamps of last committed transactions that read or wrote the data item. On the other hand, transactions that have a timestamp interval they expressed as a pair of [lower bound (*lb*), upper bound (*ub*)]. The timestamp interval indicates the validity interval of a transaction. The timestamp intervals are also used to indicate the serialization order between transactions. For example, if *Ti* (with timestamp interval [*lbi*, *ubi*]) is serialized before *Tj* (with timestamp interval [*lbj*, *ubj*]), denoted $Ti \rightarrow Tj$, then the following relation must hold: $ubi < lbj$.

6.1 Adjustment of Timestamp Interval

A timestamp interval (i.e. [0, ∞]) is assigned to each starting transaction. Because the transaction continues the execution through its lifetime in the system, the assigned timestamp interval should be adjusted cope with the serialization dependencies. The serialization dependencies might need to be adjusted while a transaction is accessing data items either in the reading phase or by being in the conflict set of a different validating transaction.

6.1.1 Adjustment at the Reading Phase

In this phase, the timestamp interval is fixed with respect to the read and write timestamps of the data item read or update. Secondly, by being in the conflict set of a different validating transaction. In such case, the timestamp interval is changed to dynamically correct the serialization order. The process of correcting the serialization order, the timestamp interval may 'shut out', i.e., become empty. In that case, the transaction cannot be successfully serialized and it needs to be restarted. It is worth noting that this is one of the major variations between conventional protocols and protocols depends on the dynamic adjustment of serialization order. In traditional *OCC* algorithms, restarts can only happen at validation times. However, in our case, transactions can restart at any other periods in case a timestamp interval shut out is identified. Figure 2 illustrates these adjustments in the reading phase. In this paper, we use the notation *TI* (*Ti*) to denote the timestamp interval of transaction *Ti* and *RTS* (*Di*) and *WTS* (*Di*) to denote the read and write stamps of data item *Di*, respectively. As a transaction successfully validates, a final timestamp is assigned to it. In *OCC*-Mix protocol we should select the final (commit) timestamp *TS* (*TV*) in such a way that room is left for backward adjustment.

6.1.2 Adjustment at the Validation Phase

The fine-tuning of the serialization order for both fixed transactions and mobile are implemented using timestamp intervals will create a partial order between transactions based on conflicts and transaction type. For example, given a validating transaction *Tv* and an active transaction *T*, let *TS* (*Tv*) be the final timestamp of the validating *y* transaction *Tv*and *TI* (*Ta*) is the timestamp interval of the active transaction Ta, Let *TI* (*Tv*) be the timestamp interval of the validating transaction and type(*Ti*) be the transaction type where type(*Ti*) ∈ {*mobile, fixed*}. We assume here that there is no blind write so there are two possible types of conflicts which are resolved using type dependent adjustment of serialization order between *Tv* and *Ta*:

Read-write conflict which occur when the $RS\ (Tv) \cap WS\ (Ta) \neq \varphi$ which can be resolved by forward adjustment. (2) write-read conflict which occur when the $RS\ (Ta)\ WS\ (Tv) \neq \varphi$ and resolved by backward adjustment.

```
Read_Phase (Ta)
   Begin
      For each Di in RS (Ta)
            Begin
               Read (Di);
               TI (Ta) = TI (Ta) ∩ [WTS (Di), ∞]
               If TI (Ta) = [] Then Restart (Ta);
            End
      For each Di in WS (Ta)
            Begin
               Pre-write (Di);
               TI (Ta) = TI (Ta) ∩ [WTS (Di), ∞] ∩ [RTS (Di), ∞]
               If TI (Ta) = [] Then Restart (Ta);
            End
   End
```

Figure 2. Adjustment of $TI\ (T_a)$ at the read phase

```
Iterate conflicting transaction (Ta, Tv)
Begin
   For all Di ∈ (RS (Tv) ∪ WS (Tv))
     Begin
       For each Ta ∈ conflicting set of the validating transaction
           Begin
             If    Di ∈ (RS (Tv) ∩ WS (Ta)) Then
                   Forward adjustment (Ta, Tv);
             If    Di ∈ (RS (Ta) ∩ WS (Tv)) Then
                   Backward adjustment (Ta, Tv);
           End
     End
End
```

Figure 3. Iterating reading set/writing set of validating Transaction

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, checking if the validating transaction has read from committed transactions. This can be done by checking the data items of read timestamp (*RTS*) and write timestamp (*WST*). The values of these data items are fetched when finish the read/write operations of the current data item. Thereafter, the algorithm iterates the set of active conflicting transactions. When accessing of the same objects in both validate and active transaction is done, the temporal time interval of the active transaction is adjusted. The process of detecting the execution of a non-serializable transaction is possible when the interval of the timestamp of an active transaction becomes empty. If the timestamp interval is empty, then the transaction will be restarted.

6.2.1.1 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. $Tv \rightarrow Ta$). 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 time stamp earlier in timestamp interval. 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 σ- value increase, opportunity to commit for the active mobile transaction increase on the account of the fixed host transaction. When the σ- value = 2 this mean that the validating fixed host transaction scarify by half of its interval to the advantage of active mobile transaction.

Example 1: Let $TI(T1) = [20, 60]$, $TS(T1) = 60$ and $TI(T2) = [10, 80)$. Let $T1$.type = fixed and $T2$.type = mobile. Assume we have a read-write conflict between transactions and σ = 2. We first make more room for the mobile transaction $T2$ and then move the mobile transaction forward.

$$TS(T1) = 20 + \left\lfloor \frac{60 - 20}{2} \right\rfloor = 40$$

$$TI(T2) = [10, 80] \cap [40, \infty] = [40, 80]$$

The resulting selected value should be within the timestamp interval. This offers a greater chance for the mobile transaction to commit in its timestamp interval. If this resulting point cannot be selected, the validating transaction is restarted. This is wasted execution, but it is required to ensure the execution of the mobile transaction. This forward adjustment can be described by the procedure in Figure 4.

```
Forward Adjustment (Ta, Tv)
   Begin
        If Tv.type = = Fixed Then
             If Ta.type = = Mobile Then
TS(Tv)' = min(TI(Tv)) + ⌊ (TS(Tv) − min(TI(Tv))) / σ ⌋
                  If TS(Tv)' > max ((TI (Ta)) Then
                       Restart (Tv);
                  Else
                       TI (Ta) = TI (Ta) ∩ [TS(Tv)', ∞]
             Else Ta.type = =Fixed
                       TI (Ta) = TI (Ta) ∩ [TS (Tv), ∞]
                       If TI (Ta) = [] Then Restart (Ta)
        Else Tv.type = =Mobile
                  TI (Ta) = TI (Ta) ∩ [TS (Tv), ∞]
                  If TI (Ta) = [] Then Restart (Ta);
End
```

Figure 4. Forward adjustment

Example 2: Forward Adjustment

Consider $Ri(x)$ is a read and $Wi(x)$ is a write operation on data item x of a transaction i, and consider vi is the validation ci is the commit of transaction i. Given the following three transactions $T1$, $T2$, and $T3$:

$T1$: $R1(D1)$ $W1(D3)$ $R1(D2)$ $v1$

$T2$: $W2(D2)$ $W2(D4)$ $v2$.

$T3$: ... $W3(D1)$ $v3$

Consider that these transactions are executed as follows: H =... $R1(D1)$ $W3(D1)$ $v3$ $W2(D2)$ $W1(D3)$ $W2(D4)$ R1 $(D2)$ $v1$ $v2$.

D_1	RTS	40
	WTS	20
D_2	RTS	50
	WTS	30
D_3	RTS	30
	WTS	65
D_4	RTS	40
	WTS	75

Figure 5 shows how the timestamp intervals for transactions are adjusted with respect to the RST and WST of the data items accessed by these transactions. Let us illustrate the forward adjustment by taking the following scenarios:

Operation		$TI(T_i)$
$R_1(D_1)$	$TI(T_1)=[0,\infty] \cap [20,\infty]$	$[20,\infty]$
$W_3(D_1)$	$TI(T_3)=[0,\infty] \cap [20,\infty] \cap [40,\infty]$	$[40,\infty]$
V_3	$TS(T_3)$= validation time=81	[40,81]
C_3	$TI(T_1)=[20,\infty] \cap [0,80]$ After backward adjustment of $TI(T_1)$	[20, 80]
$W_2(D_2)$	$TI(T_2)=[0,\infty] \cap [30,\infty] \cap [50,\infty]$	$[50,\infty]$
$W_1(D_3)$	$TI(T_1)=[20,80] \cap [30,\infty] \cap [65,\infty]$	[65,80]
$W_2(D_4)$	$TI(T_2)=[50,\infty] \cap [40,\infty] \cap [75,\infty]$	$[75,\infty]$
$R_1(D_2)$	$TI(T_1)=[65,80] \cap [30,\infty] \cap [50,\infty]$	[65,80]
V_1	$TS(T_1)$= validation time=100	

Figure 5. Processing for example 2

(1) $T1$ fixed and $T2$ mobile (active: mobile, validating: fixed, read/write conflict).

(2) $T1$ fixed and $T2$ fixed (active: fixed, validating: fixed, read/write conflict).

(3) $T1$ mobile and $T2$mobile (active: mobile, validating: mobile, read/write conflict).

(4) $T1$ mobile and $T2$ fixed (active: fixed, validating: mobile, read/write conflict).

Let the validation time of transaction $T1$=100. Because $100 \notin TI(T1)$, we select $max(TI(T1))$ to be the final commit timestamp. So $TS(T1) = 80$. At the validation time of transaction $T1$, we find that $(RS(T1) \cap WS(T2) = \{D2\}$. So, transaction $T2$ is in the conflict active set of $T1$ with a read-write conflict, which results in a forward adjustment of transaction $T2$. In scenario (1), since the validation transaction is fixed and the active is mobile, we first ensure that the validation of the fixed transaction $T1$ will not result in restarting the mobile transaction $T2$ by checking that $TS(T_1)' < \text{Max}(TI(T2))$.

$$\text{TS}(\text{T}_1)' = \min(\text{TI}(\text{T}_1)) + \left\lfloor \frac{\text{TS}(\text{T}_1) - \min(\text{TI}(\text{T}_1))}{\sigma} \right\rfloor$$

Assume $\sigma = 2$, then

$$\text{TS}(\text{T}_1)' = \min([80, \infty)) + \left\lfloor \frac{80 - 65}{2} \right\rfloor = 67$$

Because the condition is satisfied (i.e. $67 < \infty$), we first make more room for the mobile transaction $T2$ and then move the transaction forward. So the time interval of $T2$ become $TI(T2) = [75, \infty] \ [67, \infty] = [67, \infty]$ and transaction $T1$ is successfully validated against the mobile transactionT2 and commit with a final timestamp $TS(T1) = 67$. For the other scenarios (2), (3) and (4) the forward adjustment of $T1$ $(T2)$ will be as follows: $TI(T2) = [75, \infty] \cap [80, \infty] = [80, \infty]$. And transaction $T1$ is successfully validated and commits with a final timestamp $TS(T1) = 80$.

Example 3: Forward Adjustment:

Consider three transactions $T1$, $T2$, and $T3$: which are different from transactions in Example 3.

$T1$: $R1(D1)$ $W1(D2)$ $R1(D3)$ $v1$

$T2$: $R2(D2)$ $W2(D4)$ $v2$.

$T3$: ... $W3(D1)$ $v3$

Now, suppose that these transactions execute as follows: $H2$ =... $R1(D1)$ $W3(D1)$ $v3$ $R2(D2)$ $W1(D2)$ $W2(D4)$ $R1$ (D3) $v2$ $v1$.

Figure 6 shows how the timestamp intervals for transactions have been adjusted with respect to the *RST* and *WST* of the data items accessed by these transactions.

Consider the same scenarios in Example 3; the forward adjustment will work as follows:

In scenario (1), since the validation transaction is fixed and the active is mobile, we first ensure that the validation of the fixed transaction $T2$ will not result in restarting the mobile transaction $T1$.

$$TS(T_2)' = \min(TI(T_2)) + \left\lfloor \frac{TS(T_2) - \min(TI(T_2))}{\sigma} \right\rfloor$$

Assume $\sigma = 2$, then

$$TS(T_2)' = \min([65,100)) + \left\lfloor \frac{100-65}{2} \right\rfloor = 82.$$

Since $TS(T_2)' > \text{Max}(TI(T1))$, that is $82 > \max([65, 80])$, the validating fixed transaction $T2$ will be restarted and give the mobile transaction $T1$ the opportunity to continue its execution and commit. For the others scenarios (2), (3) and (4) forward adjustment of $TI(T1)$ will be as follows:

$TI(T1) = [65, 80] \cap [100, \infty] = []$, since the $TI(T1)$ is empty, Transaction $T1$ will be restarted and $T2$ commit with a final timestamp $TS(T2) = 100$.

Operation		TI(T_i)
$R_1(D_1)$	TI (T_1) =[0, ∞] ⋂ [20, ∞]	[20, ∞]
$W_3(D_3)$	TI (T_3) =[0, ∞] ⋂[20, ∞] ⋂ [40, ∞]	[40, ∞]
V_3	TS(T_3)= validation time=81	[40,81]
C_3	TI (T_1) =[20, ∞] ⋂ [0,80]	[20, 80]
	After backward adjustment of TI (T_1)	
R_2 (D_2)	TI (T_2) =[0, ∞] ⋂ [30, ∞]⋂ [50, ∞]	[50, ∞]
$W_1(D_2)$	TI (T_1) =[20,80] ⋂[30, ∞] ⋂ [50, ∞]	[50,80]
$W_2(D_4)$	TI (T_2) =[50, ∞] ⋂ [40, ∞]⋂ [65, ∞]	[65, ∞]
R_1 (D_3)	TI (T_1) =[65,80] ⋂[30, ∞] ⋂ [50, ∞]	[65,80]
V_2	TS(T_1)= validation time=100	

Figure 6. Processing for example 3

D_1	RTS	40
	WTS	20
D_2	RTS	50
	WTS	30
D_3	RTS	30
	WTS	65
D_4	RTS	40
	WTS	65

6.1.2.2 Backward Adjustment

A write-read conflict between Ta and Tv can be resolved by adjusting the serialization order between Tv and Ta by adjusting the timestamp interval of the active transaction backward, (i.e. $Ta \rightarrow Tv$). If the validating is a mobile transaction and the active conflicting is a fixed transaction, backward adjustment is correct. If the validating is fixed transaction and the active conflicting transaction is mobile, then back-ward 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. Thus, 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 the scarce and expensive resources. Backward adjustment can be described by procedure in Figure 7.

```
Backward Adjustment (Ta, Tv)
 Begin
   If (Tv.type == Fixed) Then
       If (Ta.type == Mobile) Then
             If (TS (Tv) - 1 < Min (TI (Ta)) Then
                    Restart (Tv);
             Else
                    TI (Ta) = TI (Ta) ⋂ [0, TS (Tv)-1]
       Else Ta.type == Fixed
              TI (Ta) = TI (Ta) ⋂ [0, TS (Tv)-1]
        If TI (Ta) = [] Then Restart (Ta)
             Else Tv.type == Mobile
              TI (Ta) = TI (Ta) ⋂ [0, TS (Tv)-1]
        If TI (Ta) = [] Then Restart (Ta)
 End
```

Figure 7. Backward Adjustment

Example 4: Backward Adjustment:

Let $Ri(x)$ and $Wi(x)$ denote a read and write operation, respectively, on the data item x by transaction i, and let vi and ci denote the validation and commit of transaction i, respectively. Consider three transactions $T1$, $T2$ and $T3$:

$T1{:}R1(D1)\ W1(D3)\ W1(D2)\ v1$

$T2{:}R2(D2)\ W2(D4) \ldots v2$

$T3{:} \ldots W3(D1)\ v3$

Now, suppose they execute as follows:

$H = \ldots R1(D1)\ W3(D1)\ v3\ R2(D2)\ W1(D3)\ W2(D4)\ W1(D2)\ v1 \ldots v2.$

Figure 8 shows how the timestamp intervals for transactions have been adjusted with respect to the RST and WST of the data items accessed by these transactions.

We will consider two cases (a) and (b) based on the $WTS(D4)$ for the following scenarios:

(1) $T1$ fixed and $T2$ mobile (active: mobile, validating: fixed, write/read conflict)

(2) $T2$ fixed and $T1$ mobile (active: fixed, validating: mobile, write/read conflict)

Case (a): $WTS(D4) = 65$:
In scenario (1), the timestamp interval of transaction $T2$

Case (a)			Case (b)		
D1	RTS		D1	RTS	40
	WTS			WTS	20
D2	RTS		D2	RTS	50
	WTS			WTS	30
D3	RTS		D3	RTS	30
	WTS			WTS	65
D4	RTS		D4	RTS	40
	WTS			WTS	75

Operation		$TI(T_i)$
$R_1(D_1)$	$TI(T_1)=[0,\infty] \cap [20,\infty]$	$[20,\infty]$
$W_3(D_1)$	$TI(T_3)=[0,\infty] \cap [20,\infty] \cap [40,\infty]$	$[40,\infty]$
V_3	$TS(T_3)$= validation time=71	[40,81]
C_3	$TI(T_1)=[40,\infty] \cap [0,70]$ After backward adjustment of $TI(T_1)$	[40, 70]
$R_2(D_2)$	$TI(T_2)=[0,\infty] \cap [30,\infty]$	$[30,\infty]$
$W_2(D_3)$	$TI(T_1)=[40,70] \cap [30,\infty] \cap [65,\infty]$	[65,70]
$W_2(D_4)$	$TI(T_2)=[30,\infty] \cap [40,\infty] \cap [65,\infty]$	$[65,\infty]$
$W_2(D_4)$	$TI(T_2)=[30,\infty] \cap [40,\infty] \cap [75,\infty]$	$[75,\infty]$
$W_1(D_2)$	$TI(T_1)=[65,70] \cap [30,\infty] \cap [50,\infty]$	[65,70]
V_1	$TS(T_1)$= validation time=90	

Figure 8. Processing for example 4

after executing $W2(D4)$ will be set to $[65,\infty]$. For transaction $T1$, because the validation time does not belong to $TI(T1)$ we chose max $TI(T1)$ as a final commit timestamp which is 70. At the validation time of $T1$, transaction $T2$ is in the active conflict set of $T1$ with write-read conflict $(WS(T1) \cap RS(T2) = \{D2\}$. So, $TI(T2)$ will be backward adjusted as follows: we first make sure that the validation of the fixed transaction still gives an opportunity for the mobile transaction to be validated (i.e.$(T1)$-1< min $(TI(T2))$ then do the adjustment $TI(T2) = [65,\infty] \cap [0, 69] = [65,69]$. So $T1$ is successfully validated and T2 still has the opportunity to be validated in the time interval $TI(T2) = [65, 69]$. In scenario (2), the validating transaction $T1$ is mobile. So, we do the backward adjustment for the fixed transaction $T2$ directly as follow: $TI(T2) = [65,\infty] \cap [0, 69] = [65, 69]$.Then check if the interval is shut out. If so the fixed transaction will be restarted.

Case (b): $WTS(D4) = 75$:
In scenario (1), the timestamp interval of transaction $T2$ after executing $W2(D4)$ will be $[75,\infty]$. For transaction $T1$, because the validation time = 90 does not belong to the $TI(T1)$ we choose max $(TI(T1))$ as the final commit timestamp which is 70. At the validation time of $T1$, transaction T2 is in the active conflict set of the $T1$ with write-read conflict $(WS(T1) \cap RS(T2) = \{D2\}$.So, $TI(T2)$ will be backward adjusted as follows: We first make sure that the validation of the fixed transaction still give an opportunity for the mobile transaction to be validated (i.e. $TS(T1)$-1<min $(TI(T2))$ then do the adjustment. since 70-1<75 and the validating transaction $T1$ will result in restarting the mobile transaction $T2$, we chose to restart $T1$. As a result, $T2$ still have an opportunity to complete its execution and it may successfully validate. In scenario (2), the validating transaction $T1$ is mobile, $T1$ is validated first then we do the backward adjustment for the fixed transaction $T2$ as follow: TI (T2) = $[75,\infty] \cap [0, 69] = []$. Since the timestamp interval of $T2$ is shut out the fixed transaction will be restarted. In general, if the validating and the active conflict transactions are not backward adjusted before, the active conflicting transaction will be never restarted by the validating transaction in such case. This is true for all possible scenarios for both mobile and fixed.

6.2 Final Timestamp Selection

In the *OCC*-Mix protocol, the final (commit) timestamp $TS(Tv)$ should be selected in such a way that room is left for backward adjustment. The proposed validation algorithm in figure 9, setting $TS(Tv)$ as the validation time if it belongs to the time interval of Tv or the maximum value from the time interval otherwise. To justify this choice consider the following example:

Let $RTS(x)$ and $WTS(x)$ are initialized as 100. Consider

transactions $T1$, $T2$ and history H, where $T1$.type= $T2$.type = mobile:

$T1$: $R1(x)$ $W1(x)$ $v1$ $c1$

$T2$: $R2(x)$ $v2$ $c2$

$H = R1(x)$ $R2(x)$ $W1(x)$ $v1$.

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

Case (a): $TS(Tv) = \max(TI(Tv))$

In a similar way, transactions $T1$ and $T2$ are forward adjusted to [100, ∞) Transaction $T1$ starts the validation at time 1000, and the final (commit) timestamp is selected to be $TS(T1)$ = validation_time =1000. Because we have one write-read conflict between the validating transaction $T1$ and the active transaction $T2$, the timestamp interval of the active transaction must be adjusted: $TI(T2) = [100, \infty) \cap [0, 999] = [100, 999]$. 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 $TS(Tv) = max(TI(Tv))$ avoids the unnecessary restart problem.

Case (b): $TS(Tv) = \min(TI(Tv))$

Transaction $T1$ executes $R1(x)$ which causes the timestamp interval of the transaction to be forward adjusted to [100, ∞) then $T2$ 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,). $T1$ then executes $W1(x)$ which causes the timestamp interval of the transaction to be forward adjusted to [100,). $T1$ starts the validation, and the final (commit) timestamp is selected to be $TS(T1) = \min([100, \infty)) = 100$. Because we have one write-read conflict between the validating transaction $T1$ and the active transaction $T2$, the timestamp interval of the active transaction must be adjusted: Thus $TI(T2) = [100, \infty) \cap [0,99] = []$. 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 ($TS(T1)$) 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 10 current read timestamps and write timestamps of accessed data items are updated and changes to the database are committed.

```
Select final time stamp (Tv)
     Begin
          If     (validation _time) ∈ TI (Tv) Then
                   TS (Tv) = validation time;
          Else
                   TS (Tv) = max (TI (Tv));
     End
```

Figure 9. Select commit timestamp

```
Update Data Item Timestamps (RS (Tv) ,WS (Tv))
   Begin
        For all Di ∈ (RS (Tv) ∪ WS (Tv))
             Begin
                 If (Di ∈ RS (Tv)) Then
                 RTS (Di) = max (RTS (Di)), TS (Tv));
                 If (Di ∈ WS (Tv)) Then
                 WTS (Di) = max (WTS (Di)), TS (Tv));
             End
```

Figure 10. 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.

7. Performance Evaluation

This section conducts a series of simulation experiments to evaluate the performance of the proposed concurrency protocols with other related protocols on mixed transactions environment. The following subsections will illustrate the simulation model, workload being used, performance metrics and finally summarize the research outcomes.

7.1 Simulation Model

The simulation model that was used in the experiments consists of stationary units and mobile part. These stationary units are classified according to either fixed hosts or base stations. The fixed host refer to database server connected to a wired-based network. The database server is reachable by two types of users; first user, is a user that uses the wired reliable network, and the second user, is a user who no access to these information servers. The base station has a wireless interface. It coordinates the communication between mobile units and stationary units. Generally, there are two types of transactions; mobile transactions which submitted by the mobile clients and wired-based client transactions which processed at the server's side. Both mobile and fixed host transactions contain both read and write operations. All data objects have the same chance of being accessed by any transaction's operation. The simulator can handle this chance by implementing two different generators. One of the generators is used by fixed hosts; and will be responsible for generating the transactions of this type of hosts. The other one is used in mobile transactions to simulate the mobile device as well as the wireless environment. When a mobile host submitted a transaction, the transaction will be sent to the base station through a wireless link and then it will be sent to the database through an existing wired network. In some scenarios, a transaction may require the user's interaction to the input data during the execution time. Based on many existing studies [23], this will increase the cost of communication setup. In order avoid re-establishing the communication setup each time the transaction needs the user's interaction, the proposed protocol keeps the communication during the execution time particularly in mobile transactions. The CPU queues the generated transactions from both generators (i.e. fixed and mobile) based on first in first out scheduling discipline. When the CPU is available, one of the transactions at the front of the CPU queue will be picked up and submitted to the CPU for processing. In order to read a data object, transactions need to be lined up in the disk queue for accessing the data. The transactions need to repeat all these stages until all operations are processed. The data object table and a transaction table need to be maintained in the system. The data object table is responsible for recording the read timestamp and the write timestamp for each data object in the database. Similarly, the transaction table is responsible for maintaining the read set, write set and the timestamp interval of each transaction.

As shown in Figure 11, the based station provides the communication channels to all mobile units located within a cell. The mobile users can move between cells. As a result, when a mobile user leaves and enters a new cell, and in order to allow the mobile unit to continue its communication without connection interruptions, the new base station should be able to provide an idle channel to this mobile unit. This is called a handoff process. However, in case that there are no idle channels available in the new cell, the mobile host will be disconnected. The disconnection of active mobile transaction, if happens, will force the transactions to rollback and then to restart, and hence

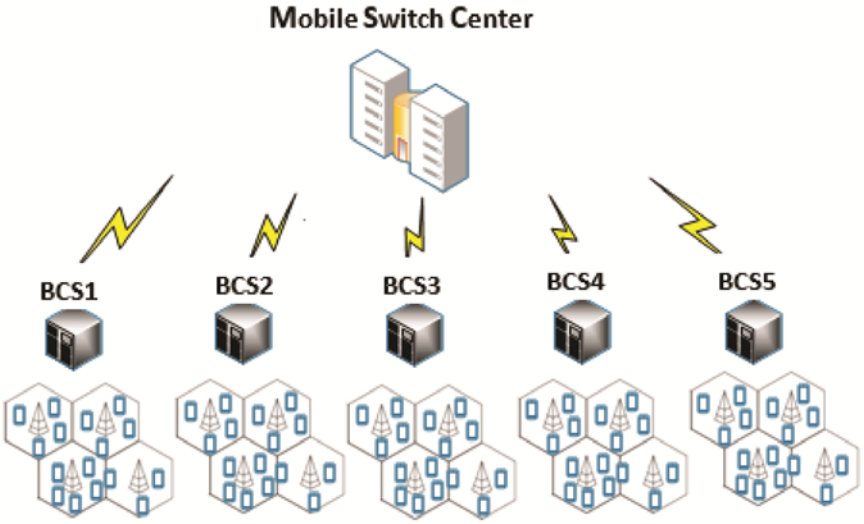

Figure 11. Wireless Part of Mixed Transaction Environment

the disconnection wastes the system's resources.

Figure 11 shows that the simulator's wireless part consists of 20 cells. Each cell is controlled by a MSS and every four MSSs are govern by one BSC, and finally every five BSCs are govern by one MSC which have a connection to the fixed host server. The following attributes are used to determine the time between arrival of the mobile transaction operations.

$$BW_t^{cell_i} = \frac{\text{BW offer by BSC}_i}{\text{Number of mobile users in cell}_i \text{ at time}_t}$$

$$TBA_t^{cell_i} = \frac{1}{BW_t^{cell_i}} * TBA_{generator}$$

The mobility of each mobile host modelled by estimating the number of visited base station during transaction execution. The visited base station for each mobile host is randomly selected when the handoff occur. This will affect the number of users under each cell during the simulation which affects the bandwidth available for the users. The numbers of users in each cell with random handoff imitated process are used to determine the time between arrivals of mobile transaction operation that uses by the mobile transaction generator during the simulation. The power consumption is measured by monitoring the three basic metrics-energy components: (i) transmission power required to send a message (i.e. operation), (ii) reception power required to receive or listen to a message, and (iii) idle power required to stay in at the active state (awake) during transaction execution. Therefore, the power consumed by any mobile host during each execution is given by the linear equation:

Power consumption (MHi) =

$$PCR(MH) = \sum_{t0}^{t_f} NopS * \beta^t + \sum_{t_0}^{t_f} NDR * \alpha + \omega * idel\_time$$

Where NopS is the number of operation sending by Ti, NDR is the number of data items received by Ti. And β, α are fixed cost associated of each operation and data item at the time of sending the operation and receiving the data item, respectively. ω is an idle power required to stay in at the active state (awake) during transaction execution.

Parameter Setting

Table 1 determines the names and meanings of all parameters that control system resources. For example, *CPUTime* and *DiskTime* parameters is used to capture the CPU and disk processing times per data page. Table 2 shows the key parameters that characterize system workload and transactions. The number of data objects accessed by a transaction and the actual data objects are determined uniformly from the database. A data object that is read is updated with the probability, *WriteProb*.

7.2 Performance Metrics

In MDBS, the major performance measure that can demonstrate the effectiveness of the protocols is the restart ratio. The restart ratio measures the average number of restarts experienced by a mobile host transaction before it can commit. This measure also indicates the amount of resources spent on restarted transactions. Reducing the restart cost not only saves resources, but also helps to soothe resource and data contention. The mobile transaction rollback frequency and the fixed transaction rollback frequency help to analyze the source

System	
Reported value test session	means of 10 times replication of each
Database size	300
CPU execution time per operation	25 ms
CPU queuing discipline	first in first out
Number of cells	20
Number of channel per cell	5
Channel Bandwidth	1 operation per millisecond

Table 1. The workload for baseline experiments

Transaction	
TBA of fixed transactions	1 transaction per 10ms
TBA of mobile transaction	1 transaction per 15 ms
Fixed host transaction length ($N_{operations}$)	3 - 10 uniformly distributed
TBA of fixed transaction operations	3.0 – 5.0 ms
Probability of disconnection	0.1, 0.2, 0.3
Mobility values	1, 2, 3, 4, and 5 BS/Transaction
Power of the mobile host	200 – 600 ms
Mobile host transaction length ($N_{operations}$)	3 – 15 uniformly distributed
Fixed transaction writes probability	0.4
Mobile transaction writes probability	0.3

Table 2. The parameters that characterize system workload and transactions

of transaction restarts. The frequencies can reflect the proportion of data conflict between mobile and fixed transactions and that among mobile transactions as the utilization of mobile transactions varies. The last performance measure specifically for the *OCC*-Mix approach is the adjustment ratio, which is the number of serialization order adjustment made per transaction. This ratio can help to understand the effectiveness of the *OCC*-Mix approach in different parameter settings. The formulas of the main performance measures are listed below.

$$\text{Power consumption ratio } (PCR) = \frac{P_{initial} - P_{final}}{P_{final}}$$

$$\text{Restart ratio} = \frac{N_{restart}}{N_{committed}}$$

$$\text{Adjustment ratio} = \frac{N_a}{N_{restart} + N_{committed}}$$

$$\text{Fixed transaction rollback frequency} = \frac{N_{fixed,\ restart}}{N_{committed}}$$

$$\text{Mobile transaction rollback frequency} = \frac{N_{mobile,\ restart}}{N_{committed}}$$

Where

$N_{restart}$: Number of restarted transaction of both types.

$N_{committed}$: Number of committed transactions of both types.

$N_{mobile,\ restart}$: Number of committed mobile transaction which caused other mobile transaction to be restarted.

$N_{fixed,\ restart}$: Number of committed mobile transactionwhich caused other fixed transaction to be restarted.

N_a: Total number of adjustment made.

7.3 Experiments and Results

7.3.1 Impact of Mobility

Figure 12 (A) shows the *PCR* as a function of the mean mobility value. This scenario shows that the 20% of transactions in the system are mobile-based transactions. When increasing the mobility value in the situation where the percentage of base stations that crossed over by a mobile transaction increase, the delay between operations of mobile transaction will increase and hence the average numbers of active transactions will increase. We noticed here the blocking of 2PL protocol is negatively effect on the *PCR* values. With respect to the *OCC* protocol, there is no blocking overhead. However, reduction of *PCR* values still exist because of restarts that is generated by mobile transaction. Since the *OCC*-Mix have no blocking overhead compared to the 2PL and have less number of restarts than *OCC*, figure 12(B) shows that the improvements of *OCC*-Mix approach over other protocols particularly when the percentage of mobile and fixed transactions are equal. Figure 12(C) shows the scenario of *PCR* of all protocols when the mobile transaction are dominated, the increase of *PCR* values in *OCC* and *OCC*-Mix are generated because of the frequent restarts. Figure 12(C) explains the reduced difference between these protocols in situation where the percentage of mobile transaction is 80%. Therefore, the best improvement that can be achieved by the *OCC*-Mix protocol when the equalling the percentages of mobile and fixed transactions.

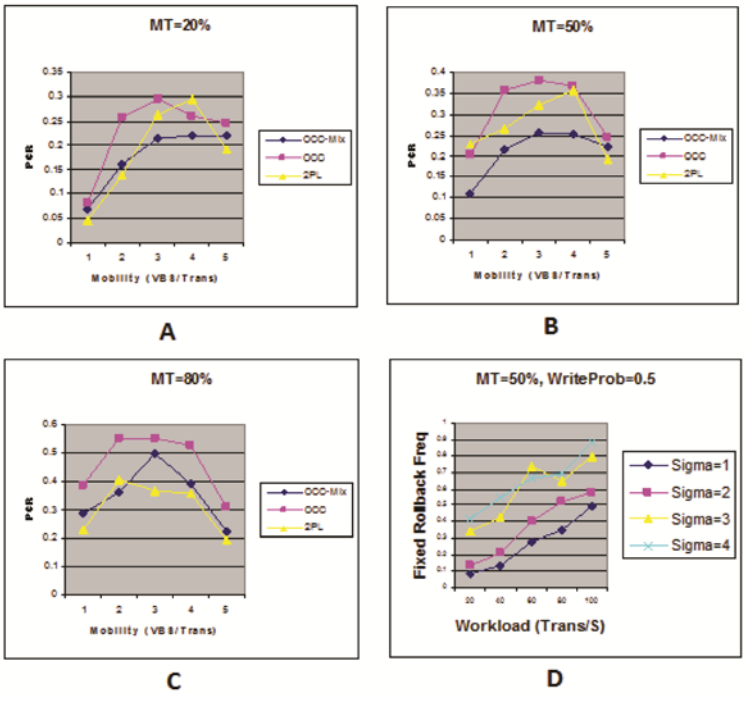

Figure 12 A. Power consumption rate (MT = 20%), Figure 12 B. Power consumption rate (MT = 50%), Figure 12 C. Power consumption rate (MT = 80%), Figure 12 D. Fixed rollback Frequency

7.3.2 Impact of σ - value

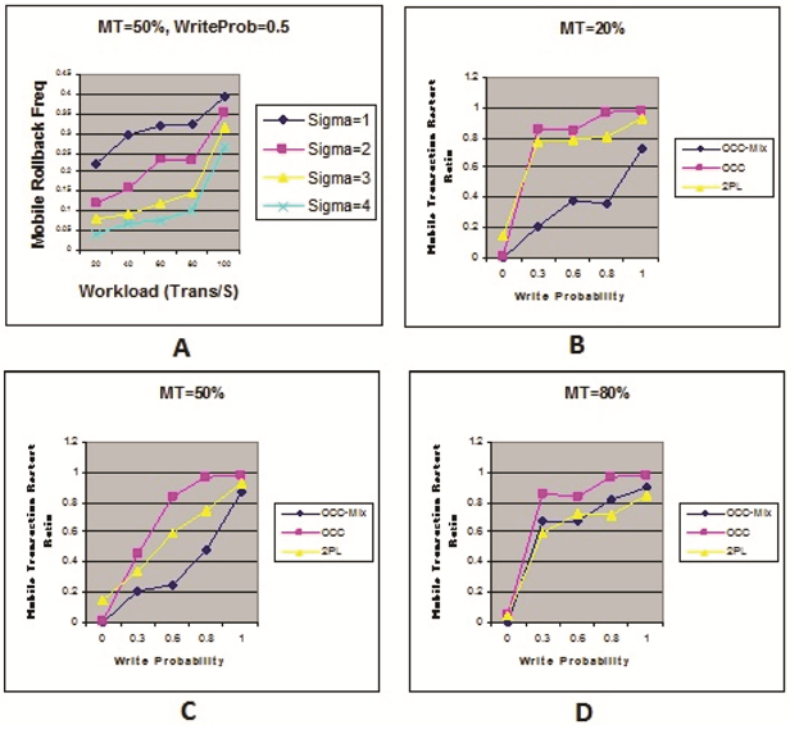

Figure 13 A. Fixed rollback Frequency, Figure 13 B. Mobile transaction restart ratio (MT = 20%), Figure 13 C. Mobile transaction restart ratio (MT = 50%), Figure 13 D. Mobile transaction restart ratio (MT = 80%)

Sigma value could be used to give an indication about the interactions between mobile and fixed transactions. Increasing the σ - value will increase the opportunity to allow fixed transactions to commit within time intervals. Figure 12(D) describes the fixed rollback frequency *FRF* when using different values of σ. This experiment determines the transactions that are aborted due to their conflict with other mobile-based transactions. Additionally, figure 12(D) shows the relationship between σ - value and the number roll backed fixed transactions. As a consequence, increasing the σ - value leads to increase the number of roll backed fixed transactions by other mobile transactions. When selecting the σ - value =2, we got the smallest value of *FRF*. Similarly, figure 13(A) shows that smaller σ - value results in more restarts in mobile-based transactions.

With respect to Mobile Rollback Frequency (MRF), adjusting an active mobile transaction that is in conflict with a bigger timestamp interval of a validating fixed transaction, will decrease the timestamp intervals of other mobile and fixed transactions which may be in conflict in the future. In summary, assigning the intermediate value of σ leads to best results in reducing the rollback frequency of both fixed and mobile, figure 12(D) and figure 13(A) summarize this conclusion.

7.3.3 Impacts write Probability

Different degree of data contention may affect the system performance when these protocols are applied. In these experiments, we simulate the transaction execution at different proportion of read and write operations in a mobile transaction. Figure 13(B) shows the restart rate for mobile transactions as the write probability varies when 20% transactions present in the system are mobile. It can be seen that there is no difference between the performances of the protocols at both ends of the write probability. When all operations are read, there is no data conflict and no adjustment is required. It makes no difference which *OCC* protocol is employed. On the other hand, when all operations are writing, the performance of the *OCC*-Mix protocol is also the same as the other two protocols because it is impossible to adjust the serialization order between the conflicting transactions since all data conflicts are serious. That is, the *OCC*-Mix protocol resembles the other two protocols when all operations are either read or write. In other words, the *OCC*-Mix protocol functions more effectively when transactions have a mix of both read and write operations. Figure 13(C) and Figure 13(D) give the restart rate when the mobile transactions percentages are 50% percent and 80%, respectively. Since there are both read and write operations in mobile transactions, when there is write-read conflict between a mobile transaction and a fixed transaction, the fixed transaction restart can be avoided by backward adjusting the fixed transaction.

One of the overheads of the *OCC*-Mix protocol is the adjustment of the serialization order for those transactions that are in conflict with the validating transaction. Figure 14(A) and Figure 14(B) give the adjustment ratio when the percentages of mobile transaction are 50% and 80%,

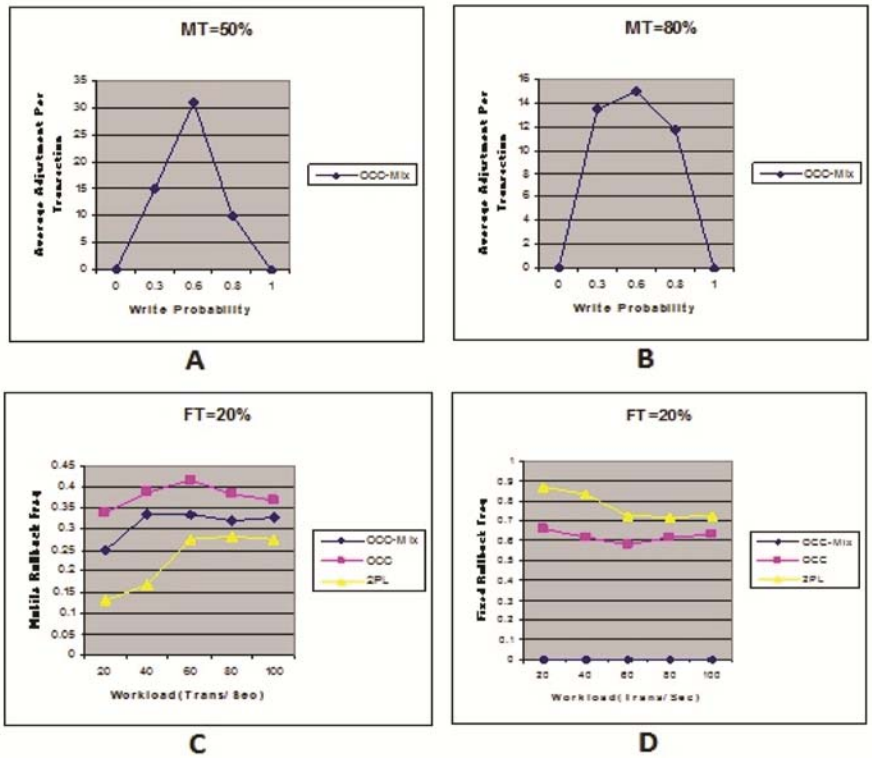

Figure 14(A). Average adjustment (MT = 50%), Figure 14(B). Average adjustment (MT = 80%), Figure 14(C). Mobile transaction rollback frequency (FT = 20%), Figure 14(D). Fixed transaction rollback frequency (FT = 20%)

respectively. This ratio in fact measures the effectiveness of the *OCC*-Mix protocol. When the ratio is zero, it means that either no dynamic adjustment is made in case of 100 percent read-only transactions or no dynamic adjustment can be made in case of 100 percent write-only transactions. The adjustment ratios reach the peak when there is a mixture of read and write operations in the system.

7.3.4 Impact of Workload

To have a deeper understanding of the data conflict between fixed and mobile transactions, two performance measures, called fixed transaction rollback frequency and mobile transaction rollback frequency, are collected. The frequencies indicate the amount of data conflict between fixed and mobile transactions and among mobile transactions, respectively. The later frequency also represents the fraction of committed mobile transactions which have roll-backed other mobile transactions. Figure 14(C) and Figure 14(D) gives the mobile and fixed transaction rollback frequency when there is 20% of transactions presences in the system are fixed. Most of data conflicts among mobile transactions. As the workload increases, data conflict intensifies and it is more likely for a transaction to rollback others in order to be committed. When the system begins to saturate, frequency decreases as the number of committed transactions decreases. For a transaction being restarted, there must be one other transaction to roll it back. Figure 15(A) and Figure 15(B) gives the frequencies when the utilization of fixed transactions is 50 percent. In Figure 15(A), it can be observed that the amount of data conflicts between mobile and fixed transactions increases as the number of fixed transactions increases until the system begins to saturate. In these two figures, it can also be observed for OCC-Mix protocol that it is more likely for a mobile transaction to rollback a fixed transaction than for a fixed transaction to rollback another fixed transaction, though the utilization of mobile transactions is equal to the fixed transaction. Since *OCC*-Mix protocol makes more room for mobile transaction, any fixed transactions that have data conflicts with a mobile transaction and its timestamp interval shut out it will be roll backed by the mobile transaction when it commits. A further increase of the utilization of mobile transactions exacerbates the situation. Figure 15(C) and Figure 15(D) give the frequencies when the utilization of fixed transactions is 80 percent. When the workload is low, under the 2PL protocol the data conflict between mobile and fixed transactions is increase. As the workload increases, data conflicts among mobile transactions increases. On the whole, it is observed that the *OCC*-Mix protocol can effectively help to reduce the number of unnecessary restarts whether they are due to data conflicts between mobile and fixed transactions or among mobile transactions. More over, even that *OCC*-Mix protocol negatively affect the number of committed fixed transaction, as the restart ratio of fixed transaction increase when the percentage of mobile transaction present in the system increase. This ratio is acceptable because the wasted resources concentrated in the fixed part of the network which can be tolerated especially when the performance gain in the scars wireless resources is high as we show in the restart ratio of mobile transaction.

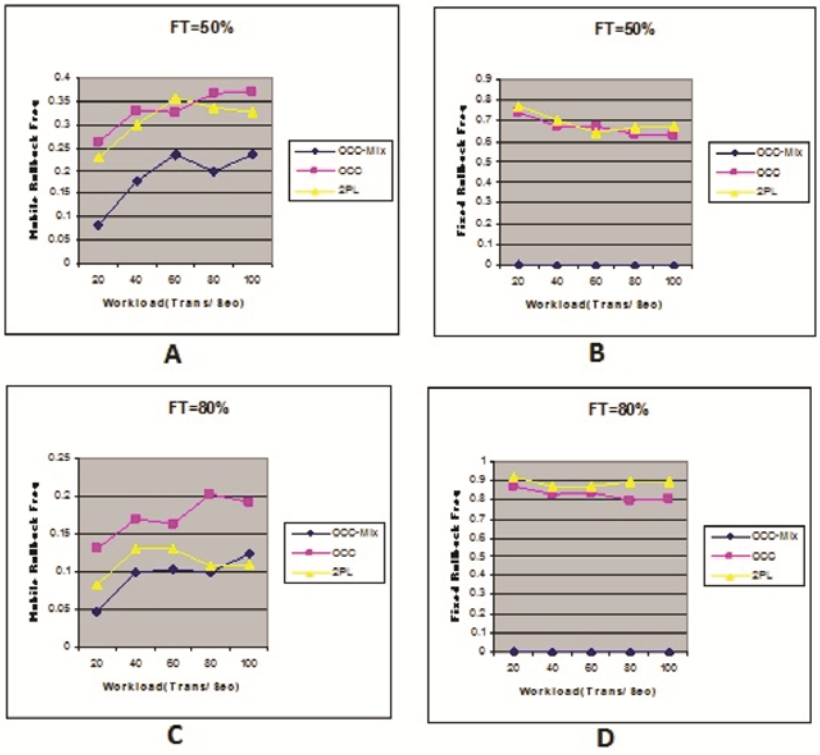

Figure 15(A). Mobile transaction rollback frequency (FT = 50%), Figure 15(B). Fixed transaction rollback frequency (FT = 50%), Figure 15(C). Mobile transaction rollback frequency (FT = 80%), Figure 16(D). Fixed transaction rollback frequency (FT = 80%)

8. Conclusion

This paper proposed the OCC-Mix protocol. The OCC-Mix could be used to overcome many problems in mixed transactions environment. The idea behind this protocol is to avoid the wasting of scars and save the resources of mobile environments. As a consequence, the OCC-Mix dynamically adjust the serialization order of the conflicting transactions with respect to the validating transaction, and hence this lead to decrease the unnecessary transaction restarts. Moreover, OCC-Mix allows mobile-based transactions to commit by making more room. The OCC-Mix protocol exploits the semantics between the operations of transaction (i.e. write and read operations). Additionally, it preserves the serializability by restarting the conflicting transactions of the validating transaction. However, the only situation that transactions needed to be restarted is when serious conflicts with the validating transaction. In case of non serious conflicts, adjusting the serialization order is required particularly for conflicting transactions with respect to the validating transaction. A set of simulation experiments were conducted to investigate the performance of the OCC-Mix approach with current optimistic concurrency control protocols. The results show that the proposed protocols outperform the traditional optimistic concurrency protocol in wide range related performance metrics. For example, OCC-Mix protocol decreases the number of mobile-based transaction restarts, which leads to a significant saving of resources. Besides, it improves the power consumption rate that is crucial to mobile computing environments.

References

[1] Lei, Xiangdong., Zhao, Yuelong., Chen, Songqiao., Yuan, Xiaoli. (2009). Concurrency control in mobiledistributed real-time database systems, *Journal of Parallel and Distributed Computing*, 69 (10).

[2] Holanda, Maristela., Brayner, Angelo., Fialho, Sergio. (2008). A Self-Adaptable Scheduler for Synchronizing Transactions in Dynamically Configurable Environments, 66 (2).

[3] Pitoura, E., Bhargava, B. K. (1999). Data Consistency in Intermittently Connected Distributed Systems, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11 (6), 896-915.

[4] Hugo, Rito., Cachopo, João. (2015). Adaptive Transaction Scheduling for Mixed Rransactional Workloads, *Parallel Computing*, 41, January.

[5] Kudo, Tsukasa., Takeda, Yui., Ishino, Masahiko., Saotome, Kenji., Kataoka, Nobuhiro. (2014). An Implementation of Concurrency Control between Batch Update and Online Entries, *Procedia Computer Science*, 35.

[6] Suh, Young-Kyoon., Richard., Snodgrass, T. (2017). Sabah Currim, "An empirical study of transaction throughput thrashing across multiple relational DBMSes", *Information Systems*, 66.

[7] Wang, Bin., Ma, Ruhui., Qi, Zhengwei., Yao, Jianguo., Guan, Haibing. (2016). A user mode CPU – GPU

scheduling framework for hybrid workloads, *Future Generation Computer Systems*, 63.

[8] Asta, Shahriar., Karapetyan, Daniel., Kheiri, Ahmed., Özcan, Ender., Andrew., Parkes, J. (2016). Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi- project scheduling problem, *Information Sciences*, 373.

[9] Li, Guohui., Yang, Bing., Chen, Jixiong. (2006). "Efficient Concurrency Control for Mobile Real-Time Transactions in Data Broadcast Environments, *Japan-China Joint Workshop on Frontier of Computer Science and Technology*.

[10] Mezzina, Antares, Claudio., Jorge., Pérez, A. (2017). Reversibility in session-based concurrency: A fresh look, *Journal of Logical and Algebraic Methods in Programming*.

[11] Jung, Sungwon., Choi, Keunha. (2009). A concurrency control scheme for mobile transactions in broadcastdisk environments, *Data & Knowledge Engineering*, 68 (10).

[12] Elizabeth, O×Neil, J., Patrick, E., OxNeil. (2016). Determining serialization order for serializable snapshot isolation, *Information Systems*, 58.

[13] Di, Sanzo., Pierangelo., Ciciani, Bruno., Palmieri, Roberto., Quaglia, Francesco., Romano, Paolo. (2012). On the analytical modeling of concurrency control algorithms for Software Transactional Memories: The case of Commit-Time-Locking, *Performance Evaluation*, 69 (5).

[14] Kavi, Krishna., Shelor, Charles., Pace, Domenico. (2015). Chapter Two - Concurrency, Synchronization, and Speculation—The Dataflow Way, *Advances in Computers*, 96.

[15] Ghica, D. R., Murawski, A. S., Ong, C. -H. L. (2006). Syntactic control of concurrency, *Heoretical Computer Science*, 350 (2–3).

[16] Imielinksi, T., Badrinath, B.R. (1994). Wireless mobile computing: challenges in data management, *Communications of ACM* 37.

[17] Celko, Joe. (2015). Chapter 2 - Transactions and Concurrency Control, Joe Celko's SQL for Smarties (Fifth Edition).

[18] Matthew, R., Lakin, Stefanovic, Darko., Phillips, Andrew. (2016). Modular verification of chemical reaction network encodings via serializability analysis, *Theoretical Computer Science*, 632.

[19] Härder, T. (1984). Observations on optimistic concurrency control schemes, *Information Systems*, 9 (2) 111-120.

[20] Dongol, Brijesh., Derrick, John. (2015). Interval-based data refinement: A uniform approach to trueconcurrency in discrete and real-time systems, *Science of Computer Programming*, 111 (2).

[21] Kung, H. T., Robinson, J. T. (1981). On Optimistic Methods for Concurrency Control, *ACM Transactions on Database Systems*, 6 (2) 213-226.

[22] Bai, Tian., Liu, YunSheng., Hu, Yong. (2008). Timestamp Vector Based Optimistic Concurrency Control Protocol for Real-Time Databases, *In*: 4th International Conference on Wireless Communications, *Networking and Mobile Computing*.

[23] Altaya., Mohammed, A., Murtada, K., Elbashir., Awadallah, Ahmed, M., Muslim, M., Ali, (2017). "Enhancement of Seamless mobility in Nested Network Mobility, *In*: International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE).

[24] Al-Qerem, A. (2014). Performance Evaluation of Transaction Processing in Mobile Data Base Systems , *International Journal of Database Management Systems* (*IJDMS*) 6 (2) April.

Author Biographies

Ahmad Al-Qerem graduated in Applied Mathematics and MSc in Computer Science at the Jordan University of Science and Technology in 1997 and 2002, respectively. After these degrees, he was appointed as full-time lecturer at the Zarqa University and also a part-time lecturer at the Arab Open University. He has also held a post in the Ministry of Labor. He obtained a PhD from Loughborough University, UK. His research interests are in performance and analytical modeling, mobile computing environments, protocol engineering, communication networks, transition to IPv6, and transaction processing. He has published several papers in various areas of computer science. Currently, he has a full academic post as Associate Professor and the Head of the Department of Internet Technology at Zarqa University, Jordan. He can be reached at ahmad_qerm@zu.edu.jo

Ala Hamarsheh is an Assistant Professor at the Faculty of Engineering and Information Technology of the Arab American University, Jenin. He obtained a PhD in engineering sciences from Vrije Universiteit Brussel (VUB)/Brussels-Belgium in 2012. He graduated in computer science at the Faculty of Science, Birzeit University, Palestine, in 2000. He obtained an MSc degree in computer science at the Kind Abdullah II School for IT, The University of Jordan, Jordan, in 2003. He has published numerous papers in international refereed journals and conferences. He can be reached at ala.hamarsheh@aauj.edu

Shadi Nashwan received his B.Sc. degree Computer Science from Alazhar University, Palestine, in 2001, and M.Sc. degree in Computer Science from the University of Jordan, Jordan, in 2003. In 2009, he received his PhD degree in computer science from the Anglia Ruskin University, United Kingdom and his PhD title was “Performance Analysis of a new Dynamic Authentication Protocol “DAKA” of 3G Mobile Systems based on a novel Cryptography Algorithm “Anglia””. Dr. Nashwan is currently the Assistant Professor in Computer Science and Information Department, Aljouf University, Saudi Arabia. His research focuses on authentication protocol of mobile network, mobility management, and wireless network security. He has published several papers in the area of authentication protocol, recovery techniques and mobility management. He can be reached at shadi_nashwan@ju.edu.sa